சி++ மொழி

C++

ந.செல்வகுமார் (B.Sc.)

சி++ மொழி

(முதற்பதிப்பு)

ந. செல்வகுமார் (B.Sc.)
(கொழும்புப் பல்கலைக்கழகம்)

வெளியீடு :
IIS (Institute of Informatics Studies)
கணினிக்கல்வி நிலையம்,
100, மணிக்கூட்டு வீதி,
யாழ்ப்பாணம்,
இலங்கை.

நூலின் பெயர் : **சி$^{++}$ மொழி**

முதற்பதிப்பு : ஜனவரி, 2003

ஆசிரியர் : ந. செல்வகுமார்

பதிப்புரிமை© : ஆசிரியர்

இந்நூலில் காணப்படும் ஆக்கங்களை ஆசிரியரின் எழுத்துமூல அனுமதியின்றி பிரதி செய்யப்படலாகாது.

வெளியீடு : யாழ் **IIS** கணினிக்கல்வி நிலையம்

பக்கங்கள் : 218 + x

விலை : ரூ. 250/=

ISBN எண் : 955-97996-0-6

This Book cannot be reproduced in any form without the written permission of the Author.

Title of the Book : **C^{++} Mozhi**

First Edition : January, 2003

Author : N. Selvakumar

Copyright© : Author

Published by : IIS Computer Institute (Jaffna)

Pages : 218 + x

Price : Rs. 250/=

ISBN : 955-97996-0-6

அணிந்துரை

சி$^{++}$ பற்றிய இந்நூலானது, இன்றைய கணினி விஞ்ஞான நூற்களஞ்சியத்தில் தகுந்த ஒரு சேர்ப்பாக அமையும் என்பதில் ஐயமில்லை. இந்நூல் தெளிவாக எழுதப்பட்டுள்ளதுடன், குறிப்பாக நிரற்படுத்துகை (Programming) இன் அடிப்படைக் கொள்கைகளை விளக்குவதில் சிறப்பாக உள்ளது.

இந்நூலினை எழுதுவதில் ஆசிரியர் இரண்டு முக்கியமான கற்கை நெறித் தீர்மானங்களை எடுத்துள்ளார். இவை நிரற்படுத்தும் மொழியையும், இந்நிரற்படுத்தும் மொழியைக் கற்பிக்கப் பயன் படுத்தும் மொழியையும் சாரும்.

நிரற்படுத்தலைக் கற்பிக்கும் பொழுது சிலர் இலகுவான Pascal, BASIC போன்ற நிரற்படுத்தும் மொழியொன்றைக் கற்பித்த பின்னர், சி$^{++}$ போன்ற கடினமானதும், வலுவானதுமான மொழி யினைக் கற்பிக்கிறார்கள். ஆனால், சிலர் கடினமான மொழிகளை நேரடியாகக் கற்பிக்கிறார்கள். இதுவே பல பல்கலைக்கழகங்களில் உள்ள வழக்கமாகும். இதையே ஆசிரியர் ந. செல்வகுமாரும் தெரிவு செய்துள்ளார்.

கணினி மொழிகளைக் கற்பிக்கும் மொழியைப் பொறுத்த மட்டில், அது மிகவும் சிக்கலானதும் தர்க்கத்துக்குரிய விடய மாகவும் காணப்படுகிறது. சிலர் முற்றாக ஆங்கில மொழியிலேயே கற்பிக்கிறார்கள். எனினும் சிலர், முற்றாகத் தமிழ் மொழியிலேயே கற்பிக்கிறார்கள். எமது தாய் மொழியில் கற்பிப்பதே சிறந்த தெனினும், ஆங்கிலத்தில் கற்பிப்பதற்கான காரணங்கள் யாதெனில், பல நவீன தகவல் தொழில்நுட்ப நூல்களும், வர்த்தக வாய்ப்புக் களும் ஆங்கில மொழியிலேயே காணப்படுகின்றன. சிலர் தமிழில் கற்பித்து, நிரல்களையும் (Programs), விஷேட சொற்களையும் (Keywords), சில பிரதான சொற்களையும் ஆங்கிலத்தில் உபயோகிக்கின்றார்கள். இதையே ஆசிரியர் பின்பற்றியுள்ளார். இது வரவேற்கத்தக்க விடயமாகும். இவ்வாறு கற்பிப்பதன் மூலம், நவீன தகவல் தொழில்நுட்ப அறிவினையும், வர்த்தக வாய்ப்புக் களையும் மாணவர்கள் இலகுவாகப் பெற முடியும்.

ஆங்கிலச் சொற்களைத் தமிழில் எழுதும் போது உச்சரிப்பு சார்ந்த பல சிக்கல்களை எதிர்நோக்க வேண்டியிருக்கும்.

உதாரணமாக, Data என்ற ஆங்கிலச் சொல்லின் உச்சரிப்பைத் தமிழில் எழுதும் போது டேட்டா, டேற்றா, இடேற்ரா, டேற்ரா போன்றவற்றைப் பயன்படுத்துகிறார்கள். இந்நூலில், ஆசிரியர் இறுதியாகக் குறிப்பிடப்பட்ட டேற்ரா என்ற முறையைத் தெரிவு செய்துள்ளார். இவ்வாறு பல ஆங்கிலச் சொற்களின் உச்சரிப்பினைத் தமிழில் எழுதியுள்ளார். இது இவ்வாறிருக்க, சிலர் விஷேட சொற்களையும் (Keywords), சில பிரதான சொற்களையும் தமிழ் மொழிமாற்றம் செய்து கற்பிக்கிறார்கள். இவ்வழியானது தமிழில் கற்போருக்கு உபயோகமாக இருப்பினும், நிரற்படுத்தல் துறையின் சிகரங்களுக்கு மாணவர்களை எடுத்துச் செல்லாது. மற்றும், வர்த்தக வாய்ப்புக்களைப் பெறுவதிலும் அவர்களுக்குச் சிரமமாக இருக்கும்.

இளம் எழுத்தாளரான ந. செல்வகுமார், இந்நூலினை எழுதுவதில் காட்டிய உற்சாகத்தையும், ஊக்கத்தையும் நான் மெச்சுகிறேன். இவர் காட்டும் வழியை மற்றவர்களும் தொடர்ந்தால், இலங்கை ஒரு கணினி அறிவு சார்ந்த நாடாக மாறும் என்பதில் ஐயமில்லை. ஆசிரியருக்கு எனது பாராட்டுக்கள்.

சா. இரத்தினஜீவன் ஹே. ஹூல் B.Sc. Eng. Cey.,
M.Sc. (Eng.) Distinction. Lond., Ph.D. C.M.U.,
D.Sc. (Eng.) Lond., IEEE Fellow,
பேராசிரியர் - மின்பொறியியல்,
பொறியியற் பீடம்,
பேராதனைப் பல்கலைக்கழகம்,
பேராதனை.

முகவுரை

இன்றைய தகவல் தொழில்நுட்ப உலகில், கணினிக்கல்வி அனைவருக்கும் அவசியமானதொன்றாக அமைந்துவிட்டது. அதனால் இன்று பலரும் கணினிக் கற்கைநெறிகளைக் கற்கிறார்கள். இவர்கள் எதிர்நோக்குகின்ற முக்கிய பிரச்சினை, கற்றலுக்கான துணை நூல்கள் தமிழில் இன்மையே!

இன்று கற்றலுக்கான துணை நூல்கள் அதிகம் ஆங்கில மொழியிலேயே காணப்படுகின்றன. சில நூல்கள் தமிழில் வெளி வந்துள்ள போதும், அவற்றிலுள்ள தமிழ் மொழிநடையை வாசித்துத் தெளிதலிலேயே போதும், போதும் என்றாகிவிடுகிறது.

இந்தத் தேவையைக் கருத்திற் கொண்டு உருவாகிய இலங்கைத் தேசிய கணினிச் சஞ்சிகைகளில் எழுத்தாளராக அறிமுகமாகி, இன்று ஒரு முழுமையான நூலையே தருமளவுக்கு வளர்ந்துள்ள ந. செல்வகுமாரின் இந்தப் புதிய முயற்சி, கணினி மொழியொன்றைத் தமிழிலேயே கற்றுத் தருவதற்கான ஆசானாக அமைகின்றது.

மிக இலகுவான மொழிநடையில் அமைந்துள்ள இந்நூல், நிலாச்சோறு ஊட்டுவது போல எளிய உதாரணங்களுடன் விளக்கிச் சொல்லப்பட்டுள்ளது. கணினி மொழி பற்றிய அடிப்படை அறிவு இல்லாதவர்கள்கூட கற்றுத் தேறக்கூடியதாக அமைந்துள்ள இந் நூலின் அமைப்பும் பாராட்டத்தக்கதாகவுள்ளது. கணினி உயர்நிலை மொழியான சி++ மொழியானது, கடினமானது என்ற மாயையை நூலாசிரியர் போக்கியிருக்கிறார்.

சி++ மொழி பயில விரும்பும் எல்லோரும் வாங்கி, ஆற அமர இருந்து படிக்கத்தக்கதாக அமைந்துள்ள இந்நூலானது, ஏனைய கணினி உயர்நிலை மொழிகளைக் கற்கப்போகின்றவர்களுக்கும் அடிப்படைத் தளமாக உதவவல்லது.

கணினி உயர்நிலை மொழியான சி++ இனைத் தமிழில் கற்க விரும்புவர்களுக்கான ஒரு வரப்பிரசாதமாக அமைந்த இந்நூலின் ஆசிரியரான ந. செல்வகுமாரின் இந்த முயற்சி பாராட்டத்தக்கது. இந்நூலாசிரியர் இது போன்ற பல கணினி உயர்நிலை மொழிக்கான நூல்களைத் தமிழில் நூலாக்கித் தரவேண்டும் என்பதே அனைவரதும் விருப்பமாகும்.

வே. நவமோகன்

(ஆசிரியர், கணினி வழிகாட்டி)

ஆசிரியர் உரை

இன்று கணினியானது எல்லாத்துறைகளிலும் பெரிதும் பயன்படுத்தப்பட்டு வருகின்றது. அதனால் மாணவர்கள், அலுவலக வேலை செய்பவர்கள் எனப் பாகுபாடின்றி அனைவரும் கணினிக் கற்கைநெறிகளைப் பயின்று வருகிறார்கள். பல புதிய தொகுப்புக்கள் (Packages) வெளிவந்துள்ள போதிலும், கணினி உயர்நிலை மொழிகளின் பயன்பாடு குறையவில்லை என்றே கூறமுடியும். ஏனெனில், தொகுப்புக்கள் மூலம் குறித்த சில பயன்பாடுகளை மட்டுமே செயற்படுத்த முடியும். எடுத்துக்காட்டாக, மைக்ரோசொவ்ற் வேர்ட் (Microsoft Word) இனைப் பயன்படுத்தி, கடிதம் சார்ந்த விடயங்களையே எழுத முடியும். ஆனால், கணினி உயர்நிலை மொழிகளைப் பயன்படுத்தி, எமக்குத் தேவையான எந்த வகையான செயற்பாடுகளையும் எழுதிப் பயன்படுத்த முடியும்.

இன்று எமது நாட்டில் கற்பிக்கப்படும் கணினிக் கற்கைநெறி களில் பலரும் விரும்பிக் கற்கும் கற்கைநெறிகள் தொகுப்புக்கள் (Packages) ஆகும். எடுத்துக்காட்டாக, எம்எஸ் ஓ.ஃ.பிஸ் (MS Office), போற்ரோஷொப் (Photoshop), பேஜ்மேக்கர் (Page-Maker) போன்றவற்றைக் குறிப்பிடலாம். இந்தத் தொகுப்புக்களை வெளியிட்ட நிறுவனங்களினால் ஏற்கனவே எழுதப்பட்ட செயற்பாடுகளை மட்டுமே எம்மால் பயன்படுத்த முடியும். எனவே, எமக்கு ஓர் புதிய செயற்பாடு தேவைப்பட்டால், இந்தத் தொகுப்புக்கள் மூலம் செய்ய முடியாது. இந்நிலையில்தான் நாம் கணினி உயர்நிலை மொழிகளை நாடவேண்டியுள்ளது. இன்று எமது நாட்டில் விசுவல் சி++ (Visual C++), விசுவல் பேசிக் (Visual Basic) போன்ற மொழிகளைப் பயன்படுத்தியே பல தொகுப்புக்கள் எழுதப்படுகின்றன.

ஜாவா (Java), விசுவல் பேசிக் (Visual Basic), விசுவல் சி++ (Visual C++), சி ஷாப்(C#) போன்ற பல புதிய கணினி உயர்நிலை மொழிகள் பயன்பாட்டுக்கு வந்தபோதிலும், இந்த மொழிகள் அனைத்தும் சி++ மொழியினை அடிப்படையாகக் கொண்டே எழுதப்பட்டவையாகும். எனவே, எந்தப் புதிய கணினி உயர்நிலை மொழிகளைக் கற்பதற்கும், இந்த சி++ மொழியானது எமக்குப் பெரிதும் உதவிபுரியும் என்பதில் ஐயமில்லை.

- நன்றி -

ந. செல்வா

பொருளடக்கம்

அணிந்துரை ... iii

முகவுரை ... v

ஆசிரியர் உரை ... vi

நுழைவாயிலில் ... x

1. அறிமுகம் ... 1

1.1 கணினி உயர்நிலை மொழிகளின் அவசியம்

1.2 கணினி உயர்நிலை மொழிகளின் அறிமுகம்

1.3 சி++ கொம்பைலர்கள் (C++ Compilers)

1.4 சி++ மொழிப் புறோகிராம் அமைப்பு

1.5 சி++ மொழிப் புறோகிராம் அடிப்படைகள்

2. மாறிகளும், மாறிலிகளும் (Variables & Constants)...19

2.1 மாறிகள், மாறிலிகளின் அடிப்படை விபர இனங்கள்

2.2 மாறிகளின் பிரயோகங்கள்

2.3 மாறிலிகளின் பிரயோகங்கள்

3. ஒப்பறேற்றர்கள் (Operators) ... 36

3.1 அசைன்மென்ற் ஒப்பறேற்றர்கள் (Assignment Operators)

3.2 அரித்மெற்றிக் ஒப்பறேற்றர்கள் (Arithmatic Operators)

3.3 றிலேஷனல் ஒப்பறேற்றர்கள் (Relational Operators)

3.4 லொஜிக்கல் ஒப்பறேற்றர்கள் (Logical Operators)

3.5 பிற்வைஸ் ஒப்பறேற்றர்கள் (Bitwise Operators)

4. கட்டுப்பாட்டுக் கட்டளைகள் (Control Statements)...46

4.1 தீர்வுசெய் கட்டளைகள் (Selection Statements)

- if .... else ....
- switch .... case

4.2 சுழற்சிக் கட்டளைகள் (Iteration Statements)

- for loop
- while loop
- do ... while loop

4.3 தாவும் கட்டளைகள் (Jump Statements)

- break
- continue
- return
- goto

5. ஃபங்ஷன்கள் (Functions) ... 65

5.1 உள்ளிணைந்த ∴பங்ஷன்கள்
(Library Functions)

5.2 நாமே உருவாக்கிக் கொள்ளும் ∴பங்ஷன்கள்
(User Defined Functions)

6. அரேக்கள், ஸ்ரக்சர்கள் மற்றும் பொயின்ரர்கள் (Arrays, Structures and Pointers) ... 82

6.1 அரேக்களின் வகைகளும், அவற்றின் பிரயோகங்களும்

6.2 ஸ்ரக்சர்களின் பிரயோகங்கள்

6.3 பொயின்ரரும், அவற்றின் பிரயோகங்களும்

7. ஒப்ஜெக்ற் ஓரியன்ரட் புறோகிராமிங் (OOP - Object Oriented Programming) இன் விளக்கங்களும், பிரயோகங்களும் ... 105

7.1 கிளாஸ்களும், ஒப்ஜெக்ற்களும் (Classes and Objects)

7.2 என்கப்சுலேஷன் (Encapsulation)

7.3 இன்ஹெரிற்ரன்ஸ் (Inheritance)

- சிங்கிள் இன்ஹெரிற்ரன்ஸ்
  (Single Inheritance)
- மல்ரி லெவல் இன்ஹெரிற்ரன்ஸ்
  (Multi-level Inheritance)
- மல்ரிபில் இன்ஹெரிற்ரன்ஸ்
  (Multiple Inheritance)
- ஹைரார்சிகல் இன்ஹெரிற்ரன்ஸ்
  (Hierarchical Inheritance)

7.4 பொலிமோபிஸம் (Polymorphism)

- ஓவலோடிங் (Overloading)
  - ∴பங்ஷன் ஓவலோடிங் (Function Overloading)
  - ஒப்பறேற்ரர் ஓவலோடிங் (Operator Overloading)
- ஓவறைடிங் (Overriding)

7.5 அப்ஸ்ரக்ற் கிளாஸ்கள் (Abstract Classes)

7.6 ∴பிரண்ட் ∴பங்ஷன்களும், ∴பிரண்ட் கிளாஸ்களும்
(Friend Functions and Friend Classes)

8. ரெம்ப்லேற்கள் (Templates) ... 171

8.1 ரெம்ப்லேற் ∴பங்ஷின்கள் (Template Functions)

8.2 ரெம்ப்லேற் கிளாஸ்கள் (Template Classes)

9. ஃபைல்கள் (Files) ... 180

10.1 ∴பைல்களில் தரவுகளை உள்ளீடு செய்தல்

10.2 ∴பைல்களிலுள்ள தரவுகளைக் கையாளுதல்

10. பிழைகளும், பிழையேந்திகளும்

(Errors and Exception Handling) ... 189

9.1 பிழைகளின் வகைகள்

9.2 பிழையேந்திகளின் செயற்பாடுகள்

11. நாமே உருவாக்கிக் கொள்ளும் ஹெடர் ஃபைல்கள்
(User defined Header files) ... 197

பிற்சேர்க்கை (Appendices)

Appendix A ... 201

உதாரணப் புறோகிராம்களும், வெளியீடுகளும்

Appendix B ... 211

அஸ்க்கீ அட்டவணை (ASCII Table)

Appendix C ... 214

விசுவல் சி++ எடிற்ரரில், சி++ மொழிப் புறோகிராம்களைச் செயற்படுத்தல்.

இன்டெக்ஸ் (Index) ... 217

நுழைவாயிலில்

இந்நூலின் மூலமாக சி++ மொழி பற்றிய அடிப்படை, இடைநிலை அறிவினைத் தமிழ் பேசும் மாணவர்களுக்கு வழங்குவதில் நான் பெரும் மகிழ்ச்சி அடைகிறேன்.

இந்நூலினை எழுதுவதற்கு முன்னர் பல கேள்விகளுக்கு எனக்கு நானே பதில் சொல்ல வேண்டியவனாய் இருக்கிறேன். முதலில், இந்நூலில் சர்வதேச கணினி கலைச்சொற்களையா? அல்லது ஆங்கிலச் சொற்களையா? அல்லது தமிழ்க் கலைச்சொற் களையா? எதைப் பயன்படுத்துவது. நிச்சயமாகத் தமிழ்மொழி மூலம் வெளியிடும் இந்நூலில், தமிழ் சொற்களையே பயன்படுத்த வேண்டும். ஆனால், இங்கு பிரச்சினை அதுவல்ல. எந்தத் தமிழ் சொல்லைப் பயன்படுத்துவது என்பதே பிரச்சினையாகும். இப்படியான சூழ்நிலையில், தகவல் தொழில்நுட்பம் சம்பந்தமான ஆங்கிலச் சொற்களை எவ்வாறு தமிழில் அழைப்பது? ஆங்கிலத்தின் மொழி பெயர்ப்பாக நானே ஒரு தமிழ்ச்சொல்லை உருவாக்குவதா? இது சிறந்ததாக எனக்குப்படவில்லை. ஒவ்வொருவரும் தமது அறிவுக்கு எட்டியவரை புதிய கலைச்சொற்களை உருவாக்கினால், குறை நீங்குவதற்குப் பதிலாகக் குழப்பமே ஏற்படும். எனவேதான், நான் இங்கு ஆங்கிலச் சொற்களை அப்படியே தமிழ் உச்சரிப்பில் எழுதியுள்ளேன். அத்துடன், அவ்வாங்கிலச் சொற்களையும் அந்தத் தமிழ் உச்சரிப்பிற்கருகில் குறிப்பிட்டுள்ளேன். எனவே, ஆங்கில அறிவு குறைந்தவர்கள்கூட இந்நூலினைப் படித்துப் பயன்பெற முடியும். சொல்லப்போனால், கணினி பற்றி அறிவு இல்லாதவர்களும் பயன்பெறும் வகையில் இந்நூலினை எழுதியுள்ளேன்.

மேலும், சி++ மொழி இவ்வளவுதான் என்று யாரும் வரையறுக்க முடியாது. எனவே, நீங்கள் இந்நூலில் உள்ள உதாரணப் புறோகிராம்களையும், மேலும் பல சி++ மொழிப் புறோகிராம்களையும் செயல்முறை ரீதியாகப் பயிற்சி செய்து, இம்மொழி பற்றிய அறிவினை வளர்த்துக் கொள்ளுங்கள். சி++ மொழியினை வெறும் வாசிப்பினால் மட்டும் கற்றுவிட முடியாது. செயல்முறை ரீதியாகப் பல வகையான சிக்கல்களுக்குரிய தீர்வினை, சி++ மொழிப் புறோகிராம் மூலம் காண முயற்சிக்க வேண்டும்.

நீங்களும் சி++ மொழியில் தேர்ச்சி பெற்று, ஒரு சிறந்த கணினி மென்பொருள் உருவாக்கும் நிபுணராக வர வேண்டும் என்பதே எனது அவாவாகும்.

- ஆசிரியர் -

1. அறிமுகம்

1.1 கணினி மொழிகளின் அவசியம்

கணினி என்பது ஓர் இயந்திரமாகும். இவ்வியந்திரமானது நாம் பயன்படுத்தும் விதத்திலேயே தனது திறனை வெளிப்படுத்தும். இக்கணினி இயந்திரமும், மனித மூளையும் இணையும் போதே செயல்முறைகளை மிக வேகமாகவும், துல்லியமாகவும் செயற்படுத்த முடிகிறது. எனவே, குறித்தொரு செயற்பாட்டினைக் கணினியின் மூலம் செய்யும் போது, அதன் செயற்திறன் கணினியில் மட்டும் தங்கியிருப்ப தில்லை, மாறாக மனித மூளையிலும் தங்கியுள்ளது.

மனிதர்கள் தமக்கிடையே கருத்துக்களைப் பரிமாறிக்கொள்வதற்கு மொழிகள் உருவாக்கப்பட்டது போல், மனிதனுக்கும் கணினிக்கும் இடையே கருத்துக்களைப் பரிமாறிக்கொள்வதற்கு கணினி உயர்நிலை மொழிகள் உருவாக்கப்பட்டன.

இக்கணினி உயர்நிலை மொழிகளானது சாதாரண மொழிகளை ஒத்ததாகும். உதாரணமாக தமிழ், ஆங்கிலம் போன்ற மொழிகளை ஒத்ததாகும். தமிழ், ஆங்கிலம் போன்ற மொழிகளில் கோடிக்கணக்கான சொற்கள் காணப்படுவது போல், கணினி உயர்நிலை மொழிகளில் 30 தொடக்கம் 60 வரையிலான சொற்கள் மட்டுமே விஷேட சொற்கள் (Keywords) ஆகக் காணப்படுகின்றன. இவ்விஷேட சொற்களைப் பயன்படுத்தியே கணினி உயர்நிலை மொழிப் புறோகிராம்கள் எழுதப்படுகின்றன.

கணினி இயந்திரமானது, மனிதனால் எழுதப்பட்ட மென்பொருள்கள் (Softwares) மூலமே கட்டுப்படுத்தப்படுகின்றது. உதாரணமாக, ஒப்பறேற்றிங் சிஸ்ரம் (Operating System) மூலமும், கணினி உதிரிப் பாகங்களுக்குரிய ட்ரைவர் மென்பொருள்கள் (Driver Softwares) மூலமும் கணினியினைக் கட்டுப்படுத்தப்படுகின்றது.

எமக்குத் தேவையான செயற்பாடுகளைக் கணினி உயர்நிலை மொழிப் புறோகிராம்கள் மூலம் மிக வேகமாகவும், துல்லியமாகவும் செயற்படுத்த முடியும். இக்கணினி உயர்நிலை மொழிப் புறோகிராம் களை யார் வேண்டுமானாலும் எழுத முடியும். ஆயினும், மிகச்சிறந்த முறையில் புறோகிராம்களை எழுத வேண்டுமாயின், சிறந்த சிந்தனை ஆற்றலும், அனுபவமும் இருக்க வேண்டும்.

1.2 கணினி மொழிகளின் அறிமுகம்

ஆரம்ப காலத்தில், கணினிக்குப் புரியும் மொழியான இயந்திர மொழி (Machine Language) இனைப் பயன்படுத்தியே புறோகிராம்கள் எழுதப்பட்டன. பின்னர், இடைநிலை மொழியான அசெம்பிளி மொழி (Assembly Language) இனைப் பயன்படுத்திப் புறோகிராம்கள் எழுதப்பட்டன. இந்த அசெம்பிளி மொழியினைக் கணினிக்குப் புரியும் மொழியான இயந்திர மொழிக்கு மாற்றுவதற்கு அசெம்பிளர் (Assembler) என்ற மொழிமாற்றி பயன்படுத்தப்பட்டது.

1957 ஆம் ஆண்டில் தான், முதலாவது உயர்நிலை மொழியான ஃபோர்ரான் (FORTRAN - Formula Translation) மொழி வெளியிடப்பட்டது. இந்த மொழியானது கணித, அறிவியல் சார்ந்த கணக்கீடுகளுக்கு மட்டுமே பயன்படுத்தப்பட்டது.

1961 ஆம் ஆண்டின் ஆரம்பப் பகுதியில் வர்த்தக நோக்கம் கருதி, கோபோல் (COBOL - COmmon Business Oriented Language) என்ற மொழி வெளியிடப்பட்டது. இன்றும், இந்த கோபோல் (COBOL) என்ற மொழியானது, பல புதிய பதிப்புக்களுடன் பெரிய நிறுவனங்களில் பயன்படுத்தப்பட்டு வருகின்றது. பின்னர், சில ஆண்டுகளுள் அல்கோல் (ALGOL), பேசிக் (Basic), சிமுலா (Simula) போன்ற பல மொழிகள் வெளியிடப்பட்டன.

1970 ஆம் ஆண்டின் நடுப்பகுதியில் போர்லாண்ட் (Borland) நிறுவனத்தினால், பஸ்கல் (Pascal) என்ற உயர்நிலை மொழி வெளியிடப்பட்டது. இந்தப் பஸ்கல் மொழியானது கற்பதற்கு மிகவும் இலகுவானதாகவும், கட்டமைப்புடையதாகவும் காணப்படுகின்றது.

போர்லாண்ட் நிறுவனத்தின் மற்றுமோர் வெளியீடான சி (C) மொழியானது, 1971 ஆம் ஆண்டு வெளியிடப்பட்டது. அன்றிலிருந்து இன்றுவரை சி மொழியைத் தழுவியே மற்றைய புதிய மொழிகள் வெளியிடப்படுகின்றன. சி மொழியினைப் பயன்படுத்திப் பல தொகுப்புக்கள் (Packages), கணினி மொழிகளுக்குரிய மொழிமாற்றிகள் (Compilers), ஒப்பறேற்றிங் சிஸ்ரம்கள் (Operating Systems) எழுதப்படுகின்றன. மற்றும் சி மொழியினைப் பயன்படுத்தியே கணினி உதிரிப்பாகங்களுக்குரிய ட்ரைவர் மென்பொருள்கள் (Driver Softwares) அதிகம் எழுதப்படுகின்றன.

முதன் முதலில் வெளிவந்த ஒப்பறேற்றிங் சிஸ்ரமான யுனிக்ஸ் (Unix), 1969 ஆம் ஆண்டு வெளியிடப்பட்டது. பின்னர், சி மொழியினைப் பயன்படுத்தி எழுதப்பட்ட யுனிக்ஸ் (Unix) ஒப்பறேற்றிங் சிஸ்ரமானது புதுப்பொலிவுடன் 1973 ஆம் ஆண்டு வெளியிடப்பட்டது.

ஒப்ஜெக்ற் ஒரியன்ரட் என்ற கட்டமைப்பைப் பயன்படுத்த முடியாது என்பதே சி மொழியில் காணப்படும் ஒரு குறைபாடாகும்.

இன்று மென்பொருள்துறையில் முன்னணியில் காணப்படும் நிறுவனமான மைக்ரோசொவ்றின் நிறுவுநர் பில்கேட்ஸால் கியூடபில்யூ பேசிக் (QW Basic) மொழி 1975 ஆம் ஆண்டு வெளியிடப்பட்டது.

1980 ஆம் ஆண்டு வீட்டுப் பாவனை கணினிகளுக்குரிய ஒப்பறேற்றிங் சிஸ்ரமான பிசி டொஸ் (PC DOS - Personal Computer Disk Operating System) இனை ஐபிஎம் (IBM), மைக்ரோசொவ்ற் (Microsoft) போன்ற நிறுவனங்கள் இணைந்து வெளியிட்டன. பின்னர், 1981 ஆம் ஆண்டு மைக்ரோசொவ்ற் நிறுவனத்தினால் எம்எஸ் டொஸ் (MS DOS - Microsoft Disk Operating System) என்ற ஒப்பறேற்றிங் சிஸ்ரம் வெளியிடப்பட்டது.

சி மொழியின் மேம்பட்ட பதிப்பாகக் கருதப்படும் சி$^{++}$ (C^{++}) மொழியானது, போர்லாண்ட் நிறுவனத்தினால் 1983 ஆம் ஆண்டு வெளியிடப்பட்டது. சி மொழியில் காணப்படுகின்ற சகல செயற்பாடுகளையும் தன்னகத்தே கொண்டிருப்பதுடன், சி மொழியில் காணப்படாதிருந்த ஒப்ஜெக்ற் ஒரியன்ரட் (Object Oriented) என்ற கட்டமைப்பும் புகுத்தப்பட்டே சி$^{++}$ மொழி வெளியிடப்பட்டது.

பின்னர் சில ஆண்டு காலமாக DBase, Foxpro, Foxbase, Clipper, Oracle போன்ற தகவல் தளத்துடன் (DataBase) சேர்ந்த தொகுப்புக்கள் வெளியிடப்பட்டன.

1991 ஆம் ஆண்டளவில், கணினித்துறையின் பொற்காலம் ஆரம்பிக்கப்பட்டது எனச் சுருக்கமாகக் கூறமுடியும். ஏனெனில், 1991 ஆம் ஆண்டு வரை கணினியுடன் தொடர்பு கொள்வதற்கு எழுத்து வடிவிலான கட்டளைகளே பயன்படுத்தப்பட்டன. அதாவது, யுனிக்ஸ் (Unix), பிசி டொஸ் (PC DOS), எம்எஸ் டொஸ் (MS DOS) போன்ற ஒப்பறேற்றிங் சிஸ்ரங்களில், கணினியுடன் தொடர்பு கொள்வதற்குரிய கட்டளைகள் யாவும் எழுத்து வடிவிலேயே பயன்படுத்தப்பட்டன. ஆனால், 1991 ஆம் ஆண்டின் இறுதியில் மைக்ரோசொவ்ற் நிறுவனத்தினால் வெளியிடப்பட்ட வின்டோஸ் ஒப்பறேற்றிங் சிஸ்ரத்தில் (Beta Version), கணினியுடன் தொடர்பு கொள்ளும் கட்டளைகள் யாவும் படக்குறியீடுகள் (Icons) மூலம் செயற்படுத்த முடியுமாயிருந்தது.

இதே 1991 ஆம் ஆண்டில், ஜாவா மொழி உருவாகக் காரணமாக அமைந்த ஓக் (Oak) மொழி வெளியிடப்பட்டது. பின்னர், இந்த ஓக் என்ற மொழியானது புதுப்பொலிவுடன் ஜாவா (Java) எனப் பெயர் மாற்றம் செய்யப்பட்டு, 1995 ஆம் ஆண்டு சன் மைக்ரோசிஸ்ரம்ஸ் (Sun Microsystems) நிறுவனத்தினால் வெளியிடப்பட்டது.

1992 ஆம் ஆண்டு மைக்ரோசொவ்ற் நிறுவனத்தின் மற்றுமொரு ஒப்பறேற்றிங் சிஸ்ரமான வின்டோஸ் 3.1 (Windows 3.1) என்ற உண்மைப் பதிப்பு (Real Version) வெளியிடப்பட்டது. அதன் பின்னர், 1994 ஆம் ஆண்டு விசுவல் பேசிக் (Visual Basic) என்ற மொழியினை மைக்ரோசொவ்ற் நிறுவனம் வெளியிட்டது.

இன்ரநெற்றின் முக்கிய அங்கமான வெப் பக்கங்களை (Web Pages) உருவாக்கப் பயன்படும் மொழியான எச்ரிஎம்எல் (HTML - HyperText Markup Language) 1993 ஆம் ஆண்டு வெளியிடப்பட்டது.

இன்று மென்பொருள்கள் தயாரிக்கும் பல நிறுவனங்களால் பயன் படுத்தப்படும் விசுவல் பேசிக் (Visual Basic), விசுவல் சி++ (Visual C++) போன்ற மொழிகள் அடங்கிய தொகுப்பான விசுவல் ஸ்ருடியோ (Visual Studio) 1997 ஆம் ஆண்டின் நடுப்பகுதியில் வெளியிடப்பட்டது.

1998 ஆம் ஆண்டு, வெப் பக்கங்களை வடிவமைப்பதற்குரிய மொழியான XML (eXtensible Markup Language) வெளியிடப்பட்டது.

மைக்ரோசொவ்ற் நிறுவனத்தின் மற்றுமொரு புதிய தொகுப்பான விசுவல் ஸ்ருடியோ.நெற் (Visual Studio.Net) 2002 ஆம் ஆண்டு பெப்ரவரி மாதம் 13 ஆம் திகதி வெளியிடப்பட்டது. இந்த விசுவல் ஸ்ருடியோ.நெற் என்ற கூட்டுத்தொகுப்பில் சி ஷாப் (C #), விபி.நெற் (VB.Net), ஏஎஸ்பி.நெற் (ASP.Net) போன்றன முக்கிய பங்கினை வகிக்கின்றன.

ஃபோர்ரான் (FORTRAN), கோபோல் (COBOL), பஸ்கல் (Pascal), சி (C), பேசிக் (Basic), சி++ (C++), விசுவல் பேசிக் (Visual Basic), விசுவல் சி++ (Visual C++), ஜாவா (Java), சி ஷாப் (C #) போன்ற மொழிகள் ஆண்டுதோறும் புதிய பரிமாணத்தில் வெளியிடப்படுகின்றன. ஆனால், இன்று வெளியாகிக் கொண்டிருக்கும் புதிய மொழிகள் அனைத்தும் சி, சி++ மொழிகளை அடிப்படையாகக் கொண்டே வெளியிடப்படுகின்றன. எனவே, நீங்கள் புதிய மொழிகளை நன்கு தெளிவாகப் புரிந்து கொள்வதற்கு, சி அல்லது சி++ மொழி பேருதவியாக அமையும் என்பதில் ஐயமில்லை.

நம்முடைய அன்றாட நடைமுறையில் ஏற்படும் சிக்கல்களுக்குத் தீர்வுகாண, கணினி உயர்நிலை மொழிகளைப் பயன்படுத்துவது மிகவும் இலகுவாக இருக்கும். பொதுவாக, கணினி உயர்நிலை மொழிகளானது விபர இனங்கள் (Data Types), கட்டளைகள் (Statements) போன்ற இரு முக்கிய கூறுகளைக் கொண்டிருக்கும். இவ்விரு கூறுகளும் மொழிகளுக்கு மொழிகள் வேறுபடுகின்றன.

சி மொழியும், சி$^{++}$ மொழியும்

சி$^{++}$ மொழியானது, சி மொழியிலும் ஒருபடி கூடிய திறன் கொண்ட மொழியாகும். சி$^{++}$ மொழிப் புறோகிராமில் சி மொழியில் காணப்படும் கட்டளைகளுடன், ஒப்ஜெக்ற் ஒரியன்ரட் என்ற கட்டமைப்பினையும் பயன்படுத்த முடியும். ஒப்ஜெக்ற் ஒரியன்ரட் என்ற கட்டமைப்பில் கிளாஸ் (Class) என்ற அமைப்பு முக்கிய அங்கம் வகிப்பதால்தான், சி$^{++}$ மொழியினை C with Classes என்றும் அழைக்கப்படுகின்றது.

மொழிமாற்றிகள் (Translators)

உயர்நிலை மொழிகளில் எழுதப்பட்ட புறோகிராம்களை கணினிக்குப் புரியும் மொழியான இயந்திர மொழிக்கு மாற்றுவதற்கு மொழிமாற்றி இன்றியமையாததொன்றாகும். பொதுவாக, கணினி உயர்நிலை மொழிகளில் எழுதப்பட்ட புறோகிராம்களை இயந்திர மொழிக்கு மாற்றுவதற்கு இரண்டு வகையான மொழிமாற்றிகள் பயன்படுத்தப்படுகின்றன.

அவையாவன,

1. கொம்பைலர் (Compiler)
2. இன்ரபிரிற்ரர் (Interpreter)

இவற்றில் கொம்பைலர் வகை மொழிமாற்றி மூலம், புறோகிராம் முழுவதும் ஒரே நேரத்தில் இயந்திர மொழிக்கு மாற்றப்படும். ஆனால், இன்ரபிரிற்ரர் வகை மொழிமாற்றி மூலம், புறோகிராமினை வரி வரியாக இயந்திர மொழிக்கு மாற்றப்படும். சி$^{++}$ மொழியில், கொம்பைலர் வகை மொழிமாற்றி பயன்படுத்தப்படுகின்றது.

1.3 சி$^{++}$ கொம்பைலர்கள் (C^{++} Compilers)

சி$^{++}$ மொழிக்குரிய கொம்பைலர்கள், பல்வேறு நிறுவனங்களால் வெளியிப்பட்டுள்ளன. இவற்றில், மூன்று வகையான கொம்பைலர்கள் மிகவும் பிரபல்யமாகக் காணப்படுகின்றன.

அவையாவன,

- ரேர்போ சி$^{++}$ கொம்பைலர் (Turbo C^{++} Compiler)
- போர்லாண்ட் சி$^{++}$ கொம்பைலர் (Borland C^{++} Compiler)
- மைக்ரோசொவ்ற் சி$^{++}$ கொம்பைலர் (Microsoft C^{++} Compiler)

சி$^{++}$ என்பது ஒரு உயர்நிலைக் கணினி மொழியாகும். ஆனால், ரேர்போ சி$^{++}$, போர்லாண்ட் சி$^{++}$, மைக்ரோசொவ்ற் சி$^{++}$ போன்றன சி$^{++}$ மொழிக்குரிய கொம்பைலரினையும், சி$^{++}$ மொழிப் புறோகிராம்களை எழுதி இயக்கக்கூடிய சூழலையும் குறிக்கும்.

போர்லாண்ட், மைக்ரோசொவ்ற் நிறுவனங்கள் சி$^{++}$ கொம்பைலரினை மட்டுமன்றி ஒருங்கிணைந்த புறோகிராம் உருவாக்கத் தளத்தினையும் (IDE - Integrated Development Environment) சேர்த்தே வெளியிட்டன. அதாவது, புறோகிராம்களை எழுதுவதற்குரிய ரெக்ஸ்ற் எடிற்றர் (Text Editor) இனையும், புறோகிராமில் உள்ள பிழைகளைக் கண்டறிந்து திருத்துவதற்கான டீபக்கர் புறோகிராம் (Debuger Program) இனையும், சி$^{++}$ மொழிக் கட்டளைகள் பற்றிய உதவிக் குறிப்புக்களையும், சி$^{++}$ மொழிப் புறோகிராம்களை கொம்பைல் (Compile) மற்றும் இணைப்புக் கள் (Links) போன்றவற்றை ஏற்படுத்தும் புறோகிராம்களையும் கொண்டிருக்கும்.

அன்சி (ANSI - American National Standards Institute) என்ற அமைப்பினால் நிர்ணயிக்கப்பட்ட சி$^{++}$ மொழிக்குரிய கீவேட்களைக் (Keyword) கொண்டவையாகவே இந்த மூன்று வகையான கொம்பைலர் களும் காணப்படுகின்றன. இவற்றில் ரேர்போ சி$^{++}$, போர்லாண்ட் சி$^{++}$ ஆகிய இரு கொம்பைலர்களும் போர்லாண்ட் நிறுவனத்தினால் வெளியிடப்பட்டவையாகும்.

சி$^{++}$ மொழிக்குரிய புறோகிராம்களை எழுதிச் செயற்படுத்துவதற்கு முதலில் மேற்குறிப்பிடப்பட்ட ரேர்போ சி$^{++}$, போர்லாண்ட் சி$^{++}$, மைக்ரோ சொவ்ற் சி$^{++}$ போன்றவற்றில் ஏதாவதொரு நிறுவனத்திற்குரிய மென்பொருளினைக் கணினியில் நிறுவ வேண்டும்.

உதாரணமாக, ரேர்போ சி$^{++}$ கணினியில் நிறுவப்படும்போது TC அல்லது TCPLUS என்ற ∴போல்டர் (Folder) உருவாக்கப்பட்டு, அங்கு பல சப் ∴போல்டர்கள் (Sub Folders) இனை உருவாக்கும். அவையாவன BIN, INCLUDE, LIB, CLASSLIB, BGI, EXAMPLES, DOC போன்ற பல சப் ∴போல்டர்கள் TC அல்லது TCPLUS என்ற பிரதான ∴போல்டரிற்குள் காணப்படும்.

TC அல்லது TCPLUS என்ற பிரதான ∴போல்டரிற்குள்ள உள்ள சப் ∴போல்டர்களின் விளக்கங்களை விரிவாகப் பார்ப்போம்.

• BIN என்ற சப் ∴போல்டரிற்குள் ரேர்போ சி$^{++}$ இயங்குவதற்குரிய அனைத்து சிஸ்ரம் ∴பைல்களும் (System Files) காணப்படும்.

உதாரணமாக, ரேர்போ சி$^{++}$ இனை இயக்குவதற்குரிய TC.EXE என்ற ∴பைலும், இந்த BIN என்ற சப் ∴போல்டரிற்குள் காணப்படும்.

• INCLUDE என்ற சப் ∴போல்டரிற்குள், சி மொழிக்குரிய ஹெடர் ∴பைல்கள் (Header Files) அடங்கலாக ஏறக்குறைய 47 ஹெடர் ∴பைல்கள் காணப்படும்.

• BGI என்ற சப் ∴போல்டரிற்குள், ரேர்போ சி$^{++}$ மொழிக்குரிய வரைவுகள் (Graphics) சார்ந்த ∴பைல்கள் காணப்படும்.

• EXAMPLES என்ற சப் ∴போல்டரிற்குள், ரேர்போ சி++ இற்குரிய பல உதாரணப் புறோகிராம்கள் காணப்படும்.

ரேர்போ சி++ இனை நிறுவிய பின்னர், முக்கியமான மாற்றம் (Setting) ஒன்றினைச் செய்ய வேண்டும். அதாவது, சி++ இற்குரிய எடிற்றர் (Editor) இற்குள் சென்று, அங்கு எடிற்றரின் மேற்பகுதியில் காணப்படும் மெனு (Menu) இலுள்ள Options இனைத் தெரிவு செய்தவுடன் பொப் - அப் மெனு (Pop - up Menu) தோன்றும். இந்தப் பொப்-அப் மெனுவில் Directories இனைத் தெரிவு செய்தவுடன், டயலொக் பொக்ஸ் (Dialog Box) ஒன்று தோன்றும். இதில் Include Directories, Library Directories, Output Directory, Source Directories போன்றவற்றுக்குரிய விபரங்களை ஒழுங்கான முறையில் பூர்த்தி செய்ய வேண்டும். (முதல் முறையாக சி++ இனைச் செயற்படுத்தும் போது மட்டுமே இந்த மாற்றங்களைச் செய்ய வேண்டும்.)

ரேர்போ சி++ இனைக் கணினியின் C ட்ரைவ் (C:\) இல், TC என்ற ∴போல்டரிற்குள் நிறுவியிருந்தால் Include Directories, Library Directories போன்றவற்றுக்குரிய விபரத்தினைக் கீழேயுள்ளவாறு பூர்த்தி செய்ய வேண்டும். (படம் - 1.1 இனைப் பார்க்கவும்)

Include Directories இல் C:\TC\INCLUDE எனவும்,

Library Directories இல் C:\TC\LIB எனவும் எழுத வேண்டும்.

மேலேயுள்ள இரு மாற்றங்களும் செய்யாது, ஒருபோதும் சி++ மொழிப் புறோகிராம்களைச் செயற்படுத்த முடியாது.

படம் - 1.1

இங்கு Include Directories என்பது, சி++ மொழியில் காணப்படும் ஹெடர் ∴பைல்கள் (Header Files) அடங்கிய ∴போல்டரினைக் குறிக்கும். எனவே, நீங்கள் நிறுவியுள்ள ரேர்போ சி++ இல் காணப்படும் ஹெடர் ∴பைல்கள் அடங்கிய ∴போல்டரான INCLUDE இனைப் பாதையுடன் குறிப்பிட வேண்டும்.

உதாரணமாக, நிறுவப்பட்ட ரேர்போ சி++ ஆனது, D:\ என்ற ட்ரைவில் TCPLUS என்ற ∴போல்டருக்குள் காணப்பட்டால் Include Directories இல் D:\TCPLUS\INCLUDE எனக் குறிப்பிட வேண்டும்.

Library Directories என்பது, சி++ மொழியில் காணப்படும் Library ∴பைல்கள் அடங்கிய ∴போல்டரினைக் குறிக்கும். எனவே, நீங்கள் நிறுவியுள்ள ரேர்போ சி++ இல் காணப்படும் Library ∴பைல்கள் அடங்கிய ∴போல்டரான LIB இனைப் பாதையுடன் குறிப்பிட வேண்டும்.

உதாரணமாக, நிறுவப்பட்ட ரேர்போ சி++ ஆனது, D:\ என்ற ட்ரைவில் TCPLUS என்ற ∴போல்டருக்குள் காணப்பட்டால் Library Directories இல் D:\TCPLUS\ LIB எனக் குறிப்பிட வேண்டும்.

அடுத்த இரு மாற்றங்களும் அவ்வளவு முக்கியமில்லை. எனினும், எமது பயன்பாட்டிற்கு இலகுவாக அமையும் என்பதற்காக இந்த இரு மாற்றங்களும் செய்யப்படுகின்றன.

Output Directory என்பது, சி++ புறோகிராம்கள் கொம்பைல் செய்யும் போது உருவாகின்ற EXE, OBJ ∴பைல்களைச் சேமிப்பதற்குப் பயன் படுத்தப்படும் ∴போல்டரினைக் குறிக்கும். எனவே, இதில் நாம் விரும்பிய ட்ரைவில் உள்ள ∴போல்டரினைக் குறிப்பிட முடியும். உதாரணமாக, Output Directory இல் D:\temp எனக் குறிப்பிடலாம்.

Source Directories என்பது, எம்மால் எழுதப்பட்ட சி++ மொழிப் புறோகிராம்களைச் சேமிப்பதற்குப் பயன்படுத்தப்படும் ∴போல்டரினைக் குறிக்கும். உதாரணமாக, Source Directories இல் D:\samples எனக் குறிப்பிடலாம்.

இந்த நூலில் காணப்படும் சி++ மொழிப் புறோகிராம்களை மேலே குறிப்பிடப்பட்ட ரேர்போ சி++, போர்லாண்ட் சி++, மைக்ரோசொவ்ற் சி++ போன்ற ஏதாவதொரு நிறுவனத்திற்குரிய தொகுப்பில் செயற்படுத்த முடியும். இங்குள்ள அனைத்து புறோகிராம்களும் Turbo 3.0 இல் செயற்படுத்தப்பட்டு, அவற்றிற்குரிய வெளியீடுகள் (Output) பரீட்சிக்கப் பட்டவையாகும்.

இனி, எவ்வாறு ரேர்போ சி++ எடிற்ரரிற்குள் நுழைவது எனப் பார்ப்போம். ரேர்போ சி++ நிறுவப்பட்ட ட்ரைவினைக் கண்டுபிடித்து, அங்கு TC அல்லது TCPLUS என்ற ∴போல்டரிற்குள் BIN என்ற

சப் ∴போல்டரினைத் தெரிவு செய்யவும். அங்குள்ள TC.EXE என்ற ∴பைலினைச் செயற்படுத்துவதன் மூலம், ரேர்போ சி++ எடிற்ரரிற்குள் நுழைய முடியும். (படம் - 1.2 இனைப் பார்க்கவும்)

படம் - 1.2

1.4 சி++ மொழிப் புறோகிராம் அமைப்பு

சி++ மொழிப் புறோகிராம்களை எவ்வாறு எழுதிச் செயற்படுத்துவது எனப் பார்ப்போம்.

சி++ எடிற்ரரின் மேல் பகுதியில் உள்ள மெனுவில் File என்பதைத் தெரிவு செய்து வருகின்ற பொப் - அப் மெனு (Pop-up Menu) இல் new இனைத் தெரிவு செய்ய வேண்டும். பின்னர், இந்த ரெக்ஸ்ற் எடிற்ரரில் சி++ மொழிக்குரிய புறோகிராமினை எழுத வேண்டும். (படம் - 1.3 இனைப் பார்க்கவும்)

படம் - 1.3

உதாரணம் (1)

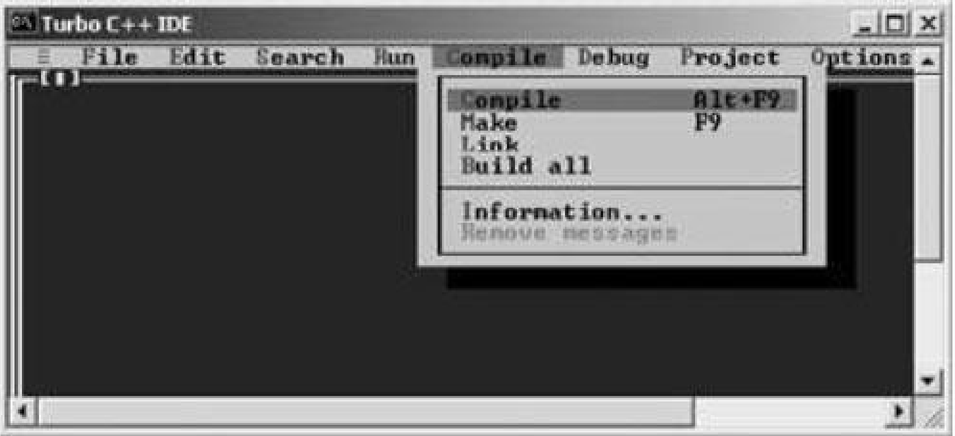
கீழேயுள்ள புறோகிராமினை அப்படியே ரேர்போ சி++ ரெக்ஸ்ற் எடிற்ரர் (Text Editor) இல் எழுதவும்.

```
#include <iostream.h>
void main()
{
    cout<<"Welcome to C++";
}
```

இந்தப் புறோகிராமினைச் செயற்படுத்திப் பார்த்தால், "Welcome to C++" என்ற வசனம் கணினித்திரையில் வெளியீடாகத் தோன்றும்.

இந்த சி++ மொழிப் புறோகிராமினை எவ்வாறு செயற்படுத்துவது என முதலில் பார்ப்போம்.

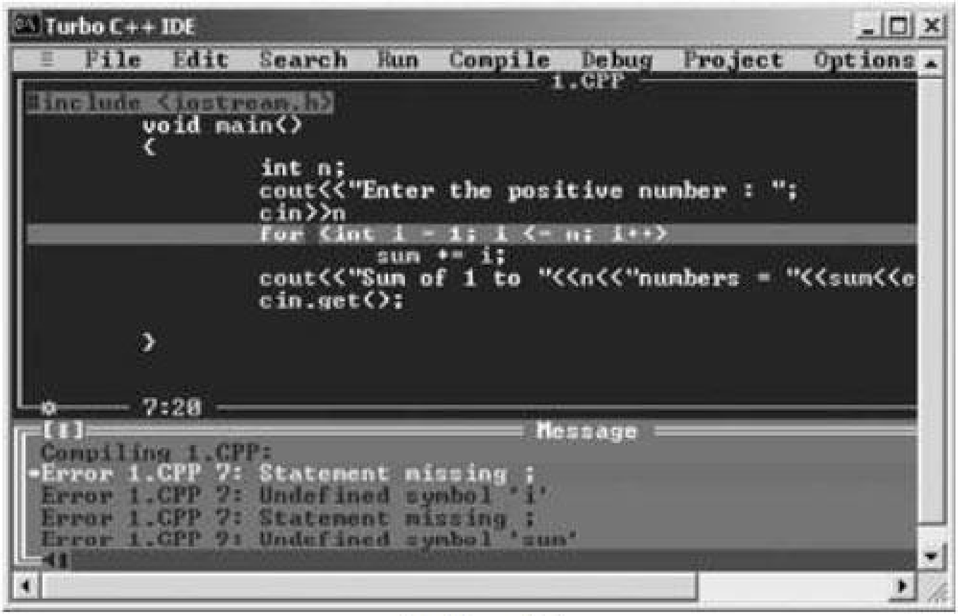
உதாரணம் (1) இல் எழுதப்பட்ட புறோகிராமானது உயர்நிலை மொழியான சி++ இல் உள்ளதால், இந்தப் புறோகிராமினை இயந்திர மொழிக்கு மாற்ற வேண்டும். எனவே, இந்தப் புறோகிராமினைக் கொம்பைல் செய்து இயந்திர மொழிக்கு மாற்ற வேண்டும். "F9" என்ற கீயினை அழுத்துவதன் மூலமோ அல்லது மெனுவிலுள்ள Compile இல் Make இனைத் தெரிவு செய்வதன் மூலமோ, இந்தப் புறோகிராமினைக் கொம்பைல், லிங் (Link) செய்ய முடியும். (படம் 1.4 இனைப் பார்க்கவும்)



படம் - 1.4

புறோகிராமினைக் கொம்பைல் செய்யும் போது கட்டளை அமைப்பில் ஏதாவது பிழைகள் காணப்பட்டால், அவற்றினை சி++ கொம்பைலர் சுட்டிக் காட்டும். (படம் - 1.5 இனைப் பார்க்கவும்)

எழுதப்பட்ட புறோகிராமில் உள்ள கட்டளை அமைப்பில் பிழை எதுவுமில்லையென்றால், Successfully என்ற டயலொக் பொக்ஸ் (Dialog box) தோன்றும். அதில், OK இனைத் தெரிவு செய்ய வேண்டும். (படம் - 1.6 இனைப் பார்க்கவும்)



படம் - 1.5

கொன்ரோல் கீ (Ctrl Key) உடன் “F9” என்ற கீயினையும் ஒன்றுசேர அழுத்துவதன் மூலமோ அல்லது மெனுவில் உள்ள Run இல் Run இனைத் தெரிவு செய்வதன் மூலமோ, இந்தப் புறோகிராமினைச் செயற்படுத்த முடியும். (படம் - 1.7 இனைப் பார்க்கவும்)

படம் - 1.6

படம் - 1.7

இந்தப் புறோகிராமினைச் செயற்படுத்தியவுடன், வெளியீட்டினைத் திரையில் காண்பித்தவுடனேயே ரெக்ஸ்ற் எடிற்ரருக்குள் மீண்டும் திரும்பி விடும். இதனைத் தவிர்ப்பதற்கு Alt கீயுடன் "F5" என்ற கீயையும் ஒன்றுசேர அழுத்தியோ அல்லது மெனுவில் உள்ள Window இல் User Screen இனைத் தெரிவு செய்வதன் மூலமோ அல்லது புறோகிராமின் இறுதி வரியாக cin.get() என்ற கட்டளையினை எழுதுவதன் மூலமோ, மீண்டும் வெளியீட்டினைத் திரையில் காட்ட முடியும். வெளியீட்டினைத் திரையில் காட்டிய பின்னர், மீண்டும் எடிற்ரருக்குள் செல்ல வேண்டுமாயின், என்ரர் கீ (Enter Key) இனை அழுத்த வேண்டும்.

இந்தப் புறோகிராமில் காணப்படும் கட்டளைகளுக்குரிய விளக்கங்களைப் பின்னர் தெளிவாகப் பார்ப்போம்.

எழுதப்பட்ட புறோகிராம்களை மீண்டும் பயன்படுத்த வேண்டுமாயின், அந்தப் புறோகிராம்களை முதலில் சேமிக்க வேண்டும். இங்கு முக்கியமாகக் கவனிக்க வேண்டியது யாதெனில், எழுதப்பட்ட புறோகிராமினை இயக்கும் போது obj, exe ஃபைல்கள் உருவாக்கப்படுமே தவிர சி++ மொழி ஃபைலான cpp ஃபைலானது உருவாக்கப்படமாட்டாது. எனவே, "F2" என்ற கீயினை அழுத்துவதன் மூலமோ அல்லது மெனுவில் உள்ள File இல் Save அல்லது Save As இனைத் தெரிவு செய்வதன் மூலமோ, இந்தப் புறோகிராமினை cpp ஃபைலாகச் சேமிக்க முடியும்.

Alt கீயுடன் X இனைச் சேர்த்து அழுத்துவதன் மூலமோ அல்லது மெனுவில் உள்ள File இல் Quit இனைத் தெரிவு செய்வதன் மூலமோ ரேர்போ சி++ எடிற்ரரிலிருந்து வெளியேற முடியும்.

சி++ மொழி உதாரணப் புறோகிராமினையும், இதில் பயன்படுத்தப் பட்ட கட்டளைகளின் விளக்கங்களையும் அடுத்துப் பார்ப்போம்.

உதாரணம் (2)

```
// Display some messages on the computer screen
#include <iostream.h>
void main()
{
    cout<<"Welcome to University of Colombo \n";
    cout<<"Faculty of Science \n";
    cout<<"Sri Lanka \n";
    cout<<"Thank You \n";
    cin.get();
}
```

1.5 சி++ மொழியில் பயன்படுத்தப்படும் அடிப்படைக் கட்டளைகள்

விளக்கக் குறிப்புகள் (Comments):

தரமான புறோகிராம்களுக்கு இந்த விளக்கக் குறிப்புக்களானது மிகவும் முக்கியமானதொன்றாகும். ஏனெனில், எதற்காக புறோகிராம் எழுதப்படுகின்றது என்பதற்குரிய விளக்கக் குறிப்பினையும், முக்கிய கட்டளைகளுக்குரிய விளக்கக் குறிப்பினையும் புறோகிராம் எழுதும் போது குறிப்பிட்டால், நீண்ட நாட்களின் பின்னர் அல்லது வேறொரு புறோகிராமர் (Programmer) மீள்திருத்தம் செய்ய இந்த விளக்கக் குறிப்பானது பேருதவியாக அமையும்.

சி++ மொழியில் இரண்டு வகையான விளக்கக் குறிப்புகள் பயன் படுத்தப்படுகின்றன. அவையாவன,

(1) // - இரண்டு முன் சாய்வுக் கோடுகள் (Two forward slashes): இந்த விளக்கக் குறிப்பானது, ஒரேயொரு வரியினை மட்டும் விளக்கக் குறிப்பாகக் குறிப்பிடுவதற்குப் பயன்படுத்தப்படுகின்றது. அதாவது, // என்ற குறியீட்டுக்குப் பின்னர், அதே வரியில் எழுதப்படும் சொற்கள் அனைத்தும் விளக்கக் குறிப்பாகவே கருதப்படும்.

(2) /* */

இந்த விளக்கக் குறிப்பானது, ஒன்றுக்கு மேற்பட்ட வரிகளை விளக்கக் குறிப்பாகக் குறிப்பிடுவதற்குப் பயன்படுத்தப்படுகின்றது.

ஒன்றுக்கு மேற்பட்ட வரிகளில் விளக்கக் குறிப்புகளை குறிப்பிடப்பட வேண்டுமாயின், ஒவ்வொரு வரிக்கும் முன்னால் // என்ற குறியீடுகளைப் பயன்படுத்தலாம் அல்லது /* என்ற குறியீட்டினை விளக்கக் குறிப்புகளுக்கு முன்பும், */ என்ற குறியீட்டினை விளக்கக் குறிப்புகளின் இறுதியிலும் இட வேண்டும்.

உதாரணப் புறோகிராம் (2) இல் உள்ள முதலாவது வரியானது விளக்கக் குறிப்பாகும். இந்த வரியினைக் குறிப்பிடாது விட்டாலும் புறோகிராம் ஒழுங்காகச் செயற்படும். பொதுவாகக் கொம்பைலர்கள் விளக்கக் குறிப்புகளைத் தவிர்த்தே புறோகிராம்களைக் கொம்பைல் (Compile) செய்கின்றன.

கீழேயுள்ளவாறு விளக்கக் குறிப்புகளைப் புறோகிராம்களில் பயன்படுத்த முடியும்.

```
1. int age; // variable age declaration
2. /* This is the second C++
      programme */
3. int x = /* 10 is assign to x */ 10;
```

ஆயினும், மேலேயுள்ள மூன்றாவது உதாரணத்தில் உள்ளவாறு கட்டளைகளுக்கு இடையில் விளக்கக் குறிப்புகளை எழுதுவதைத் தவிர்ப்பது சிறந்ததாகும். சி++ மொழிப் புறோகிராமில், விளக்கக் குறிப்பு களை எந்த இடத்திலும் பயன்படுத்த முடியும்.

ஹெடர் ஃபைல்கள் (Header Files):

சி++ மொழியில் ஹெடர் ∴பைல்கள் முக்கியமானவையாகும். சி++ மொழிக்குரிய கொம்பைலரினை உருவாக்கும் போதே, பல ஹெடர் ∴பைல்கள் இணைக்கப்பட்டு வெளியிடப்பட்டுள்ளன. இந்த ஹெடர் ∴பைல்கள் ஒவ்வொன்றும் பலதரப்பட்ட கட்டளைகளையும், ∴பங்ஷன் களையும் தன்னகத்தே கொண்டுள்ளன. இவ்வாறான ஹெடர் ∴பைல்களை எமது தேவைக்கு ஏற்றால் போல், எம்மால் உருவாக்க முடியும்.

சி++ மொழியில் ஏற்கனவே உள்ளிணைக்கப்பட்ட ஹெடர் ∴பைல் கள் ஏறக்குறைய 47 உள்ளன. சி மொழியில் காணப்பட்ட 18 ஹெடர் ∴பைல்களும் இதில் அடங்கும்.

உதாரணமாக, iostream.h என்ற ஹெடர் ∴பைலில்தான் cout என்ற கட்டளையும், அதற்கு அடுத்தால் போல் காணப்படும் << என்ற ஒப்பறேற்ரரும், cin என்ற கட்டளையும், அதற்கு அடுத்தால் போல் காணப்படும் >> என்ற ஒப்பறேற்ரரும் காணப்படுகின்றது. எனவேதான், அனேகமான சி++ மொழிப் புறோகிராமில், iostream.h என்ற ஹெடர் ∴பைலானது உட்புகுத்தப்படுகின்றது. இவ்வாறு பல கட்டளைகளையும், ∴பங்ஷின்களையும் தன்னகத்தே கொண்ட பல ஹெடர் ∴பைல்கள் சி++ மொழியில் உண்டென்பது குறிப்பிடத்தக்கதோர் விடயமாகும்.

iostream.h என்ற ஹெடர் ∴பைலின் விரிவாக்கம் யாதெனில், input and output stream ஆகும். இங்கு .h என்பது ஹெடர் ∴பைல் (Header File) இனைக் குறிக்கும்.

இந்த ஹெடர் ∴பைலில் காணப்படுகின்ற cout (சி அவுட் என உச்சரிக்க வேண்டும்) என்பது கணினித்திரையில் வெளியீட்டினைக் காட்டுவதற்கும், cin (சி இன் என உச்சரிக்க வேண்டும்) என்பது தரவு களை உள்ளீடு செய்வதற்கும் வரையறுக்கப்பட்ட கட்டளைகளாகும். cout, cin போன்ற கட்டளைகளை எமது புறோகிராமில் பயன்படுத்த வேண்டுமாயின், முதலில் iostream.h என்ற ஹெடர் ∴பைலினைப் புகுத்த வேண்டும்.

மேலும், சி++ மொழியில் உள்ளிணைக்கப்பட்ட ஹெடர் ∴பைல்களை எமது புறோகிராமில் பயன்படுத்தும் போது <, > போன்ற குறியீடுகளுக்கு இடையிலேயே ஹெடர் ∴பைலினைக் குறிப்பிட வேண்டும். நாம் எழுதிய ஹெடர் ∴பைல்களை ", " போன்ற இரு மேற்கோள் குறியீடுகளுக்கு இடையில் குறிப்பிட வேண்டும்.

உதாரணமாக, சி++ மொழியில் ஏற்கனவே உள்ளிணைக்கப்பட்ட ஹெடர் ∴பைல்களை எமது புறோகிராமில் பயன்படுத்த வேண்டுமாயின்,

```
#include <iostream.h>
#include <string.h>
#include <math.h>
```

எனக் குறிப்பிட வேண்டும்.

உதாரணமாக, நாம் எழுதிய ஹெடர் ∴பைல்களைப் புறோகிராமில் பயன்படுத்த வேண்டுமாயின்,

```
#include "calculation.h"
#include "utility1.h"
#include "test.h"
```

எனக் குறிப்பிட வேண்டும்.

இங்கு calculation.h, utility1.h, test.h போன்றன எம்மால் எழுதப்பட்ட ஹெடர் ∴பைல்களாகும்.

∴பங்ஷன்கள் (Functions)

பொதுவாகக் கணினி மொழிகளில், பல கட்டளைகளின் தொகுப்பே புறோகிராம் என அழைக்கப்படுகின்றது. ஆனால், சி++ மொழியில் ஒன்று அல்லது ஒன்றுக்கு மேற்பட்ட ∴பங்ஷன்களின் தொகுப்பே புறோகிராம் என அழைக்கப்படுகின்றது. சி++ மொழிப் புறோகிராமில் குறைந்தது ஒரு ∴பங்ஷனாவது காணப்பட வேண்டும்.

∴பங்ஷன்கள் என்றால் என்ன?

குறித்தொரு செயற்பாட்டினைச் செய்வதற்கு எழுதப்படும் புறோகிராம் ∴பங்ஷன் என அழைக்கப்படும். இந்த ∴பங்ஷன்களை மற்றைய கணினி உயர்நிலை மொழிகளில் மெதட் (Method), புரொசீஜர் (Procedure), சப்றூற்றின் (Subroutine) எனவும் அழைக்கப்படுகின்றன.

சி$^{++}$ மொழியில் வரையறுக்கப்படும் ∴பங்ஷன்கள் பொதுவாக ஒரு மதிப்பினை விடையாகத் திருப்பியனுப்ப வேண்டும். இவ்வாறு திருப்பியனுப்புகின்ற மதிப்பு ஒரு குறித்த விபர இனத்தினைக் (Data Type) கொண்டிருக்க வேண்டும். எந்த மதிப்பினையும் திருப்பியனுப்பாத ∴பங்ஷன்களை வொய்ட் (void) ∴பங்ஷன் என அழைக்கப்படுகின்றது. அதாவது void என்பது, எந்தவொரு பெறுமானத்தையும் ∴பங்ஷினுக்குத் திருப்பியனுப்பாத சந்தர்ப்பத்தில் பயன்படுத்தப்படுகின்றது.

சி$^{++}$ மொழிப் புறோகிராமில் எழுதப்படும் மெயின் ∴பங்ஷன்களை வொய்ட் (void) அல்லது முழு எண் (int) விபர இனமாக வரையறுக்க முடியும். எந்தவொரு விபர இனத்தினையும் குறிப்பிடாது எழுதப்படும் மெயின் ∴பங்ஷன்கள் இயல்பாக முழு எண் விபர இனத்தினையே பெற்றுச் செயற்படும்.

உதாரணம் (1)

```
void main()
{
    .....;
    .....;
}
```

உதாரணம் (2)

```
int main()
{
    .....;
    return 0;
}
```

உதாரணம் (3)

```
 main()
{
    .....;
    return 0;
}
```

மேலேயுள்ள மூன்று உதாரணங்களிலும் குறிப்பிட்டதுபோல் மெயின் ∴பங்ஷன்களை எழுத முடியும். இறுதியாகவுள்ள இரு உதாரணங்களும் முழு எண் விபர இன மெயின் ∴பங்ஷன்கள் என்பதால், இறுதிக் கட்டளையாக return என்ற கட்டளை எழுதப்பட்டுள்ளது.

மிகப் பெரிய செயல்திட்டத்திற்கு எழுதப்படும் புறோகிராமில் ஒன்றுக்கு மேற்பட்ட மெயின் ∴பங்ஷன்கள் பயன்படுத்தப்படுகின்றன. இவ்வாறான சந்தர்ப்பங்களில் எழுதப்படும் மெயின் ∴பங்ஷன்கள் முழு எண் விபர இனமாக (Integer data type) எழுதப்படுகின்றன.

இந்நூலிலுள்ள அனைத்து புறோகிராம்களின் மெயின் ∴பங்ஷன் களும் வொய்ட் (void) ∴பங்ஷனாகவே எழுதப்பட்டுள்ளது. ஏனெனில், இங்குள்ள புறோகிராம்கள் அனைத்தும், தனித் தனியாகச் செயற்படும் விதமாகவே எழுதப்பட்டுள்ளது.

சி++ மொழியில் வரையறுக்கப்படும் ∴பங்ஷனுக்குள் எழுதப்படும் கட்டளைகள் யாவும், இரட்டை அடைப்புக்குறிகள் (Curly Brackets) இற்குள் காணப்பட வேண்டும். ∴பங்ஷன்கள் பற்றிய மேலதிக விளக்கங் களைப் பின்னர் வரும் அத்தியாயத்தில் பார்ப்போம்.

சி++ மொழிப் புறோகிராமில் உள்ளீடு செய்யப்பட்ட தரவுகளை, எவ்வாறு வெளியீடாகக் கணினித்திரையில் காட்ட முடியும் என்பதற் குரிய உதாரணத்தினை அடுத்துப் பார்ப்போம்.

```
#include <iostream.h>
void main()
{
    char name[30];
    int age;
    float bsal;
    cout<<"Enter your name : ";
    cin>>name;
    cout<<"Enter your age : ";
    cin>>age;
    cout<<"Enter your basic salary: ";
    cin>>bsal;
    cout<<"Your Personal Details "<<endl;
    cout<<"******************** "<<endl;
    cout<<"Your name  :  "<<name<<endl;
    cout<<"Your age :  "<<age<<endl;
    cout<<"Your basic salary : "<<bsal<<endl;
    cin.get();
}
```

இந்தப் புறோகிராமினைச் செயற்படுத்திப் பார்த்தால், முதலில் பெயர், வயது, ஆரம்பச் சம்பளம் போன்றவற்றினை உள்ளீடு செய்யுமாறு கேள்விகள் கேட்கப்படும். இதற்கு நாம் உள்ளீடாக மூன்று பெறுமானங் களையும் உள்ளீடு செய்ய வேண்டும். இறுதியாக, நாம் உள்ளீடாகக் கொடுத்த தரவுகளை அப்படியே கணினித்திரையில் வெளியீடாகக் காண்பிக்கும். இப்புறோகிராமில் காணப்படும் கட்டளைகளைப் பின்னர் தெளிவாகப் பார்ப்போம்.

சி++ மொழியில் புறோகிராம்கள் எழுதும் போது முக்கியமாகக் கவனிக்க வேண்டிய அம்சங்கள்:

- சி++ மொழியானது, எழுத்துணர்திறன் (Case Sensitive) உடைய மொழியாகும். எனவே, சி++ மொழியில் எழுதப்படும் புறோகிராம்கள் ஆங்கிலப் பெரிய எழுத்துக்களினதும் (Capital Alphabet Letters), ஆங்கிலச் சிறிய எழுத்துக்களினதும் (Small Alphabet Letters) வேறுபாட்டினை உணரும். சி++ மொழியின் கட்டளைகள் பொதுவாக ஆங்கிலச் சிறிய எழுத்துக்களினால் எழுதப்படுகின்றன.
- சி++ மொழியில் எழுதப்படும் கட்டளைகளின் முடிவில் அரைப் புள்ளி (; - Semicolon) இடம்பெற வேண்டும்.

சி++ மொழிப் புறோகிராம் செயற்பட்டுக் கொண்டிருக்கும் போது இடைநடுவே நிறுத்துவதற்கு, கொன்றோல் கீ (Ctrl Key) உடன் C என்ற கீயையும் ஒன்றுசேர்த்து அழுத்த வேண்டும் அல்லது கொன்றோல் கீ (Ctrl Key) உடன் Break என்ற கீயையும் ஒன்றுசேர்த்து அழுத்த வேண்டும்.

2. மாறிகளும் மாறிலிகளும் (Variables and Constants)

புறோகிராம் இயங்கும் போது பெறுமானங்களைத் தற்காலிகமாகச் சேமித்து வைக்கப் பயன்படும் இடமே மாறிகள் அல்லது மாறிலிகள் என அழைக்கப்படுகின்றன.

மாறிகள் (Variables)

புறோகிராமில் குறித்தவொரு பெறுமானம் மாறிக்கொண்டிருந்தால், அவை மாறிகளாக வரையறுக்கப்பட்டுச் சேமிக்கப்படும்.

உதாரணமாகப் பல மாணவர்களின் பெயர், சுட்டிலக்கம், பெறுபேறுகள் போன்றவற்றைச் சேமிப்பதற்கு மாறிகள் வரையறுக்கப்படுகின்றன. ஏனெனில், பல மாணவர்களின் விபரங்களைச் சேமிக்கும் போது மாணவர் பெயர், சுட்டிலக்கம், பெறுபேறுகள் போன்றன மாணவருக்கு மாணவர் மாறிக் கொண்டேயிருக்கும்.

மாறிலிகள் (Constants)

குறித்தவொரு பெறுமானம், புறோகிராம் முடியும் வரை மாறாமல் காணப்பட்டால், அவை மாறிலிகளாக வரையறுக்கப்பட்டுச் சேமிக்கப் படும்.

உதாரணமாக, அறேயில் உள்ள மூலகங்களின் எண்ணிக்கை புறோகிராம் செயற்பட்டு முடியும் வரை மாற்றக்கூடாது என்பதால், அறேயிலுள்ள மூலகங்களின் எண்ணிக்கையினை மாறிலியாக வரையறுப்பது சிறந்த முறையாகும். ஏனெனில், மீண்டும் இந்தப் புறோகிராமினைச் செயற்படுத்த முன்னர், மூலகங்களின் எண்ணிக்கை யினை அதிகரிக்க வேண்டுமாயின், மாறிலியில் மட்டும் மாற்றம் செய்தால் போதுமானதாகும்.

மாறிகள், மாறிலிகளின் பெயர்கள்

சி++ மொழியில் அறிவிக்கப்படும் மாறிகள், மாறிலிகள், ஃபங்ஷன் கள், ஸ்ரக்சர்கள், கிளாஸ்கள் போன்றவற்றின் பெயர்கள் கீழேயுள்ள மூன்று விதிமுறைகளுக்கு இணங்க அமைய வேண்டும்.

- ஆங்கில அகர எழுத்துக்கள் (English Alphabet Letters), எண்கள் (Numbers), "_" (Underscore) ஆகியவை தவிர மற்றைய எந்தக் குறியீடுகளையும் பயன்படுத்தக்கூடாது.
- பெயரின் தொடக்கத்தில் ஆங்கில அகர எழுத்துக்கள், "_" (Underscore) ஆகியவை மட்டுமே இடம்பெற முடியும்.
- சி++ மொழிக்குரிய சிறப்புச் சொற்கள் (Keywords) இனை மாறிகள், மாறிலிகளின் பெயர்களாகப் பயன்படுத்தக்கூடாது.

உதாரணமாக =, +, -, ~, `, ", >, <, /, \, |, @, $, %, &, ^, ", ;, :, #, *, (,), {, }, [,] , ? போன்ற விஷேட குறியீடுகளையும், இடைவெளிகளையும் மாறிகள், மாறிலிகளின் பெயர்களில் பயன்படுத்தக்கூடாது.

சி++ மொழிக்குரிய சிறப்புச் சொற்கள் (C++ Keywords)

சி++ மொழியில் கிட்டத்தட்ட 48 சிறப்புச் சொற்கள் (Keywords) காணப்படுகின்றன. அவையாவன,

asm	double	new	switch
auto	else	operator	template
break	enum	private	this
case	extern	protected	throw
catch	float	public	try
char	for	register	typedef
class	friend	return	union
const	goto	short	unsigned
continue	if	signed	virtual
default	inline	sizeof	void
delete	int	static	volatile
do	long	struct	while

அட்டவணை - 2.1

சி++ மொழியில் உள்ள சிறப்புச் சொற்கள் அனைத்தும், ஆங்கிலச் சிறிய எழுத்துக்களினால் குறிப்பிடப்பட வேண்டும்.

சிறப்புச் சொற்களில் சேர்க்கப்படாத true, false, null ஆகிய சொற்கள் மாறிகள், மாறிலிகளின் மதிப்புக்களாகப் பயன்படுத்தப்படுவதால், இந்தச் சொற்களையும் மாறிகள், மாறிலிகளின் பெயர்களாகப் பயன் படுத்தக்கூடாது.

உதாரணமாக,

- மாறிகள், மாறிலிகள், ஃபங்ஷன்கள், ஸ்ரக்சர்கள், கிளாஸ்கள் போன்றவற்றின் பெயர்களாகப் பயன்படுத்தப்படக்கூடியவை:

num4, total_salary, _name, num4m, x

- மாறிகள், மாறிலிகள், ஃபங்ஷன்கள், ஸ்ரக்சர்கள், கிளாஸ்கள் போன்றவற்றின் பெயர்களாகப் பயன்படுத்த முடியாதவை:

4num, total-salary, full name, num/m, sel45$

பொதுவாக மாறிகளின் பெயர்கள் (Variable names) ஆங்கிலச் சிறிய எழுத்துக்களினால் குறிப்பிடப்படுவது வழக்கமாகும். எனினும், இரண்டு சொற்களை மாறியின் பெயராக கொடுக்கப்படும் போது, இரண்டு சொற்களையும் வேறுபடுத்திக் காட்டுவதற்காக இரண்டாவது சொல்லின் முதல் எழுத்தை ஆங்கிலப் பெரிய எழுத்தாகக் குறிப்பிடலாம் அல்லது "_" (Underscore) இனை இரண்டு சொற்களுக்குமிடையில் குறிப்பிடலாம்.

உதாரணமாக: age, name, fullName, basic_salary, lastName என மாறிகளின் பெயர்களைக் குறிப்பிடுவது சிறந்த முறையாகும்.

பொதுவாக மாறிலிகளின் பெயர்கள் (Constant Names) ஆங்கிலப் பெரிய எழுத்துக்களினால் குறிப்பிடப்படுவது வழக்கமாகும்.

உதாரணமாக PI, COUNT, G, MAX_LEN, OTRATE என மாறிலிகளின் பெயர்களைக் குறிப்பிடுவது சிறந்த முறையாகும்.

மாறிகள், மாறிலிகள், ஃபங்ஷன்கள், ஸ்ரக்சர்கள், கிளாஸ்கள் போன்றவற்றின் பெயர்கள் 32 எழுத்துக்களுக்குக் குறைவாக இருக்க வேண்டும் என முன்னர் வெளியிட்ட சி++ கொம்பைலர்கள் வரையறுத் திருந்தன. ஆனால், தற்போது வெளியிடப்படும் சி++ கொம்பைலர்களில் மாறிகள், மாறிலிகள், ஃபங்ஷன்கள், ஸ்ரக்சர்கள், கிளாஸ்கள் போன்றவற்றின் பெயர்கள் 128 எழுத்துக்கள் வரை அமைய முடியும். எனினும் மாறிகள், மாறிலிகள், ஃபங்ஷன்கள், ஸ்ரக்சர்கள், கிளாஸ்கள் போன்றவற்றின் பெயர்கள் நீண்டதாகக் கொடுத்தால், தவறுகள் ஏற்பட வாய்ப்புண்டு.

2.1 சி++ மொழியில் பயன்படுத்தப்படும் விபர இனங்கள் (C++ Data Types)

சி++ மொழியிலுள்ள விபர இனங்களை நான்கு குழுக்களாக வகைப்படுத்த முடியும்.

அவையாவன,

- முழு எண் வகைகள் (Integer number types)
- தசம எண் வகைகள் (Float number types)
- எழுத்து வகைகள் (Character types)
- இனம் சாராத வகைகள் (Void types)

இந்த 4 வகையான விபர இனக் குழுக்களில், இனம் சாராத வகையினைப் ஃபங்ஷனில் மட்டுமே பயன்படுத்த முடியும். மற்றைய மூன்று வகையான விபர இனக் குழுக்களை மாறிகள், மாறிலிகள், ஃபங்ஷன்கள் போன்றவற்றில் பயன்படுத்த முடியும்.

முழு எண் வகைகள் (Integer number types)

இந்த முழு எண் வகைகள், மூன்று பிரிவாகப் பிரிக்கப்பட்டுள்ளன. அவையாவன short, int, long ஆகும்.

முழு எண் விபர இன வகைகள்	தேவைப்படும் நினைவகத்தின் அளவு	பெறுமான வீச்சுக்கள் (Ranges)
short	2 bytes	-32768 இலிருந்து +32767
int	2 bytes	-32768 இலிருந்து +32767
long	4 bytes	-2,147,483,648 இலிருந்து +2,147,483,648

அட்டவணை - 2.2

சி++ மொழியில் int, short போன்ற முழு எண் விபர இனங்கள் ஒரே பெறுமான வீச்சினைக் கொண்டிருக்கும்.

அட்டவணை - 2.2 இல் உள்ள விபர இனங்களுக்குரிய பெறுமான வீச்சுக்கள் ஒப்பறேற்றிங் சிஸ்ரத்தினைப் பொறுத்து மாறுபடுகின்றன. அட்டவணை -2.2 இல் உள்ள பெறுமான வீச்சுக்கள், மைக்ரோசொவ்ற் நிறுவனத்தின் ஒப்பறேற்றிங் சிஸ்ரத்தினை மையமாகக் கொண்டவை யாகும். இதனுடன் ஒப்பிடுகையில், யுனிக்ஸ் (Unix) என்ற ஒப்பறேற்றிங் சிஸ்ரத்தில் int, short போன்ற முழு எண் விபர இனங்களின் பெறுமான வீச்சுக்கள் வேறுபடுகின்றன.

தசம எண் வகைகள் (Float number types)

தசம எண் வகைகளும், மூன்று பிரிவாகப் பிரிக்கப்பட்டுள்ளன. அவையாவன float, double, long double ஆகும்.

முழு எண் விபர இன வகைகள்	தேவைப்படும் நினைவகத்தின் அளவு	பெறுமான வீச்சுக்கள் (Ranges)
float	4 bytes	3.4×10^{-38} இலிருந்து 1.7×10^{38}
double	8 bytes	1.7×10^{-308} இலிருந்து 1.7×10^{308}
long double	10 bytes	1.7×10^{-4936} இலிருந்து 1.7×10^{4936}

அட்டவணை - 2.3

எழுத்து வகைகள் (Character types)

எழுத்து வகையான char என்ற விபர இனத்தில், அஸ்க்கீ (ASCII

- American Standard Code Information Interchange) எழுத்துக்களைச் சேமிக்க முடியும். அதாவது, கீபோட் (Keyboard) இல் உள்ள அனைத்து எழுத்துக்கள், இலக்கங்கள், விஷேட குறியீடுகள் (A - Z, a-z, 0-9, @, #, ^, &, *, ...) போன்றவை char விபர இனமாக வரையறுக்கப் படுகின்றன. இந்த char என்ற விபர இனத்துக்கு 1 பைற் (byte) இடம், நினைவகத்தில் தேவைப்படுகின்றது.

2.2 மாறிகளின் பிரயோகங்கள்

சி++ மொழிப் புறோகிராமில் எவ்வாறு மாறிகளை (Variables) வரையறுப்பது என அடுத்துப் பார்ப்போம்.

மாறிகளை வரையறுக்கும் போது, முதலில் விபர இனத்தினையும் (உதாரணமாக: int, long, char, double, float) பின்னர், மாறியின் பெயரினையும் குறிப்பிடப்பட வேண்டும்.

உதாரணமாக,

```
int age;
int marks;
double salary;
char sex;
```

சி++ மொழிப் புறோகிராமில், ஒரே வரியில் ஒன்றுக்கு மேற்பட்ட மாறிகளை வரையறுக்க முடியும்.

உதாரணமாக,

```
int age, marks, total;
char sex, ans;
double basic_sal, net_sal, otRate;
```

மாறிகளை வரையறுக்கும் போது, இந்த மாறிகளுக்குரிய ஆரம்பப் பெறுமானங்களைக் கொடுக்க முடியும்.

உதாரணமாக,

```
int age = 20, marks = 92, total = 0;
double basic_sal = 20000, net_sal=25000, otRate=100.00;
```

ஆனால், மாறிகளை வரையறுக்கும் போது மாறிகளுக்குரிய பெறுமானத்தினை கீழேயுள்ளவாறு கொடுக்க முடியாது.

உதாரணமாக, int x=y=z=10; என்ற கட்டளையினைப் பயன்படுத்த முடியாது.

விபர இனங்களுக்கு முன்னால் unsigned என்ற சி++ மொழி கீவேட் (Keyword) இனை ஒரு இடைவெளி விட்டு எழுதினால், இந்த மாறியில் நேர் எண்களை (Positive numbers) மட்டுமே சேமிக்க முடியும். ஆனால், முன்னரைவிட இரு மடங்கு அதிகமான எண்களை இந்த மாறியில் தற்போது சேமிக்க முடியும்.

உதாரணமாக, unsigned int age என வரையறுக்கப்பட்ட age என்ற மாறியில், 0 இலிருந்து 65536 வரையிலான நேர் எண்களைச் சேமிக்க முடியும். அதாவது, சாதாரணமாக int விபர இன மாறியில் சேமிப்பதைவிட இரு மடங்கு நேர் பெறுமானத்தினைத் தற்போழுது சேமிக்க முடியும். இவ்வாறு long, float, double, char போன்ற விபர இனங்களில், இந்த unsigned என்ற சி++ மொழி சிறப்புச்சொல்லினைப் பயன்படுத்த முடியும்.

unsigned என்ற சி++ கீவேட் (Keyword) இனைப் போல் signed என்ற சி++ கீவேட்டும் உண்டு. இந்த signed என்ற சொல்லும் விபர இனத்துக்கு முன்னால் குறிப்பிடப்படுகின்றது. விபர இனத்துக்கு முன்னால் signed என்ற சொல் குறிப்பிட்டிருந்தால் நேர், மறை எண்களை இந்த மாறியில் சேமிக்க முடியும்.

2.3 மாறிலிகளின் பிரயோகங்கள்

சி++ மொழிப் புறோகிராமில் எவ்வாறு மாறிலிகள் (Constants) வரையறுக்கப்படுகின்றன என அடுத்துப் பார்ப்போம்.

மாறிலிகளை வரையறுக்கும் போது முதலில் const என்ற சி++ மொழி கீவேட் (Keyword) இனையும், அடுத்ததாக விபர இனத்தினை யும் (உதாரணமாக: int, long, char, double, float) பின்னர், மாறிலியின் பெயரினையும் குறிப்பிடப்பட வேண்டும்.

உதாரணமாக,

```
const  float PI = 3.141;
const  int MAX_LEN = 100;
```

என மாறிலிகளை வரையறுக்க வேண்டும்.

மாறிலிக்குக் கொடுக்கப்பட்ட பெறுமானத்தினைப் புறோகிராம் செயற்பட்டு முடியும் வரை மாற்றக்கூடாது என்பது குறிப்பிடத்தக்க விடயமாகும். தவறுதலாக மாறிலிகளுக்குரிய பெறுமானத்தினை மாற்றினால், இந்தப் புறோகிராமினைக் கொம்பைல் (Compile) செய்யும் போதே, கொம்பைலர் (Compiler) பிழையினைச் சுட்டிக்காட்டும்.

சி மொழியிலிருந்து சி++ மொழி வெளியிடப்பட்டிருப்பதனால், சி மொழியில் பயன்படுத்தப்படும் அநேகமான கட்டளைகளை சி++ மொழி யிலும் பயன்படுத்த முடியும் என்பதை நாம் அறிவோம். மாறிலிகளை சி மொழியில் வரையறுப்பது போல் சி++ மொழியிலும் வரையறுக்க முடியும்.

உதாரணமாக,

```
#define PI 3.141;
#define MAX 200;
```

என சி மொழியில் வரையறுப்பதுபோல், சி++ மொழியில் வரையறுக்க முடியும். எனினும், முதலாவதாகக் குறிப்பிட்ட const என்ற முறையினைப் பயன்படுத்துவது சிறந்ததாகும்.

விபர இன மாற்றம் (Casting)

ஒரு விபர இனத்திலுள்ள மாறியின் பெறுமானத்தினை வேறொரு விபர இனப் பெறுமானமாக மாற்ற முடியும். இதையே சி++ மொழியில் காஸ்ற்றிங் (Casting) என அழைக்கப்படுகின்றது.

உதாரணமாக, எழுத்து விபர இன பெறுமானத்தினை முழு எண் விபர இனப் பெறுமானமாக மாற்ற வேண்டுமாயின், (int)'a' எனக் குறிப்பிட வேண்டும். அதாவது,

char ch = 'a';

int x = (int)ch; அல்லது int x = int(ch); எனக் குறிப்பிட முடியும். இவ்வாறு, தசம எண்ணை முழு எண்ணாகவும், எழுத்தினை முழு எண்ணாகவும் மாற்ற முடியும்.

உதாரணமாக,

```
double num = 45.67;
int x = (int)num;   // x = 45
int y = (int)33.34; // y = 33
```

சி++ மொழியில் char விபர இனத்தைச் சேர்ந்த மாறியில் முழு எண்ணினைச் சேமிக்க முடியும். இவ்வாறு int விபர இனத்தைச் சேர்ந்த மாறியில் எழுத்தினைச் சேமிக்க முடியும்.

உதாரணமாக,

char ch = 65; என்ற கட்டளையானது ஒழுங்காகச் செயற்படும். அதாவது, 65 இன் அஸ்க்கீ (ASCII) மதிப்பான "A" என்ற பெறுமானம் ch என்ற மாறியில் சேமிக்கப்படும்.

int i = 'A' எனவும் வரையறுக்க முடியும். அதாவது, i என்ற முழு எண் மாறியில் பெறுமானம் 65 சேமிக்கப்படும்.

மாறிகள், மாறிலிகள் உள்ளடக்கிய உதாரணப் புறோகிராமினை அடுத்துப் பார்ப்போம்.

```
// Find the area of the circle
#include <iostream.h>
void main()
{
    float r, area;
    const float PI=3.141;
    cout<< "Enter the radius of circle: ";
    cin>>r;
    area = PI*r*r;
    cout<<"Area of the circle ="<<area<<endl;
    cin.get();
}
```

இந்தப் புறோகிராமிலுள்ள முதலாவது கட்டளையானது, iostream.h என்ற ஹெடர் ∴பைல் (Header File) இனைப் புகுத்தப் பயன்படுத்தப் பட்டுள்ளது. இந்த ஹெடர் ∴பைலினை புகுத்தினால் மட்டுமே cin, cout போன்ற கட்டளைகளையும் >>, << போன்ற ஒப்பறேற்றர்களையும் பயன்படுத்த முடியும்.

மெயின் (main) என்ற ∴பங்ஷனிற்குள் காணப்படும் முதலாவது கட்டளையில் வட்டத்தின் ஆரையையும், பரப்பளவையும் சேமித்து வைப்பதற்கு முறையே r, area போன்ற இரு தசம தான மாறிகள் வரையறுக்கப்பட்டுள்ளன. அடுத்த கட்டளையின் மூலமாக, PI என்ற மாறிலி வரையறுக்கப்பட்டு, அதில் 3.141 என்ற பெறுமானம் சேமிக்கப் பட்டுள்ளது. அடுத்துள்ள கட்டளைகள், வட்டத்தின் ஆரையினை உள்ளீடு செய்யவும், வட்டத்தின் பரப்பளவைக் கணிப்பிடுவதற்கும் எழுதப்பட்டுள்ளன. இறுதிக் கட்டளையானது, வெளியீட்டினைத் திரையில் நிறுத்தி வைப்பதற்குப் பயன்படுத்தப்பட்டுள்ளது.

இந்தப் புறோகிராமினைச் செயற்படுத்திப் பார்த்தால், முதலில் வட்டத்தின் ஆரையினை உள்ளீடு செய்யுமாறு கேள்வி கேட்கப்படும். உதாரணமாக, உள்ளீடாக 7 இனைக் கொடுத்து என்ரர் கீ (Enter key) இனை அழுத்தினால், வட்டத்தின் பரப்பளவை 154.0 எனக் கணித்து விடையைக் கணினித்திரையில் காண்பிக்கும்.

மாறிகளின் வகைகள் (Type of Variables)

ஒரு புறோகிராமானது இயங்கும்போது தற்காலிகமாக மதிப்புக் களைச் சேமித்து வைக்கப் பயன்படும் இடமே மாறிகள் என அழைக்கப் படுகின்றன என ஏற்கனவே பார்த்துள்ளோம்.

இந்த மாறிகளை லோக்கல் மாறிகள் (Local Variables), குளோபல் மாறிகள் (Global Variables) என இரு வகையாகப் புறோகிராமில் பயன்படுத்த முடியும்.

லோக்கல் மாறிகள் (Local Variables) :

குறித்ததொரு ∴பங்ஷனில் மட்டும் பயன்படுத்தக்கூடிய மாறிகள் லோக்கல் மாறிகள் என அழைக்கப்படும்.

உதாரணமாக,

```
#include <iostream.h>
void main()
{
        int x, y;  // x and y are local variables
        x = 10, y = 35;
}
```

குளோபல் மாறிகள் (Global Variables) :

குறித்ததொரு புறோகிராமில் காணப்படும் எல்லா ஃபங்ஷின்களிலும் பயன்படுத்தக்கூடிய மாறிகள், குளோபல் மாறிகள் என அழைக்கப்படும். உதாரணமாக,

```
#include <iostream.h>
int  x = 20;
double salary = 45000.00;  // salary is a global variable
void test1()
{
     x = 40;
     cout<<" x = "<<x<<endl;  // x = 40
}
void test2()
{
     cout<<" x = "<<x<<endl;  // x = 20
}
void main()
{
     test1();
     test2();
     cout<<" x = "<<x<<endl;  // x = 20
     cout<<"Salary ="<<salary<<endl;  //Salary=45000.00
}
```

இந்தப் புறோகிராமினைச் செயற்படுத்திப் பார்த்தால், கீழேயுள்ளவாறு வெளியீடானது காண்பிக்கப்படும்.

```
x = 40
x = 20
x = 20
Salary = 45000.00
```

ஸ்கோப் ரிசொலுஷன் ஒப்பறேற்றர் (:: - Scope Resolution Operator)

குளோபல் மாறியிலுள்ள பெறுமானத்தினை, இந்த ஒப்பறேற்றரினைப் பயன்படுத்தி ஃபங்ஷின்களில் கையாள முடியும்.
உதாரணமாக,

```
#include <iostream.h>
int x = 30;  //  x is a global variable
void main()
{
    int x = 20;
```

```
        cout<<"Local variable x value ="<<x<<endl; // x = 20
        cout<<"Global variable x value ="<<::x<<endl; //x=30
    }
```

இந்தப் புறோகிராமினைச் செயற்படுத்திப் பார்த்தால், கீழேயுள்ளவாறு வெளியீடானது காண்பிக்கப்படும்.

```
Local variable x value = 20
Global variable x value = 30
```

ஒரு ∴பங்ஷனிலுள்ள லோக்கல் மாறிகளை அந்த ∴பங்ஷனில் மட்டுமே பயன்படுத்த முடியும். ஆனால், குளோபல் மாறிகளை எந்த ∴பங்ஷன்களிலும் பயன்படுத்த முடியும்.

ஒரே பெயர் கொண்ட இரு லோக்கல் மாறிகளை ஒரு ∴பங்ஷனில் வரையறுக்க முடியாது. எனினும், வெவ்வேறு ∴பங்ஷன்களில் ஒரே பெயர் கொண்ட லோக்கல் மாறிகள் வரையறுக்க முடியும். உதாரணமாக,

```
#include <iostream.h>
int  x = 10;  //  x is a global variable
void test1()
{
    int x = 20;
    cout<<" x = "<<x<<endl;  // x = 20
}
void test2()
{
    int x = 30;
    cout<<" x = "<<x<<endl;  // x = 30
}
void main()
{
    test1();
    test2();
    int  x = 40;
    cout<<" x = "<<x<<endl;  // x = 40
    cout<<" Global variable x = "<<::x<<endl;  // x = 10
}
```

இவ்வாறு, வெவ்வேறு ∴பங்ஷன்களில் ஒரே பெயர் கொண்ட லோக்கல் மாறிகளை வரையறுக்க முடியும்.

இந்தப் புறோகிராமினைச் செயற்படுத்திப் பார்த்தால், கீழேயுள்ளவாறு வெளியீடானது காண்பிக்கப்படும்.

```
x = 20
```

```
x = 30
x = 40
Global variable x = 10
```

அடுத்ததாக, சி++ மொழி சிறப்புச் சொல்லான sizeof இனைப் பார்ப்போம்.

இந்த sizeof என்ற கட்டளையின் மூலமாக, சி++ மொழி விபர இனங்களுக்குரிய நினைவக இடத்தின் அளவினைப் பெற முடியும்.

உதாரணமாக, sizeof(char) என்ற கட்டளையின் மூலமாக char விபர இனத்திற்குத் தேவையான நினைவக இடத்தின் அளவினைப் பெற முடியும். இவ்வாறு int, long, float, double போன்ற விபர இனங்களுக்குத் தேவையான நினைவக இடத்தின் அளவினையும் பெற முடியும். இதற்குரிய புறோகிராமினை அடுத்துப் பார்ப்போம்.

```
#include <iostream.h>
void main()
{
    char ch;
    int i;
    int x[20];
    cout << sizeof (ch) <<", "; // size of character ch
    cout << sizeof (i) <<", ";   // size of integer i
    cout << sizeof (float) <<", "; // size of  float
    cout << sizeof (double) << ", "; // size of double
    cout << sizeof (x) << ", "; // size of integer array x
}
```

இந்தப் புறோகிராமுக்குரிய வெளியீடானது 1, 2, 4, 8, 40 ஆகும். அதாவது char, int, float, double, integer array போன்ற விபர இனங்களுக்குத் தேவையான நினைவக இடத்தின் அளவினை பைற்ஸ் (Bytes) சார்பாக வெளிக்காட்டும்.

typedef கட்டளை :

ஏற்கனவே சி++ மொழியில் காணப்படும் விபர இனங்கள், கிளாஸ், ஸ்ரக்சர் போன்ற பெயர்களுக்குப் புதிய பெயரினை வழங்குவதே typedef என்ற கட்டளையின் செயற்பாடாகும்.

உதாரணமாக, unsign int விபர இனத்துக்குப் பதிலாக uint எனப் புதிய பெயரினைக் கொடுக்க முடியும்.

typedef unsigned int uint; என்ற கட்டளையினை எழுதிய பின்னர், uint x,y; என x, y மாறிகளை வரையறுக்க முடியும். இவ்வாறு typedef என்ற கட்டளையின் மூலம் கிளாஸ், ஸ்ரக்சர் போன்றவற்றை வரையறுக்க முடியும்.

உதாரணம் (1)

```
typedef struct
{
    char name[30];
    int age;
    char result;
} Student;
Student s;
```

உதாரணம் (2)

```
typedef struct
{
    int day, month, year;
} date;
date d1, d2, d3;
date d = {26,11,1954};
```

இவ்வாறு typedef என்ற கட்டளையின் மூலமாக ஸ்ரக்சரிற்குப் புதிய பெயரினை வழங்க முடியும்.

உதாரணம் (3)

```
typedef class
{
    char empNo[10];
    char name[30];
    int age;
    double bsal, netsal;
} Employee;
Employee e;
```

இவ்வாறு typedef என்ற கட்டளையின் மூலமாக கிளாஸிற்குப் புதிய பெயரினை வழங்க முடியும்.

சி++ மொழியில் புறோகிராம்களை எழுதும் போது கட்டளை அமைப்புக்கள் ஒழுங்கான முறையில், எளிதில் புரிந்து கொள்ளும் வகையில் எழுதப்பட வேண்டும்.

உதாரணமாக, கீழேயுள்ள புறோகிராமினைச் சற்று அவதானித்துப் பாருங்கள்.

```
#include <iostream.h>
void main(){ int x, y, max; cout<<"Enter two numbers: ";
cin>>x>>y; if (x>y) max = x; else max = y;
cout<<"Maximum = "<<max<<endl; }
```

இந்தப் புறோகிராமில் காணப்படும் கட்டளைகளில் எதுவித பிழையுமில்லை. எனினும், கட்டளைகள் சரியான அமைப்பில் எழுதப்பட வில்லை. இவ்வாறு புறோகிராம்களை எழுதுவதால் ஏற்படக்கூடிய பிரச்சினைகள் கீழே தரப்பட்டுள்ளன.

- இந்தப் புறோகிராமில், ஏதாவது கட்டளை அமைப்பு (Syntax) பிழைகள் காணப்பட்டால் கண்டுபிடிப்பது கடினமாக இருக்கும்.
- இந்தப் புறோகிராமினை வேறொருவர் புரிந்து கொள்வது மிகவும் கடினமாக இருக்கும்.
- இவ்வாறு புறோகிராமினை எழுதினால், மீண்டும் இந்தப் புறோகிராமினை மீள்திருத்தம் செய்யும்போது எமக்குக் குழப்பம் ஏற்பட வாய்ப்புண்டு.

இந்த மூன்று பிரச்சினைகளையும் நிவர்த்தி செய்வதற்காகவே, புறோகிராம்களை எழுதும்போது கட்டளைகளின் அமைப்பு ஒழுங்கான முறையில் அமைய வேண்டும் என வலியுறுத்தப்படுகிறது.

புறோகிராம்களை எழுதும்போது, கட்டளைகளைக் கீழேயுள்ளவாறு எழுதுவது சிறந்ததாகும்.

```
#include <iostream.h>
void main()
{
    int x, y, max;
    cout<<"Enter two numbers : ";
    cin>>x>>y;
    if (x>y)
        max = x;
    else
        max = y;
    cout<<"Maximum = "<<max<<endl;
}
```

இவ்வாறு புறோகிராம்களை எழுதும்போது கட்டளைகளுக்கு இடையில் இடைவெளிகள் விட்டு, அடுத்தடுத்த வரிகளில் கட்டளை களை எழுத வேண்டும்.

நீங்கள் எந்தக் கணினி உயர்நிலை மொழிகளைப் பயன்படுத்தி புறோகிராம்களை எழுதினாலும், மேலேயுள்ள புறோகிராம் அமைப்பினைப் பின்பற்றுவது சாலச் சிறந்ததாகும்.

தரவுகளை உள்ளீடு செய்வதற்குப் பயன்படுத்தப்படும் cin என்ற கட்டளையில் காணப்படும் ஃபங்ஷன்கள் :

சி++ மொழியில் தரவுகளை உள்ளீடு செய்வதற்கு cin என்ற

கட்டளை பயன்படுத்தப்படுகிறது என்பதை நீங்கள் அறிவீர்கள். cin என்பது console in என்பதன் சுருக்கமாகும். கணினியின் உள்ளீட்டுச் சாதனமான கீபோட் (Keyboard) இலிருந்து பெறப்படும் தரவுகள் cin என்ற கட்டளை மூலம் நினைவகத்தில் சேமித்து வைக்கப்படுகிறது. இந்தக் கட்டளை மூலம் ஒரு தரவினை அல்லது பல தரவுகளை உள்ளீடாகப் பெற முடியும்.

உதாரணமாக, ஒரு தரவினை மட்டும் உள்ளீடு செய்து, இந்த உள்ளீடு செய்யப்பட்ட தரவினை ஒரு மாறியில் சேமிக்க வேண்டுமாயின்,

```
int age;
cin>>age;
```

எனக் கட்டளைகளை எழுத வேண்டும்.

உதாரணமாக, பல தரவுகளை உள்ளீடு செய்து, இந்த உள்ளீடு செய்யப்பட்ட தரவுகளை பல மாறிகளில் சேமிக்க வேண்டுமாயின்,

```
int a, b, c, d;
cin>>a>>b>>c>>d;
```

எனக் கட்டளைகளை எழுத வேண்டும். இங்கு முதலாவது தரவை உள்ளீடு செய்த பின்னர், என்ரர் கீ (Enter key) இனை அழுத்தியோ அல்லது ஸ்பேஸ் பார் (Space bar) இனை அழுத்தியோ மற்றைய தரவுகளை உள்ளீடு செய்ய வேண்டும்.

இந்த cin என்ற கட்டளையில், பல ஃபங்ஷன்கள் காணப்படுகின்றன. இவற்றினை உதாரணங்கள் மூலம் தெளிவாகப் பார்ப்போம்.

cin.get() : இந்த கட்டளையின் மூலமாக எழுத்துக்கோவை (String) இல் உள்ள குறிப்பிட்ட எண்ணிக்கையான எழுத்துக்களை மட்டும் சேமிப்பதற்குப் பயன்படுத்த முடியும்.

உதாரணமாக, cin.get() என்ற கட்டளையினை ஒரு எழுத்தினை மட்டும் சேமிப்பதற்குப் பயன்படுத்த முடியும்.

```
char ch;
ch = cin.get();
```

இந்தக் கட்டளையைப் புறோகிராம்களின் இறுதி வரியாகப் பயன் படுத்துவதன் மூலம், என்ரர் கீ (Enter key) இனை அழுத்தும் வரை புறோகிராமின் வெளியீட்டினைத் திரையில் நிறுத்தி வைக்க முடியும்.

உதாரணமாக புறோகிராம்களின் இறுதி வரியாக,

```
cout<<"Press enter key to continue";
cin.get();
```

எனக் கட்டளைகளை எழுத முடியும்.

உதாரணமாக,

```
char  name[50];
```

cin.get(name, 20) என்ற கட்டளையின் மூலம், உள்ளீடாகக்

கொடுக்கப்பட்ட முதல் 19 எழுத்துக்களை name என்ற மாறியில் சேமிக்க முடியும்.

உதாரணமாக,

char firstName[30];

cin.get (firstName, sizeof (firstName)) என்ற கட்டளையின் மூலம், இந்த அறேயில் சேமிக்கக்கூடிய 29 எழுத்துக்களை firstName என்ற மாறியில் சேமிக்க முடியும்.

இவ்வாறு குறிப்பிட்ட எண்ணிக்கையான எழுத்துக்களை மாறியில் சேமிப்பதற்கு, cin.get() என்ற கட்டளையினைப் பயன்படுத்த முடியும்.

உதாரணமாக,

char name[40];

cin.get (name, 20, '.');

என்ற கட்டளையின் மூலமாக, name என்ற மாறியில் முதல் 19 எழுத்துக்கள் அல்லது "." இனை உள்ளீடாகக் கொடுக்க முன்னர் உள்ளீடு செய்யப்பட்ட எழுத்துக்களைச் சேமிக்க முடியும். அதாவது, இந்தக் கட்டளையின் மூலமாகக் குறிப்பிட்ட எண்ணிக்கையான எழுத்துக்கள் அல்லது குறிப்பிட்ட எழுத்து உள்ளீடு செய்ய முன்னர் கொடுக்கப்படும் எழுத்துக்கோவையினை ஒரு மாறியில் சேமிக்க முடியும்.

உதாரணமாக,

```
#include <iostream.h>
void main()
{
    char name[25];
    cout<<"Enter name : ";
    cin.get(name, 5);
    cout<<name<<endl;
    cin.get(name, 20, '.');
    cout<<name<<endl;
}
```

இந்த புறோகிராமினைச் செயற்படுத்தி Sivakumar.N என உள்ளீடு செய்தால், முதலில் Siva எனவும், பின்னர் kumar எனவும் திரையில் வெளியீடாகக் காண்பிக்கும்.

cin என்ற கட்டளையில் காணப்படும் get() ∴பங்ஷன் போன்ற செயற்பாடுகளை, getline() என்ற ∴பங்ஷின் மூலமும் செயற்படுத்த முடியும்.

உதாரணமாக,

char name[30];

cin.getline (name, 12);

என்ற கட்டளையின் மூலமாக name என்ற மாறியில் முதல் 11 எழுத்துக்களை மட்டும் சேமிப்பதற்குப் பயன்படுத்த முடியும்.

அடுத்து, cin என்ற கட்டளையில் காணப்படும் ignore() என்ற ∴பங்ஷனின் செயற்பாடுகளை உதாரணங்கள் மூலம் பார்ப்போம்.

ignore() என்ற ∴பங்ஷன் மூலம், உள்ளீடு செய்யப்பட்ட எழுத்துக் கோவையில் குறிப்பிட்ட எண்ணிக்கையான எழுத்துக்களைத் தவிர்த்து மாறியில் சேமிக்க முடியும்.

உதாரணமாக,

```
char name[40];
name = "Sivakumar";
cin.ignore(name, 4);
```

என்ற கட்டளையின் மூலமாக name என்ற மாறியில் முதல் 4 எழுத்துக்கள் தவிர்ந்த மற்றைய எழுத்துக்களான "kumar" என்ற எழுத்துக்கோவை சேமிக்கப்படும்.

ignore() என்ற ∴பங்ஷினுக்குரிய மற்றுமொரு உதாரணத்தினைப் புறோகிராம் ரீதியாகப் பார்ப்போம்.

திகதியினை உள்ளீடு செய்யும் போது, முதலில் மாதத்தினையும் அடுத்து "/" என்ற குறியீட்டினையும், பின்னர் நாட்களையும் அடுத்து "/" என்ற குறியீட்டினையும், இறுதியாக ஆண்டினையும் கொடுத்தால், மாதங்கள், நாட்கள், ஆண்டுகள் போன்றவற்றுக்குரிய பெறுமானங்களை தனித்தனி மாறிகளில் சேமிப்பதற்குரிய புறோகிராமினைப் பார்ப்போம்.

```
#include <iostream.h>
void main()
{
    int month, day, year;
    cout<<"Enter date (mm/dd/yy) :";
    cin>>month;
    cin.ignore();
    cin>>day;
    cin.ignore();
    cin>>year;
    cout<<" Month - "<<month<<endl;
    cout<<" Day - "<<day<<endl;
    cout<<" Year - "<<year<<endl;
}
```

இந்த புரோகிராமினைச் செயற்படுத்தி உள்ளீடாக 7/23/1983 இனைக் கொடுத்தால்,

Month - 7

Day - 23

Year - 1983 என வெளியீடாகத் திரையில் காண்பிக்கும்.

சி++ மொழியில் cout என்ற கட்டளையின் மூலம், சொற்தொடரினை வெளியீடாகக் கணினித்திரையில் காட்ட முடியும் என்பதை ஏற்கனவே நாம் பார்த்துள்ளோம்.

நீண்ட சொற்தொடரானது இரண்டு வரிகளில் வருமாயின், கீழே உள்ளவாறு கட்டளையினை எழுத வேண்டும்.

```
cout << " Area of  the rectangle \
        is "<<area<<endl;
```

இவ்வாறு, இரண்டு வரிகளில் எழுதப்படும் போது "\" இனைக் குறிப்பிட வேண்டும்.

உதாரணமாக,

```
#include <iostream.h>
void main()
{
   cout<<"Welcome to University of  Colombo, \
           Faculty of Science, Sri Lanka."<<endl;
   cout<<"Thanks"<<endl;
}
```

3. ஒப்பறேற்ரர்கள் (Operators)

சி++ மொழியில் பயன்படுத்தப்படும் ஒப்பறேற்ரர்கள், இங்கு ஐந்து பிரிவுகளாகப் பிரித்து ஆராயப்படவுள்ளன.

1. அசைன்மென்ற் ஒப்பறேற்ரர்கள் (Assignment Operators)
2. அரித்மெற்ரிக் ஒப்பறேற்ரர்கள் (Arithmetic Operators)
3. றிலேஷனல் ஒப்பறேற்ரர்கள் (Relational Operators)
4. லொஜிக்கல் ஒப்பறேற்ரர்கள் (Logical Operators)
5. பிற்வைஸ் ஒப்பறேற்ரர்கள் (Bitwise Operators)

3.1 அசைன்மென்ற் ஒப்பறேற்ரர்கள் (Assignment Operators)

மாறிகளில் அல்லது மாறிலிகளில், பெறுமானங்களைச் சேமிக்க அசைன்மென்ற் ஒப்பறேற்ரர்கள் பயன்படுத்தப்படுகின்றன.

அசைன்மென்ற் ஒப்பறேற்ரர்களைப் பின்வருமாறு அட்டவணைப் படுத்திப் பார்ப்போம்.

ஒப்பறேற்ரர்	செயற்பாடு
=	பெறுமானங்களை மாறிகளில் சேமிப்பதற்குப் பயன்படும்.
+=	குறித்த பெறுமானம் கூட்டப்பட்டு மாறியில் சேமிக்கப்படும்.
−=	குறித்த பெறுமானம் கழிக்கப்பட்டு மாறியில் சேமிக்கப்படும்.
*=	குறித்த பெறுமானம் பெருக்கப்பட்டு மாறியில் சேமிக்கப்படும்.
/=	குறித்த பெறுமானம் வகுக்கப்பட்டு மாறியில் சேமிக்கப்படும்.
%=	குறிப்பிட்ட மதிப்பால் வகுக்கும் போது வரும் மீதியானது மாறியில் சேமிக்கப்படும்.

அட்டவணை - 3.1

உதாரணமாக, basic_salary என்ற மாறியில் 25000 இனைச் சேமிக்க வேண்டுமாயின், basic_salary = 25000 எனக் குறிப்பிடப்பட வேண்டும்.

```
age = 28;
net_salary = 25000.00;
```

இவ்வாறு மாறிகளில் பெறுமானங்களைச் சேமிப்பதற்கு, "=" என்ற ஒப்பறேற்றரானது பயன்படுத்தப்படுகிறது. அத்துடன், குறித்ததொரு மாறியில் உள்ள பெறுமானத்தினை மற்றொரு மாறியில் சேமிக்கவும் முடியும்.

உதாரணமாக,

x = 40;

y = x;

அத்துடன் +=, -=, *=, /=, %= போன்ற மேலும் பல அசைன்மென்ற் ஒப்பறேற்றர்கள் மூலமும், மாறிகளில் பெறுமானங்களைச் சேமிக்க முடியும்.

உதாரணமாக, x+=3 எனப் புறோகிராமில் எழுதினால், x இனது ஆரம்பப் பெறுமானம் 3 இனால் கூட்டப்பட்டு x என்ற மாறியில் சேமிக்கப்படும். அதாவது, x = x + 3 ஆகும். இவ்வாறு,

x -= 2 எனின் x = x-2

x *= 3 எனின் x = x*3

x /= 4 எனின் x = x/4

x %= 2 எனின் x = x%2 ஆகும்.

3.2 அரித்மெற்றிக் ஒப்பறேற்றர்கள் (Arithmetic Operators)

இவை பொதுவாகக் கணக்கீடுகளுக்குப் பயன்படுத்தப்படுகின்றன. இவற்றைப் பின்வருமாறு அட்டவணைப்படுத்திப் பார்ப்போம்.

ஒப்பறேற்றர்	செயற்பாடு
+	எண்களைக் கூட்டுவதற்குப் பயன்படும்.
-	எண்களைக் கழிப்பதற்குப் பயன்படும்.
*	எண்களைப் பெருக்குவதற்குப் பயன்படும்.
/	எண்களை வகுப்பதற்குப் பயன்படும்.
%	வகுக்கும் போது மீதியைப் பெறுவதற்குப் பயன்படும்.
++	ஒவ்வொன்றாகக் கூட்டுவதற்குப் பயன்படும்.
_ _	ஒவ்வொன்றாகக் குறைப்பதற்குப் பயன்படும்.

அட்டவணை - 3.2

அட்டவணை 3.2 இல் காணப்படும் முதல் மூன்று ஒப்பறேற்றர்கள் (Operators) பற்றிக் கூறத் தேவையில்லை. ஏனெனில், இவை ஏற்கனவே அறிமுகமான ஒப்பறேற்றர்களாகும். எனவே, அட்டவணையில் உள்ள ஏனைய ஒப்பறேற்றர்களை உதாரணங்கள் மூலம் தெளிவாகப் பார்ப்போம்.

“/” என்ற ஒப்பறேற்றரானது, வகுப்பதற்குப் பயன்படுத்தப்படுகிறது. ஆனால், இரண்டு முழு எண்களை வகுப்பதற்கு, “/” என்ற ஒப்பறேற்றர் இனைப் பயன்படுத்தினால், விடையினை முழு எண்ணாகப் பெற முடியும்.

உதாரணமாக, 15/4 இனது விடை 3 ஆகும். அதாவது, 15 இல் 4 எத்தனை முறைகள் என்பதனை மட்டுமே விடையாகத் தரும். ஆனால், இந்த இரு எண்களில் ஏதாவதொரு எண் தசம எண்ணாக இருப்பின், உண்மையான தசம தான விடையைப் பெற முடியும்.

உதாரணமாக, 19.0/4 அல்லது 19/4.0 அல்லது 19.0/4.0 என எழுதினால், விடையாக 4.75 இனைப் பெற முடியும்.

ஒரு எண்ணினை மற்றைய எண்ணால் வகுக்கும் போது மீதியாக வரும் பெறுமானத்தினைப் பெறுவதற்கு “%” என்ற ஒப்பறேற்றரானது பயன்படுத்தப்படுகிறது. இந்த ஒப்பறேற்றரானது இரண்டு எண்களும் முழு எண்களாகக் காணப்பட்டால் மட்டுமே பயன்படுத்த முடியும்.

உதாரணமாக, 15 % 6 எனின், 3 ஐ விடையாகத் தரும். அதாவது, 15 ஐ 6 இனால் வகுக்கும் போது மீதியாக வரும் பெறுமானமான 3 இனை விடையாகத் தரும்.

“++” என்ற ஒப்பறேற்றரினைப் பயன்படுத்தி, ஒரு மாறியில் உள்ள பெறுமானத்தினை ஒவ்வொன்றாகக் கூட்டுவதற்குப் பயன்படுத்த முடியும். இந்த “++” என்ற ஒப்பறேற்றரினை மாறிக்கு முன்னாலும், பின்னாலும் குறிப்பிட முடியும். அதாவது, “++” என்ற ஒப்பறேற்றரினை ++x, x++ என இரு வகையாகக் குறிப்பிட முடியும்.

உதாரணமாக,

```
int x = 7;
int y = x++;
```

இங்கு x என்ற முழு எண் மாறியில், ஆரம்பப் பெறுமானமாக 7 வரையறுக்கப்பட்டுள்ளது. பின்னர், மாறி x இன் பெறுமானம் ஒன்றினால் அதிகரிக்கப்பட்டு 8 ஆக மாற்றப்படும். “++” என்ற ஒப்பறேற்றரானது x இற்குப் பின்னால் குறிப்பிட்டிருப்பதனால், x இன் ஆரம்பப் பெறுமானமான 7 இனை y என்ற மாறியில் கொடுத்த பின்னர்தான், x இன் பெறுமானம் ஒன்றால் கூட்டப்படும். எனவே, இந்த உதாரணத்தில் y என்ற மாறியில் 7 என்ற பெறுமானமும், x என்ற மாறியில் 8 என்ற பெறுமானமும் சேமிக்கப்படும்.

“++” என்ற ஒப்பறேற்றரானது, x என்ற மாறிக்கு முன்னால் குறிப்பிடப் பட்டிருந்தால், x இனது பெறுமானம் ஒன்றால் கூட்டப்பட்ட பின்னர்தான் y என்ற மாறியில் சேமிக்கப்படும். எனவே, இங்கு y என்ற மாறியிலும், x என்ற மாறியிலும் பெறுமானம் 8 சேமிக்கப்படும்.

இவ்வாறுதான், "--" என்ற ஒப்பறேற்றரானது ஒவ்வொன்றாகக் குறைப்பதற்குப் பயன்படுத்தப்படும். இந்த "--" என்ற ஒப்பறேற்றரினை மாறிக்கு முன்னாலும், பின்னாலும் குறிப்பிட முடியும். அதாவது, "--" என்ற ஒப்பறேற்றரினை --x, x-- என இரு வகையாகக் குறிப்பிட முடியும்.

உதாரணமாக,

```
int x = 5;
int y = x--;
```

இங்கு x என்ற முழு எண் மாறியின் ஆரம்பப் பெறுமானமாக 5 வரையறுக்கப்பட்டுள்ளது. பின்னர், மாறி x இன் பெறுமானத்தை ஒன்றால் குறைத்து 4 ஆக மாற்றப்படும். "--" என்ற ஒப்பறேற்றரானது x இற்குப் பின்னால் குறிப்பிட்டிருப்பதால், x இன் ஆரம்பப் பெறுமானமான 5 இனை y என்ற மாறிக்குக் கொடுத்த பின்னர்தான், x இன் பெறுமானம் ஒன்றால் குறைக்கப்படும். எனவே, இந்த உதாரணத்தில் y என்ற மாறியில் 5 என்ற பெறுமானமும், x என்ற மாறியில் 4 என்ற பெறுமானமும் சேமிக்கப்படும்.

"--" என்ற ஒப்பறேற்றரானது, x என்ற மாறிக்கு முன்னால் குறிப்பிட்டிருந்தால், x இனது பெறுமானம் ஒன்றால் குறைக்கப்பட்ட பின்னர்தான் y என்ற மாறியில் சேமிக்கப்படும். எனவே, இங்கு x என்ற மாறியிலும், y என்ற மாறியிலும் பெறுமானம் 4 சேமிக்கப்படும்.

உதாரணமாக: int x = 3, y = 6, a = 4, b = 7, c = 5; என x, y, a, b, c போன்ற முழு எண் மாறிகளில், ஆரம்பப் பெறுமானங்கள் சேமிக்கப் பட்டுள்ளன.

(1) y / x = ? (2) y % x = ? (3) x / y = ? (4) x % y = ?
(5) b % x = ? (6) x / a = ? (7) b / a = ? (8) a % b = ?
(9) b = ++x எனின் x = ?, b = ?(10) x = c-- எனின் x = ?, c = ?

மேலேயுள்ள உதாரணங்களுக்குரிய விடைகள்:

(1) y / x = 6/3 விடை 2 (2) y % x = 6%3 விடை 0
(3) x / y = 3/6 விடை 0 (4) x % y = 3%6 விடை 3
(5) b % x = 7 %3 விடை 1 (6) x / a = 3/4 விடை 0
(7) b / a = 7/4 விடை 1 (8) a % b = 4%7 விடை 4
(9) x = 3
b = ++x எனின் x = 4, b = 4
(10) c = 5
x = c-- எனின் x = 5, c = 4

சி++ மொழியில் ++, --, % போன்ற ஒப்பறேற்றர்களை முழு எண்களுக்கு மட்டுமே பயன்படுத்த முடியும் என்பது குறிப்பிடத்தக்க விடயமாகும். உதாரணமாக, 4.3%3 எனக் கட்டளை எழுதுவது தவறாகும்.

3.3 ரிலேஷனல் ஒப்பரேற்றர்கள் (Relational Operators)

இவை பொதுவாக இரண்டு மதிப்புகளை ஒப்பிடுவதற்குப் பயன் படுத்தப்படுகின்றன. இவற்றைப் பின்வருமாறு அட்டவணைப்படுத்திப் பார்ப்போம்.

ஒப்பரேற்றர்	செயற்பாடு
== (equal to)	இரு மதிப்புக்களும் சமனா எனச் சோதனை செய்யப் பயன்படும்.
!= (not equal to)	இரு மதிப்புக்களும் சமனற்றவையா எனச் சோதனை செய்யப் பயன்படும்.
>	இரு மதிப்புக்களை ஒப்பிடுகையில், முதலாவது மதிப்பு பெரியதா?
<	இரு மதிப்புக்களை ஒப்பிடுகையில், முதலாவது மதிப்பு சிறியதா?
>=	இரு மதிப்புக்களை ஒப்பிடுகையில், முதலாவது மதிப்பு பெரிது அல்லது சமனா?
<=	இரு மதிப்புக்களை ஒப்பிடுகையில், முதலாவது மதிப்பு சிறிது அல்லது சமனா?

அட்டவணை - 3.3

அட்டவணை- 3.3 இல், முதலாவதாகக் கூறப்பட்ட ஒப்பரேற்றரான == (Equal to) இனை முதலில் பார்ப்போம்.

இந்த ஒப்பரேற்றரானது, இரண்டு மதிப்புக்களும் சமனா எனச் சோதனை செய்வதற்குப் பயன்படுத்தப்படுகின்றது. அதாவது, a == b எனின், மாறிகள் a, b இல் உள்ள இரண்டு மதிப்புக்களும் சமனா எனச் சோதனை செய்யும். இங்கு a = b என ஒற்றைக்குறியினைப் பயன்படுத்துவது தவறாகும். ஏனெனில், சி++ மொழியில் ஒற்றைக் குறியினைப் பயன்படுத்தினால், அசைன்மென்ற் ஒப்பரேற்றர் (Assignment Operator) ஆகத் தொழிற்படும்.

அட்டவணை - 3.3 இல் குறிப்பிடப்பட்ட ரிலேஷனல் ஒப்பரேற்றர்களின் வெளியீடானது, பூலியன் (Boolean) மதிப்பாகவே காணப்படும். அதாவது, உண்மை (True) அல்லது பொய் (False) என்ற மதிப்புக்களை மட்டுமே கொண்டிருக்க முடியும்.

இந்த ரிலேஷனல் ஒப்பரேற்றர்கள் தீர்வுசெய் கட்டளையான if ... இலும், இற்றரேஷன் கட்டளைகளான for ... , while ... , do ... while போன்றவற்றிலும் அதிகம் பயன்படுத்தப்படுகின்றன. இவற்றினை இனிவரும் அத்தியாயங்களில் பார்ப்போம்.

3.4 லொஜிக்கல் ஒப்பரேற்றர்கள் (Logical Operators)

இரண்டு அல்லது இரண்டுக்கு மேற்பட்ட நிபந்தனைகளைச் சோதனை செய்வதற்கு இந்த லொஜிக்கல் ஒப்பரேற்றர்கள் பயன்படுத்தப்படுகின்றன. இவற்றைப் பின்வருமாறு அட்டவணைப்படுத்திப் பார்ப்போம்.

ஒப்பரேற்றர்	செயற்பாடு
&& (and)	இரு நிபந்தனைகளை ஒன்று சேர்த்துச் சோதனை செய்யப் பயன்படுத்தப்படும்
|| (or)	இரு நிபந்தனைகளில் ஏதேனும் ஒரு நிபந்தனை சரியா எனச் சோதனை செய்யப் பயன்படுத்தப்படும்
! (not)	நிபந்தனைக்கு எதிர்மாறானதா எனச் சோதனை செய்யப் பயன்படுத்தப்படும்.

அட்டவணை - 3.4

இந்த லொஜிக்கல் ஒப்பரேற்றர்களின் முடிவுகளைக் கீழேயுள்ள அட்டவணை மூலம் இலகுவான முறையில் பெற முடியும்.

A	B	A || B	A && B	!A
F	F	F	F	T
T	F	T	F	F
F	T	T	F	T
T	T	T	T	F

அட்டவணை - 3.5

இங்கு T - True, F - False ஆகும். அதாவது, || என்ற லொஜிக்கல் ஒப்பரேற்றரினை A, B போன்ற இரு நிபந்தனைகளுக்கிடையில் பயன் படுத்தினால் A, B போன்ற இரு நிபந்தனைகளும் பொய் (false) ஆகக் காணப்பட்டால் மட்டுமே, இறுதி முடிவு பொய்யாக அமையும். மற்றைய சந்தர்ப்பங்கள் யாவற்றிலும் இறுதி முடிவு உண்மையாகவே அமையும்.

&& என்ற லொஜிக்கல் ஒப்பரேற்றர் A, B போன்ற இரு நிபந்தனை களுக்கிடையில் பயன்படுத்தப்பட்டால் A, B போன்ற இரு நிபந்தனை களும் உண்மையாகக் காணப்பட்டால் மட்டுமே இறுதி முடிவு உண்மையாக அமையும். மற்றைய சந்தர்ப்பங்கள் யாவற்றிலும் இறுதி முடிவு பொய்யாகவே அமையும்.

3.5 பிற்வைஸ் ஒப்பரேட்ரர்கள் (Bitwise Operators)

பிற்வைஸ் ஒப்பரேற்றர்கள் என்றால், ஒரு மாறியின் மதிப்பை பைனரி (Binary) பிற்களில் (Bits) கையாளுதலாகும். இவற்றைக் கீழேயுள்ள அட்டவணை மூலம் பார்ப்போம்.

ஒப்பரேற்றர்கள் (Operators)	செயற்பாடு
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR (Exclusive OR)
<<	Bitwise Shift Left
>>	Bitwise Shift Right

அட்டவணை - 3.6

பிற்வைஸ் ஒப்பரேற்றர்கள் பிற்களின் (Bits) துணை கொண்டு இயங்குகின்றன. அதாவது, 0 அல்லது 1 ஆகும்.

பிற்வைஸ் ஒப்பரேற்றர்களின் முடிவுகளைக் கீழேயுள்ள அட்டவணை யின் மூலம் பார்ப்போம்.

a	b	a & b	a | b	a ^ b
0	0	0	0	0
1	0	0	1	1
0	1	0	1	1
1	1	1	1	0

அட்டவணை - 3.7

உதாரணமாக: a = 6, b = 10 எனின், a & b, a | b என்பவற்றின் பெறுமானங்கள் எவ்வாறு கணிக்கப்படுகின்றன எனப் பார்ப்போம்.

இந்த இரு எண்களையும் முதலில் பைனரி (Binary) எண்களாக மாற்றி எழுத வேண்டும். உதாரணமாக: a = 0110, b = 1010 ஆகும்.

அட்டவணை - 3.7 இல் உள்ள முடிவுகளிலிருந்து,

a & b = 0010 = 2

a | b = 1110 = 14 போன்ற விடைகளைப் பெற முடியும்.

மேலே கூறப்பட்ட பிற்வைஸ் ஒப்பரேற்றர்களுக்குரிய மேலும் பல உதாரணங்களை அட்டவணை - 3.8 இன் மூலம் பார்ப்போம்.

a	b	binary a	binary b	a & b	a | b	a ^ b
6	5	110	101	100 = 4	111 = 7	011 = 3
2	4	010	100	000 = 0	110 = 6	110 = 6
9	7	1001	0111	0001 = 1	1111 =15	1110 =14
11	2	1001	0010	0000 = 0	1011 =11	1011 =11
3	8	0011	1000	0000 = 0	1011 =11	1011 =11

அட்டவணை - 3.8

அடுத்து, ஷிவ்ற் (Shift) ஒப்பறேற்றர்களான றைற் ஷிவ்ற் (Right Shift), லெவ்ற் ஷிவ்ற் (Left Shift) போன்றவற்றினைப் பார்ப்போம்.

1. றைற் ஷிவ்ற் (Right Shift) ஒப்பறேற்றர் (>>)

உதாரணமாக, 8>>2 என்பதன் விடை எவ்வாறு கணிக்கப்படுகின்றது எனப் பார்ப்போம்.

8 என்பதனை பைனரியில் எழுதினால், 1000 ஆகும். இந்த பைனரி எண்ணை இருமுறை வலப் பக்கம் இடமாற்றம் செய்ய வேண்டும். அதாவது, 0010 ஆகும். எனவே, இதன் விடை 2 ஆகும். இவற்றினைக் கீழேயுள்ளவாறும் கணிப்பிட முடியும்.

8>>2 என்பதன் கருத்து $8/2^2 = 8/4 = 2$ ஆகும்.

18>>3 என்பதன் கருத்து $18/2^3 = 18/8 = 2$ ஆகும்.

2. லெவ்ற் ஷிவ்ற் (Left Shift) ஒப்பறேற்றர் (<<)

8 << 2 எனின், 8 இன் பைனரி இலக்கத்தை, இருமுறை இடது பக்கம் இடமாற்றம் செய்ய வேண்டும். அதாவது, 100000 (அடி இரண்டு) ஆகும். எனவே, இதன் விடை 32 ஆகும். இவற்றினைக் கீழேயுள்ள வாறும் கணிப்பிட முடியும்.

8<<2 என்பதன் கருத்து $8*2^2 = 8*4 = 32$ ஆகும்.

12<<3 என்பதன் கருத்து $12*2^3 = 12*8 = 96$ ஆகும்.

சி++ மொழியில் மேலும் சில ஒப்பறேற்றர்கள் காணப்படுகின்றன. அவையாவன *, →, &, . போன்றனவாகும்.

"*" என்ற ஒப்பறேற்றரானது, ஒரு விபர இனத்தின் பெயருக்கு அடுத்தோ அல்லது ஒரு மாறியின் பெயருக்கு முன்பாகவோ வருமாயின், அது பொயின்ரர் (Pointer) ஆகத் தொழிற்படும். அதாவது, int *a எனின், a என்பது ஒரு முழு எண்ணைச் சுட்டும் பொயின்ரர் ஆகத் தொழிற்படும்.

"&" என்ற ஒப்பறேற்றரானது, நினைவக முகவரியைக் குறிப்பிடு வதற்குப் பயன்படுத்தப்படுகின்றது. அதாவது, &a எனக் குறிப்பிட்டால்,

a இற்குரிய முகவரி, நினைவகத்தில் எங்குள்ளது என்பதனைக் குறிக்கும்.

., → போன்ற ஒப்பறேற்றர்கள், ஒரு ஒப்ஜெக்ரிலுள்ள ஃபங்ஷன்கள் (Functions) அல்லது அற்றிபியூற்கள் (Attributes) என்பவற்றைச் சுட்டிக் காட்டப் பயன்படுத்தப்படுகின்றன. இவற்றைப் பின்னர் தெளிவாகப் பார்ப்போம்.

ஒப்பறேற்றர்களின் முன்னுரிமை:

மாறிகள், மாறிலிகள், மதிப்புக்கள் ஆகியவற்றுடன் ஒப்பறேற்றர்கள் சேர்ந்து காணப்படுதல் எக்ஸ்பிரஷன் (Expression) என அழைக்கப் படுகின்றது. இந்த எக்ஸ்பிரஷனிலுள்ள ஒப்பறேற்றர்களின் முன்னுரிமை எவ்வாறு கொடுக்கப்படுகின்றது என அட்டவணை மூலம் பார்ப்போம்.

ஒப்பறேற்றர்கள்	முன்னுரிமை
++, --	1
*, /, %	2
+, -	3
>, <, >=, <=	4
==, !=	5
&&	6
||	7
=, +=, -=, *=, /=, %=	8

அட்டவணை - 3.9

இந்த அட்டவணையில், முக்கிய ஒப்பறேற்றர்கள் மட்டுமே ஆராயப் பட்டுள்ளது. எக்ஸ்பிரஷனில் உள்ள ஒப்பறேற்றர்களின் முன்னுரிமை சமனாகக் காணப்பட்டால், இடது பக்கத்திலுள்ள ஒப்பறேற்றர் முதலில் செயற்படும்.

உதாரணமாக, 5/3*3-4+9/2 என்ற எக்ஸ்பிரஷனைக் கருதுவோம். இந்த எக்ஸ்பிரஷனில், முதலாவதாகவுள்ள "/" என்ற ஒப்பறேற்றரே முதலில் செயற்படும். பின்னர், முறையே "*", "/", "-", "+" போன்ற ஒப்பறேற்றர்கள் செயற்படும். எனவே, இந்த உதாரணத்துக்குரிய இறுதிப் பெறுமானம் 3 ஆகும்.

உதாரணமாக: int x = 3, y = 2, a = 4, b = 5; என x, y, a, b போன்ற முழு எண் மாறிகளில் ஆரம்பப் பெறுமானங்கள் சேமிக்கப்பட்டுள்ளன.

(1) x + y / a = ? (2) b / x + y*a = ? (3) x / y*a*b = ?

(4) a*b+ y% x =? (5) x + a*y/b = ? (6) x + b / a + y = ?

மேலேயுள்ள உதாரணங்களுக்குரிய விடைகள்:

(1) x + y / a =3+0 விடை 3 (2) b / x + y*a = 1+8 விடை 9

(3) x / y*a*b = 1*4*5 விடை 20 (4) a*b+ y% x = 20+2 விடை 22

(5) x + a*y/b = 3 +8/5 விடை 4 (6) x+b/a+y = 3+1+2 விடை 6

ரேர்னரி ஒப்பறேற்றர் (Ternary Operator)

? : என்கிற இரட்டைக் குறியீடுகள், நிபந்தனைகளைச் சோதனை செய்து குறிப்பிட்ட பணிகளைச் செயற்படுத்துவதற்குப் பயன்படுத்தப் படுகிறது. இந்த ஒப்பறேற்றரினை ரேர்னரி ஒப்பறேற்றர் (Ternary Operator) என அழைக்கப்படுகிறது. இந்த ரேர்னரி ஒப்பறேற்றர் if ... else ... என்ற கட்டளையினை ஒத்த செயற்பாட்டினைக் கொண்டிருக்கும்.

உதாரணம் (1):

int max = (a>b) ? a : b; என ஒரே ஒரு கட்டளையின் மூலம், இரு இலக்கங்களில் மிகப் பெரிய இலக்கத்தினை max என்ற மாறியில் சேமிக்கப் பயன்படுத்த முடியும்.

உதாரணம் (2):

cout<< (a>b) ? b : a; என ஒரே ஒரு கட்டளையின் மூலம், இரு இலக்கங்களில் மிகச் சிறிய இலக்கத்தினைத் திரையில் காட்ட முடியும்.

உதாரணம் (3):

cout<<(m >= 70) ? “D”: (m >= 60) ? “C” : (m > 40) ? “C”: “F”; என ஒரே ஒரு கட்டளையின் மூலம், மாணவனுக்குரிய பெறுபேறுகளைத் திரையில் காட்ட முடியும்.

உதாரணம் (4):

cout<< (n%2 == 0) ? “Even”: “Odd”; என ஒரே ஒரு கட்டளையின் மூலம், கொடுக்கப்பட்ட இலக்கம் இரட்டை எண்ணா? அல்லது ஒற்றை எண்ணா? எனத் திரையில் காட்ட முடியும்.

4. கட்டுப்பாட்டுக் கட்டளைகள் (Control Statements)

சி++ மொழியில் மூன்று வகையான கட்டுப்பாட்டுக் கட்டளைகள் காணப்படுகின்றன. அவையாவன,

1. தீர்வுசெய் கட்டளைகள் (Selection Statements)
   உதாரணமாக: if ..., if ... else ..., if ... else if ... else ..., switch ... case ...
2. சுழற்சிக் கட்டளைகள் (Iteration Statements)
   உதாரணமாக: for ... , while ... , do .... while
3. தாவும் கட்டளைகள் (Jump Statements)
   உதாரணமாக: break, continue, return, goto

4.1 தீர்வுசெய் கட்டளைகள் (Selection Statements)

கட்டுப்பாட்டுக் கட்டளைகளில், முதலில் தீர்வுசெய் கட்டளைகளைப் பார்ப்போம்.

சி++ மொழியில் இரண்டு வகையான தீர்வுசெய் கட்டளைகள் பயன் படுத்தப்படுகின்றன. அவையாவன,

1. if .... else ....
2. switch ... case ...

முதலில், தீர்வுசெய் கட்டளையான if ..... இனைப் பார்ப்போம். இந்த if என்ற கட்டளை அமைப்பானது, சி++ மொழியில் மூன்று வகையாகப் பயன்படுத்தப்படுகின்றது. அவையாவன,

1.
```
if (நிபந்தனை அல்லது நிபந்தனைகள்)
{
    ----- ;
    ----- ;
}
```

2.
```
if (நிபந்தனை அல்லது நிபந்தனைகள்)
{
    ----- ;
    ----- ;
}
else
{
    ----- ;
    ----- ;
}
```

3. if (நிபந்தனை அல்லது நிபந்தனைகள்)

```
{
    ----- ;
    ----- ;
}
else if (நிபந்தனை அல்லது நிபந்தனைகள்)
{
    ----;
    ----;
}
else
{
    ----;
    ----;
}
```

பொதுவாக if என்ற கட்டளையானது, நிபந்தனை அல்லது நிபந்தனை களைச் சோதனை செய்து, இறுதி முடிவுக்கு அமைய சில பணிகளைச் செயற்படுத்துவதற்குப் பயன்படுத்தப்படுகிறது.

முதலாவதாகக் கூறப்பட்ட if என்ற கட்டளையானது, ஒரு நிபந்தனை அல்லது பல நிபந்தனைகளைச் சோதனை செய்து இறுதி முடிவு உண்மையாகக் காணப்பட்டால் சில பணிகளைச் செயற்படுத்துவதற்குப் பயன்படுத்தப்படுகிறது. இந்த if என்ற கட்டளையிலுள்ள நிபந்தனை களின் இறுதி முடிவு தவறாகக் காணப்பட்டால், என்ன செய்ய வேண்டும் எனக் குறிப்பிடப்படவில்லை. அதாவது, நிபந்தனையைச் சோதனை செய்யும் போது உண்மையாக (True) இருக்கலாம் அல்லது பொய்யாக (False) இருக்கலாம். ஆனால், உண்மையாக இருக்கும் போது மட்டுமே முதலாவதாகக் கூறப்பட்ட if என்ற கட்டளை நிறைவேற்றப்படும். பொய்யாக இருந்தால், ஒரு செயற்பாட்டையும் நிறைவேற்றாது.

சி++ மொழியில், முழு எண் மதிப்புக்களே பூலியன் (Boolean) இனமாகக் கையாளப்படுகின்றன. 0 என்ற மதிப்பு பொய் (False) எனவும், 0 தவிர்ந்த மற்றைய முழு எண் மதிப்புக்கள் உண்மை (True) எனவும் கருதப்படுகிறது.

if என்ற கட்டளைக்குள் ஒரேயொரு கட்டளை மட்டும் காணப்பட்டால், “{”, “}” போன்றன பயன்படுத்த வேண்டிய அவசியமில்லை.

உதாரணமாக,

```
if (age >=18)
      cout<<"You are a matured boy ";
```

இந்த உதாரணத்தில், வயதானது 20 ஐ விடக் கூடுதலாகக் காணப்பட்டால் “You are a matured boy” என்ற வாக்கியத்தைத்

திரையில் காண்பிக்கும். வயதானது 20 ஐ விடக் குறைவாகக் காணப்பட்டால், எந்த வாக்கியத்தையும் திரையில் காண்பிக்காது.

மேலேயுள்ள உதாரணத்தில், ஒரு கட்டளை மட்டுமே உள்ளபடியால் "{", "}" போன்றன பயன்படுத்தப்படவில்லை.

முதலாவதாகக் கூறப்பட்ட if என்ற கட்டளையினைப் பயன்படுத்தி நான்கு முழு எண் பெறுமானங்களில், மிகச் சிறிய எண்ணை எவ்வாறு கண்டுபிடிப்பது என்பதனை ஒரு உதாரணப் புறோகிராம் மூலம் பார்ப்போம்.

```
#include <iostream.h>
// find the minimum number
void main()
{
     int a, b, c, d;
     cout<<"Enter four integer numbers : ";
     cin>>a>>b>>c>> d ;
     int min = a;
     if (min > b)
          min = b;
     if (min > c)
          min = c;
     if (min > d)
          min = d;
     cout<<"The minimum  number is "<<min<<endl;
}
```

இந்தப் புறோகிராமை இயக்கியவுடன், முதலில் நான்கு முழு எண்களை உள்ளீடு செய்யுமாறு கேள்வி கேட்கப்படும். நீங்கள் நான்கு முழு எண்களையும் உள்ளீடாகக் கொடுத்தால், இந்த நான்கு முழு எண்களில் மிகச் சிறிய இலக்கத்தை வெளியீடாகத் திரையில் காண்பிக்கும்.

இந்தப் புறோகிராமில், முதலில் min என்ற மாறியில் முதல் இலக்கத்தினைச் சேமிக்கும் விதமாகக் கட்டளை எழுதப்பட்டுள்ளது. பின்னர், இரண்டாவது இலக்கத்தினை இந்த min என்ற மாறியில் உள்ள பெறுமானத்துடன் ஒப்பிட்டுப் பார்த்து, இதில் சிறியதை மீண்டும் min என்ற மாறியில் சேமிக்கப்படும். இவ்வாறு மேலும் இருமுறை செய்யும் போது, இறுதியாக min என்ற மாறியில் மிகச் சிறிய இலக்கம் சேமிக்கப்படும். இறுதி வரியானது, min என்ற மாறியில் உள்ள பெறுமானத்தைத் திரையில் காட்டுவதற்கு எழுதப்பட்டுள்ளது. இந்த min என்ற மாறியில் இறுதியாக உள்ள எண்ணானது நீங்கள் உள்ளீடு செய்த நான்கு எண்களில் மிகச் சிறிய எண்ணாகும்.

அடுத்ததாக, if ... else ... என்ற தீர்வுசெய் கட்டளையினைப் பார்ப்போம்.

இந்தக் கட்டளையானது, நிபந்தனை அல்லது நிபந்தனைகளைச் சோதனை செய்து உண்மையாக இருந்தால் சில பணிகளைச் செயற்படுத்தும், பொய்யாக இருந்தால் வேறு சில பணிகளைச் செயற்படுத்தும்.

உதாரணமாக,

```
if (age >= 18)
     cout << " You are a matured boy";
else
     cout << " You are not a matured boy";
```

வயதானது 18 அல்லது 18 ஐ விடக் கூடவாக இருப்பின், திரையில் You are a matured boy என்ற வசனத்தைக் காண்பிக்கும். வயதானது 18 ஐ விடக் குறைவாக இருப்பின், You are not a matured boy எனத் திரையில் காண்பிக்கும்.

இறுதியாகக் குறிப்பிடப்பட்ட if ... else if ... else ... என்ற கட்டளை யானது, முதல் நிபந்தனையைச் சோதனை செய்து உண்மை எனின், சில பணிகளைச் செய்யும். முதல் நிபந்தனை பொய் என்றால், மீண்டும் ஒரு நிபந்தனையைச் சோதனை செய்து இந்நிபந்தனை உண்மை எனின், வேறு சில பணிகளைச் செய்யும். இவ்வாறு தொடர்ந்து நிபந்தனைகளைப் பல படிகளில் சோதனை செய்வதற்கு இக்கட்டளை பயன்படுத்தப்படுகின்றது. இதனை நெஸ்ரட் இ∴வ் கட்டளை (Nested if Statement) என அழைக்கப்படுகின்றது.

உதாரணமாக,

```
if ((marks)>100 || (marks<0))
     cout <<" Your marks is invalid ";
else if (marks >= 70)
     cout <<" Your grade is very good ";
else if (marks >= 50)
     cout <<"Your grade is good ";
else
     cout << "Your grade is bad ";
```

ஒரு மாணவனது மதிப்பெண்ணினை உள்ளீடாகக் கொடுக்கும் போது, மதிப்பெண்களுக்கு ஏற்றால்போல் வெவ்வேறு நிபந்தனைகள் செயற்படுத்தப்படும்.

மேலே கூறப்பட்ட மூன்று if கட்டளைகளில், ஒன்றுக்கு மேற்பட்ட நிபந்தனைகளைச் சோதனை செய்ய வேண்டுமெனின் && , ||, ! போன்ற லொஜிக்கல் ஒப்பறேற்றர்களைப் பயன்படுத்த வேண்டும்.

உதாரணமாக, மாணவனின் மதிப்பெண் வீச்சுக்கு ஏற்றால்போல் நிபந்தனைகளைச் சோதனை செய்யும் விதமாகக் கட்டளையினைக் கீழேயுள்ளவாறு எழுத முடியும்.

```
if ((marks >= 70) && (marks <=100))
     cout<<"Your grade is good";
```

இந்த if ... else if ... else ... என்ற கட்டளையினைப் பயன்படுத்தி எழுதப்பட்ட ஒரு உதாரணப் புறோகிராமினைப் பார்ப்போம்.

```
#include <iostream.h>
void main()
{
     double bsal, nsal;
     cout<<"Enter the Basic Salary : ";
     cin>>bsal;
     if (bsal > 30000)
          nsal = bsal- 0.5*bsal;
     else  if (bsal > 20000)
          nsal = bsal - 0.3*bsal;
     else  if (bsal >10000)
          nsal = bsal - 0.2*bsal;
     cout<<"Net Salary = "<<nsal<<endl;
}
```

மேலே கூறப்பட்ட உதாரணத்தில், ஆரம்பச் சம்பளத்தை உள்ளீடாகக் கொடுக்கும் போது குறிப்பிட்டளவு பணத்தொகையினை பாதுகாப்பு நிதியாக சம்பளத்திலிருந்து அரசாங்கம் எடுப்பதனால், எமக்குக் கிடைக்கும் தேறிய சம்பளமானது ஆரம்பச் சம்பளத்தில் இருந்து பாதுகாப்பு நிதியைக் கழிப்பதன் மூலம் பெறும் விதமாக இந்தப் புறோகிராம் கட்டளைகள் எழுதப்பட்டுள்ளன.

அடுத்த தீர்வுசெய் கட்டளையான switch ... case என்ற கட்டளையைத் தெளிவாகப் பார்ப்போம்.

இந்த switch ... case கட்டளையானது, பல படிகளில் நிபந்தனை களைச் சோதனை செய்யப் பயன்படுத்தப்படுகின்றது.

switch ... case கட்டளையின் அமைப்பு:

```
switch (மாறி)
{
     case பெறுமானம் 1:
           statements;
           break;
     case பெறுமானம் 2:
           statements;
```

```
            break;
        case பெறுமானம் 3:
            statements;
            break;
        ......
        ......
        default:
            statements;
    }
```

மாறி எனக் குறிப்பிடப்பட்ட இடத்தில், முழு எண்களைக் கையாளும் மாறிகளையும், எழுத்துக்களை கையாளும் மாறிகளையும் மட்டுமே பயன்படுத்த முடியும். இங்கு தசம எண்களைக் கையாளும் மாறிகளைப் பயன்படுத்த முடியாது.

மாறியில் கொடுக்கப்படும் பெறுமானத்திற்கு அமைய case இற்குரிய கட்டளைகள் செயற்படும்.

உதாரணமாக மாதங்களுக்குரிய எண்ணினை (அதாவது, 1- ஜனவரி, 2 -பெப்ரவரி, 3 - மார்ச், 4 -ஏப்பிரல், ...) உள்ளீடு செய்யும் போது, அந்த மாதத்தில் காணப்படும் நாட்கள் எத்தனை என்பதனை வெளியீடாகக் காட்டுவதற்குரிய சி++ மொழிப் புறோகிராமினைப் பார்ப்போம்.

```
#include <iostream.h>
void main()
{
    int  month, days, y;
    cout<<"Enter the month number : ";
    cin>>month;
    if (month == 2)
    {
        cout<<"Enter the year : ";
        cin>>y;
        if ((y%4 == 0) && ((y%100 != 0) || (y%400 == 0)))
            days = 29;
        else
            days = 28;
    }
    else
    {
        switch (month)
        {
```

```
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
                days = 31;
                break;
            case 4:
            case 6:
            case 9:
            case 11:
                days = 30;
                break;
            default :
                days = 0;
        }
    }
    cout<<"Month "<<month<<" = "<<days<<" days"<<endl;
    cin.get();
}
```

இந்தப் புறோகிராமிலுள்ள முதல் இரு கட்டளைகளும் ஏற்கனவே பலமுறை ஆராயப்பட்டவையாகும். எனவே, இந்தப் புறோகிராமிலுள்ள மற்றைய கட்டளைகளுக்குரிய விளக்கங்களினைப் பார்ப்போம்.

மெயின் ஃபங்ஷனிலுள்ள முதலாவது கட்டளையில் மாத எண், நாட்கள், ஆண்டு போன்றவற்றினைச் சேமிப்பதற்குரிய முழு எண் மாறிகள் வரையறுக்கப்பட்டுள்ளன. பின்னர், அடுத்த இரு கட்டளைகள் மூலம் முறையே மாதத்திற்குரிய எண்ணினை உள்ளீடு செய்யுமாறு கேட்கப்படும் கட்டளையும், month என்ற மாறியில் மாதத்திற்குரிய எண்ணினைச் சேமிப்பதற்குரிய கட்டளையும் எழுதப்பட்டுள்ளன.

அடுத்ததாகவுள்ள கட்டளைகள்தான் இங்கு முக்கியமானவையாகும். குறித்த ஆண்டில் உள்ள பெப்ரவரி மாதத்தில் எத்தனை நாட்கள் உள்ளன என்பது, அக்குறித்த ஆண்டு லீப் ஆண்டா? அல்லது இல்லையா? என்பதனைப் பொறுத்து அமையும். எனவேதான் ஆண்டினை உள்ளீடு செய்யுமாறு கேட்கப்படும் கேள்வியும், y என்ற மாறியில் ஆண்டினைச் சேமிப்பதற்குரிய கட்டளையும் எழுதப்பட்டுள்ளன. அடுத்த கட்டளையானது, அக்குறித்த ஆண்டு லீப் ஆண்டா? அல்லது இல்லையா? என்பதனைச் சோதனை செய்ய எழுதப்பட்டுள்ளது.

லீப் ஆண்டு என்றால் என்ன என்பதை முதலில் பார்ப்போம். ஒரு குறித்த ஆண்டை நான்கினால் வகுக்கும் போது மீதியாகப் பூச்சியமாகவும், அத்துடன் அக்குறிப்பிட்ட ஆண்டை 100 ஆல் வகுக்கும் போது பூச்சியமல்லாமலும் அல்லது அந்த ஆண்டை 400 ஆல் வகுக்கும் போது பூச்சியமாகவும் வருமாயின், அந்த வருடம் லீப் வருடம் என அழைக்கப்படுகின்றது.

லீப் ஆண்டாக இருந்தால், பெப்ரவரி மாதத்தில் 29 நாட்களும், லீப் ஆண்டு இல்லாதிருந்தால், பெப்ரவரி மாதத்தில் 28 நாட்களும் காணப்படும்.

இவற்றை சி++ மொழியில் எழுதுவோமாயின்,

```
if ((y%4 == 0) && ((y%100 != 0) || (y%400 == 0)))
      days =29 ;
else
      days = 28;
```

அடுத்ததாகவுள்ள switch ... case என்ற கட்டளையானது, ஒவ்வொரு மாதத்திலும் உள்ள நாட்கள் எத்தனை என்பதனைக் கணிப்பிட உதவுகிறது.

switch என்ற சி++ மொழி கீவேட்டிற்கு அடுத்தால் போல், அடைப்புக் குறிகளுக்குள் மாத எண்ணினைச் சேமித்து வைத்திருக்கும் மாறியான month இனைக் குறிப்பிட வேண்டும். (இங்கு முக்கியமாகக் கவனிக்க வேண்டியது யாதெனில், அடைப்புக்குறிகளுக்குள் மட்டுமே மாறியினைக் குறிப்பிட வேண்டும். மற்றும், இந்த மாறியின் விபர இனமானது int,short, char ஆக மட்டுமே காணப்பட வேண்டும். மாறாக float, double போன்ற விபர இனங்களை உடைய மாறிகளைப் பயன்படுத்த முடியாது)

மாத எண்களுக்கு அமைய switch ... case கட்டளை செயற்பட்டு, அந்த மாத எண்ணுக்குரிய நாட்கள் எத்தனை என்பதனை days என்ற மாறியில் சேமிக்கும். அத்துடன், break என்ற கட்டளையினைக் குறிப்பிட வேண்டும். இந்த break என்ற கட்டளையானது switch ... case என்ற கட்டுப்பாட்டுக்கு வெளியே கொண்டு செல்வதற்குப் பயன்படுகின்றது. இந்த break கட்டளையானது குறிப்பிடப்படாதிருந்தால், அடுத்ததாகவுள்ள case கட்டளைகளைச் சோதனை செய்யாது, அப்படியே செயற்படுத்தும். இதன் விளக்கத்தினை அடுத்த உதாரணம் மூலம் பார்ப்போம்.

புறோகிராமின் இறுதியில், உள்ளீடாகக் கொடுக்கும் மாதத்திற்குரிய நாட்கள் எத்தனை என்பதனைத் திரையில் காண்பிப்பதற்குரிய கட்டளைகள் எழுதப்பட்டுள்ளன.

இந்தப் புறோகிராமினைச் செயற்படுத்தியவுடன், முதலில் மாதத்திற் குரிய எண்ணினை உள்ளீடு செய்யுமாறு கேள்வி கேட்கப்படும். உள்ளீடாக 1, 3, 5, 7, 8, 10, 12 ஆகிய எண்களில் ஏதாவதொரு எண்ணினை உள்ளீடு செய்தால் days என்ற மாறியில் 31 என்ற பெறுமானமும் 4, 6, 9, 11 ஆகிய எண்களில் ஏதாவதொரு எண்ணினை உள்ளீடு செய்தால் days என்ற மாறியில் 30 என்ற பெறுமானமும், 2 என்பதை உள்ளீடு செய்தால், லீப் வருடம் எனின், days என்ற மாறியில் 29 நாட்களும், லீப் வருடம் இல்லாவிட்டால் days என்ற மாறியில் 28 நாட்களும் சேமிக்கப்படும்.

தீர்வுசெய் கட்டளையான switch ... case இல் break கட்டளையானது பயன்படுத்தப்படாதிருந்தால் ஏற்படக்கூடிய விளைவினை உதாரணம் மூலம் பார்ப்போம்.

```
#include <iostream.h>
void main()
{
      int x = 3;
      switch (x)
      {
            case 1:
                  cout << " One ";
            case 2:
                  cout << " Two ";
            case 3:
                  cout << " Three ";
            case 4:
                  cout << " Four ";
            case 5:
                  cout << " Five ";
           default:
                  cout << " Zero ";
       }
      cin.get();
}
```

இந்தப் புறோகிராமிலுள்ள switch ... case கட்டளையில் break என்ற கட்டளை பயன்படுத்தப்படாததால், இதற்குரிய வெளியீடானது

Three

Four

Five எனத் தோன்றும். ஏனெனில், switch ... case கட்டளையில், முதலில் x இனது பெறுமானம் 3 ஆகவுள்ள case கட்டளைகள்

செயற்படுத்தப்படும். இந்த case கட்டளைக்குள் break கட்டளை இல்லை என்பதால், அடுத்தடுத்துள்ள case கட்டளைகள் சோதனை செய்யப்படாது அப்படியே செயற்படும். எனவேதான் x இனது பெறுமானம் 3 ஆகவுள்ள case இற்கு அடுத்துள்ள case கட்டளைகள் செயற்பட்டு வெளியீடானது திரையில் காண்பிக்கப்பட்டுள்ளன.

4.2 சுழற்சிக் கட்டளைகள் (Iteration Statements)

கட்டுப்பாட்டுக் கட்டளைகளில் அடுத்து இற்றரேஷன் (Iteration) கட்டளை களைப் பார்ப்போம்.

இற்றரேஷன் என்றால், ஒரே வேலையைத் திரும்பத் திரும்ப நிபந்தனைக்கு ஏற்ப பலமுறை செய்வதாகும். அதாவது, இற்றரேஷன் என்றால், சுழற்சி எனப் பொருள்படும்.

சி++ மொழியில் மூன்று வகையான இற்றரேஷன் கட்டளைகள் பயன்படுத்தப்படுகின்றன.

அவையாவன,

1. for ....
2. while ...
3. do ... while

முதலில் for ... என்ற இற்றரேஷன் (Iteration) கட்டளையினைப் பார்ப்போம்.

```
for (ஆரம்பப் பெறுமானம்; நிபந்தனை; சுழற்சி)
{
    for loop இற்குரிய உள்ளடக்கப் பகுதி (Body of for loop)
}
```

ஆரம்பப் பெறுமானம் (Initialization) : ஒருமுறை மட்டுமே செயற்படும். இந்த மதிப்பானது, லூப்பின் ஆரம்ப மதிப்பினை (Initial value) மாறியில் சேமிக்கப் பயன்படுகிறது.

நிபந்தனை (Condition) : ஒரு பூலியன் (Boolean) மதிப்பான உண்மை (True) அல்லது பொய் (False) இனைப் பொறுத்து, லூப்பானது திரும்பத் திரும்பச் செயற்படும்.

சுழற்சி (Iteration) : ஆரம்ப மாறி அல்லது லூப்பினைக் கட்டுப்படுத்தும் மாறியின் ஆரம்ப மதிப்பை ஒவ்வொன்றாக, இரண்டாக, மூன்றாக, ... கூட்ட அல்லது குறைக்கப் பயன்படுகிறது.

for என்ற இற்றரேஷன் கட்டளையானது, எத்தனை முறை சுழற்சி நடைபெறும் எனத் தெரிந்தால் மட்டுமே பயன்படுத்த முடியும்.

உதாரணமாக,

```
for (int i=1; i<10; i++)
{
    // for loop இற்குரிய உள்ளடக்கப் பகுதி (Body of loop)
}
```

இந்த for என்ற லூப்பிலுள்ள முதல் கட்டளையானது i என்ற மாறியினை முழு எண் விபர இனமாக வரையறுத்து, ஆரம்பப் பெறுமானம் 1 இடப்பட்டுள்ளது. இந்தக் கட்டளையானது ஒருமுறை மட்டுமே செயற்படும். அடுத்ததாகவுள்ள கட்டளை ஒரு நிபந்தனை யாகும். அதாவது, i என்ற மாறியானது 10 ஐ விடக் குறைவாக இருக்கும் சந்தர்ப்பங்களில் மட்டுமே for என்ற லூப் செயற்படும். அடுத்து, for என்ற லூப்பின் உள்ளடக்கப் பகுதிக் கட்டளைகள் செயற்படும். அடுத்ததாக கட்டளை i++ ஆனது செயற்படும். அதாவது, i என்ற மாறியினை ஒவ்வொன்றாகக் கூட்டுவதற்குப் பயன்படுத்தப் பட்டுள்ளது. பின்னர், மீண்டும் நிபந்தனைக் கட்டளையான i<10 இனைச் சோதனை செய்து உண்மையாக இருக்கும் சந்தர்ப்பத்தில், மீண்டும் லூப்பின் உள்ளடக்கப் பகுதிக் கட்டளைகள் செயற்படும். இவ்வாறு நிபந்தனைக் கட்டளையான i<10 இனைச் சோதனை செய்து, பொய்யாக வரும் வரை இந்த லூப்பானது தொடர்ந்து செயற்படும்.

உதாரணமாக, 1 தொடக்கம் 100 வரையுள்ள இரட்டை எண்களைத் திரையில் வெளியீடாகக் காட்டுவதற்கும், இறுதியில் அதன் கூட்டுத் தொகையினை வெளியீடாகக் காட்டுவதற்குமுரிய புறோகிராமினைப் பார்ப்போம்.

```
#include <iostream.h>
void main()
{
    int sum = 0;
    for (int i = 2; i<=100; i+=2)
    {
        cout<< i <<", ";
        sum += i;
    }
    cout<<"Sum of first 50 even numbers="<<sum<<endl;
    cin.get();
}
```

இந்தப் புறோகிராமினைச் செயற்படுத்திப் பார்த்தால், வெளியீடாக

2, 4, 6, 8, ... , 100

Sum of first 50 even numbers = 5100 எனக் கணினித்திரையில்

காண்பிக்கும். இந்தப் புறோகிராமில், sum என்ற மாறியானது இரட்டை எண்களின் கூட்டுத்தொகையினை சேமித்து வைக்கப் பயன்படுத்தப் பட்டுள்ளது. for என்ற லூப்பிற்குள் உள்ள முதலாவது கட்டளையான i என்ற முழு எண் மாறியானது, ஆரம்பப் பெறுமானமாக 2 இனைச் சேமிக்கின்றது. i ஆனது 100 இலும் குறைவு அல்லது சமனாக வரும் வரை for என்ற லூப்பானது செயற்படும். அடுத்து i += 2 என்ற கட்டளையின் நோக்கம் யாதெனில், i என்ற மாறியினை இரண்டு, இரண்டாகக் கூட்டுவதற்குப் பயன்படுத்தப்பட்டுள்ளது.

for என்ற லூப்பிற்குள் முதலாவதாகவுள்ள கட்டளையானது, திரையில் இரட்டை எண்களை வெளியீடாகக் காட்டுவதற்குப் பயன் படுத்தப்பட்டுள்ளது. அடுத்தாகவுள்ள வரியான sum += i என்ற கட்டளையானது இரட்டை எண்களைக் கூட்டுவதற்கு பயன்படுத்தப் பட்டுள்ளது. இறுதியில், அதாவது for என்ற லூப் முடிவடைந்த பின்னர் இரட்டை எண்களின் கூட்டுத்தொகையைத் திரையில் காட்டுவதற்குரிய கட்டளை எழுதப்பட்டுள்ளது. for என்ற லூப்பிற்குள் ஒரே ஒரு கட்டளை மட்டும் காணப்பட்டால், “{”, “}” போன்ற குறிகள் பயன்படுத்தத் தேவையில்லை.

for என்ற லூப்பினை இறங்குவரிசையாக செயற்படும் விதமாகவும் பயன்படுத்த முடியும்.

உதாரணமாக,

```
for (int i =100; i>=1; i -= 2)
```

இங்கு ஆரம்பப் பெறுமானம் 100 ஆனது i என்ற மாறியில் சேமிக்கப் பட்டுள்ளது. பின்னர், இரண்டிரண்டாகக் குறைந்து i என்ற மாறியானது ஒன்றாக வரும் சந்தர்ப்பம் வரை இந்த for லூப்பானது செயற்படும்.

for லூப்பில் பல மாறிகள் பயன்படுத்தப்படலாம். அதாவது, இரண்டு அல்லது மூன்று மாறிகள் பயன்படுத்த முடியும்.

உதாரணம் (1)

```
for (int i =1, j = 10; i<j ; i +=2, j++)
{
     .....;
     .....;
}
```

ஆரம்ப மதிப்பினைக் கையாளும் மாறிகளைக் கமா (Comma) மூலம் வேறுபிரிக்க வேண்டும்.

உதாரணம் (2)

```
for (int i =3, j = 5; i < j && j<20; i +=2, j++)
{
```

```
        .....;
        .....;
    }
```

நிபந்தனைக் கட்டளைகளின் இறுதி முடிவு பூலியனாக அமைய வேண்டும். ஆதாவது, உண்மை (True) அல்லது பொய் (False) ஆக அமைய வேண்டும்.

for லூப்பைப் பயன்படுத்தி, நாம் உள்ளீடாகக் கொடுக்கப்படும் எழுத்துக்கோவையிலுள்ள இலக்கங்களின் எண்ணிக்கையை காண்பதற் குரிய உதாரணப் புறோகிராமினை அடுத்துப் பார்ப்போம்.

```
#include <iostream.h>
int  isDigit(char ch)
{
    if  ((ch >= 48) && (ch<=56))
        return 1;
    else
        return 0;
}
void main()
{
    char  a[80];  // String variable a
    int count = 0;   //  count digits
    cout <<" Enter the string : ";
    cin>>a;
    for (int i = 0; a[i] != '\0'; i++)
        if  ( isDigit(a[i]) )  count++;
    cout<<"Total number of digits in the string= "<<count;
}
```

பொதுவாகப் பல புறோகிராங்களுக்கு இந்த for லூப்பானது பேருதவிபுரிகின்றது. எனவே, நீங்கள் இந்த for லூப்பின் நெளிவு சுளிவுகளை நன்கு அறிந்து வைத்திருப்பது சிறந்ததாகும்.

அடுத்து, நாம் while என்ற இற்றரேஷன் கட்டளையினை உதாரணம் மூலம் பார்ப்போம்.

while என்ற லூப்பானது, நிபந்தனைகளை முதலில் உண்மையா? எனச் சோதனை செய்த பின்னர் தான் செயற்படும்.

உதாரணமாக,

```
while (நிபந்தனைகள்)
{
    ....;
    ....;
}
```

while என்ற லூப்பில், எத்தனை தரம் செயல்பட வேண்டும் என்பது முன்கூட்டியே தெரியத் தேவையில்லை. மாறாக, நிபந்தனை உண்மை யாக இருக்கும் வரை தொடர்ந்து செயற்படும். மற்றும் நிபந்தனை முதலில் செயற்படுவதால், சில சந்தர்ப்பத்தில் ஒரு முறையேனும் while என்ற லூப்பிற்குள் உள்ள கட்டளைகள் செயற்படாமல் போகலாம்.

உதாரணமாக, 1 தொடக்கம் 100 வரையுள்ள ஒற்றை எண்களைத் திரையில் வெளியீடாகக் காட்டுவதற்கு இந்த while என்ற லூப்பானது எவ்வாறு பயன்படுத்தப்படுகின்றது எனப் பார்ப்போம்.

```
#include <iostream.h>
void main()
{
     int i = 1;
     while ( i <= 100 )
     {
         cout<< i <<", ";
         i += 2;
     }
     cin.get();
}
```

இந்தப் புறோகிராமினைச் செயற்படுத்திப் பார்த்தால், வெளியீடாக 1, 3, 5, 7, ... ,99 எனக் கணினித்திரையில் காண்பிக்கும்.

அடுத்து while என்ற லூப்பிற்குரிய இன்னுமொரு உதாரணத்தினைப் பார்ப்போம்.

```
#include <iostream.h>
void main()
{
     char  ch = 'y';
     while ( ch == 'y' )
     {
         ..... ;
         ..... ;
     }
     cout<<"Do you want to continue (y/n) ? "
     cin>>ch;
}
```

இந்தப் புறோகிராமினைச் செயற்படுத்திப் பார்த்தால், 'y' என்ற எழுத்தினைத் தவிர வேறெந்த ஒரு எழுத்தினை அழுத்தினாலும் while லூப்பை விட்டு வெளியேறும்.

அடுத்ததாக, do ... while என்ற லூப்பைப் பார்ப்போம். இங்கு நிபந்தனைகள் லூப்பின் இறுதியில் காணப்படும். மற்றும்படி while என்ற லூப் மாதிரியே செயற்படும். இந்த do ... while லூப்பானது குறைந்தது ஒரு முறையாவது செயற்படும்.

உதாரணமாக,

```
do
{
    ....;
    ....;
}
while (நிபந்தனைகள்)
```

இங்கு நிபந்தனை உண்மையெனின், மீண்டும் லூப் செயற்படும்.

அடுத்து, do ... while என்ற லூப்பிற்குரிய உதாரணத்தினைப் பார்ப்போம்.

```
#include <iostream.h>
void main()
{
    char  ch = 'y';
    do
    {
        ..... ;
        ..... ;
    }
    while (ch == 'y')
    cout<<"Do you want to continue (y/n) ? "
    cin>>ch;
}
```

இந்தப் புறோகிராமினைச் செயற்படுத்திப் பார்த்தால் 'y' என்ற எழுத்தினைத் தவிர வேறு எந்த ஒரு எழுத்தினை அழுத்தினாலும் do ... while லூப்பை விட்டு வெளியேறும். இங்கு குறைந்தது ஒரு முறையாவது லூப்பானது செயற்படும்.

முதன்மை எண்களை (Prime numbers) வெளியீடாகக் காண்பிக்கக் கூடிய புறோகிராமினைப் பார்ப்போம்.

ஒன்றாலும், தன்னாலும் வகுபடக்கூடிய எண்கள் யாவும் முதன்மை எண்களாகும். முதன்மை எண்களில், 2 தவிர்ந்த மற்றைய எண்கள் யாவும் ஒற்றை எண்களாகும். எனவே, புறோகிராமினை எழுதும் போது ஒற்றை எண்களை மட்டும் கருதினால் போதுமாகும்.

```
#include <iostream.h>
void main()
{
    int n,j,r;
    cout<<"Enter a positive number:";
    cin>>n;
    cout<<"2, ";
    for (int i = 3; i<n; i+=2)
    {
        j=3;
        do
        {
            if (i>j)
            {
                r = i%j;
                j +=2;
            }
            else
            {
                cout<<i<<", ";
                r = 0;
            }
        }
        while (r != 0);
    }
}
```

இங்கு n என்பது உள்ளீடாகக் கொடுக்கும் முழு எண் மாறியாகும். இந்த n என்ற மாறியில் உள்ளீடு செய்யும் இலக்கத்திலும் குறைவாக வுள்ள முதன்மை எண்கள் அனைத்தையும் திரையில் காண்பிக்கும்.

4.3 தாவும் கட்டளைகள் (Jump Statements)

சி++ மொழியில் நான்கு வகையான தாவும் கட்டளைகள் உள்ளன. அவையாவன break, continue, return, goto போன்றவையாகும். ஏற்கனவே, break என்ற கட்டளையை switch .. case இலும், return என்ற கட்டளையை ஃபங்ஷனிலும் பயன்படுத்தியதைப் பார்த்தோம். உதாரணமாக,

```
int sum(int a, int b)
{
    return a+b;
}
```

ஃபங்ஷனுக்கு ஒரு பெறுமானத்தைக் கொடுப்பதுடன், ஃபங்ஷன்னை விட்டு வெளியேறுவதற்கு return என்ற கட்டளை பயன்படுத்தப் பட்டுள்ளது. அதாவது, return என்ற கட்டளையானது இரண்டு செயற்பாடு களுக்குப் பயன்படுத்தப்படுகின்றது. அவையாவன, குறித்தொரு பெறுமானத்தை ஃபங்ஷனுக்குக் கொடுப்பதற்கும், ஃபங்ஷன்னைவிட்டு வெளியேறுவதற்கும் பயன்படுத்தப்படுகிறது.

ஃபங்ஷன்னைவிட்டு வெளியேற return என்ற கட்டளையினைப் பயன்படுத்தும் உதாரணத்தினை அடுத்துப் பார்ப்போம்.

```
void test (int a, int b)
{
    int c = a + b;
    cout<<"Before the return statement ";
    return;
    cout<<"This statement will not be executed!";
}
```

இந்தப் புறோகிராமினைச் செயற்படுத்திப் பார்த்தால், வெளியீடானது Before the return statement இனை மட்டும் திரையில் காண்பிக்கும்.

அடுத்து break இற்கும், continue இற்கும் இடையிலுள்ள வித்தியாசத்தை உதாரணம் மூலம் பார்ப்போம்.

```
void main()
{
    for (int i = 1; i <=10; i++)
    {
        if (i == 5) break;
        cout<<i<<", ";
    }
}
```

இந்தப் புறோகிராமினைச் செயற்படுத்திப் பார்த்தால், வெளியீடானது 1, 2, 3, 4 போன்ற இலக்கங்களை மட்டுமே திரையில் காண்பிக்கும். அதாவது, இங்கு 4 என்ற இலக்கத்திற்கு பின்னர் உள்ள இலக்கங் களை வெளியீடாகக் காண்பிக்காது.

```
void main()
{
    for (int i=1; i<=10; i++)
    {
        if (i == 5) continue;
        cout<<i<<", ";
    }
}
```

இந்தப் புறோகிராமினைச் செயற்படுத்திப் பார்த்தால், வெளியீடானது 1, 2, 3, 4, 6, 7, 8, 9 10 போன்ற இலக்கங்களைத் திரையில் காண்பிக்கும். அதாவது, 5 என்ற இலக்கம் வெளியீடாகக் காண்பிக்காது.

இங்கு break என்ற கட்டளையானது முற்றாக லூப்பை விட்டு வெளியேற்றுவதற்கும், continue என்ற கட்டளையானது குறிப்பிட்ட நிலையில் மட்டும் செயற்படாது, மற்றைய நிலைகளில் செயற்படவும் பயன்படுத்தப்பட்டுள்ளது. இந்த இரண்டு உதாரணங்கள் மூலம் தெளிவாக break, continue போன்ற இரு கட்டளைகளுக்கும் இடையில் உள்ள வித்தியாசம் விளங்கியிருக்கும் என நினைக்கிறேன்.

அடுத்து, goto என்ற தாவும் கட்டளை (Jump Statement) இனை உதாரணம் மூலம் பார்ப்போம்.

goto என்ற கட்டளையானது, குறிப்பிடத்தக்க புறோகிராம் வரிகளைச் செயற்படுத்தாது தாண்டுவதற்குப் பயன்படுத்தப்படுகின்றது.

goto என்ற தாவும் கட்டளைக்குரிய உதாரணத்தினை அடுத்துப் பார்ப்போம்.

```
#include <iostream.h>
void main()
{
    int x = 40;
    goto label1;
    x += 10;
    cout<<"These statements will not work<<endl";
    cout<<"x = "<<x<<endl;
    label1:
    x += 15;
    cout<<"These statements will work"<<endl;
    cout<<"x = "<<x<<endl;
}
```

இந்தப் புறோகிராமினைச் செயற்படுத்திப் பார்த்தால் வெளியீடாக,

These statements will work

x = 55 போன்றவற்றினை மட்டுமே திரையில் காண்பிக்கும்.

இந்த புறோகிராமிலுள்ள முதலாவது கட்டளையான int x = 40 இனை ஏற்கனவே பலமுறை ஆராயப்பட்டதால், அடுத்த கட்டளையான goto label1 இனைப் பார்ப்போம். இங்குள்ள label1 என்பது ஒரு பெயராகும். எனவே, label1 என்பதற்குப் பதிலாக, எநதப் பெயரையும்

பயன்படுத்தலாம். இந்தக் கட்டளையானது அடுத்து செயற்படவிருக்கும் கட்டளையினைத் தீர்மானிக்கும். எனவே, label1 என்ற கட்டளை இந்தப் புறோகிராமில் எங்குள்ளதோ, அந்தக் கட்டளைக்கு அடுத்துள்ள கட்டளைகள் செயற்படும்.

goto என்ற கட்டளையானது, break என்ற கட்டளையினை ஒத்த செயற்பாட்டினைக் கொண்டிருக்கும்.

இந்த goto என்ற கட்டளையினைப் புறோகிராம்களில் பயன்படுத்துவதைத் தவிர்ப்பது நல்லதாகும்.

5. ∴பங்ஷின்கள் (Functions)

கணினி மூலமாகக் குறித்தொரு பணியினைச் செயற்படுத்துவதற்கு புறோகிராம் எழுதப்படுகின்றது. அக்குறித்த பணி மிகப் பெரிய, மிகச் சிக்கலான பணியாக இருந்தால், அந்தப்பணியைச் சிறு, சிறு கூறுகளாகப் பிரித்து, ஒவ்வொரு சிறு பணியையும் தனித் தனி சிறு புறோகிராம் மூலம் செய்து முடிக்கின்றோம். இச்சிறு பணிக்குரிய புறோகிராமே சி++ மொழியில் ∴பங்ஷின் என அழைக்கப்படுகின்றது.

சி++ மொழிக்குரிய புறோகிராமானது மிகப் பெரிய அளவில் காணப்பட்டாலும், அவற்றை மெயின் ∴பங்ஷினில் எழுதிச் செயற்படுத்த முடியும். ஆனால், பல்வேறு காரணங்களுக்காக அந்தப் புறோகிராமினைப் பல ∴பங்ஷின்களாகப் பிரித்துப் பயன்படுத்துகின்றோம்.

அவையாவன,

1. புறோகிராமொன்றில், குறித்ததொரு வேலையைத் திரும்பத் திரும்பச் செய்ய வேண்டிய தேவை ஏற்படும் போது மீண்டும், மீண்டும் அதற்குரிய கட்டளைகளை எழுதுவதற்குப் பதிலாக, தனியாக ஒரு ∴பங்ஷினை எழுதி, இப்புறோகிராமில் பல தடவைகள் பயன்படுத்த முடியும்.

2. பெரிய புறோகிராமைச் சிறு, சிறு கூறுகளாக்கி எழுதும் போது சிக்கல்கள், பிழைகள் போன்றவற்றைத் தவிர்க்க முடியும். மற்றும், புறோகிராமினை பிழைதிருத்தம் (Debugging) செய்வதற்கு இலகுவாக இருக்கும்.

3. ஒரு புறோகிராமில் மட்டுமல்லாமல், எத்தனையோ பல புறோகிராம் களில் சில குறிப்பிட்ட பணியைச் செய்ய வேண்டியிருக்கலாம். அந்தப் பணிக்காக ஒரு ∴பங்ஷினை எழுதி, ஹெடர் ∴பைலில் (Header file) சேமிக்கப்பட்ட பின்னர், எந்தப் புறோகிராம்களிலும் அந்த ∴பங்ஷினைப் பயன்படுத்த முடியும்.

இவ்வாறு பல நன்மைகளைப் ∴பங்ஷின்கள் மூலம் பெற முடியும். சி++ மொழியில் இரண்டு வகையான ∴பங்ஷின்கள் பயன்படுத்தப் படுகின்றன. அவையாவன,

1. உள்ளிணைந்த ∴பங்ஷின்கள்
   (Library Functions)
2. நாமே உருவாக்கிக் கொள்ளும் ∴பங்ஷின்கள்
   (User Defined Functions)

5.1 உள்ளிணைந்த ஃபங்ஷன்கள் (Library Functions)

சி++ மொழியில் ஏற்கனவே வரையறுக்கப்பட்டுள்ள ∴பங்ஷின்கள் உள்ளிணைந்த ∴பங்ஷின்கள் என அழைக்கப்படுகின்றன. அதாவது, ஒரு குறித்த சிறு பணியை நிறைவேற்றுவதற்காக கணினி மொழிக்குரிய கொம்பைலரினை உருவாக்கியவர்களே அதற்குரிய ∴பங்ஷின்களை எழுதி, அம்மொழியோடு உள்ளிணைத்து வைத்திருப்பார்கள். இவற்றைத் தான் உள்ளிணைந்த ∴பங்ஷன்கள் (Library Functions) என அழைக்கப் படுகின்றன. இதன் மூலம் இலகுவாக ஒரு சொல் கட்டளை போல் புறோகிராமில் கையாள முடியும்.

இந்த உள்ளிணைந்த ∴பங்ஷன்களைப் புறோகிராம்களில் பயன் படுத்திக்கொள்ள வேண்டுமாயின், புறோகிராமின் தொடக்கத்தில் இந்த உள்ளிணைந்த ∴பங்ஷன் காணப்படும் ஹெடர் ∴பைல் (Header File) இனை எழுத வேண்டும்.

உதாரணமாக, ஒரு எண்ணின் வர்க்கமூலம் எமக்குப் புறோகிராமில் தேவையெனின், sqrt() என்ற உள்ளிணைந்த ∴பங்ஷன் மூலம் கணிப்பிட முடியும். இந்த sqrt() என்ற உள்ளிணைந்த ∴பங்ஷன், math.h என்ற ஹெடர் ∴பைலில் உள்ளதென்பதால், புறோகிராமின் தொடக்கத்தில் #include <math.h> என்ற வரியினை எழுதிய பின்னர்தான் sqrt() என்ற ∴பங்ஷினைப் புறோகிராமில் பயன்படுத்த முடியும்.

கீழேயுள்ள உதாரணம் மூலம், உள்ளிணைந்த ∴பங்ஷின்கள் எவ்வாறு புறோகிராமில் பயன்படுத்தப்படுகின்றது எனப் பார்ப்போம்.

```
# include <iostream.h>
# include <math.h>
void main()
{
    int num;
    cout<<"Enter the number :  ";
    cin>>num;
    cout<<"Square root of  "<<num<<" = "<<sqrt(num);
    cin.get()
}
```

சி++ மொழியில் பல ஹெடர் ∴பைல்கள் உள்ளன. அவையாவன: iostream.h, math.h, string.h, iomainp.h, fstream.h, ... போன்றனவாகும்.

சி++ மொழியில் இணைந்துள்ள முக்கியமான ஹெடர் ∴பைல்கள் (Header File) இன் விளக்கங்களைச் சற்றுப் பார்ப்போம்.

- iostream.h - உள்ளீட்டு, வெளியீட்டு பணிகளைக் கொண்ட ∴பங்ஷன்கள் இந்த ஹெடர் ∴பைலில் காணப்படுகின்றன.

• dir.h - டொஸ் (DOS) என்ற ஒப்பறேற்றிங் சிஸ்ரத்திற்குரிய கட்டளைகளைக் கையாளும் ∴பங்ஷன்கள் இந்த ஹெடர் ∴பைலில் காணப்படுகின்றன.

• math.h - கணக்கீடுகள் தொடர்பான ∴பங்ஷின்கள் இந்த ஹெடர் ∴பைலில் காணப்படுகின்றன.

• string.h - எழுத்துக்கோவை (String) தொடர்பான ∴பங்ஷின்கள் இந்த ஹெடர் ∴பைலில் காணப்படுகின்றன.

• time.h - நேரம் தொடர்பான ∴பங்ஷின்கள் இந்த ஹெடர் ∴பைலில் காணப்படுகின்றன.

• fstream.h - ∴பைல்கள் தொடர்பான ∴பங்ஷின்கள் இந்த ஹெடர் ∴பைலில் காணப்படுகின்றன.

• graphics.h - கிர∴பிக்ஸ் தொடர்பான ∴பங்ஷின்கள் இந்த ஹெடர் ∴பைலில் காணப்படுகின்றன.

இவைதவிர மேலும் பல சி$^{++}$ மொழி ஹெடர் ∴பைல்கள் காணப்பட்ட போதிலும், அவை புறோகிராமில் அதிகம் பயன்படுத்தப்படாதவையாகும்.

முதலில் இங்கு math.h என்ற ஹெடர் ∴பைலினைச் சற்றுத் தெளிவாகப் பார்ப்போம்.

math.h என்ற ஹெடர் ∴பைலில் கிட்டத்தட்ட 20 ∴பங்ஷின்கள் உள்ளன. இவற்றில் முக்கியமான சில ∴பங்ஷின்களை மட்டும் இங்கு பார்ப்போம்.

1. sqrt() - ஒரு நேர் எண்ணை sqrt() என்ற ∴பங்ஷினிற்குக் கொடுத்தால், அந்த இலக்கத்தின் வர்க்கமூலத்தினை விடையாகப் பெற முடியும்.

உதாரணமாக: sqrt(9) = 3.0, sqrt(25) = 5.0, sqrt(2.25) = 1.5 ஆகும்.

2. abs() - ஒரு எண்ணை abs() என்ற ∴பங்ஷினிற்குக் கொடுத்தால், அந்த இலக்கத்தினை நேர் எண்ணாக மாற்றிப் பெற முடியும்.

உதாரணமாக: abs(-4.9) = 4.9, abs(4.3) = 4.3, abs(-21) = 21 ஆகும். ஒருவருடைய வயதைத் தவறுதலாக மறைப் பெறுமானமாக உள்ளீடு செய்திருந்தால், அந்த எண்ணை நேர் எண் பெறுமானமாக மாற்ற இந்த abs() என்ற ∴பங்ஷின் பயன்படுத்தப்படுகின்றது.

3. pow() - உதாரணமாக: pow(2,5) =32 ஆகும். அதாவது, 2 இன் 5 ஆம் அடுக்கிற்குரிய விடையைப் பெற முடியும். pow(5,3) = 125 ஆகும்.

4. floor() - ஒரு எண்ணை floor() என்ற ஃபங்ஷனிற்குக் கொடுத்தால், அந்த இலக்கத்திலும் கிட்டிய குறைந்த முழு எண்ணினை விடையாகத் தரும்.

உதாரணமாக: floor(4.7) = 4.0, floor(-4.9) = -5.0, floor(8.9)= 8.0 ஆகும்.

5. ceil() - ஒரு எண்ணை ceil() என்ற ஃபங்ஷனிற்குக் கொடுத்தால், அந்த இலக்கத்திலும் கிட்டிய கூடிய முழு எண்ணினை விடையாகத் தரும்.

உதாரணமாக: ceil (4.9) = 5.0, ceil (-4.6) = -4.0, ceil (6.4) = 7.0 ஆகும்.

அடுத்து, string.h என்ற ஹெடர் ஃபைலில் காணப்படும் ஃபங்ஷன் களைப் பார்ப்போம்.

இந்த string.h என்ற ஹெடர் ஃபைலில் காணப்படுகின்ற முக்கியமான ஃபங்ஷன்கள் strcmp(), strlen(), strcpy(), strupr(), strlwr() போன்றனவாகும்.

1. strcmp() - இரண்டு எழுத்துக்கோவைகளை ஒப்பிடுவதற்குப் பயன்படுத்தப்படுகின்றன.

strcmp() இன் கட்டளை அமைப்பு:

strcmp (முதலாவது எழுத்துக்கோவை, இரண்டாவது எழுத்துக்கோவை)

உதாரணமாக,

- strcmp("Hello", "Hello") எனின், 0 இனை விடையாகத் தரும். ஏனெனில், இரண்டு எழுத்துக்கோவைகளும் ஒரேமாதிரி காணப்படுவதால், 0 இனை விடையாகக் காண்பிக்கும்.
- strcmp("Hello", "HELLO") எனின், 1 இனை விடையாகத் தரும். ஏனெனில், இவ்விரண்டு எழுத்துக்கோவைகளில், இரண்டாவது எழுத்துக் கோவையிலுள்ள எழுத்துக்களின் ASCII பெறுமானமானது, முதலாவது எழுத்துக்கோவையிலுள்ள எழுத்துக்களின் ASCII பெறுமானத்திலும் பெரிது என்பதால், விடையாக 1 இனைத் தரும்.
- strcmp("HELLO", "Hello") எனின், -1 இனை விடையாகத் தரும். ஏனெனில், இவ்விரண்டு எழுத்துக்கோவைகளில், இரண்டாவது எழுத்துக் கோவையில் உள்ள எழுத்துக்களின் ASCII பெறுமானமானது, முதலாவது எழுத்துக்கோவையிலுள்ள எழுத்துக்களின் ASCII பெறுமானத்திலும் சிறியது என்பதால், விடையாக -1 இனைத் தரும்.

2. strlen() - எழுத்துக்கோவையில் காணப்படும் எழுத்துகளின் எண்ணிக்கையினைப் பெறுவதற்குப் பயன்படுத்தப்படுகின்றது.

strlen() இன் கட்டளை அமைப்பு :

strlen (எழுத்துக்கோவை)

உதாரணமாக,

strlen("Welcome") எனின், 7 இனை விடையாகத் தரும். அதாவது, எழுத்துக்கோவையில் காணப்படும் எழுத்துகளின் எண்ணிக்கையினை விடையாகத் தரும்.

strlen("Welcome to Jaffna") எனின், 17 இனை விடையாகத் தரும்.

3. strcpy() - இரண்டாவது மாறியில் உள்ள எழுத்துக்கோவையினை முதலாவது மாறியில் பிரதிசெய்யப் பயன்படுத்தப்படுகின்றது.

strcpy() இன் கட்டளை அமைப்பு:

strcmp (முதலாவது எழுத்துக்கோவை, இரண்டாவது எழுத்துக்கோவை)

உதாரணமாக,

char s1[10] = "C++";
char s2[10] = "Java";
strcpy (s2,s1); எனின் s1,s2 மாறிகளில் "C++" என்ற சொல் சேமிக்கப்பட்டிருக்கும்.

strcpy() என்ற ∴பங்ஷன் எவ்வாறு சி++ மொழிப் புறோகிராமில் பயன்படுத்தப்படுகிறது என்பதற்குரிய உதாரணத்தினைப் பார்ப்போம்.

```
//A program is to display the operation of the strcpy()
# include<iostream.h>
# include<string.h>
void main()
{
    char string1[10]="String1";
    char string2[10]="String2";
    cout<<"Before the copy operation :"<<endl;
    cout<<"String 1:\t"<<string1<<endl;
    cout<<"String 2:\t"<<string2<<endl;
    strcpy (string2,string1);
    cout<<"After the copy operation :"<<endl;
    cout<<"String 1:\t"<<string1<<endl;
    cout<<"String 2:\t"<<string2<<endl;
}
```

வெளியீடு (Output):

```
Before the copy operation :
String 1 :        String1
```

```
        String 2 :    String2
    After the copy operation :
        String 1 :    String1
        String 2 :    String1
    Press any key to continue
```

அடுத்து strupr(), strlwr() போன்ற ஃபங்ஷின்களின் செயற்பாடு களைப் பார்ப்போம்.

strupr(), strlwr() போன்ற ஃபங்ஷன்கள் முறையே எழுத்துக் கோவையினை ஆங்கிலப் பெரிய எழுத்துக்கள் (Capital Alphabet Letters) கொண்ட எழுத்துக்கோவையாகவும், ஆங்கிலச் சிறிய எழுத்துக்கள் (Small Alphabet Letters) கொண்ட எழுத்துக்கோவையா கவும் மாற்றும்.

உதாரணமாக,

```
char s1[10] = "colombo";
char s2[10] = "JAFFNA";
char s3[10], s4[10];
```

strupr(s3,s1); எனின், s3 என்ற மாறியில் "COLOMBO" என ஆங்கிலப் பெரிய எழுத்துக்கள் கொண்ட எழுத்துக்கோவை சேமிக்கப்படும்.

strlwr(s4,s2); எனின், s4 என்ற மாறியில் "jaffna" என ஆங்கிலச் சிறிய எழுத்துக்கள் கொண்ட எழுத்துக்கோவை சேமிக்கப்படும்.

5.2 நாமே உருவாக்கிக் கொள்ளும் ஃபங்ஷின்கள் (User Defined Functions)

உள்ளிணைந்த ஃபங்ஷின்களில் காணப்படாத ஒரு செயற்பாடு எமக்கு புறோகிராமொன்றில் தேவைப்படின், நாமே இந்த செயற்பாட்டுக் குரிய ஃபங்ஷினை உருவாக்கிப் புறோகிராமில் பயன்படுத்த முடியும். ஃபங்ஷின்களை எவ்வாறு உருவாக்குவது எனப் பார்ப்போம்.

```
return_type   function_name (Parameter list)
{
    function body
}
```

return_type - ஃபங்ஷின்களுக்குள் காணப்படும் கட்டளைகள் செயற்பட்டு இறுதியாகத் திருப்பி அனுப்பும் மதிப்பின் விபர இனத்தினைக் குறிக்கும்.

உதாரணமாக void, int, float, char, double போன்ற விபர இனங்கள் return_type ஆகக் குறிப்பிட முடியும். இங்கு வொய்ட் (void) என்பது, ஒரு பெறுமானமும் திருப்பியனுப்பாத ஃபங்ஷின்களில் பயன்படுத்தப்படும்.

function_name - ஃபங்ஷன்களின் பெயரினைக் குறிக்கும். இந்த ஃபங்ஷன்களின் பெயர்கள் எவ்வாறு அமைய வேண்டும் என இரண்டாம் அத்தியாயத்தில் தெளிவாக ஆராயப்பட்டுள்ளது.

Parameter list - ஃபங்ஷன்களுக்கு உள்ளீடாகக் கொடுக்கப்படும் மாறிகளைக் குறிக்கும். சில சமயம் ஃபங்ஷனுக்கு ஒரு பராமீற்றரும் காணப்படாதிருக்கலாம்.

function body - ஃபங்ஷன்களுக்குள் எழுதப்படும் கட்டளைகளைக் குறிக்கும்.

ஃபங்ஷனுக்குரிய ஒரு உதாரணத்தினை அடுத்துப் பார்ப்போம். மூன்று எண்களில், மிகப் பெரிய எண்ணைத் தெரிவு செய்வதற்குரிய செயற்பாடு எமக்குத் தேவையெனின், இதற்குரிய ஃபங்ஷனை நாமே எழுதிப் புறோகிராமில் பயன்படுத்த முடியும்.

```
int max (int x, int y, int z)
{
    int  m = x;
    if  (y>m)  m = y;
    if  (z>m)  m = z;
    return  m;
}
```

இந்த ஃபங்ஷனிலுள்ள முதல் வரியானது மேலதிக மாறி m உருவாக்கப்பட்டு, அதில் முதலாவது எண் சேமிக்கப்படுகின்றது. பின்னர், இரண்டாவது எண்ணையும், m என்ற மாறியிலுள்ள எண்ணையும் சோதனை செய்து, இரண்டாவது எண் பெரிது எனின், m என்ற மாறியில் இரண்டாவது எண் சேமிக்கப்படுகின்றது. பின்னர், மூன்றாவது எண்ணையும் m என்ற மாறியிலுள்ள எண்ணையும் சோதனை செய்து, மூன்றாவது எண் பெரிது எனின், m என்ற மாறியில் மூன்றாவது எண் சேமிக்கப்படுகின்றது. இந்த ஃபங்ஷனின் இறுதி வரியானது, m என்ற பெறுமானத்தை ஃபங்ஷனுக்குத் திருப்பி அனுப்பும் விதமாகக் கட்டளை எழுதப்பட்டுள்ளது.

சி++ மொழியில் நாமே உருவாக்கிப் பயன்படுத்தும் ஃபங்ஷன்களில் மூன்று முக்கிய அம்சங்களைக் கையாள வேண்டும்.

1. நாமே உருவாக்கிய ஃபங்ஷனை மெயின் ஃபங்ஷனுக்கு மேலே எழுதியிருந்தால், ஃபங்ஷனின் முன்வடிவம் (Proto type - புறொற்றோ ரைப்) வரையறுக்கப்படத் தேவையில்லை. ஆனால், நாமே உருவாக்கிய ஃபங்ஷனை மெயின் ஃபங்ஷனுக்குக் கீழே எழுதியிருந்தால், கட்டாயம் ஃபங்ஷனின் முன்வடிவம் (Proto type) வரையறுக்கப்பட வேண்டும்.

2. ∴பங்ஷனுக்கு அனுப்ப வேண்டிய பராமீற்றர்களை (Parameters) முன்வடிவத்திலும் (Proto type), ∴பங்ஷன் வரையறுப்பிலும் (அதாவது, ∴பங்ஷன் எழுதும் இடத்திலும்) ஒரே மாதிரி அறிவிக்க வேண்டும்.

3. ∴பங்ஷன் திருப்பி அனுப்பும் மதிப்பானது, எந்த விபர இனத்தைச் சார்ந்தது என்பதை ∴பங்ஷன் அறிவிப்பிலும், வரையறுப்பிலும் கட்டாயம் குறிப்பிட வேண்டும்.

உதாரணமாக, மூன்று முழு எண்களில் மிகச் சிறிய எண்ணைக் கண்டுபிடிப்பதற்குரிய ∴பங்ஷனை அடுத்துப் பார்ப்போம்.

```
int min(int x, int y, int z)
{
    int  m = x;
    if (m>y)
        m = y;
    if (m>z)
        m = z;
    return  m;
}
```

மேலேயுள்ள உதாரணத்தில், min என்ற ∴பங்ஷன் பெயரின் விபர இனம் (Data type) முழு எண்ணாகும். அதாவது, மூன்று முழு எண்களில் மிகச் சிறிய எண்ணை min என்ற ∴பங்ஷன் பெயருக்குத் திருப்பி அனுப்பப் பயன்படுத்தப்பட்டுள்ளது. அதனால்தான் min என்ற ∴பங்ஷன் பெயரும் முழு எண்ணுக்குரிய விபர இனமாக வரையறுக்கப் பட்டுள்ளது.

இரு உதாரணங்கள் மூலம், நாமே உருவாக்கிக் கொள்ளும் ∴பங்ஷன்கள் பற்றிய விளக்கங்களைத் தெளிவாகப் பார்ப்போம்.

உதாரணம் - 1

```
#include <iostream.h>
void main()
{
    int cube(int);   // proto type of cube() function
    int  x;
    cout<<"Enter a number : ";
    cin>>x;
    cout<<" Cube of  "<<x<<" = "<<cube(x)<<endl;
    cin.get();
}
int cube(int n)
{
    return  n*n*n;
}
```

உதாரணம் - 2

```
#include <iostream.h>
int cube(int n)
{
   return  n*n*n;
}
void main()
{
    int x;
    cout<<"Enter a number : ";
    cin>>x;
    cout<<" Cube of "<<x<<" = "<<cube(x)<<endl;
    cin.get();
}
```

இந்த இரு உதாரணங்களிலும் cube() என்ற ∴பங்ஷனின் தொடக்க வரியில் int cube(int x) என வரையறுக்கப்பட்டுள்ளது. இங்கு cube என்பது ∴பங்ஷனின் பெயராகும். int x என அடைப்புக்குறிக்குள் இருப்பது, cube() என்ற ∴பங்ஷன் ஏற்கும் பராமீற்றர் (Parameter) ஆகும். ஒன்றுக்கு மேற்பட்ட பராமீற்றர்கள், ஒரு ∴பங்ஷனுக்கு இருக்க முடியும். மேலேயுள்ள உதாரணங்கள் இரண்டிலும் ஒரேயொரு பராமீற்றர் மட்டுமே காணப்படுகின்றது. இந்த ∴பங்ஷன்கள், int மதிப்பினைப் பராமீற்றராக ஏற்று, int மதிப்பினை விடையாகத் தரும் விதமாகக் கட்டளைகள் எழுதப்பட்டுள்ளன.

இனி, ∴பங்ஷனின் உள்ளடக்கப் பகுதிக்குச் (Body of the Function) செல்வோம். இங்கு ∴பங்ஷன் என்ன பணிகளை ஆற்றுகின்றது என்பதனைக் குறிப்பிட வேண்டும். இந்தப் பகுதிக்குள் ஒரு கட்டளையோ அல்லது பல கட்டளைகளோ இருக்க முடியும். ஆனால், இக்கட்டளைகள் யாவும் இரட்டை அடைப்புக்குறிக்குள் எழுதப்பட வேண்டும். ஒவ்வொரு ∴பங்ஷனுக்குள்ளும் return என்கின்ற கட்டளை இடம்பெற வேண்டிய கட்டாயமில்லை. return என்ற கட்டளை பயன்படுத்துவதன் நோக்கம் யாதெனில், ∴பங்ஷனை முடிவுறச்செய்து, ஒரு குறித்த பெறுமானத்தை ∴பங்ஷன் பெயருக்குத் திருப்பி அனுப்புவதற்காகும். வொய்ட் (void) ∴பங்ஷனுக்கு return என்ற கட்டளை அவசியமில்லை.

உதாரணமாக,

```
void display()
{
    cout <<" Welcome  to  Jaffna ";
}
```

இந்த உதாரணத்தை பார்க்கவும். இங்கு return என்ற கட்டளை பயன்படுத்தப்படவில்லை. ஏனெனில், display() என்ற ஃபங்ஷன் void இனத்தைச் சார்ந்ததாகும். எனவே, ஒரு பெறுமானத்தையும் display() என்ற ஃபங்ஷன் பெறாது. மெயின் ஃபங்ஷனும் void இனத்தைச் சார்ந்ததாக வரையறுக்கப்பட்டிருப்பதால் தான், அங்கு return என்கிற கட்டளை பயன்படுத்தப்படவில்லை.

முதலில் கூறப்பட்ட உதாரணங்கள் இரண்டிலும், ஒரு எண்ணின் கனத்தினைக் கணிப்பதற்குரிய ஃபங்ஷன் எழுதப்பட்டுள்ளது. முதலாவதாக உள்ள உதாரணத்தில், ஃபங்ஷனின் முன்வடிவம் (Function Proto type) மெயின் ஃபங்ஷனுக்குள் முதல் கட்டளையாக வரையறுக்கப்பட்டுள்ளது. cube() என்ற ஃபங்ஷனை மெயின் ஃபங்ஷனுக்கு மேலே எழுதியிருந்தால், முன்வடிவம் எழுதத் தேவை யில்லை. அதாவது, இரண்டாவது உதாரணத்தைப் போல் எழுதப் பட்டிருந்தால், ஃபங்ஷன் முன்வடிவம் எழுதத் தேவையில்லை.

ஃபங்ஷன் விபர இனமானது எதுவாகவும் இருக்க முடியும். அதாவது int, float, double, void, char போன்ற எந்த விபர இனமாகவும் இருக்க முடியும்.

return என்ற கட்டளையின் மூலம் ஒன்றுக்கு மேற்பட்ட மதிப்புக் களை ஒரு ஃபங்ஷனுக்குத் திருப்பி அனுப்ப முடியாது. எனவே, ஒன்றுக்கு மேற்பட்ட மதிப்புக்களைத் திருப்பி அனுப்ப வேண்டும் என்றால் என்ன செய்ய முடியும்?

உதாரணமாக, இரு எண்களை இடம்மாற்றுவதற்குரிய (Interchange) ஃபங்ஷன் ஒன்றை எழுதி, இரு எண்களையும் திருப்பி அனுப்ப வேண்டு மெனின் என்ன செய்யலாம்?

இதற்குத்தான் சி++ மொழியில், முகவர் (Reference) மாறி பயன்படுத்தப்படுகின்றது. சி++ மொழியில் "&" என்பது முகவர் குறியீடு (Reference Operator) ஆகும்.

```
#include <iostream.h>
void main()
{
    //Proto type of a function called swap()
    void swap(int &, int &);
    int  a, b;
    cout<<"Enter two numbers : ";
    cin>>a>>b;
    cout<<" a = "<<a<<endl;
```

```
        cout<<" b = "<<b<<endl;
        swap (a, b);
        cout<<" a = "<<a<<endl;
        cout<<" b = "<<b<<endl;
        cin.get();
    }
    void swap(int &x, int &y)
    {
        int  t = x;
        x = y;
        y = t;
    }
```

மேலேயுள்ள உதாரணத்தில், முதலில் ∴பங்ஷன் முன்வடிவம் (Proto type) அறிவிக்கப்பட்டுள்ளது. இங்கு ∴பங்ஷனிலுள்ள இரு பராமீற்றர்களை அறிவிக்கும்போது சாதாரண மாறியாக அறிவிக்காது, முகவர் மாறியாக அறிவிக்க வேண்டும். ஏனெனில், சாதாரண மாறியாக இந்த இரு பராமீற்றர்களையும் அறிவித்திருந்தால், ∴பங்ஷனுக்குள் மட்டுமே இரு எண்களையும் இடமாற்றம் (Interchange) செய்ய முடியும். இந்த ∴பங்ஷனுக்கு வெளியே இரு எண்களும் ஒருபோதும் தமக்குள் இடமாற்றம் செய்யாது. எனவேதான், இரு மாறிகளையும் முகவர் மாறியாக வரையறுத்துள்ளோம். அதாவது, மாற்றங்கள் யாவற்றையும் ∴பங்ஷனுக்கு வெளியிலும் எடுத்துச் செல்லுவதற்காகும்.

swap() ∴பங்ஷனுக்குள் உள்ள முதலாவது வரியானது t என்ற மேலதிக மாறியை உருவாக்கி, அதில் முதல் மாறியிலுள்ள பெறுமானமானது சேமிப்பதற்குப் பயன்படுத்தப்பட்டுள்ளது. பின்னர், முதலாவது மாறியில், இரண்டாவது மாறியில் உள்ள பெறுமானம் சேமிக்கப்படும். இறுதியாக, இரண்டாவது மாறியில், t என்ற மேலதிக மாறியிலுள்ள பெறுமானத்தினைச் சேமிக்கும் விதமாகக் கட்டளைகள் எழுதப்பட்டுள்ளன.

இங்கு int &x எனக் குறிப்பிடுவதற்குப் பதிலாக int& x எனவும், int&x எனவும் குறிப்பிட முடியும்.

றிகர்சிவ் ஃபங்ஷன்கள் (Recursive Functions)

ஒரு ∴பங்ஷன் தன்னைத்தானே அழைத்துக் கொள்ளக்கூடிய றிகர்சிவ் ∴பங்ஷன் (Recursive Function) பற்றி விரிவாகப் பார்ப்போம்.

இந்த றிகர்சிவ் ∴பங்ஷன் செயற்படும் போது, ஸ்ரக் (Stack) என்ற அமைப்பினை நினைவகத்தில் உருவாக்கி, அதில் இடைநிலை மதிப்புக்கள் சேமிக்கப்பட்டுகின்றது. இறுதியில் இந்த ஸ்ரக்கிலுள்ள

மதிப்புக்கள் ஒவ்வொன்றாக வெளியே எடுத்து இறுதிக் கணிப்பீட்டுக்குப் பயன்படுத்தப்படுகின்றது. அதாவது, LIFO (Last In First Out) என்ற ஸ்ரக்கின் நிபந்தனைக்கு ஏற்ப இறுதியாக ஸ்ரக்கிற்கு அனுப்பப்பட்ட மதிப்பு முதலில் கணிப்பீட்டுக்குப் பயன்படுத்தப்படுகின்றது.

இந்த றிகா்சிவ் ∴பங்ஷனுக்கு அதிக நினைவக இடம் எடுப்பதுடன் புரிந்துகொள்வது கடினமாக இருப்பினும், மிகவும் சிக்கலான கணிதம் சார்ந்த பிரச்சினைகளுக்குரிய தீர்வினை றிகா்சிவ் ∴பங்ஷன் மூலம் மிக இலகுவாக எழுத முடியும்.

உதாரணமாக, ஒரு முழு எண்ணுக்குரிய ∴பக்ரோறியல் (Factorial) மதிப்பினைக் கண்டறிய றிகா்சிவ் என்ற தத்துவத்தைப் பயன்படுத்துவது சிறந்ததாகும். இதற்குரிய புறோகிராமினை அடுத்துப் பார்ப்போம்.

```
// Factorial function by using recursive method
#include <iostream.h>
void main()
{
    int fact(int);  // Proto type of a function called fact()
    int n;
    cout<<"Enter a number : ";
    cin>>n;
    cout<<" Factorial of "<<n<<" = "<<fact(n)<<endl;
    cin.get();
}
int fact(int n)
{
   if (n == 0)
      return 1;
   else
      return n*fact(n-1);
}
```

இந்த உதாரணத்திற்கு 4 என்ற இலக்கத்தினை உள்ளீடு செய்தால், முதலில் இந்த இலக்கம் பூச்சியமா? எனச் சோதனை செய்யும். இல்லையெனின், அடுத்த கட்டளையான, n*fact(n-1) செயற்படும். அதாவது, 4*fact(3) ஆகும். மேலே கூறப்பட்டது போல், ∴பங்ஷன் மீண்டும், மீண்டும் செயற்பட்டு எப்பொழுது fact(0) என வருகிறதோ அப்பொழுதுதான் ∴பங்ஷன் அழைப்பை நிறுத்தும். பின்னா் reverse order இல் கட்டளைகளைப் பூர்த்தி செய்யும்.

fact(4) = 4 * fact(3)

fact(3) = 3 * fact(2)

fact(2) = 2 * fact(1)

fact(1) = 1 * fact(0)

ஆனால் fact(0) = 1 ஆகும்.

fact(4) = 4 * fact(3)

= 4 * 3* fact(2)

= 4 * 3 * 2* fact(1)

= 4 * 3 * 2 * 1 = 24

இந்த புறோகிராமில், return என்ற கட்டளை இருமுறை பயன் படுத்தப்பட்டுள்ளது. எனினும், ஏதாவதொரு return கட்டளை மட்டுமே ∴பங்ஷின் அழைப்பின் போது செயற்படுத்தப்படும்.

சி++ மொழியில் ஒரு ∴பங்ஷனுக்குள் இன்னுமொரு ∴பங்ஷினை அழைக்க முடியுமா?

ஒரு ∴பங்ஷனுக்குள் இன்னுமொரு ∴பங்ஷினை அழைக்க முடியும். எனினும், ஒரு ∴பங்ஷனுக்குள் இன்னுமொரு ∴பங்ஷினை வரையறுக்க முடியாது.

உதாரணமாக, மெயின் ∴பங்ஷனுக்குள் ∴பங்ஷினின் முன்வடிவ அறிவிப்பை (Proto type declaration) மட்டுமே செய்துள்ளோம். மெயின் ∴பங்ஷனுக்குள் முன் அறிவிப்பில் கூறப்பட்ட ∴பங்ஷினை வரையறுக்கவில்லை.

இன்லைன் ஃபங்ஷன்கள் (Inline Functions)

சி++ மொழியில் எழுதப்படும் ∴பங்ஷனுக்கு முன்னால், inline என்ற சி++ மொழிச் சிறப்புச்சொல் (C++ Keyword) இனைச் சேர்த்தால், இன்லைன் ∴பங்ஷனாகத் தொழிற்படும். இந்த இன்லைன் ∴பங்ஷனானது புறோகிராமினைக் கொம்பைல் செய்யும்போதே எங்கெல்லாம் இந்த இன்லைன் ∴பங்ஷன்கள் அழைக்கப்பட்டுள்ளதே அங்கெல்லாம் ∴பங்ஷனுடைய கட்டளைகள் புறோகிராமில் சேர்க்கப்பட்டு செயற்படத் தயாராகிவிடும்.

உதாரணம் (1)

```
inline double square(double n)
{
   return n*n;
}
```

உதாரணம் (2)

```
inline int min(int a, int b)
{
   return (a>b) ? b : a;
}
```

சி++ மொழியில் எழுதப்பட்ட சாதாரண ∴பங்ஷனைப் புறோகிராமில் அழைக்கும்போது, உடனே எங்கு ∴பங்ஷன் எழுதப்பட்டுள்ளதோ அங்கு ∴பங்ஷன் கட்டளைகள் செயற்பட்டு விடைகளை அழைக்கப்பட்ட இடத்துக்கு அனுப்பும். இவ்வாறு பல தடவைகள் ஒரு ∴பங்ஷனை அழைக்கும்போது மீண்டும், மீண்டும் ∴பங்ஷன் எழுதப்பட்ட இடத்துக்குச் சென்று கட்டளைகளைச் செயற்படுத்தும். ஆனால், இன்லைன் ∴பங்ஷன்களானது புறோகிராமினைக் கொம்பைல் செய்யும் போதே செயற்படத் தயாராகிவிடும். எனவேதான், சாதாரண ∴பங்ஷன்களைவிட மிக வேகமாக இன்லைன் ∴பங்ஷன்கள் செயற்படுகின்றன.

மக்றோ ஃபங்ஷன்கள் (Macro Functions)

சி++ மொழியில் வரையறுக்கப்படும் சிறிய ∴பங்ஷன்களை மக்றோ ∴பங்ஷனாக எழுதிப் புறோகிராமில் பயன்படுத்த முடியும்.

உதாரணம் (1):

```
#include <iostream.h>
#define add(a,b) (a+b)  // macro function add()
void main()
{
    cout<<add(50,75)<<endl;
}
```

உதாரணம் (2):

```
#include <iostream.h>
#define square(x, y) (x*y) //macro function square()
#define max(a, b) (a>b ? a: b) //macro function max()
void main()
{
    cout<<max(500,75)<<endl;
    cout<<max(100,175)<<endl;
    cout<<square(5,15)<<endl;
    cout<<max(25,15)<<endl;
    cout<<square(10.4,15.7)<<endl;
}
```

இவ்வாறு சிறிய ∴பங்ஷன்களை மக்றோ ∴பங்ஷனாக எழுதிப் புறோகிராமில் பயன்படுத்த முடியும்.

சி++ புறோகிராமில் பாஸ்வேட் (Password) இனைப் பெற்று, அந்த பாஸ்வேட் சரியாக இருந்தால் மட்டுமே தொடர்ந்து புறோகிராம் செயற்படுவதற்குரிய புறோகிராமினைப் பார்ப்போம்.

சி++ மொழியில் getch() என்ற ஃபங்ஷன் மூலம் ஒரு எழுத்தினை உள்ளீடு செய்ய முடியும். ஆனால், இந்த உள்ளீடு செய்யப்பட்ட எழுத்தினைத் திரையில் காட்டாது. putch() என்ற ஃபங்ஷன் மூலம் ஒரு குறித்த எழுத்தினை, உள்ளீடு செய்யப்பட்ட எழுத்திற்குப் பிரதியீடாகத் திரையில் காண்பிக்க முடியும்.

கீழேயுள்ள புறோகிராமினை அப்படியே ரேர்போ சி++ ரெக்ஸ்ற் எடிற்ரரில் எழுதவும்.

```
#include <iostream.h>
#include <conio.h> // for clrscr() function
#include <string.h> // for strcmp() function
void main()
{
    clrscr();
    char pass[20];
    int i=0;
    cout<<"Enter the password : ";
    pass[i]=getch();
    while (pass[i] !=13)
    {
        putch('*');
        i++;
        pass[i]=getch();
    }
    pass[i]='\0';
    if (strcmp(pass,"admin123") == 0)
        cout<<"\n Welcome! your password is correct";
    else
        cout<<"\n Sorry!!! Access denied. ";
    cin.get();
}
```

இந்தப் புறோகிராமினைச் செயற்படுத்தியவுடன், முதலில் பாஸ்வேட்டினை உள்ளீடு செய்யுமாறு கேள்வி கேட்கப்படும். இதற்கு நீங்கள் இருபது எழுத்துக்களுக்குக் குறைவான எழுத்துக்களை உள்ளீடு செய்த பின்னர், என்ரர் கீ (Enter Key) இனை அழுத்த வேண்டும். உள்ளீடு செய்யப்பட்ட பாஸ்வேட், admin123 ஆகக் காணப்பட்டால் தொடர்ந்து புறோகிராம் செயற்படும்.

∴பங்ஷன்களை வரையறுக்கும் போதே ஆரம்பப் பெறுமானங்களைக் கொடுக்க முடியும்.

உதாரணம் (1)

```
#include <iostream.h>
void main()
{
    void printLine(char = '*', int = 50);
    printLine();
    cout<<"University of Colombo \n";
    printLine('=');
    cout<<"Faculty of Science \n";
    printLine('-',30);
}
void printLine(char ch, int n);
{
    for (int i=1; i <= n; i++)
        cout<<ch;
    cout<<endl;
}
```

இந்தப் புறோகிராமினைச் செயற்படுத்தினால் வெளியீடாக,

```
*********************
University of Colombo
=================
Faculty of Science
------------------------
```

போன்றவற்றைத் திரையில் காண்பிக்கும்.

உதாரணம் (2)

```
#include <iostream.h>
int sum (int=20, int=5); // proto type of function
void main()
{
  // assumes second argument is default
  cout<<sum(30)<<endl;
  cout<<sum()<<endl; // assume both are default arguments
  cout<<sum(34, 59)<<endl; // ignore the default arguments
}
int sum (int x, int y)
{
  return x+y;
}
```

இந்தப் புறோகிராமினைச் செயற்படுத்தினால் வெளியீடாக 35, 25, 93 போன்றவற்றைத் திரையில் காண்பிக்கும்.

ஸ்ரரிக் மாறிகள் (Static Variables)

இந்த ஸ்ரரிக் மாறியானது லோக்கல் மாறியாக வரையறுத்தாலும் இவை புறோகிராம் முடியும் வரை நினைவகத்தில் இடம் விடுவிக்கப் படுவதில்லை.

உதாரணமாக,

```
#include <iostream.h>
double getavg (double d)
{
    static double total=0; // static variables are initialied
    static int count=0; // only once per program
    count++;
    total+=d;
    return (total/count);
}
void main()
{
    double x, avg;
    do
    {
        cout<<"Enter a number : ";
        cin>>x;
        avg=getavg(x);
        cout<<"New average is "<<avg<<endl;
    }
    while (x! = 0); // input 0 then exit  do ... while loop
}
```

இந்தப் புறோகிராமுக்கு உள்ளீடாக 20, 10, 45, 0 போன்றவற்றினைக் கொடுத்தால், வெளியீடுகள் கீழேயுள்ளவாறு காண்பிக்கப்படும்.

```
New average is 20
New average is 15
New average is 25
```

இந்தப் புறோகிராமில் total, count போன்றன ஸ்ரரிக் மாறியாக வரையறுத்துள்ளதாலேயே வெளியீடுகள் இவ்வாறான பெறுமானங் களைக் காண்பித்துள்ளது.

6. அறேக்கள், ஸ்ரக்சர்கள் மற்றும் பொயின்ரர்கள் (Arrays, Structures and Pointers)

6.1 அறேக்கள் (Arrays):

சி++ மொழியில் ஒரே விபர இனத்திலமைந்த பல மாறிகளை உருவாக்க அறே (Array) பேருதவியாக அமைகின்றது.

உதாரணமாக,

```
int x;
x = 5;
x = 10;
```

இந்த உதாரணத்தில், x என்பது முழு எண் விபர இனமாக வரையறுக்கப்பட்டு, இந்த மாறியில் பெறுமானம் 5 சேமித்து வைக்கப் பட்டுள்ளது. அடுத்துவரும் வரியின் மூலம், x என்ற மாறியின் பெறுமானம் 10 ஆக மாற்றப்பட்டுள்ளது. அதாவது, முதலிருந்த 5 என்ற மதிப்பு அழிக்கப்பட்டு, இறுதியாகக் கொடுத்த 10 என்ற பெறுமானம் x என்ற மாறியில் சேமித்து வைக்கப்பட்டுள்ளது. இவ்வாறான சந்தர்ப்பங்களில், அதாவது, பல பெறுமானங்களைக் குறித்தவொரு பெயரில் சேமிப்பதற்கு அறே என்ற கட்டளை பேருதவியாக அமைகின்றது.

ஒரே விபர இனம் கொண்ட பல மாறிகளை ஒரு குறித்தொரு பெயரில் வரையறுப்பதையே அறே (Array) என அழைக்கப்படுகின்றது. அறேயிலுள்ள மாறிகள் அனைத்தும் தொடர்ச்சியான நினைவக இடத்திலேயே காணப்படும்.

உதாரணமாக,

```
int marks[50];
```

இந்த marks என்ற அறேயில், 50 முழு எண் மதிப்புக்களைச் சேமிக்க முடியும். இந்த 50 முழு எண் மதிப்புக்களையும் marks[0], marks[1], marks[2], ..., marks[49] என 50 மாறிகளில் சேமிக்க முடியும்.

இந்த 50 மதிப்புக்களையும் உள்ளீடு, வெளியீடு செய்வதற்கு for லூப்பினைப் பயன்படுத்துவது சிறந்ததாகும்.

உதாரணமாக 50 மதிப்புக்களையும் உள்ளீடு செய்வதற்கு,

```
for (int i = 0; i<50; i++)
    cin>>marks[i];
```

எனக் கட்டளைகளை எழுத முடியும்.

உதாரணமாக 50 மதிப்புக்களையும் வெளியீடு செய்வதற்கு,

```
for (int i = 0; i<50; i++)
    cout<<marks[i];
```

எனக் கட்டளைகளை எழுத முடியும்.

உதாரணமாகப் பல்கலைக்கழகம் ஒன்றில் கற்கும் 100 மாணவர்களின் சுட்டிலக்கங்கள், மதிப்பெண்கள் போன்றவற்றினைச் சேமித்து வைக்க வேண்டுமாயின், எமக்கு 200 மாறிகள் தேவைப்படுகின்றன. இவ்வாறான சிக்கல்களைத் தீர்ப்பதற்காகவே அறே என்ற கட்டளை பயன்படுத்தப்படுகின்றது.

```
int marks[100];
```

இந்தக் கட்டளைன் மூலமாக முழு எண்களுக்குரிய 100 மாறிகள் உருவாக்கப்பட்டுள்ளன. இதில் முதலாவது மாணவனின் மதிப்பெண், marks[0] என்ற மாறியில் காணப்படும். அதேவேளை 100 ஆவது மாணவனின் மதிப்பெண் marks[99] என்ற மாறியில் காணப்படும்.

அறேக்களில் int, short, double, float, char போன்ற மதிப்புக்களையும் சேமித்து வைக்க முடியும்.

உதாரணமாக,

```
double salary[20];
char ch[10];
```

எழுத்துக்கோவை (String)

ஒரு எழுத்துக்கோவை (String) இனைச் சேமித்து வைக்க char என்ற அறேயினைப் பயன்படுத்த முடியும்.

உதாரணமாக,

```
char name[10];
```

இவ்வாறு ஓர் அறேயினை வரையறுத்தால், 9 எழுத்துக்கள் கொண்ட பெயரை name என்ற மாறியில் சேமிக்க முடியும். பத்தாவது இடத்தில் "\0" என்ற எழுத்தினைச் சேமித்து வைக்கப்படும். இவ்வாறு சேமிப்பது, எத்தனை எழுத்துக்கள் அந்த அறேயில் உண்டு என்பதைக் கண்டுபிடிப்பதற்கு உதவியாக அமைகின்றது.

char name[30] என வரையறுத்தால், இம்மாறியிலுள்ள எழுத்துக்களை name[0], name[1], ..., name[29] எனத் தனித் தனியாகவும் கையாள முடியும். மற்றும், char என்ற அறேயைப் பொறுத்தவரை name என்ற மாறியினை உள்ளீடு செய்வதற்கும், வெளியீடாகத் திரையில் காட்டுவதற்கும் நேரடியாகப் பயன்படுத்த முடியும். இவ்வகைச் செயற்பாடு char என்ற அறேயின் தனிச் சிறப்பாகும்.

உதாரணமாக,

```
cin >>name;
cout <<name;
```

இவ்வகைக் கட்டளைகளை char விபர இன அறேக்களைத் தவிர மற்றைய அறேக்களில் பயன்படுத்த முடியாது.

உதாரணமாக,

```
int n[20];
```

என வரையறுத்த பின்னர்,

```
cin>>n;
cout<<n;
```

போன்ற கட்டளைகளைப் பயன்படுத்த முடியாது. அதாவது int, float, double ஆகிய அறேக்களில் உள்ள உறுப்புக்களை தனித்தனி உறுப்புக்களாக மட்டுமே கையாள முடியும்.

அறே என்ற கட்டளைக்குரிய உதாரணப் புறோகிராமினை அடுத்துப் பார்ப்போம்.

உதாரணமாக, 10 மாணவர்களது பெயர்களையும், மதிப்பெண் களையும் உள்ளீடாகக் கொடுத்து, இவர்களில் 75 மதிப்பெண்களுக்குக் கூடிய மதிப்பெண்களைப் பெற்ற மாணவர்களை மாத்திரம் வெளியீடாகக் காட்டுவதற்குரிய புறோகிராமினைப் பார்ப்போம்.

```
#include <iostream.h>
void main()
{
    int marks[10];
    char  name[10] [50];
    for (int i=0; i<10; i++)
    {
        cout<<"Enter name : ";
        cin>>name[i];
        cout<<"Enter marks : ";
        cin>>marks[i];
    }
    for ( i = 0; i<10; i++)
        if  (marks[i]>75)  cout<<name[i]<<endl;
}
```

10 மாணவர்களது பெயர்களையும், மதிப்பெண்களையும் சேமித்து வைப்பதற்கு இரண்டு அறேக்கள் பயன்படுத்தப்பட்டுள்ளன. இதில் 10 மாணவர்களது பெயர்களைச் சேமிப்பதற்கு இரு பரிமாண அறே பயன்படுத்தப்பட்டுள்ளது. இரு பரிமாண அறேக்குரிய விளக்கத்தினைப் பின்னர் தெளிவாகப் பார்ப்போம். if என்ற தீர்வுசெய் கட்டளையானது, 75 மதிப்பெண்களுக்குக் கூடுதலாக பெற்ற மாணவர்களைத் திரையில் காட்டப் பயன்படுத்தப்பட்டுள்ளது.

இங்கு இரண்டு for லூப்கள் பயன்படுத்தப்பட்டுள்ளன. முதலாவது for லூப்பானது, 10 மாணவர்களினதும் பெயர்கள், மதிப்பெண்கள் போன்றவற்றைச் சேமிப்பதற்குப் பயன்படுத்தப்பட்டுள்ளது. மற்றைய

for லூப்பானது, 75 மதிப்பெண்களுக்குக் கூடுதலாகப் பெற்ற மாணவர் களின் விபரங்களை வெளியீடாகக் காட்டுவதற்குப் பயன்படுத்தப் பட்டுள்ளது.

அறே ஒன்றினை வரையறுக்கும் பொழுதே அதன் தொடக்க மதிப்புக்களைக் கொடுக்க முடியும்.

உதாரணமாக,

```
int m[4] = {40, 70, 90, 29}
char ch [5] = { 'a', 'e', 'i', 'o', 'u'}
```

இவ்வாறு, அறேக்களில் தொடக்க மதிப்பினைச் சேமிக்கும் போது அறேக்களிலுள்ள உறுப்புக்களின் எண்ணிக்கை அறிவிக்கப்பட்ட பெறுமானத்திலும் பார்க்கக் கூடுதலாகவோ அல்லது குறைவாகவோ காணப்பட்டால், எதிர்பாராத முடிவுகளைத் தரலாம். எனவே, இவ்வாறு அறேக்களின் தொடக்க மதிப்பினை அறிவிக்கும் போது உறுப்புக்களின் எண்ணிக்கையைக் குறிப்பிடாமல் விடுவது சிறந்ததாகும்.

உதாரணமாக,

```
int x[] = {10, 45, 490, 129}
char address[] = { 'J', 'a', 'f', 'f', 'n','a'}
float avg[]={11.23, 77.78, 34.56}
```

அறேயினை வரையறுக்கும் பொழுது தொடக்க மதிப்பினை கொடுக்கும் சந்தர்ப்பத்தில் மட்டும், உறுப்புக்களின் எண்ணிக்கையைக் குறிப்பிடத் தேவையில்லை. ஆனால், அறேயினை வரையறுக்கும் பொழுது தொடக்க மதிப்பினை கொடுக்காத சந்தர்ப்பத்தில், உறுப்புக் களின் எண்ணிக்கையைக் கட்டாயமாகக் குறிப்பிட வேண்டும்.

உதாரணமாக,

```
int x[];
float m[];
```

என அறேயினை வரையறுக்க முடியாது.

அறேயினைப் பயன்படுத்தும் பொழுது ஏற்படும் வழுக்கள்:

- அறேக்களில் வரையறுக்கப்பட்ட உறுப்புக்களின் எண்ணிக்கையை விடக் கூடிய எண்ணிக்கையான உறுப்புக்களைப் பயன்படுத்தும் போது வழு ஏற்படலாம்.

உதாரணமாக, int n[10]; என அறேயினை வரையறுத்த பின்னர், n [11] = 90; என எழுதினால், இந்தப் புறோகிராமினைக் கொம்பைல் செய்யும் போதே பிழையினைச் சுட்டிக் காட்டும். ஏனெனில், இந்த n என்ற அறேயில், 10 இலக்கங்களை மட்டுமே சேமிக்க முடியும். அதாவது, n[0], n[1], ..., n[9] ஆகிய உறுப்புக்களைப் பயன்படுத்த முடியும்.

◆ அறே ஒன்றினை வரையறுக்கும் பொழுது, அதிலுள்ள உறுப்புக் களின் எண்ணிக்கையைக் குறிப்பிடாமல் இருந்தாலும் வழு ஏற்படலாம்.

உதாரணமாக,

```
int m[];
char name[];
```
போன்றவற்றைக் குறிப்பிடலாம்.

◆ அறே ஒன்றினை வரையறுக்கும் பொழுது ஆரம்பப் பெறுமானம் கொடுக்கப்படும் சந்தர்ப்பங்களில், கூடுதலாக அல்லது குறைவாக உறுப்புக்களின் எண்ணிக்கை கொடுக்கும் போதும் வழு ஏற்படலாம்.

உதாரணமாக,

```
int n[3] = {40, 70, 90, 70, 30};
int x[4] = {14, 11};
```

அறேக்களில், நாம் இதுவரை ஒரு பரிமாண அறேக்களை மட்டுமே பார்த்தோம். இனி, இரு பரிமாண அறே (Two dimentional array) இனைப் பார்ப்போம்.

அறேக்களில் ஒரு பரிமாண அறே (One dimentional array), இரு பரிமாண அறே (Two dimentional array), முப்பரிமாண அறே (Three dimentional array), ... எனப் பல வகையான அறேக்கள் உண்டு.

சி++ மொழியில், எவ்வாறு இரு பரிமாண அறேயினை வரையறுப்பது என்பதனை அடுத்துப் பார்ப்போம்.

```
int n[3][4];
```

முழு எண் இனத்திற்குரிய n என்ற அறேயில், 3 வரிசைகள் (Rows), 4 நிரல்கள் (Columns) இற்கு அமைய உறுப்புக்களைச் சேமிக்க முடியும். அதாவது, n[0][0], n[0][1], n[0][2], ..., n[2][3] என அறேயில் 12 உறுப்புகளைச் சேமிக்க முடியும்.

```
for (int i = 0; i<3; i++)
    for (int j = 0; j<4; j++)
        cin >>n[i][j]
```

போன்ற கட்டளைகள் மூலம், இரு பரிமாண அறேக்களிலுள்ள உறுப்புக்களுக்கு மதிப்பினை உள்ளீடு செய்ய முடியும்.

எழுத்துக்கோவைகளின் அறே
(Array of String)

எழுத்துக்கோவைகளின் அறே என்பது, char விபர இனத்தின் இரு பரிமாண அறே ஆகும்.

உதாரணமாக,

```
char array_name[number of string][number of characters];
char name[50][20];
```

இந்த இரு பரிமாண அறேயில், 50 பெயர்களைச் சேமிக்க முடியும். இங்கு ஒவ்வொரு பெயரும், 19 எழுத்துக்கள் அல்லது அதிலும் குறைவான எழுத்துக்கள் கொண்ட பெயரைச் சேமிக்க முடியும்.

உதாரணமாக, 10 மாணவர்களின் பெயர்களை உள்ளீடு செய்து, அந்த 10 மாணவர்களின் பெயர்களை வெயியீடாகக் கணினித் திரையில் காண்பிப்பதற்குரிய புறோகிராமினைப் பார்ப்போம்.

```
#include <iostream.h>
void main()
{
    char  name[10][30];
    for (int i = 0; i<10; i++)
    {
        cout<<"Enter name : ";
        cin>>name[i];
    }
    for (i=0; i<10; i++)
        cout<<name[i]<<endl;
}
```

இரு பரிமாண அறே ஒன்றினை வரையறுக்கும் பொழுதே அதன் தொடக்க மதிப்புக்களைக் கொடுக்க முடியும்.

உதாரணமாக,

```
int a[3][2] = {40, 70,
               34, 56,
               64, 76
              };
```

அல்லது

```
int a[3][2] = {40, 70, 34, 56, 64, 76 };
```

அல்லது

```
int a[][2] = {40, 70, 34, 56, 64, 76 };
```

இவ்வாறு இரு பரிமாண அறேக்களில் தொடக்க மதிப்பினைச் சேமிக்க முடியும். இந்த இரு பரிமாண அறேயினை மெற்றிக்ஸ் (Matrix) எனவும் அழைக்கப்படுகின்றது.

உதாரணம் (1)

இரு பரிமாண அறேயினைப் பயன்படுத்தி, இரண்டு மெற்றிக்ஸ் (Matrices) இனைக் கூட்டுவதற்கு, கழிப்பதற்கு, பெருக்குவதற்குரிய சி++ மொழிப் புறோகிராமினை அடுத்துப் பார்ப்போம்.

```
#include <iostream.h>
void main()
{
    int a[5][5], b[5][5], c[5][5];
    int i,j,c1,c2,r1,r2;
    cout<<"Enter row and column size of martrix A: ";
    cin>>r1>>c1;
    cout<<"Enter row and column size of martrix A: ";
    cin>>r2>>c2;
    if ((c1==c2) && (r1==r2)) // Check if matrices can be added
    {
        // read matrix A
        cout<<"Matrix A is "<<r1<<"x"<<c1<<" matrix \n";
        cout<<"Enter the elements of the matrix A \n";
        for (i=0; i<r1; i++)
            for (j=0; j<c1; j++)
                cin>>a[i][j];
        // read matrix B
        cout<<"Matrix B is "<<r2<<"x"<<c2<<" matrix \n";
        cout<<"Enter the elements of the matrix B \n";
        for (i=0; i<r2; i++)
            for (j=0; j<c2; j++)
                cin>>b[i][j];
        // Addtion of two matrices
        // Matrix C = Matrix A + Matrix B
        for (i=0; i<r1; i++)
            for (j=0; j<c1; j++)
                c[i][j]= a[i][j] + b[i][j];
        //Printing the matrix C
        cout<<"Summation matrix C "<<endl;
        for (i=0; i<r1; i++)
        {
            for (j=0; j<c1; j++)
                cout<<c[i][j]<<" ";
            cout<<endl;
        }
        // Subtraction of two matrices
        // Matrix C = Matrix A - Matrix B
```

```
        for (i=0; i<r1; i++)
            for (j=0; j<c1; j++)
                c[i][j]= a[i][j] - b[i][j];
        //Printing subtraction matrix C
        cout<<"Subtraction matrix C"<<endl;
        for (i=0; i<r1; i++)
        {
            for (j=0; j<c1; j++)
                cout<<c[i][j]<<" ";
            cout<<endl;
        }
        // Multiplication of two matrices
        // Matrix C = Matrix A * Matrix B
        for (i=0; i<r1; i++)
            for (j=0; j<c1; j++)
            {
                c[i][j]=0;
                for (int k=0;k<c1;k++)
                    c[i][j] = c[i][j] + a[i][k]*b[k][j];
            }
        //Printing multiplication matrix C
        cout<<"Multiplication matrix C"<<endl;
        for (i=0; i<r1; i++)
        {
            for (j=0; j<c1; j++)
                cout<<c[i][j]<<" ";
            cout<<endl;
        }

    }
    else
        cout<<"These two matrix cannot be added, \
                substracted and multiplicated"<<endl;
}
```

உதாரணம் (2)

ஒரு வகுப்பறையிலுள்ள 100 மாணவர்களின் பெயர்களையும், மற்றும் கணினி பாடத்திற்குரிய மதிப்பெண்களையும் உள்ளீடாகக் கொடுக்கும் போது, அவர்கள் பெற்ற மதிப்பெண்களுக்கு ஏற்ப, அவர்களுக்குரிய வகுப்புநிலை (Rank) இனைப் பெயருடன் சேர்த்து

வெளியீடாகக் கணினித்திரையில் காட்ட வேண்டும்.

இங்கு 100 மாணவர்களின் பெயர், மற்றும் மதிப்பெண் போன்ற வற்றைச் சேமிப்பதற்கு 100 உறுப்புக்களைக் கொண்ட இரு அறேக்களை வரையறுக்க வேண்டும். இதில், பெயர்களைச் சேமிப்பதற்கு இரு பரிமாண அறே பயன்படுத்தப்பட்டுள்ளது. பின்னர், மதிப்பெண்களின் பெறுமானத்திற்கு ஏற்ப இறங்கு வரிசைப்படுத்த வேண்டும். மதிப்பெண்களின் இறங்கு வரிசைக்கு ஏற்ப மாணவனின் வகுப்பு நிலையைக் கொடுக்க வேண்டும்.

```
#include <iostream.h>
void main()
{
    int marks[100];
    char  name[100] [50];
    for (int i = 0; i<100; i++)
    {
        cout<<"Enter name : ";
        cin>>name[i];
        cout<<"Enter Computer Science marks : ";
        cin>>marks[i];
    }
    cout<<" Name          Rank "<<endl;
    cout<<" *****         *****"<<endl;
    for ( i = 0; i<99; i++)
        for ( j = i+1; j<100; j++)
            if  (marks[j] > marks[j-1])
            {
                int  temp = name[j];
                name[j] = name[j-1];
                name[j-1] = temp;
            }
     for ( i = 0; i<100; i++)
        cout<<name[i]<<" - "<<i+1<<endl;
}
```

இந்தப் புறோகிராமில், முதலிலுள்ள for லூப்பானது 100 மாணவர் களின் பெயர்களையும், மதிப்பெண்களையும் உள்ளீடாகக் கொடுப்பதற்கு பயன்படுத்தப்பட்டுள்ளது. அடுத்ததாக இரு for லூப்கள் பயன்படுத்தப்பட்டதன் நோக்கம் யாதெனில், மதிப்பெண்களின் பெறுமானத்திற்கு ஏற்ப இறங்கு வரிசைப்படுத்துவதற்காகும். இந்த

இரு for லூப்பிற்கும் அடுத்தாற் போலுள்ள if என்ற தீர்வுசெய் கட்டளையானது, முதல் இரு மதிப்பெண்களையும் ஒப்பிட்டு, அதில் முதலிலுள்ள மதிப்பெண், இரண்டாவதாக உள்ள மதிப்பெண்ணை விடச் சிறியது எனின், அவ்விரு மதிப்பெண்களுக்குரிய பெயரினை ஒத்துமாற்றம் (Swap) செய்யும். அடுத்து, இரண்டாவது மதிப்பெண்ணை யும் மூன்றாவது மதிப்பெண்ணையும் ஒப்பிட்டு, முதலில் கூறியபடி ஒத்துமாற்றம் செய்யும். இவ்வாறு 99 தடவைகள் செய்வதன் மூலம், 100 ஆவது இடத்தில், மிகக் குறைந்த மதிப்பெண்ணைப் பெற்ற மாணவனின் பெயர் மாற்றப்பட்டுவிடும். இவ்வாறு 99 தடவைகள் செய்வதன் மூலம் மதிப்பெண்களுக்கேற்ப மாணவர்களின் பெயர்கள் யாவும் இறங்கு வரிசைப்படுத்தப்பட்டுவிடும். இறுதியாகவுள்ள for லூப்பானது, மாணவர்களின் பெயர், வகுப்புநிலை போன்றவற்றைக் கணினித்திரையில் வெளியீடாகக் காட்டுவதற்குப் பயன்படுத்தப் பட்டுள்ளது.

6.2 ஸ்ரக்சர் (Structure)

அடுத்ததாக, சி++ மொழியில் பயன்படுத்தப்படும் ஸ்ரக்சர் (Structure) என்ற கட்டமைப்பினைப் பார்ப்போம்.

பல வகையான விபர இனங்களை ஒரு குறித்த கட்டமைப்பில் வரையறுக்கப்படுவதையே சி++ மொழியில் ஸ்ரக்சர் என அழைக்கப் படுகின்றது.

ஒரே வகையான விபர இனங்களை ஒரு குறித்த பெயரில் வரையறுப்பது அரே (Array) எனவும், பல வகையான விபர இனங் களை ஒரு குறித்த பெயரில் வரையறுப்பது ஸ்ரக்சர் (Structure) எனவும் அழைக்கப்படுகின்றது. அதாவது, ஒத்த விபரங்களைத் தனித் தனி மாறிகளில் (Variables) கையாளுவதற்குப் பதிலாக ஒரு குழுவாக வரையறை செய்வதே ஸ்ரக்சர் ஆகும்.

சி++ மொழியில் ஏற்கனவே காணப்படும் int, short, long, float, double, char போன்ற விபர இனங்களைப் போல், ஸ்ரக்சர் (Structure) என்பது நாம் வரையறுத்துப் பயன்படுத்தப்படும் புதிய விபர இனமாகும்.

உதாரணமாக, ஒரு மாணவனுக்குரிய பெயர், சுட்டிலக்கம், வயது, மதிப்பெண்கள் போன்ற 4 மாறிகளையும், ஒரே அமைப்பில் கொண்டு வருவதற்கு ஸ்ரக்சர் என்ற கட்டளை அமைப்பு பயன்படுத்தப்படுகின்றது. இவற்றினை எவ்வாறு ஸ்ரக்சராக வரையறுக்க முடியும் என்பதனைப் பார்ப்போம்.

```
struct Student{
    char name[30];
```

```
        char indexNo[10];
        int age;
        int marks;
    };
```

மேலேயுள்ள Student என்ற விபர இனத்திற்குரிய மாறியினை Student s என உருவாக்க முடியும்.

உதாரணமாக, Student s என வரையறுத்த பின்னர், s.name s.indexNo, s.age, s.marks என s என்ற மாணவனுக்குரிய பெயர், சுட்டிலக்கம், வயது, மதிப்பெண்கள் ஆகியவற்றுக்குரிய பெறுமானங் களைத் தனித் தனியாகக் கையாள முடியும்.

உதாரணமாக, அந்த மாணவனுக்குரிய பெயர், வயது போன்றன வற்றை உள்ளீடு செய்ய வேண்டுமாயின்,

cin >>s.name;

cin >>s.age; போன்ற கட்டளைகளை எழுத வேண்டும்.

அந்த மாணவனுக்குரிய பெயர், வயது போன்றவற்றை வெளியீடாகத் திரையில் காட்டுவதற்கு,

cout <<s.name;

cout <<s.age ; போன்ற கட்டளைகளை எழுத வேண்டும்.

int, long, double, char போன்ற விபர இனங்களுக்குரிய அறேக்களை உருவாக்குவது போல், ஸ்ரக்சர் விபர இனத்திற்குரிய அறேயினையும் உருவாக்க முடியும்.

உதாரணமாக,

```
Student s1[20];
Student s2[30];
```

ஸ்ரக்சரிற்கு எவ்வாறு ஆரம்ப மதிப்புக்கள் கொடுக்கப்படுகின்றது என அடுத்துப் பார்ப்போம்.

Student s = {"Siva", "96/s/6380", 25, 87};

இவ்வாறு மாணவனுக்குரிய பெயர், சுட்டிலக்கம், வயது, மதிப்பெண் ஆகியவற்றுக்குரிய ஆரம்பப் பெறுமானங்களைக் கொடுக்க முடியும்.

ஒரு ஸ்ரக்சர் மாறியிலுள்ள பெறுமானங்களை இன்னுமொரு ஸ்ரக்சர் மாறிக்குக் கொடுக்க முடியும்.

உதாரணமாக,

```
Student s1 = {"Kumar", " 96/s/6385", 35, 94};
Student s2 = s1;
```

ஸ்ரக்சர் விபர இனத்திற்குரிய அறேயில், எவ்வாறு ஆரம்ப மதிப்புக் கள் கொடுக்கப்படுகின்றது எனப் பார்ப்போம்.

உதாரணமாக,

```
Student s[3] = {
                "Kumar", " 96/s/6385", 23, 94,
                "Raja", "96/s/6386",23, 67,
                "Selva", "96/s/6388",23, 100,
                };
```

ஸ்ரக்சர் என்ற கட்டமைப்புக்குள், இன்னுமொரு ஸ்ரக்சர் எவ்வாறு வரையக்கப்படுகிறது எனப் பார்ப்போம்.

```
struct Date{
    int day, month, year;
};
struct Employee{
    char empNo[10];
    char name[30];
    Date dateOfBirth;
    double salary;
};
```

இரண்டாவதாக அறிவிக்கப்பட்ட Employee என்ற ஸ்ரக்சரிற்குள் முதலாவதாக அறிவிக்கப்பட்ட Date என்ற ஸ்ரக்சர் பயன்படுத்தப் பட்டுள்ளது.

```
Employee  e;
e = {"Siva", "M131", {26,5,1967},23000};
```

என Employee என்ற ஸ்ரக்சரிலிருந்து உருவாக்கப்பட்ட e என்ற மாறிக்குப் பெறுமானங்களைக் கொடுக்க முடியும்.

உதாரணமாக 10 மாணவர்களின் பெயர், வயது, சுட்டிலக்கம், மதிப்பெண்கள் போன்றவற்றை உள்ளீடாகக் கொடுத்து, அவர்களில் 75 மதிப்பெண்களுக்குக் கூடிய மதிப்பெண்களைப் பெற்ற மாணவர் களை மட்டும் வெளியீடாகத் திரையில் காட்டுவதற்கான புறோகிராமினை அடுத்துப் பார்ப்போம்.

```
#include <iostream.h>
void main()
{
    struct Student{
        char name[30];
        char indexNo[10];
        int age;
        int marks;
    };
```

```
        Student  s[10];
        for (int i = 0; i<10; i++)
        {
            cout<<"Enter the student's name : ";
            cin>>s[i].name;
            cout<<"Enter the index number : ";
            cin>>s[i].indexNo;
            cout<<"Enter the student's marks : ";
            cin>>s[i].marks;
        }
        for (i=0;i<10; i++)
            if (s[i].marks>75)
                cout<<s[i].name<<" "<<s[i].marks<<endl;
    }
```

மாணவனின் பெயர், சுட்டிலக்கம், மதிப்பெண் ஆகியவற்றைத் தனித் தனியாக மூன்று மாறிகளில் கையாள முடியும். எனினும், இங்கு 3 மாறிகளுக்கும் இடையே ஒரு தொடர்புள்ளபடியால், இவற்றை ஸ்ரக்சர் என்ற குழுவாக வரையறுத்துள்ளோம். பின்னர், இந்த மாணவனைப் போல் 10 மாணவர்களுக்குரிய விபரத்தைப் பெற வேண்டும் என்பதற்காக 10 மாணவர்களுக்குரிய ஒரு அறேயானது வரையறுக்கப் பட்டுள்ளது. அடுத்ததாகவுள்ள கட்டளைகள் அனைத்தும் ஏற்கனவே விளக்கமாக ஆராயப்பட்டவையாகும்.

மேலேயுள்ள உதாரணத்தில், ஸ்ரக்சர் என்னும் விபர இனத்திற்குள் மாறிகளை மட்டுமே வரையறுத்துள்ளோம். எனினும், மாறிகளுடன் ஃபங்ஷன்களையும் சேர்த்து ஸ்ரக்சர் என்ற வரையறைக்குள் எழுதிக் கையாள முடியும்.

உதாரணமாக,

```
#include <iostream.h>
void main()
{
    struct Employee{
        char empNo[10];
        char name[30];
        int age;
        double salary;
        void getData()
```

```
        {
            cout<<"Enter the employee number: "
            cin>>empNo;
            cout<<"Enter the name : "
            cin>>name;
            cout<<"Enter the age : "
            cin>>age;
            cout<<"Enter the salary : "
            cin>>salary;
        }
        void displayData()
        {
            cout<<" Employee Details "<<endl;
            cout<<" *************** "<<endl;
            cout<<"Employee number : "<<empNo<<endl;
            cout<<"Name : "<<name<<endl;
            cout<<"Age : "<<age<<endl;
            cout<<"Salary : "<<salary<<endl;
        }
    };
    Employee e;
    e.getData();
    e.displayData();
}
```

மேலேயுள்ள உதாரணத்தில், ஸ்ரக்சரிற்குரிய ஃபங்ஷன்கள் ஸ்ரக்சரிற்குள்ளேயே எழுதப்பட்டுள்ளது. எனினும், இந்த ஸ்ரக்சரிற்குரிய ஃபங்ஷன்களை ஸ்ரக்சருக்கு வெளியிலும் எழுத முடியும்.

ஸ்ரக்சருக்கு வெளியே ஃபங்ஷன்களை எழுத வேண்டுமாயின், ஃபங்ஷனின் முன்வடிவம் (Proto type) ஸ்ரக்சரிற்குள் அறிவிக்கப்பட வேண்டும்.

உதாரணமாக,

```
    struct Employee
    {
        -----------;
        -----------;
        void getData();      // Function proto type
        void displayData();     // Function proto type
    };
```

```
        void Employee::getData()
        {
            ----------;
            ----------;
        }
        void Employee::displayData()
        {
            ----------;
            ----------;
        }
    void main()
    {
        -----------
        -----------
    }
```

இவ்வாறு ∴பங்ஷனின் முன்வடிவத்தை ஸ்ரக்சரிற்குள் அறிவித்த பின்னர், ஸ்ரக்சரிற்கு வெளியே அந்த ∴பங்ஷன்களை வரையறுக்க முடியும்.

ஒரு மிகப்பெரும் செயல்திட்டத்துக்காக எழுதப்படும் புறோகிராம் களுக்கு ஒன்றுக்கு மேற்பட்ட புறோகிராமர்கள் தேவைப்படலாம். எனவே, ஒவ்வொரு புறோகிராமரும் தமக்கு ஏற்றவாறு டேற்றாவை மாற்ற முற்படின், இச்செயல்திட்டத்தில் பல பிழைகள் ஏற்பட வாய்ப்புண்டு. எனவே, இப்பிரச்சினைகளைத் தீர்வு செய்வதற்கு பிரைவேற் (private) அல்லது புரொரெக்ரட் (protected) என்ற குணங்களைச் சேர்ப்பதன் மூலம் டேற்றாவைப் பாதுகாக்க முடியும். இந்தக் குணங்களை ஸ்ரக்சரிற்குள் வரையறுக்கப்படும் ∴பங்ஷன்கள், பண்புகள் போன்ற வற்றுக்குப் பயன்படுத்த முடியும். இவற்றைப் பற்றிய விளக்கங்களைப் பின்னர் தெளிவாக கிளாஸ் என்ற கட்டமைப்பில் பார்ப்போம்.

6.3 பொயின்ரர் (Pointer)

சி$^{++}$ மொழிப் புறோகிராம்களில், பொயின்ரர்கள் பல சந்தர்ப்பங்களில் பேருதவி புரிகின்றன. எனினும், சில கவனயீனப் பிழைகளினால் பல பின்விளைவுகள் இப்பொயின்ரர்களினால் ஏற்படுகின்றன.

புறோகிராமில் உருவாக்கப்படும் மாறிகள், கணினி நினைவகத்தில் இரு நிலைகளில் தமது நினைவக ஒதுக்கீடுகளைச் செய்கின்றன.

1. புறோகிராம் கொம்பைல் செய்யப்படுகின்ற நேரத்தில் (Compile Time) புறோகிராமில் பயன்படுத்தப்படும் மாறிகளுக்குத் தேவையான நினைவகம் ஒதுக்கீடு செய்யப்படுகின்றது.

உதாரணமாக,

```
int x, y;
x = 10;
y = 490;
double z = 4.9;
```

2. கொம்பைல் (Compile) செய்யப்பட்ட புறோகிராம் செயற்பட்டுக் கொண்டிருக்கும் நேரத்தில் (Run Time), மாறிகளுக்குரிய நினைவகம் ஒதுக்கீடு செய்யப்படுகின்றது. இதற்கு சி++ மொழியில் பொயின்ரர்கள் பயன்படுகின்றன. இது டைனமிக் மெமரி ஒதுக்கீடு (Dynamic Memory Allocation) எனவும் அழைக்கப்படுகின்றது.

உதாரணமாக,

```
int *x;
char *name;
```

முதலில் கூறப்பட்ட முறைப்படி ஒதுக்கப்படுகின்ற நினைவக இடமானது, புறோகிராம் செயற்பட்டு முடிகின்ற வரையில் விடுவிக்கப்படுவதில்லை. அதாவது, இந்தப் புறோகிராம் செயற்பட்டு முடிந்த பிறகே, இங்கு பயன்படுத்தப்பட்ட நினைவக இடத்தை வேறு பணிகளுக்குப் பயன்படுத்த முடியும். ஆனால், இரண்டாவதாகக் கூறப்பட்ட முறையில் ஒதுக்கப்பட்ட நினைவகத்தை, இந்தப் புறோகிராமிற்குள்ளேயே விடுவிக்க முடியும். எனவே இந்த விடுவிக்கப்பட்ட நினைவகத்தை இதே புறோகிராமில், வேறொரு தேவைக்குப் பயன்படுத்த முடியும். இதனால், நினைவகத்தைச் சிக்கனமாக புறோகிராமில் பயன்படுத்தலாம்.

உதாரணமாக, எமக்குப் பட்டியல் (List) ஒன்றை உருவாக்க வேண்டியுள்ளது. ஆனால், எத்தனை உறுப்புக்கள் இந்தப் பட்டியலில் உள்ளது என்பது எமக்குத் தெரியாது. இச்சந்தர்ப்பத்தில் இரண்டாவதாகக் கூறப்பட்ட முறை விரும்பத்தக்கது.

பொயின்ரர் (Pointer) என்றால் என்ன?

கணினி நினைவகத்தில் (RAM - Random Access Memory) உருவாக்கப்பட்டுள்ள மாறியின் முகவரி கையாளப்படும் ஒரு இடத்தை பொயின்ரர் மூலம் செயற்படுத்த முடியும்.

மாறியின் முகவரியை ஒரு பொயின்ரரிடம் கொடுத்துவிட்டு, அந்தப் பொயின்ரரின் மூலம் அந்த மாறியில் மாற்றங்களைச் செய்ய முடியும். அதாவது, மாறிகளை மறைமுகமாக கையாள உதவுவதே பொயின்ரர் ஆகும்.

எவ்வாறு பொயின்ரரினை வரையறுப்பது?

பொயின்ரரினை "*" என்ற ஒப்பறேற்ரரினைப் பயன்படுத்தி வரையறுக்க வேண்டும்.

உதாரணமாக i என்ற முழு எண் பொயின்ரரினை, int *i என வரையறுக்க வேண்டும்.

உதாரணமாக,

```
int *p;
```

இங்கு "*" என்பது, பொயின்ரர் ஒப்பறேற்றர் (Pointer Operator) ஆகும். p என்பது, ஓர் முழு எண் மதிப்பினைக் கையாளும் நினைவகத்தின் இருப்பிடத்தினைக் குறிக்கும். p இனைப் பொயின்ரர் மாறி (Pointer Variable) எனவும் அழைக்க முடியும். இந்த p இல் முழு எண் மதிப்பின் முகவரியினைச் சேமிக்க முடியும்.

உதாரணமாக,

```
int x = 10;  // integer variable x
int *p;  // integer pointer i
```

p = &x என வரையறுக்க முடியும். ஏனெனில், &x என்பது x என்ற மாறியின் முகவரி இடத்தைக் குறிக்கும். p என்ற முழு எண் பொயின்ரரில் (அதாவது முகவரியில்) x என்ற மாறிக்குரிய முகவரி யினைக் கொடுப்பதற்கு p = &x என எழுத முடியும்.

p = x என எழுதுவது தவறாகும். ஏனெனில், x என்பது மாறிக் குரிய பெறுமானத்தினைக் கையாள உதவும் இடமாகும். ஆனால், p என்பது கணினி நினைவக இடத்தினைக் கையாள உதவும் இடமாகும்.

```
int x = 75;
int *y = &x;  // address of  x is assigned to y
```

இந்த உதாரணத்தின் முதல் கட்டளையின் மூலம், x என்ற முழு எண் விபர இன மாறியில், 75 என்ற மதிப்பு சேமிக்கப்பட்டுள்ளது. அடுத்த கட்டளையின் மூலம், y என்ற முழு எண் பொயின்ரரில், x என்ற மாறியின் முகவரி சேமிக்கப்பட்டுள்ளது. இங்கு "&" என்பது, நினைவகத்தில் உள்ள முகவரியினைச் சுட்டிக் காட்ட பயன்படும் ஒப்பரேற்றராகும்.

சி++ மொழியில், டைனமிக் (Dynamic) நினைவக இடங்களை ஒதுக்கீடு செய்ய new என்ற ஒப்பறேற்றர் (Operator) பயன்படுத்தப் படுகின்றது.

உதாரணமாக,

```
int  x;
int *y = new  int;
```

மேலே கூறப்பட்ட இரு கட்டளைகளும், நினைவகத்தில் இரண்டு பைற்களைக் (2 Bytes) கையாளும். ஆனால், முதலாவதாகக் கூறப்பட்ட கட்டளையானது, கொம்பைல் செய்கின்ற போதே

நினைவகத்திற்குரிய இடத்தைத் தீர்மானித்துவிடும். இரண்டாவது கட்டளையானது, புறோகிராம் செயற்படுத்தப்படும் போதுதான் நினைவகத்திற்குரிய இடத்தைத் தீர்மானிக்கும். இரண்டாவதாக எழுதப்பட்ட கட்டளையின் மூலமாக உருவாக்கப்பட்ட மாறிகளை, புறோகிராம் செயற்பட்டுக் கொண்டிருக்கும் போதே நினைவக இடத்தை விட்டு விடுவிக்க முடியும்.

நினைவக இடங்களை விடுவிப்பதற்கு delete என்ற கட்டளையினைப் பயன்படுத்த வேண்டும்.

உதாரணமாக,

```
int *x = new int;
```

delete x; என்ற கட்டளை மூலம் x என்ற நினைவக இடத்தை விடுவிக்க முடியும்.

உதாரணமாக,

```
int y;
```

delete y; என எழுதுவதன் மூலம், y என்ற நினைவக இடத்தை விடுவிக்க முடியாது. ஏனெனில், இந்த y என்ற மாறியானது புறோகிராம் செயற்பட்டு முடிவடையும் வரை விடுவிக்க முடியாது.

new என்ற ஒப்பறேற்றர் மூலம் ஒதுக்கப்பட்ட நினைவக இடத்தை மட்டுமே delete என்ற கட்டளை மூலம் விடுவிக்க முடியும்.

பொயின்ரராக வரையறுக்கப்பட்ட மாறியில், எவ்வாறு பெறுமானங்களைச் சேமிக்கப்படுகின்றது என்பதற்குரிய உதாரணத்தினை அடுத்துப் பார்ப்போம்.

```
int *x = new int;
*x = 100
```

போன்ற இரு கட்டளைக்குப் பதிலாக, int *x = new int(100) என்ற கட்டளை மூலமும் வரையறுக்க முடியும்.

முழு எண் பொயின்ரர் போன்று தசம எண் பொயின்ரர், எழுத்துப் பொயின்ரர் போன்றவற்றையும் வரையறுக்க முடியும்.

உதாரணமாக,

```
char *a = new char ('B');
float *b = new float;
double *c = new double;
```

10 முழு எண் பொயின்ரர்கள் தேவையெனின், ஒரு முழு எண் அறேயை உருவாக்கிப் பயன்படுத்த முடியும்.

```
int *x = new int[10];
```

பின்னர்,

```
for (int i = 0; i<10; i++)
     *x++ = 0;
```

போன்ற கட்டளை மூலம் ஒவ்வொரு முழு எண் பொயின்ரருக்கும், 0 என்ற பெறுமானத்தினைக் கொடுக்க முடியும்.

```
char *name;
char name[30]
```

மேலே கூறப்பட்ட இரு கட்டளைகளும் name என்ற மாறியில், பெயரினைச் சேமிக்க உதவுகின்றன. இதில், முதலாவது கட்டளையின் மூலம், தேவையான அளவு நீளமான பெயரைச் சேமிக்க முடியும். ஆனால், இரண்டாவது கட்டளை மூலம் 29 எழுத்துக்களோ அல்லது அதிலும் குறைவான எழுத்துக்கள் கொண்ட பெயரை மட்டுமே name என்ற மாறியில் சேமிக்க முடியும்.

உதாரணமாக,

```
char *name;
cout<<"Enter name ";
cin>>name;
```

இரண்டு பெறுமானங்களைத் தமக்குள் பரிமாற்றம் (swap) செய்து கொள்வதற்கு ஒரு ∴பங்ஷன் எழுத வேண்டுமாயின், பொயின்ரரின் துணையுடன் மிக இலகுவாக எழுத முடியும்.

```
void swap(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}
```

இந்த swap() என்ற ∴பங்ஷனில், return என்ற கட்டளை பயன் படுத்தப்படவில்லை. ஏனெனில், இந்த ∴பங்ஷன் எந்தவொரு மதிப்பினையும் return எனற கட்டளையைப் பயன்படுத்தித் திருப்பி அனுப்பவில்லை. எனினும், இரண்டு பெறுமானங்களையும் தமக்குள் பரிமாற்றம் செய்து, அவற்றைத் திருப்பியனுப்பும் செயற்பாட்டிற்கு பொயின்ரர் உதவி புரிகின்றது.

டைனமிக் அறே (Dynamic Array)

புறோகிராம் செயற்பட்டுக் கொண்டிருக்கும் போது அறேயில் எத்தனை உறுப்புக்கள் உள்ளன என்பது தீர்மானிக்கப்படுவதையே டைனமிக் அறே என அழைக்கப்படுகின்றது.

சாதாரண மாறியினைப் பயன்படுத்தி, டைனமிக் அறேயினை உருவாக்க முடியாது.

உதாரணமாக,

```
int x[n];
cin >>n;
for (int i = 0; i<n ; i++)
    x[i] = 0;
```

என அறேயினை வரையறுக்க முடியாது. அதாவது, அறேயில் n என்பது மாறிலி (Constant) ஆக இருக்க வேண்டும். எனவே, நாம் பொயின்ரரினைப் பயன்படுத்தி, இந்த டைனமிக் அறேயினை உருவாக்க முடியும்.

உதாரணமாக,

```
int *x = new int[n]
cin >> n;
for (int i = 0; i<n; i++)
    x[i] = 0;
```

என்ற கட்டளைகள் மூலம் டைனமிக் அறேயினை உருவாக்க முடியும்.

இந்த உதாரணத்திற்கு உள்ளீடாக 30 என்ற பெறுமானத்தினை உள்ளீடு செய்தால், 30 உறுப்புக்கள் கொண்ட ஒரு டைனமிக் அறே உருவாக்கப்பட்டு, அந்த 30 உறுப்புக்களுக்கும் 0 என்ற ஆரம்ப மதிப்பு கொடுக்கப்படும்.

புறோகிராம் செயற்பட்டுக் கொண்டிருக்கும் போது, உள்ளீடு செய்யும் எண்ணிக்கைக்கு அமைய அறேயிலுள்ள மூலகங்களின் எண்ணிக்கை அமையும்.

பொயின்ரர் ஃபங்ஷன்கள் (Pointer Functions)

சி++ மொழியில் எழுதப்படும் ∴பங்ஷன் முழு எண்ணைத் திருப்பி அனுப்புவது போன்று, பொயின்ரரினைத் திருப்பி அனுப்ப முடியும். இதையே பொயின்ரர் ∴பங்ஷன் என அழைக்கப்படுகின்றது. திருப்பி அனுப்பும் பொயின்ரரானது, முழு எண் பொயின்ரர் அல்லது தசம தான பொயின்ரர் அல்லது எழுத்து பொயின்ரராக இருக்கலாம். இந்தப் பொயின்ரர் ∴பங்ஷனானது, அறேயினைத் திருப்பி அனுப்பும் ∴பங்ஷனை ஒத்ததாகும். உதாரணமாக, முழு எண் பொயின்ரர் என்பது, முழு எண் பொயின்ரரினை ∴பங்ஷனுக்குத் திருப்பி அனுப்புவதாகும். அதாவது, ∴பங்ஷனுக்கு முழு எண் அறேயினைத் திருப்பி அனுப்பும்.

உதாரணமாக,

```
int* addMatrix(int* x, int* y, int n, int m)
```

```
    {
        int* a = x+m*n;
        int* z=a;
        for (int i=0; i<n i++)
            for (int j=0; j<m; j++)
              *z++ = (*x++) + (*y++);
        return (r);
    }
```

இவ்வாறு தசம தான பொயின்ரர் ∴பங்ஷன்கள் அல்லது எழுத்து பொயின்ரர் ∴பங்ஷன்கள் வரையறுக்க முடியும். அதாவது, தசம தான அறேயினை அல்லது எழுத்து அறேயினை ∴பங்ஷனுக்குத் திருப்பி அனுப்பும்.

அடுத்து, சி++ மொழியிலுள்ள எனும் (enum), யூனியன் (union) போன்ற கட்டளைகளைப் பார்ப்போம்.

சி++ மொழியில் நாமே உருவாக்கிக் கொள்ளும் விபர இனங்கள் (User Defined data types) போல் struct, union, enum, class போன்ற கட்டளைகள் தொழிற்படும்.

ஏற்கனவே ஸ்ரக்சர் பற்றித் தெளிவாகப் பார்த்துள்ளோம். எனவே, இங்கு யூனியன் (union) என்ற கட்டளையினை முதலில் பார்ப்போம்.

union என்ற கட்டளை அமைப்பானது, ஸ்ரக்சர் என்ற கட்டளை அமைப்பு போன்று காணப்பட்டாலும், செயற்பாட்டில் வேறுபாடு உண்டு.

உதாரணம் - 1

```
struct Student{
    char name[30];
    char indexNo[10];
    int age;
    char result;
} s;
```

என Student என்ற ஸ்ரக்சரிலிருந்து s என்ற மாறி உருவாக்கப் பட்டிருந்தால், இந்த s என்ற மாறிக்கு நினைவகத்தில் மொத்தம் 43 பைற்கள் (Bytes) ஒதுக்கப்படும். அதாவது, 30+10 +2+1 = 33 ஆகும்.

உதாரணம் - 2

```
union Student{
    char name[30];
    char indexNo[10];
    int age;
    char result;
} s;
```

என Student என்ற யூனியன் (union) இலிருந்து s என்ற மாறி உருவாக்கப்பட்டிருந்தால், இந்த s என்ற மாறிக்கு நினைவகத்தில் 30 பைற்கள் (Bytes) மட்டுமே ஒதுக்கப்படும். அதாவது, யூனியனிலுள்ள உறுப்புக்களில் மிகக் கூடிய நினைவகத்தினைக் கொண்ட உறுப்பின் நினைவக இடத்தினை மட்டுமே ஒதுக்கப்படும்.

அடுத்து, enum என்ற கட்டளையினைப் பார்ப்போம். enum என்ற கட்டளையானது குறிப்பிட்ட வகை தரவுகளின் சேர்க்கைக்குரிய விபர இனத்தினைக் கையாள உதவுகின்றது.

உதாரணமாக,

```
enum  weekday {mon,tue,wed,thu,fri};
enum  shape {circle, square, triangle};
```

என வரையறுத்து weekday, shape போன்ற விபர இனத்துக்குரிய மாறிகளை உருவாக்க முடியும்.

உதாரணமாக,

```
weekday day;
shape a;
```

int x என வரையறுப்பது போன்று, weekday day என வரையறுக்க முடியும். day என்ற மாறியில், weekday என்ற விபர இனத்துக்குரிய பெறுமானத்தினைச் சேமிக்க முடியும். புறோகிராமில், day = fri; என எழுத முடியும். அதாவது, day என்ற மாறியில் mon, tue, wed, thu, fri போன்ற பெறுமானங்களைச் சேமிக்க முடியும்.

அடுத்து, enum என்ற கட்டளைக்குரிய உதாரணப் புறோகிராமினைப் பார்ப்போம்.

```
#include <iostream.h>
enum  weekday{mon,tue,wed,thu,fri};
// mon - 0, tue - 1, wed - 2, thu - 3, fri - 4
void main()
{
    weekday  d1,d2;
    d1= mon;
    d2 = fri;
    int d = d2 - d1;
    cout<<d<<endl;
}
```

இந்தப் புறோகிராமில், weekday என்ற விபர இனத்துக்குரிய இரு மாறிகள் d1, d2 வரையறுக்கப்பட்டுள்ளன. பின்னர், d1 என்ற மாறியில் mon என்ற பெறுமானத்தினையும், d2 என்ற மாறியில் fri

என்ற பெறுமானத்தினையும் சேமிக்கும் விதமாகக் கட்டளைகள் எழுதப் பட்டுள்ளன. அடுத்த வரியில், d1 இற்கும் d2 இற்கும் இடையிலுள்ள வித்தியாசத்தை சேமிப்பதற்கு d என்ற மாறி வரையறுக்கப்பட்டுள்ளது. எனவே, இங்கு d என்ற மாறியில் பெறுமானம் 4 சேமிக்கப்படும். அதாவது, d = 4 - 0 = 4 ஆகும்.

enum இற்குரிய மேலும் சில உதாரணங்களைப் பார்ப்போம்.

உதாரணம் - 1

```
enum  shape{circle, square, triangle, rectangle};
```
என வரையறுத்த பின்னர்,
```
shape s;   // s is a type of shape
s = square; // now value of square assigned to s, s = 1.
```

உதாரணம் - 2

```
enum  colour{red, blue, green, yellow, orange};
```
என வரையறுத்த பின்னர்,
```
colour background;  // background is a type of colour
background=green;//now value of green assigned to background.
```

உதாரணம் - 3

```
enum  boolean{false, true};
```

பூலியன் (Boolean) என்ற விபர இனம் சி++ மொழியில் இல்லை என்பதால், உதாரணம் -3 இல் உள்ளவாறு enum கட்டளையினை எழுதிய பின்னர், புறோகிராமில் பயன்படுத்த முடியும்.

உதாரணமாக,
```
boolean  d1, d2;
d1 = "true";
d2 = "false";
```

7. ஒப்ஜெக்ற் ஓரியன்ரட் புறோகிராமிங் (Object Oriented Programming)

சி++ மொழியில் பயன்படுத்தப்படும் ஒப்ஜெக்ற் ஓரியன்ரட் புறோகிராமிங் (OOP - Object Oriented Programming) பற்றிய விளக்கங்களைத் தெளிவாகப் பார்ப்போம்.

முதலில், ஒப்ஜெக்ற் ஓரியன்ரட் புறோகிராமிங் என்றால் என்ன என்பதையும், இதனால் கிடைக்கும் நன்மைகள் என்ன என்பதையும் பார்ப்போம்.

ஆரம்ப காலகட்டத்தில் வெளியிடப்பட்ட மொழிகள் மூலம் எழுதப்பட்ட புறோகிராம்கள் யாவும் பல மொடியூலர்களின் (Modulars) சேர்க்கைகளால் உருவாக்கப்பட்டவையாகும். ஆனால், தற்பொழுது வெளியாகும் புதிய மொழிகள் மூலம் எழுதப்படும் பெரும்பாலான புறோகிராம்கள் **ஒப்ஜெக்ற் ஓரியன்ரட் புறோகிராமிங்** என்ற தத்துவத்துக்கு அமையவே எழுதப்படுகின்றன.

மொடியூலர் புறோகிராமிங் (Modular Programming) போலல்லாது, OOP மூலம் ஒரு முழு சிஸ்ரத்தின் ஒரு பகுதி வேலையைக் குறித்தொரு ஒப்ஜெக்ற் மூலம் நிறைவேற்ற முடியும்.

ஒப்ஜெக்ற் ஓரியன்ரட் புறோகிராமில், ஒவ்வொரு ஒப்ஜெக்ற்றும் தன்னகத்தே பண்புகளையும் (Attributes), செயல்முறைகளையும் (Functions) கொண்டிருக்கும்.

உதாரணமாக, மைக்ரோசொவ்ற் வின்டோஸ் (Microsoft Windows) ஒப்பறேற்றிங் சிஸ்ரத்தினைக் கருதுவோம். இந்த ஒப்பறேற்றிங் சிஸ்ரத்தில் காணப்படும் ஒவ்வொரு வின்டோவுக்கும் அகலம், உயரம் போன்ற பண்புகளும் (Attributes), பெரிதாக்குதல், சிறிதாக்குதல், நகர்த்துதல், திறத்தல், மூடுதல் போன்ற செயல்முறைகளும் (Functions) உள்ளன. இங்கு காணப்படும் ஒவ்வொரு வின்டோவும், ஒரு தனி ஒப்ஜெக்ற் (Object) போன்று தொழிற்படும். இவ்வாறான பல செயற்பாடுகளுக்கு இந்த ஒப்ஜெக்ற் ஓரியன்ரட் புறோகிராமிங் பயன்படுத்தப்படுகிறது. இன்று வெளியிடப்படும் அனைத்து தொகுப்புகளும் (Packages) ஒப்ஜெக்ற் சார்ந்ததாகவே காணப்படுகின்றன.

OOP இன் நன்மைகள்

◎ ஒரு ஒப்ஜெக்ற்றில் வரையறுக்கப்பட்ட டேற்றா பாதுகாக்கப்பட்டு (அதாவது, டேற்றா மறைக்கப்பட்டு - Data Hiding) பிற ஒப்ஜெக்ற்றிலுள்ள

∴பங்ஷன்களினால் அணுக முடியாது தடுக்கப்படுகிறது. இதன் மூலம், புறோகிராமில் ஏற்படும் எதிர்பாராத விளைவுகளைத் தவிர்க்க முடிகிறது.

உதாரணமாக, ஒரு மிகப் பெரும் செயல்திட்டத்துக்காக எழுதப்படும் புறோகிராமுக்கு ஒன்றுக்கு மேற்பட்ட புறோகிராமர்கள் தேவைப்படலாம். இங்கு, ஒவ்வொரு புறோகிராமரும் தமக்கு ஏற்றவாறு டேற்றாவை மாற்ற முற்படின், இப்புறோகிராமில் பல எதிர்பாராத விளைவுகள் ஏற்பட வாய்ப்புக்கள் உண்டு.

◎ OOP முறையைப் பயன்படுத்தி எழுதப்படும் புறோகிராம்களின் வரிகள் (Source Codes) குறைவாகவே காணப்படுகின்றன. மற்றும், இவற்றை மீள்திருத்தம் செய்வதும் இலகுவாக இருக்கும். எனவே, இப்புறோகிராம் எழுதுவதற்குரிய காலம், விலை போன்றவற்றையும் கட்டுப்படுத்த முடியும்.

அடுத்து, OOP இன் அடிப்படைத் தத்துவங்களான கிளாஸ் (Class), என்கப்சுலேஷன் (Encapsulation), இன்ஹரிற்ரன்ஸ் (Inheritance), பொலிமோபிஸம் (Polymorphism) போன்றவற்றைப் பல உதாரணங்கள் மூலம் விரிவாகப் பார்ப்போம்.

7.1 கிளாஸ் (Class)

ஒரே தன்மையான பண்புகளையும், செயற்பாடுகளையும் தன்னகத்தே கொண்ட ஒரு தொகுதி கிளாஸ் என அழைக்கப்படுகின்றது.

உதாரணமாக,

◎ ஒரு மாணவனை எடுத்தால், அந்த மாணவனுக்குப் பெயர், முகவரி, வயது, திறமை போன்ற பல பண்புகளும் அத்துடன் நடத்தல், உண்ணுதல், படித்தல் போன்ற செயற்பாடுகளும் காணப்படுகிறது.

◎ கார் ஒன்றை எடுத்தால், அந்தக் காருக்கு நிறம், வகை, வேகம் போன்ற பல பண்புகளுடன் ஓடுதல், நிறுத்துதல், இயக்குதல் போன்ற செயற்பாடுகளும் காணப்படுகிறது.

மேலே கூறப்பட்ட இரு உதாரணங்களுக்குமுரிய புறோகிராம்கள் எழுதும் போது, இரண்டையும் கிளாஸாக வரையறுப்பது சிறந்ததாகும். ஏனெனில், கிளாஸாக வரையறுப்பதன் மூலம் பல நன்மைகளைப் பெற முடியும். இவற்றைப் பின்னர் விரிவாகப் பார்ப்போம்.

கிளாஸ் ஒன்றினைப் புறோகிராமில் வரையறுப்பதற்கு முன்னர், கிளாஸ் வரிப்படத்தை (Class diagram) வரையப்பட வேண்டும். அதன் பின்னர் கிளாஸிற்குரிய புறோகிராமை எழுதுவது இலகுவாக இருக்கும்.

கிளாஸ் வரிப்படம்

கிளாஸ் வரிப்படத்தில், முதலில் கிளாஸினது பெயரினைக் குறிப்பிட வேண்டும். பின்னர், அற்ரிபியூற்களையும் (Attributes), ∴பங்ஷன் களையும் (Functions) குறிப்பிட வேண்டும். அதாவது, கீழேயுள்ளவாறு கிளாஸ் வரிப்படத்தை (Class diagram) வரைய வேண்டும்.

கிளாஸ் பெயர் (Class Name)
பண்புகள் (Attributes)
செயற்பாடுகள் (Functions)

உதாரணம் (1)

Employee	} Class Name
empNo name address salary	} Attributes
addEmp() ammendEmp() deleteEmp()	} Functions

உதாரணம் (2)

Student
regNo indexNo name
readData() printData()

சி++ மொழியில் எவ்வாறு கிளாஸ் ஒன்றை வரையறுப்பது என அடுத்துப் பார்ப்போம்.

முதலில் class என்ற சி++ மொழிக் கீவேட்டினை எழுதிய பின்னர், கிளாஸினது பெயரை எழுத வேண்டும். மேலும் கிளாஸிற்குள்

முதலில் அற்ரிபியூற்களையும் பின்னர், ∴பங்ஷன்களையும் எழுத வேண்டும். கிளாஸ் வரையறை முடிந்தவுடன் இறுதியாக அரைப்புள்ளி (; - Semicolon) கட்டாயம் இட வேண்டும்.

Student என்ற கிளாஸ் வரையறுக்கப்படும் முறையினை சி++ மொழிப் புறோகிராம் ரீதியாகப் பார்ப்போம்.

```
class Student
{
    char name[30];
    char regNo[10];
    char indexNo[10];
    int age;
    void readData()
    {
        cout << " Enter name : ";
        cin >> name;
        cout << " Enter Reg. number : ";
        cin >> regNo;
        cout << "Enter index number : ";
        cin >> indexNo;
        cout << "Enter age : ";
        cin >> age;
    }
    void display()
    {
        cout << " Name : "<< name << endl;
        cout << " Reg. No : " << regNo << endl;
        cout << " Index No : " << indexNo << endl;
        cout << " Age : " << age<< endl;
    }
};
```

இந்த உதாரணத்திலுள்ள கிளாஸிற்குள் நான்கு அற்ரிபியூற்களும், இரண்டு ∴பங்ஷன்களும் எழுதப்பட்டுள்ளன.

ஒரே வகையான விபர இனங்களை (Data types) ஒரு மாறியின் பெயரில் வரையறுப்பதை அறே (Array) என அழைக்கப்படுகின்றது. பல வகையான விபர இனங்களையும், ∴பங்ஷன்களையும் ஒரு மாறியின் பெயரில் ஸ்ரக்சர் அல்லது கிளாஸாக வரையறுக்க முடியும்.

கிளாஸ், ஸ்ரக்சரிற்குள் வரையறுக்கப்படும் அற்ரிபியூற்களின் பாதுகாப்பிற்காகவும், ∴பங்ஷன்களின் செயற்பாடுகளுக்கும் அவை மூன்று பிரிவுகளுக்குள் வரையறுக்கப்படுகின்றன.

அவையாவன,

1.பிரைவேற் (private)
2.பப்பிளிக் (public)
3.புரொரெக்ரட் (protected)

பிரைவேற் (private)

கிளாஸ் ஒன்றில் அற்றிபியூற்கள், ∴பங்ஷன்கள் போன்றவற்றை private என்ற பிரிவுக்குள் வரையறுத்தால், அக்கிளாஸிற்குள் மட்டுமே அந்த அற்றிபியூற்கள், ∴பங்ஷன்கள் போன்றவற்றைப் பயன்படுத்த முடியும். கிளாஸிற்கு வெளியில் காணப்படும் மெயின் ∴பங்ஷனில் அல்லது இன்ஹெரிற் (Inherit) செய்யப்பட்ட கிளாஸில், அந்த பிரைவேற் (private) அற்றிபியூற்கள், ∴பங்ஷன்கள் பயன்படுத்தப்பட முடியாது.

பப்பிளிக் (public)

ஒரு கிளாஸில் public என்ற பிரிவுக்குள் அற்றிபியூற்கள், ∴பங்ஷன் கள் வரையறுக்கப்பட்டிருந்தால், அக்கிளாஸிற்குரிய அற்றிபியூற்கள், ∴பங்ஷன்கள் என்பவற்றை எந்த இடத்திலும் பயன்படுத்த முடியும். அதாவது, இன்ஹெரிற் செய்யப்பட்ட கிளாஸில் அல்லது மெயின் ∴பங்ஷனில் அந்த அற்றிபியூற்கள், ∴பங்ஷன்கள் போன்றவற்றைப் பயன்படுத்த முடியும்.

புரொரெக்ரட் (protected)

கிளாஸ் ஒன்றில், protected என்ற பிரிவுக்குள் அற்றிபியூற்கள், ∴பங்ஷன்கள் வரையறுக்கப்பட்டிருந்தால், அக்கிளாஸில் அல்லது இன்ஹெரிற் செய்யப்பட்ட கிளாஸில் அந்த அற்றிபியூற்கள், ∴பங்ஷன் கள் போன்றவற்றைப் பயன்படுத்த முடியும். ஆனால், மெயின் ∴பங்ஷனில் பயன்படுத்த முடியாது.

பொதுவாக அற்றிபியூற்கள் private என்ற பிரிவுக்குள்ளும், ∴பங்ஷன் கள் public என்ற பிரிவுக்குள்ளும் வரையறுக்கப்படுகின்றன. ஏனெனில், அற்றிபியூற்களில் தவறுதலாக மாற்றம் செய்யப்பட்டால், அந்தப் புறோகிராமில் பல பிரச்சினைகள் ஏற்பட வாய்ப்புக்கள் உண்டு.

உதாரணமாக ஒரு மாணவருக்குரிய பெயர், சுட்டிலக்கம், மதிப்பெண் கள் போன்ற அற்றிபியூற்கள், இந்த அற்றிபியூற்களை உள்ளீடு, வெளியீடு செய்வதற்கு readData(), display() போன்ற ∴பங்ஷன்கள் Student என்ற கிளாஸில் வரையறுத்துப் புறோகிராமில் கீழேயுள்ளவாறு எழுத முடியும்.

```
class Student
{
```

```
    private:
        char name[40];
        char indexNo[10];
        int marks;
    public:
        void readData();
        void display();
};
```

மேலேயுள்ள உதாரணத்தில் பெயர், சுட்டிலக்கம், மதிப்பெண்கள் போன்ற அற்ரிபியூற்கள் private என்ற பிரிவுக்குள்ளும் readData(), display() போன்ற ∴பங்ஷன்கள் public என்ற பிரிவுக்குள்ளும் வரையறுக்கப்பட்டுள்ளன.

அடுத்து, சி++ மொழியில் ஏற்கனவே நாம் ஆராய்ந்த ஸ்ரக்சரிற்கும், கிளாஸிற்கும் இடையிலுள்ள வித்தியாசங்களைப் பார்ப்போம்.

◆ ஸ்ரக்சரில் வரையறுக்கப்படும் டேற்ரா (Data), ∴பங்ஷன்கள் (Functions) போன்றன இயல்பாக public என்ற தகுதியைப் பெற்றுச் செயற்படும். ஆனால், கிளாஸில் வரையறுக்கப்படும் டேற்ரா, ∴பங்ஷன் கள் போன்றன இயல்பாக private என்ற தகுதியைப் பெற்றுச் செயற்படும்.

- **உதாரணம் - 1**

```
struct Student
{
    char name[40];
    int age;
    void getData();
    void printData();
};
```

- **உதாரணம் - 2**

```
class Student
{
    char name[40];
    int age;
    void getData();
    void printData();
};
```

உதாரணம் -1 இல், Student என்ற ஸ்ரக்சரில் name, age போன்ற இரு அற்ரிபியூற்களும் getData(), printData() போன்ற இரு ∴பங்ஷன் களும் வரையறுக்கப்பட்டுள்ளன. இவை நான்கும் இயல்பாக public

என்ற தகுதியைப் பெற்றுச் செயற்படும். ஆனால், உதாரணம்-2 இல் உள்ளது போல் கிளாஸாக வரையறுத்திருந்தால் அற்றிபியூற்களும், ∴பங்ஷன்களும் இயல்பாக private என்ற தகுதியைப் பெற்றுச் செயற்படும்.

- கிளாஸாக வரையறுக்கப்பட்டிருந்தால், அந்தக் கிளாஸினை அடிப்படையாகக் கொண்டு இன்னுமொரு புதிய கிளாஸினை உருவாக்க முடியும். ஆனால், ஸ்ரக்சராக வரையறுக்கப்பட்டிருந்தால், புதிய ஸ்ரக்சரை இன்ஹெரிற் (Inherit) செய்து பெற முடியாது.

எனவே, மேலே கூறப்பட்ட இரு நன்மைகளையும் பார்க்கும் போது ஸ்ரக்சராக வரையறுப்பதைவிடக் கிளாஸாக வரையறுப்பது சிறந்த தாகும்.

பொதுவாகப் புறோகிராமர்கள், தரவுகளின் தொகுதியை ஸ்ரக்சரா கவும், தரவுகள், ∴பங்ஷன்கள் போன்றவற்றின் தொகுதியைக் கிளாஸாகவும் வரையறுக்கிறார்கள்.

அடுத்து, கிளாஸிற்குரிய உதாரணம் ஒன்றைப் புறோகிராம் ரீதியாகப் பார்ப்போம்.

Student என்ற கிளாஸினை வரையறை செய்த பின்னர், ஒரு ஒப்ஜெக்ற்றினை உருவாக்கி, அதற்குரிய தகவல்களை உள்ளீடாகக் கொடுத்து, அவற்றினைத் திரையில் வெளியீடாகக் காண்பிப்பதற்குரிய புறோகிராம் ஒன்றைப் பார்ப்போம்.

```
#include <iostream.h>
class  Student
{
    private:    // this statement is optional
          char name[30];
          char indexNo[10];
          int marks;
      public:
          void  readData()
          {
              cout << " Enter the name : ";
              cin >> name;
              cout << "Enter the index number : ";
              cin >> indexNo;
              cout<< "Enter the marks : ";
              cin >> marks;
          }
```

```
        void display();
        {
            cout << " Student Report : "<<endl;
            cout << " Name : "<< name << endl;
            cout << " Index No : " << indexNo << endl;
            cout << " Maths Marks : " << marks << endl;
        }
};
void main()
{
    Student s;
    s.readData();
    s.display();
}
```

இந்த Student என்ற கிளாஸிலுள்ள முதல் வரியில் private என்ற சொல் எழுதப்பட்டுள்ளது. ஆனால், இந்த private என்ற சொல்லை எழுத வேண்டிய அவசியமில்லை. ஏனெனில், கிளாஸாக வரையறுத் திருப்பதால், இயல்பாகவே அவை private என்ற தகுதியைப் பெற்றுச் செயற்படும்.

இங்கு Student என்பது ஒரு விபர இனமாகும். s என்பது Student என்ற விபர இனத்தில் அறிவிக்கப்பட்டுள்ள ஒரு மாறியாகும். இந்த மாறி s இனை ஒப்ஜெக்ற் (Object) என அழைக்கப்படுகின்றது.

உதாரணமாக, செங்கற்களை உருவாக்கப் பயன்படும் அச்சு போன்ற அமைப்பே கிளாஸின் அமைப்பாகும். இந்த அச்சிலிருந்து உருவாக்கப்படும் ஒவ்வொரு செங்கற்கள் போன்றே கிளாஸிலிருந்து உருவாக்கப்படும் ஒப்ஜெக்ற்கள் ஆகும்.

மெயின் என்ற ஃபங்ஷனின் முதல் வரியில் Student என்ற கிளாஸிலிருந்து ஒரு ஒப்ஜெக்ற் s உருவாக்கப்பட்டுள்ளது.

Student என்ற கிளாஸிலிருந்து ஓர் உறுப்பினை ஒப்ஜெக்ற் மாறியாகவும், பொயின்ரர் (Pointer) மாறியாகவும் அறிவிக்க முடியும்.

Student s; என s என்ற ஒப்ஜெக்ற் மாறியாக அறிவித்திருந்தால்,

```
s.readData();
s.display();
```

என s என்ற ஒப்ஜெக்ற்றின் உறுப்புக்களைக் கையாள முடியும். எனினும், Student என்ற கிளாஸில் private என்ற பிரிவுக்குள் name, indexNo, marks போன்ற அற்றிபியூற்களை அறிவித்திருப்பதால் s.name, s.indexNo, s.marks என கையாள முடியாது. ஏனெனில், private

என்ற பிரிவுக்குள் அறிவிக்கப்பட்டவற்றை ஒருபோதும் கிளாஸிற்கு வெளியே பயன்படுத்த முடியாது. மாறாகப் பயன்படுத்தினால், புறோகிராமினைக் கொம்பைல் செய்யும் போதே பிழையினைச் சுட்டிக் காட்டும்.

Student *s = new Student; என s என்ற ஒப்ஜெக்ற்றினைப் பொயின்ரர் மாறியாக அறிவித்திருந்தால்,

```
s->readData();
s->display();
```

என s என்ற ஒப்ஜெக்ற்றின் உறுப்புக்களைக் கையாள முடியும். ஒப்ஜெக்ற்களை பொயின்ரர் மாறிகள் மூலம் கையாளும் போது new என்ற ஒப்பறேற்றர் மூலம் நினைவக இடம் ஒதுக்கப்படுகின்றது.

ஒரு கிளாஸிலிருந்து ஒன்றுக்கு மேற்பட்ட ஒப்ஜெக்ற்களை உருவாக்க முடியும்.

உதாரணமாக, 50 ஒப்ஜெக்ற்களை உருவாக்க வேண்டுமாயின், 50 ஒப்ஜெக்ற்களுக்குரிய அறேயினை வரையறுக்க வேண்டும். அதாவது, Student s[50]; என வரையறுக்க வேண்டும்.

சாதாரண மாறிகளுக்கு ஆரம்பப் பெறுமானங்கள் கொடுப்பது போல், கிளாஸிலிருந்து உருவாக்கப்படும் ஒப்ஜெக்ற்களுக்கும் ஆரம்பப் பெறுமானங்களைக் கொடுக்க முடியும்.

உதாரணமாக,

Student s = {"siva", "s5283", 90} என Student இலிருந்து உருவாக்கப்பட்ட s என்ற ஒப்ஜெக்ற்றிற்கு ஆரம்பப் பெறுமானங்களாக "siva", "s5283", 90 போன்றன கொடுக்கப்பட்டுள்ளன.

மேலேயுள்ள உதாரணத்தில், கிளாஸிற்குரிய ∴பங்ஷன்களை கிளாஸிற்குள்ளேயே எழுதியியுள்ளோம். எனினும், இந்தக் கிளாஸிற் குரிய ∴பங்ஷன்களை கிளாஸிற்கு வெளியிலும் எழுத முடியும்.

கிளாஸிற்கு வெளியே ∴பங்ஷன்களை எழுத வேண்டுமாயின், ∴பங்ஷனின் முன்வடிவம் (Proto type) கிளாஸிற்குள் அறிவிக்கப்பட வேண்டும்.

உதாரணமாக,

```
class Student
{
    private:
        -----------
        -----------
    public:
```

```
                void readData();  // Function proto type
                void display();  // Function proto type
        };
    void Student::readData()
    {
        -------;
        -------;
    }
    void Student::display()
    {
        -------;
        -------;
    }
    void main()
    {
        -------;
        -------;
    }
```

இவ்வாறு ∴பங்ஷனின் முன்வடிவத்தைக் கிளாஸிற்குள் அறிவித்த பின்னர், கிளாஸிற்கு வெளியே இந்த ∴பங்ஷன்களை வரையறுக்க முடியும்.

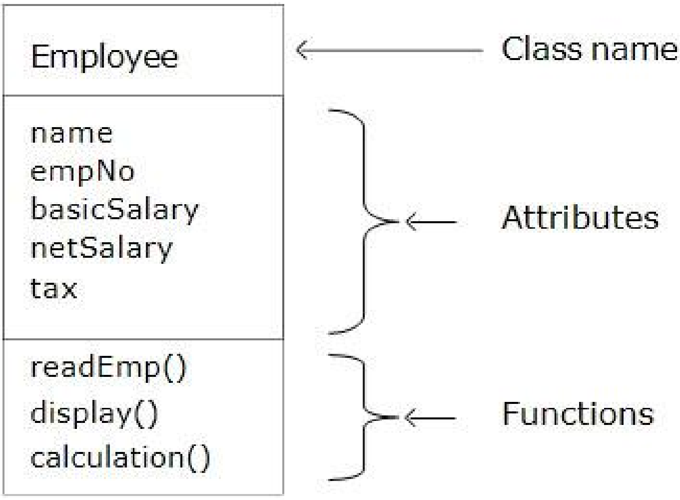
அடுத்து, கிளாஸிற்குரிய மேலும் ஒரு உதாரணத்தைப் புறோகிராம் ரீதியாகப் பார்ப்போம்.

ஒரு ஊழியரைக் கிளாஸாகக் கருதுமிடத்து, அவருக்குரிய அல்லது கிளாஸிற்குரிய பண்புகளாக (அற்றிபியூற்கள் - Attributes) name, empNo, netSalary, basicSalary, tax என்பவற்றைக் குறிப்பிடலாம்.

basicSalary 10,000/= ஐ விடக் கூடுதலாகவும் 20,000/= ஐ விடக் குறைவாகவும் இருப்பின், 8% வரி அறவிடப்படுகிறது. basicSalary 20,000/= அல்லது 20,000/= விடக் கூடுதலாக இருப்பின் வரியாக 12% அறவிடப்படுகிறது.

இந்த கிளாஸில் மூன்று செயற்பாடுகளை (Functions) வரையறுப்பது சிறந்ததாகும். அவையாவன readEmp(), calculation(), display() போன்றனவாகும்.

வழக்கம்போல் நாம் கிளாஸிற்குரிய புறோகிராமை வரையறுக்க முன்னர், கிளாஸிற்குரிய கிளாஸ் வரிப்படத்தை (Class diagram) வரைய வேண்டும். ஏனெனில், கிளாஸ் வரிப்படம் வரைந்த பின்னர், கிளாஸிற்குரிய புறோகிராமை எழுதுவது இலகுவாக இருக்கும்.




```
#include <iostream.h>
class Employee
{
    private:
        char name[40];
        char empNo[10];
        double basicSalary, netSalary, tax;
    public:
        void readEmp();
        void calculation();
        void display();
};
void Employee::readEmp()
{
    cout << " Enter the employee name : ";
    cin >> name;
    cout<< " Enter the employee number : ";
    cin >> empNo;
    cout << "Enter the Basic Salary : ";
    cin >> basicSalary;
}
void Emplyee::calculation()
{
    if (basicSalary <= 10000) tax = 0 ;
    else if (basicSalary < 20000) tax= 0.08;
```

```
    else if (basicSalary>=20000) tax = 0.12;
    netSalary = basicSalary - tax * basicSalary;
}
void Employee::display()
{
    cout<<"Employee name:" << name << endl;
    cout<<"Employee number:"<< empNo <<endl;
    cout<<"Basic salary:"<< basicSalary <<endl;
    cout<<"Net salary:"<< netSalary <<endl;
}
void main()
{
    Employee e[10];
    for (int i=0; i < 10; i ++)
    {
        e[i].readEmp();
        e[i].calculation();
    }
    cout<<"Employee Details "<<endl;
    for (i=0; i<10; i++)
        e[i].display();
}
```

மேலேயுள்ள புறோகிராமில், Employee என்ற கிளாஸிற்குரிய பண்புகள் யாவும் private என்ற கட்டமைப்புக்குள் மட்டும் வரையறுக்கப்பட்டுள்ளதன் காரணம் யாதெனில், எந்தவொரு புறோகிராமரும் தவறுதலாகக் கிளாஸிற்குரிய பண்பில் மாற்றத்தை ஏற்படுத்தாமல் தடுப்பதற்காகும்.

மெயின் ∴பங்ஷனிற்குள் காணப்படும் முதலாவது கட்டளை மூலம் e என்ற அறேயில், Employee என்ற கிளாஸிற்குரிய 10 ஒப்ஜெக்ற்கள் உருவாக்கப்படும்.

மேலேயுள்ள உதாரணத்தில் மெயின் ∴பங்ஷனில், Employee என்ற கிளாஸின் உறுப்புக்களை ஒப்ஜெக்ற் மாறியாக அறிவிக்காது, பொயின்ரர் மாறியாகவும் அறிவிக்க முடியும்.

உதாரணமாக, Employee *e = new Employee என பொயின்ரர் மாறியாக அறிவிக்க முடியும்.

பொயின்ரர் மாறியாக வரையறுக்கப்படுவதால் பல நன்மைகள் உண்டு. அவற்றில் ஒன்றை நாம் இங்கு பார்ப்போம்.

புறோகிராம் ஒன்றை உருவாக்கும் போது, எத்தனை உறுப்புக்கள் உண்டு என்பது தெரியாதிருக்கும் சந்தர்ப்பத்தில், பொயின்ரர் மாறியாக வரையறுப்பது புத்திசாலித்தனமாகும். ஏனெனில், பொயின்ரர் மாறியானது புறோகிராம் செயற்படுத்தும் போதுதான் உறுப்புக்களின் எண்ணிக்கையைத் தீர்மானித்துக் கொள்ளும்.

ஒப்ஜெக்ற் மாறியாக வரையறுக்கும் போது, புறோகிராமைக் கொம்பைல் (Compile) செய்யும் போதே எத்தனை உறுப்புக்கள் தேவை என்பது தெரிந்திருக்க வேண்டும். ஒப்ஜெக்ற் மாறியாக வரையறுப்பதால், சில பிரதிகூலங்கள் ஏற்பட வாய்ப்புண்டு. அதாவது, புறோகிராமை செயற்படுத்தும் போது 1000 ஒப்ஜெக்ற்கள் தேவை என அறிவிக்கப்பட்ட பின்னர், 200 ஒப்ஜெக்ற்கள் மட்டுமே எமக்குத் தேவைப்பட்டால், மீதி 800 ஒப்ஜெக்ற்களுக்குரிய நினைவக இடம் தேவையில்லாமல் வீணாகும். இதைத் தவிர்ப்பதற்கு கிளாஸிற்குரிய ஒப்ஜெக்ற்களைப் பொயின்ரர் மாறியாக வரையறுப்பது சிறந்ததாகும்.

மேலேயுள்ள உதாரணத்தைப் பொயின்ரர் மாறியாக வரையறுப்பதற்கு மெயின் ∴பங்ஷனில் மட்டும் திருத்தம் செய்தால் போதுமாகும். புறோகிராமைச் செயற்படுத்தும் போது எத்தனை உறுப்புக்கள் உண்டு என்பது எமக்குத் தெரியாத சந்தர்ப்பத்திற்குரிய புறோகிராம் கீழே எழுதப்பட்டுள்ளது.

```
void main()
{
    char ch;
    int n = 0;
    do
    {
        Employee *e = new Employee;
        e->readEmp();  // read all Employees
        e->calculation(); // calculation for all Employees
        n++;  // count the number of objects
        cout<<"Do you want to append a new record (y/n)?";
        cin>>ch;
    }
    while ((ch == 'Y') || (ch == 'y'));
    for (int i = 1; i <= n; i++)
        e-> display(); // display all the employees
}
```

இந்தப் புறோகிராமானது எத்தனை தடவை செயற்படும் என்பதனைப் புறோகிராமினைச் செயற்படுத்திக் கொண்டிருக்கும் சந்தர்ப்பத்தில் பயன்பாட்டாளர் தீர்மானிக்க வேண்டும். அதாவது, பயன்பாட்டாளர் 'Y' அல்லது 'y' இனைத் தவிர்த்து, ஏதாவதொரு எழுத்தை அழுத்தும் போது மட்டுமே புறோகிராம் செயற்படுவது நிறுத்தப்படும்.

அடுத்து, கிளாஸிற்குரிய கொன்ஸ்ரக்ரர் (Constructor), டீஸ்ரக்ரர் (Destructor) போன்றவற்றைப் பார்ப்போம்.

கொன்ஸ்ரக்ரர் (Constructor)

கொன்ஸ்ரக்ரர் என்றால், ஒரு கிளாஸிலிருந்து ஒப்ஜெக்ற்களை உருவாக்கும் பொழுதே, கொன்ஸ்ரக்ரருக்குரிய கட்டளைகள் யாவும் செயற்படுத்தப்படும். பொதுவாக கொன்ஸ்ரக்ரர் ஒப்ஜெக்ற்களுக்குரிய பண்புகளுக்கு (Attributes) ஆரம்பப் பெறுமானங்களைக் கொடுப்பதற்குப் பயன்படுத்தப்படுகிறது.

கொன்ஸ்ரக்ரர் என்பது கிளாஸிற்குரிய மெம்பர் ∴பங்ஷன் (Member Function) ஆகும். ஆனால், இந்த மெம்பர் ∴பங்ஷனானது கிளாஸின் பெயராக இருக்க வேண்டும். மேலும், ஒரு கிளாஸிலிருந்து ஒப்ஜெக்ற்களை உருவாக்கும் போது அந்த கொன்ஸ்ரக்ரர் தானாகவே செயற்படும். அதாவது, கொன்ஸ்ரக்ரரைப் புறோகிராமில் ∴பங்ஷன்கள் அழைப்பது போல் அழைக்கத் தேவையில்லை.

கொன்ஸ்ரக்ரர் ஒன்றை வரையறுக்கும் போது மூன்று முக்கிய விடயங்களைக் கவனிக்க வேண்டும்.

1. கொன்ஸ்ரக்ரரின் பெயரானது கிளாஸின் பெயராக அமைய வேண்டும்.
2. கொன்ஸ்ரக்ரர் என்ற மெம்பர் ∴பங்ஷனானது, ஒரு பெறுமானத்தையும் திருப்பியனுப்பக் கூடாது. அதாவது, return என்ற கட்டளையை இங்கு பயன்படுத்தக்கூடாது.
3. கொன்ஸ்ரக்ரர் என்ற மெம்பர் ∴பங்ஷனை public என்ற பிரிவுக்குள் வரையறுக்க வேண்டும். பொதுவாக, private அல்லது protected என்ற பிரிவுக்குள் கொன்ஸ்ரக்ரர்கள் வரையறுக்கப்படுவதில்லை.

கொன்ஸ்ரக்ரர் (Constructor) இனது கட்டளை அமைப்பு:

```
class class_name
{
    private:
        ...... // private members
    public:
```

```
        class_name(); // constructor prototype
         ...... // other functions
};
class_name::class_name() // constructor definitions
{
    // constructor body definition
}
```

அடுத்து நாம் கொன்ஸ்ரக்ரருக்குரிய உதாரணப் புறோகிராமினைப் பார்ப்போம்.

மாணவனுக்குரிய கிளாஸ் ஒன்றை வரையறுத்து, இந்தக் கிளாஸிலிருந்து ஒப்ஜெக்ற்களை உருவாக்கும் பொழுது கிளாஸிற்குள் இருக்கும் பண்புகளான பெயர், சுட்டிலக்கம், மூன்று மதிப்பெண்கள் போன்றவற்றிற்கு ஆரம்பப் பெறுமானங்களைக் கொடுப்பதற்குரிய புறோகிராமானது கீழே தரப்பட்டுள்ளது.

```
#include <iostream.h>
class Student
{
   private:
      char *name, *indexNo;
      int m1,m2,m3;
   public:
      Student()  // constructor
      {
         name = "Selva";
         indexNo = "S 5283";
         m1 = 70;
         m2 = 90;
         m3 = 65;
      }
      void display()
      {
         cout<<"Name : "<<name <<endl;
         cout<<"Index No : "<< indexNo <<endl;
         cout<<"Marks 1 : "<<m1 << endl;
         cout<<"Marks 2 : "<< m2 << endl;
         cout<<"Marks 3 : "<< m3 << endl;
      }
};
void main()
{
```

```
    Student *s = new Student();
    s->display();
}
```

கொன்ஸ்ரக்ரானது பல பராமீற்றர்களைக் கொண்டிருக்கலாம். உதாரணமாக Student(char name[80], int mark, int age) என மூன்று பராமீற்றர்கள் Student என்ற கொன்ஸ்ரக்ரர் கொண்டிருக்க முடியும். இதற்குரிய உதாரணப் புறோகிராமினை அடுத்துப் பார்ப்போம்.

```
#include <iostream.h>
class Student
{
    private:
        char *name;
        int marks;
        int age;
    public:
        Student (char *na, int m, int a)
        {
            name = na;
            marks = m;
            age = a;
        }
        void display()
        {
            cout <<"Name : " <<name << endl;
            cout <<"marks : " << marks << endl;
            cout <<"Age : " << age << endl;
        }
};
void main()
{
    Student *s = new Student ("Siva", 90, 24);
    s->display();
}
```

மேலேயுள்ள புறோகிராமில், Student என்ற கிளாஸிலிருந்து s என்ற பொயின்ரர் ஒப்ஜெக்ற்றினை உருவாக்கும் பொழுதே, ஆரம்பப் பெறுமானமாக Siva, 90, 24 போன்றவை கொடுக்கப்பட்டுவிடும்.

டிஸ்ரக்ரர் (Destructor)

கிளாஸ் ஒன்றிலிருந்து உருவாக்கப்பட்ட ஒப்ஜெக்ற்களை நினைவகத்திலிருந்து முற்றாக நீக்குவதற்கு டிஸ்ரக்ரர் பயன்படுத்தப்

படுகிறது. அதாவது, கிளாஸ் ஒன்று செயற்பட்டு முடிவடையும் போது, ஒப்ஜெக்ற்களுக்குரிய நினைவகத்தை விடுவிப்பது இதன் பணியாகும்.

டிஸ்ரக்ரரானது, கொன்ஸ்ரக்ரர் போன்று பல பண்புகளைக் கொண்டதாகும். அதாவது, டிஸ்ரக்ரரின் பெயரானது கிளாஸின் பெயராகவும், ஒரு பெறுமானத்தையும் திருப்பி அனுப்பாத ஒரு மெம்பர் ∴பங்ஷன் (Member Function) ஆகவும் காணப்படும். எனினும், டிஸ்ரக்ரர் ஒரு பராமீற்றர் (Parameter) ஐயும் கொண்டிருக்காது.

இந்த டிஸ்ரக்ரர், "~" (ரைல்ட் - tilde) என்ற குறியீட்டுடன் கிளாஸின் பெயராக அமைய வேண்டும்.

டிஸ்ரக்ரர் (Destructor) இனது கட்டளை அமைப்பு:

```
class class_name
{
    private:
        ...... // private members
    public:
        ~class_name(); // Destructor prototype
        ...... // other functions
};
class_name::~class_name() // Destructor definitions
{
    //  Destructor body definition
}
```

இந்த டிஸ்ரக்ரரை வரையறுக்கும் போது, கிளாஸிற்கு முன் "~" என்ற குறியீட்டினை இட வேண்டும். ஆனால், கொன்ஸ்ரக்ரர் போன்று டிஸ்ரக்ரருக்குள் கட்டளைகள் இருக்க வேண்டிய அவசியமில்லை.

உதாரணமாக,

```
~ Student()
{
}
```

அடுத்து, கிளாஸ் ஒன்றில் கொன்ஸ்ரக்ரர், டிஸ்ரக்ரர் போன்றவை எவ்வாறு வரையறுக்கப்படுகின்றன என்பதற்குரிய உதாரணத்தினைப் பார்ப்போம்.

```
class Student
{
   private:
      char *name;
      int marks;
   public:
```

```
    Student()
    {
        cout<<"Enter the name: ";
        cin>>name;
        cout<<"Enter the marks: ";
        cin>>marks;
    }
    void display()
    {
        cout <<"name : " <<name<<endl;
        cout <<"marks : " <<marks<<endl;
    }
    ~Student()
    {
        cout << "Destructor works : "<<endl;
    }
};
```

கிளாஸிலிருந்து ஒப்ஜெக்ற்கள் உருவாக்கப்படும் பொழுதே அந்த ஒப்ஜெக்ற்களுக்குரிய பண்புகளுக்குப் (Attributes) பெறுமானத்தைக் கொடுப்பதற்கு கொன்ஸ்ரக்ரர் பயன்படுத்த முடியும் எனவும், கிளாஸிலிருந்து உருவாக்கப்பட்ட ஒப்ஜெக்ற்களுக்குரிய நினைவக இடத்தை முற்றாக விடுவிக்க டீஸ்ரக்ரர் பயன்படுத்த முடியும் எனவும் பார்த்தோம்.

இந்த டீஸ்ரக்ரரின் மூலம் நினைவகத்தில் (RAM - Random Access Memory) உள்ள ஒப்ஜெக்ற்களுக்குரிய நினைவக இடத்தை எந்த நேரத்திலும் விடுவிக்க முடியும். இதற்கு delete என்ற கட்டளை பயன்படுத்தப்படுகிறது.

உதாரணமாக கொன்ஸ்ரக்ரர், டீஸ்ரக்ரர் போன்றன ஒரு கிளாஸில் வரையறுக்கப்பட்ட பின்னர், இக்கிளாஸிலிருந்து உருவாக்கப்பட்ட ஒப்ஜெக்ற்றினை delete என்ற கட்டளை மூலம் நினைவகத்திலிருந்து விடுவிக்க முடியும்.

உதாரணமாக,

Student *s = new Student;

delete s; என கட்டளை எழுதுவதன் மூலம் நினைவகத்திலிருக்கும் s என்ற ஒப்ஜெக்ற்ருக்குரிய நினைவகத்தை விடுவிக்க முடியும்.

அடுத்து கொன்ஸ்ரக்ரர், டீஸ்ரக்ரர் போன்றன இயல்பாக எவ்வாறு செயல்படுகின்றன என்பதற்குரிய உதாரணப் புறோகிராமினைப் பார்ப்போம்.

```
#include <iostream.h>
class Customer
```

```
{
    char *name; // my;yJ char name [30];
    char *telnum;
public:
    Customer(char *n, char *tel)
    {
        cout <<"Now create an object" <<endl;
        name = n;
        telnum = tel;
    }
    void display()
    {
        cout<<"Customer's name:" << name <<endl;
        cout<<"Customer's telephone no:" << telnum <<endl;
    }
    ~ Customer()
    {
        cout <<"Now dispose the object" << endl;
    }
};
void main()
{
    Customer *c1 = new Customer ("Siva ","411978");
    Customer *c2 = new Customer ("Kumar", "491498");
    cl.display();
    delete c1;
    c2. display();
    delete c2;
}
```

இந்தப் புறோகிராமினைச் செயற்படுத்திப் பார்த்தால், முதலில் உருவாக்கப்பட்ட c1 என்ற ஒப்ஜெக்ற்ருக்குரிய கொன்ஸ்ரக்ரர் செயற்படும். எனவே, வெளியீடானது திரையில் கீழேயுள்ளவாறு தோன்றும்.

```
Now create an object
Customer name : Siva
Customer telephone no : 411978
```

பின்னர், c1 என்ற கிளாஸிற்குரிய டிஸ்ரக்ரர் செயற்பட்டு Now dispose the object என்ற வெளியீட்டையும் திரையில் காண்பிக்கும். அடுத்து, இரண்டாவது ஒப்ஜெக்ற் c2 இன் செயற்பாடானது, முதல் ஒப்ஜெக்ற் போன்றே வெளியீட்டைக் காண்பிக்கும்.

delete c1, delete c2 போன்ற கட்டளைகள் குறிப்பிடப்படாதிருந்தால், இரு ஒப்ஜெக்ற்களும் உருவாக்கப்பட்டு விடையினை வெளியீடாகக்

காட்டிய பின்னரே இறுதியாக c1, c2 போன்ற ஒப்ஜெக்ற்களுக்குரிய டீஸ்ரக்ரர் செயற்படும். எனவே வெளியீடாக,

```
Now create an object
Customer's name : Siva
Customer's telephone no : 411978
Now create an object
Customer's name : Kumar
Customer's telephone no: 491498
Now dispose the object
Now dispose the object
```

போன்றன திரையில் தோன்றும்.

delete என்ற கட்டளையைப் பயன்படுத்துவதால், கிளாஸிலிருந்து உருவாக்கப்பட்ட ஒப்ஜெக்ற்களுக்குரிய நினைவக இடத்தை எந்த நேரத்திலும் விடுவிக்க முடியும்.

சி++ மொழியில் this என்ற கீவேட்டானது, ஒப்ஜெக்ற்றைச் சுட்டும் பொயின்ரராகத் தொழிற்படும். டேற்ரா மெம்பர்களின் பெயரும், கொன்ஸ்ரக்ரர் அல்லது ஃபங்ஷனில் காணப்படும் பராமீற்றர்களின் பெயரும் ஒன்றாக இருந்தால், இந்த this என்ற கீவேட் (Keyword) பயன்படுத்தப்படுகின்றது.

உதாரணமாக, Student என்ற கிளாஸில் name, marks போன்ற இரண்டு டேற்ரா மெம்பர்களும், கொன்ஸ்ரக்ரரும் காணப்படுகின்றது. இந்த கொன்ஸ்ரக்ரருக்கு இரண்டு பராமீற்றர்களான name, marks போன்றன வரையறுக்கப்படுகின்றன.

```
class Student
{
    private:
        char* name;
        int marks;
    public:
        Student (char* name, int marks)
        {
            this->name = name;
            this->marks = marks;
        }
};
```

ஒரு கிளாஸிலுள்ள டேற்ரா மெம்பர்களைச் சுட்ட this என்ற கீவேட் பயன்படுத்தப்படுகின்றது.

கிளாஸில் வரையறுக்கப்பட்ட ஸ்ரரிக் மாறிகள்

கிளாஸிலுள்ள அற்றிபியூற்றானது ஸ்ரரிக் ஆகக் காணப்பட்டால், இந்த அற்றிபியூற் கிளாஸினது மாறி (Class Variable) போல் தொழிற்படும். கிளாஸிற்குள் ஸ்ரரிக் அற்றிபியூற் பயன்படுத்துவதன் நோக்கம் யாதெனில், இக்கிளாஸில் எத்தனை ஒப்ஜெக்ற்கள் உருவாக்கப்பட்டுள்ளது எனக் கணிப்பதற்காகும்.

உதாரணமாக,

```
class Student
{
        static int count; // count objects of this class
        .......
    public:
        Student()
        {
            count++;
            ........
        }
};
```

7.2 என்கப்சுலேஷன் (Encapsulation)

அடுத்து, ஒப்ஜெக்ற் ஓரியன்ரட் புறோகிராமின் அடிப்படைத் தத்துவங்களில் ஒன்றான என்கப்சுலேஷன் (Encapsulation) பற்றிப் பார்ப்போம்.

ஒரு கிளாஸிலுள்ள பண்புகளும், ஃபங்ஷன்களும் கிளாஸ் என்ற பண்பு மூலம் ஒரே பொருளாக மூடப்பட்டு, பாதுகாப்பாக வைக்கப் படுகின்றது. இதுவே என்கப்சுலேஷன் என அழைக்கப்படுகின்றது. அதாவது, டேற்றாவையும், அவற்றைக் கையாளும் ஃபங்ஷன்களையும் ஒன்று சேர்த்து ஒரு கப்சூல் (Capsule) இல் மூடி வைப்பது போன்று கிளாஸினுள் ஒன்று சேர்த்து மூடி வைக்கப்படுவதே என்கப்சுலேஷன் என அழைக்கப்படுகின்றது.

பொதுவாக டேற்றா, private அல்லது protected என்ற பிரிவுக்குள் வரையறுக்கப்படுகின்றன. இதனால்தான் ஒப்ஜெக்ற்றுக்குரிய டேற்றாவை ஃபங்ஷனால் மட்டுமே கையாளப்படுகின்றன.

டேற்றா பாதுகாக்கப்பட்டு (அதாவது, டேற்றா மறைக்கப்பட்டு - Data Hiding) பிற ஒப்ஜெக்ற்றிலுள்ள ஃபங்ஷன்களினால் அணுக முடியாது தடுக்கும் முறையே என்கப்சுலேஷன் என அழைக்கப்படுகின்றது.

7.3 இன்ஹெரிற்ரன்ஸ் (Inheritance)

புதிய கிளாஸ் ஒன்றை, ஏற்கனவே வரையறுக்கப்பட்ட கிளாஸிலிருந்து உருவாக்கப்படுவதையே இன்ஹெரிற்ரன்ஸ் என அழைக்கப்படுகின்றது. இப்புதிய கிளாஸானது, ஏற்கனவே வரையறுக்கப்பட்ட கிளாஸில் காணப்படும் பண்புகள் (Attributes), ∴பங்ஷன்கள் (Functions) போன்றவற்றுடன், மேலும் பல புதிய பண்புகளையும், ∴பங்ஷன்களையும் கொண்டிருக்கும்.

ஒரு கிளாஸ் அல்லது பல கிளாஸ்களை அடிப்படையாகக் கொண்டு ஒரு புதிய கிளாஸினை உருவாக்க முடியும். இங்கு அடிப்படைக் கிளாஸினை பேஸ் கிளாஸ் (Base Class) எனவும், உருவாக்கப்பட்ட புதிய கிளாஸினை, டிரைவ்ட் கிளாஸ் (Derived Class) எனவும் அழைக்கப்படுகின்றது.

இந்த பேஸ் கிளாஸானது சுப்பர் கிளாஸ் (Super Class) அல்லது பேரன்ற் கிளாஸ் (Parent Class) எனவும், டிரைவ்ட் கிளாஸானது சப் கிளாஸ் (Sub Class) அல்லது சைல்ட் கிளாஸ் (Child Class) எனவும் அழைக்கப்படுகின்றது.

ஒப்ஜெக்ற் ஓரியன்ரட் புறோகிராமில் காணப்படும் மிக முக்கியமான சிறப்பியல்புகளில் ஒன்று, மீள்பயன்பாடு (Reusability) ஆகும். இந்த இன்ஹெரிற்ரன்ஸ் என்பது மீள்பயன்பாட்டிற்குப் பயன்படுத்தப்படும் ஒரு செயற்பாடாகும்.

சி++ மொழியில் நான்கு வகையான இன்ஹெரிற்ரன்ஸ் முறைகள் பயன்படுத்தப்படுகின்றன.

அவையாவன,

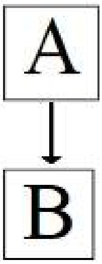
- சிங்கிள் இன்ஹெரிற்ரன்ஸ் (Single Inheritance)
- மல்ரி - லெவல் இன்ஹெரிற்ரன்ஸ் (Multi-level Inheritance)
- மல்ரிபில் இன்ஹெரிற்ரன்ஸ் (Multiple Inheritance)
- ஹைரார்சிகல் இன்ஹெரிற்ரன்ஸ் (Hierarchical Inheritance)

❖ சிங்கிள் இன்ஹெரிற்ரன்ஸ் (Single Inheritance)

ஒரே ஒரு பேஸ் கிளாஸிலிருந்து ஒரு டிரைவ்ட் கிளாஸ் மட்டும் உருவாக்கப்படுவதையே சிங்கிள் இன்ஹெரிற்ரன்ஸ் என அழைக்கப்படுகின்றது. அதாவது, ஒரு பேஸ் கிளாஸில் காணப்படும் அனைத்து பண்புகளும், ∴பங்ஷன்களும் டிரைவ்ட் கிளாஸிற்குக் கடத்தப்படுவதாகும்.

சிங்கிள் இன்ஹெரிற்ரன்ஸ் என்றால், ஒரே ஒரு சுப்பர் கிளாஸிலிருந்து மட்டுமே பண்புகளையும், ∴பங்ஷன்களையும் சப் கிளாஸிற்குக் கொடுக்க முடியும். அத்துடன், தமக்குரிய மேலும் பல புதிய பண்பு களையும், ∴பங்ஷன்களையும் இந்த சப் கிளாஸ் கொண்டிருக்க முடியும்.

அடுத்து, சிங்கிள் இன்ஹெரிற்ரன்ஸிற்குரிய கிளாஸ் வரிப்படத்தைப் (Class Diagram) பார்ப்போம்.



சிங்கிள் இன்ஹெரிற்ரன்ஸிற்குரிய ஒரு உதாரணத்தை சி++ மொழிப் புறோகிராம் ரீதியாகப் பார்ப்போம்.

Person என்ற பெயரில் ஒரு பேஸ் கிளாஸினை உருவாக்கி, அதனை அடிப்படையாகக் கொண்டு Student என்ற டிரைவ்ட் கிளாஸினை உருவாக்கிக் கையாளும் முறையைத் தெளிவாக ஒரு உதாரணப் புறோகிராம் மூலம் பார்ப்போம்.

```
#include <iostream.h>
class  Person
{
   protected:
      char *name;
      int age;
      char *icnumber;
};
class Student:public Person
{
      char *indexnum;
      char *results;
   public:
      void readData();
      void display();
};
void Student::readData()
{
```

```
    cout <<"Enter the name : ";
    cin >>name;
    cout <<"Enter the age : ";
    cin >>age;
    cout <<"Enter the IC number : ";
    cin >>icnumber;
    cout <<"Enter the index number : ";
    cin >>indexnum;
    cout <<"Enter the results : ";
    cin >>results;
}
void Student::display()
{
    cout << "Student's Personal Details"<<endl;
    cout << "Name :" << name<<endl;
    cout << "Age  :" << age <<endl;
    cout << "Ic number :" << icnumber <<endl;
    cout << "Index number :" << indexnum <<endl;
    cout << "Results :" << results <<endl;
}
 void main()
 {
    Student *s = new Student;
    s->readData();
    s->display();
 }
```

இங்கு Person என்ற ஒரு பேஸ் கிளாஸிலிருந்து Student என்ற ஒரேயொரு டிரைவ்ட் கிளாஸ் உருவாக்கப்பட்டிருப்பதால், இதனை சிங்கிள் இன்ஹெரிற்ரன்ஸ் என அழைக்க முடியும்.

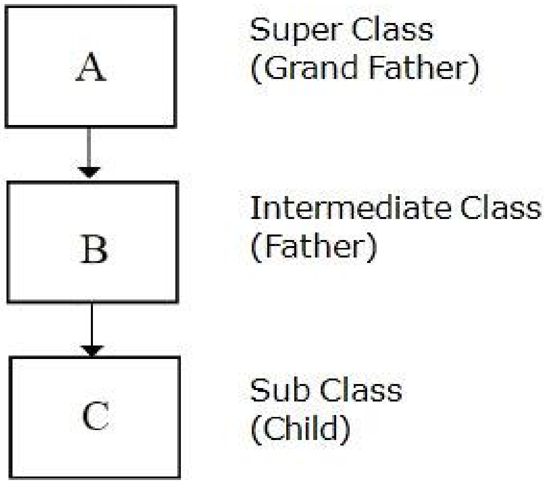
சுப்பர் கிளாஸில் (Super Class) வரையறுக்கப்பட்ட பண்புகள், பண்புகளைக் கையாளும் ∴பங்ஷன்கள் போன்றவற்றை சப் கிளாஸில் (Sub Class) பயன்படுத்த முடியும். இதனால் புறோகிராமிற்குரிய சோர்ஸ் கோட் (Source Code) குறைவாகக் காணப்படும்.

இந்த உதாரணத்தில், Person என்ற பேஸ் கிளாஸில் காணப்படும் பண்புகளான name, age, icnumber போன்றனவும், இந்தப் பண்புகளைக் கையாளும் ∴பங்ஷன்களையும் சப் கிளாஸான Student இல் பயன்படுத்த முடியும்.

மல்ரி - லெவல் இன்ஹெரிற்ரன்ஸ் (Multi - level Inheritance)

பல நிலைகளில் கிளாஸினை உருவாக்கும் முறையே மல்ரி - லெவல் இன்ஹெரிற்ரன்ஸ் என அழைக்கப்படுகின்றது. அதாவது, 1வது கிளாஸிலிருந்து 2வது கிளாஸையும், 2வது கிளாஸிலிருந்து 3வது கிளாஸையும் உருவாக்குவதாகும்.

மல்ரி - லெவல் இன்ஹெரிற்ரன்ஸிற்குரிய கிளாஸ் வரிப்படத்தைப் (Class Diagram) பார்ப்போம்.



மேலேயுள்ள வரிபடத்தில், A என்ற கிளாஸானது சுப்பர் கிளாஸ் (Super Class) எனவும், B என்ற கிளாஸானது இடைநிலைக் கிளாஸ் (Intermediate Class) எனவும், C என்ற கிளாஸானது சைல்ட் கிளாஸ் (Child Class) எனவும் அழைக்கப்படுகின்றது.

A என்ற கிளாஸில் காணப்படும் பண்புகள், ∴பங்ஷன்கள் அனைத்தும் கிளாஸ் B இற்குக் கடத்தப்படும். இவ்வாறு B என்ற கிளாஸில் காணப்படும் பண்புகள், ∴பங்ஷன்கள் அனைத்தும் கிளாஸ் C இற்குக் கடத்தப்படும். அதாவது, கிளாஸ் A, B போன்றனவற்றில் காணப்படும் அனைத்து பண்புகளும், ∴பங்ஷன்களும் கிளாஸ் C இற்குக் கடத்தப்படும்.

அடுத்து, இதற்குரிய சி++ மொழிக் கட்டளைகளைப் பார்ப்போம்.

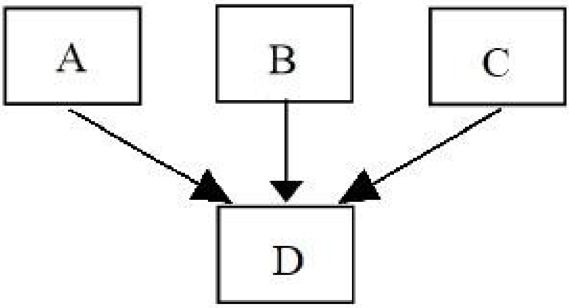
```
class A
{
   ..........;
   ..........;
};
```

```
class B:public A
{
   ........;
   ........;
};
class C:public B
{
   ........;
   ........;
};
```

மல்ரிபில் இன்ஹெரிற்ரன்ஸ் (Multiple Inheritance)

ஒன்றுக்கு மேற்பட்ட பேஸ் கிளாஸ்களிலிருந்து, ஒரு டிரைவ்ட் கிளாஸினை உருவாக்கும் முறையே மல்ரிபில் இன்ஹெரிற்ரன்ஸ் என அழைக்கப்படுகின்றது. எனவே, இதில் ஒன்றிற்கு மேற்பட்ட பேஸ் கிளாஸில் உள்ள பண்புகள் (Attributes), ∴பங்ஷன்கள் (Functions) டிரைவ்ட் கிளாஸிற்குக் கடத்தப்படுகின்றன. மற்றும் பல புதிய பண்பு களும், ∴பங்ஷன்களும் இந்த டிரைவ்ட் கிளாஸில் காணப்படும்.

மல்ரி - லெவல் இன்ஹெரிற்ரன்ஸிற்குரிய கிளாஸ் வரிப்படத்தை அடுத்துப் பார்ப்போம்.



மேலேயுள்ள வரிப்படத்தில் A, B, C என்பன பேஸ் கிளாஸ்கள் எனவும், D டிரைவ்ட் கிளாஸ் எனவும் அழைக்கப்படுகின்றன.

A, B, C ஆகிய கிளாஸ்களில் காணப்படுகின்ற பண்புகள், ∴பங்ஷன்கள் D என்ற கிளாஸிற்குக் கடத்தப்படுகின்றன. மற்றும் D என்ற கிளாஸ் பண்புகளையும், ∴பங்ஷன்களையும் தன்னகத்தே கொண்டிருக்கும்.

இனி இதற்குரிய சி++ மொழிக் கட்டளைகளைப் பார்ப்போம். முதலில் பேஸ் கிளாஸ்கள் A, B, C போன்றவற்றுக்குரிய கட்டளைகளை எழுத வேண்டும்.

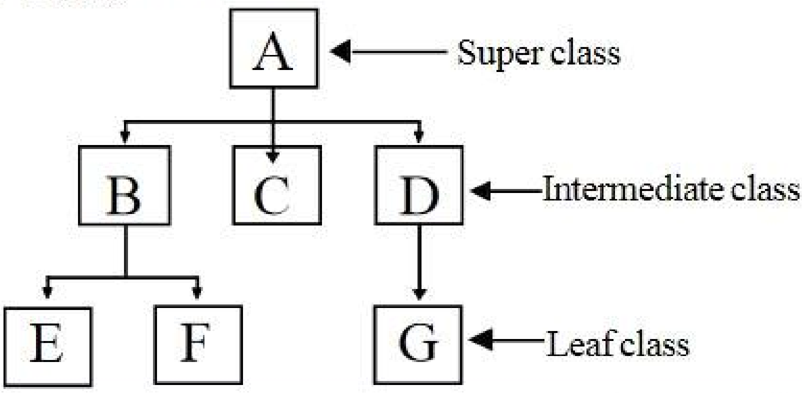
```
class A
{
    .............;
    .............;
};
class B
{
    .............;
    .............;
};
class C
{
    .............;
    .............;
};
```

பின்னர், கிளாஸ் D இற்குரிய கட்டளைகளை எழுத வேண்டும்.

```
class D:public A, public B, public C
{
    .............;
    .............;
};
```

ஹைரார்சிகல் இன்ஹெரிற்ரன்ஸ் (Hierarchical Inheritance)

பல மல்ரிலெவல் இன்ஹெரிற்ரன்ஸ், பல சிங்கிள் இன்ஹெரிற்ரன்ஸ் போன்றவற்றின் சேர்க்கையே ஹைரார்சிகல் இன்ஹெரிற்ரன்ஸ் என அழைக்கப்படுகின்றது. அதாவது, இந்த இன்ஹெரிற்ரன்ஸ் முறையானது ஒரு பெரிய ஒன்றையொன்று சார்ந்திருக்கும் கட்டமைப்புடைய புறோகிராம் களை உருவாக்கப் பயன்படுத்தப்படுகின்றது. இதற்கான கிளாஸ் வரிப் படத்தை அடுத்துப் பார்ப்போம்.



மேலேயுள்ள வரிப்படத்தில், A என்ற கிளாஸானது சுப்பர் கிளாஸ் (Super class) எனவும், B, C, D போன்ற கிளாஸ்கள் இடைநிலைக் கிளாஸ் (Intermediate class) எனவும், E, F, G போன்ற கிளாஸ்கள் லீ∴வ் கிளாஸ் (Leaf class) எனவும் அழைக்கப்படுகின்றது.

A, B போன்ற கிளாஸ்களில் காணப்படும் பண்புகள், ∴பங்ஷன்கள் E, F போன்ற கிளாஸ்களிற்குக் கடத்தப்படுகின்றன. அத்துடன் E, F போன்ற கிளாஸ்கள் தமக்குரிய பண்புகளையும், ∴பங்ஷன்களையும் தன்னகத்தே கொண்டிருக்கும்.

அடுத்து, இதற்குரிய சி++ மொழிக் கட்டளைகளைப் பார்ப்போம். முதலில், A என்ற கிளாஸினை வரையறுக்க வேண்டும். பின்னர் B, C, D போன்ற கிளாஸ்களுக்குரிய கட்டளைகளை எழுத வேண்டும். இறுதியாக E, F, G போன்ற கிளாஸ்களுக்குரிய கட்டளைகளை எழுத வேண்டும்.

```
class A
{
    ............;
    ............;
};
class B: public A
{
    ............;
    ............;
};
class C: public A
{
    ............;
    ............;
};
class D: public A
{
    ............;
    ............;
};
class E: public B
{
    ............;
```

```
    .............;
};
class F: public B
{
    .............;
    .............;
};
class G: public D
{
    .............;
    .............;
};
```

சி++ மொழியில் நான்கு வகையான இன்ஹெரிற்ரன்ஸ் முறைகள் பயன்படுத்தப்படுகின்றன எனத் தெளிவாகப் பார்த்தோம். ஆனால், அவற்றுக்குரிய உதாரணப் புறோகிராம்களைப் பார்க்கவில்லை. எனவே, அவற்றுக்குரிய உதாரணங்களை அடுத்துப் பார்ப்போம்.

சிங்கிள் இன்ஹெரிற்ரன்ஸிற்குரிய உதாரணப் புறோகிராமினை முதலில் பார்ப்போம்.

```
#include <iostream.h>
class A
{
  public:
  A()
  {
     cout<<"Constructor of the base class A executed first \n";
  }
};
class B : public A
{
  public:
  B()
  {
    cout<<"Constructor of the derived class B executed second";
  }
};
```

```
void main()
{
      B obj;
}
```

இந்தப் புறோகிராமினைச் செயற்படுத்திப் பார்த்தால், முதலில் வெளியீடாக A என்ற கிளாஸிற்குரிய கொன்ஸ்ரக்ரர் செயற்பட்டு "Constructor of the base class A executed first" என வெளியீட்டைக் காண்பிக்கும். பின்னர், B என்ற கிளாஸிற்குரிய கொன்ஸ்ரக்ரர் செயற்பட்டு "Constructor of the derived class B executed second" என வெளியீட்டைக் காண்பிக்கும்.

இந்தப் புறோகிராமில், A என்ற பேஸ் கிளாஸ் (Base Class) இல் ஒரு பப்பிளிக் (public) கொன்ஸ்ரக்ரர் வரையறுக்கப்பட்டுள்ளது. பின்னர், A என்ற கிளாஸிலிருந்து B என்ற டிரைவ்ட் கிளாஸ் (Derived Class) உருவாக்கப்பட்டு, அதிலும் ஒரு பப்பிளிக் கொன்ஸ்ரக்ரர் வரையறுக்கப் பட்டுள்ளது.

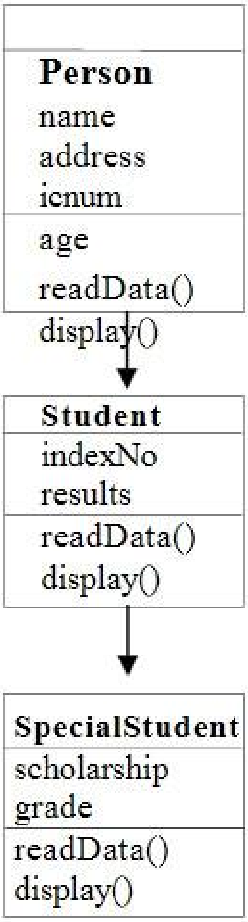
மெயின் ∴பங்ஷனிலுள்ள கட்டளையின் மூலம், B என்ற கிளாஸி லிருந்து obj என்ற ஒப்ஜெக்ற் உருவாக்கப்பட்டுள்ளது. A என்ற கிளாஸிலிருந்து B என்ற கிளாஸ் உருவாக்கப்பட்டதால், A என்ற கிளாஸிற்குரிய பண்புகளும், ∴பங்ஷன்களும் இந்த obj என்ற ஒப்ஜெக்ற்றில் காணப்படும். இந்த இரு கிளாஸ்களிலும் கொன்ஸ்ரக்ரர் கள் வரையறுக்கப்பட்டிருப்பதால், கிளாஸிலிருந்து ஒப்ஜெக்ற்களை உருவாக்கும் பொழுதே கொன்ஸ்ரக்ரர் செயற்படும். எனவேதான், இவ்விரு கிளாஸிற்கும் உரிய கொன்ஸ்ரக்ரர்கள் முதலில் செயற்பட்டன.

அடுத்து, மல்ரி - லெவல் இன்ஹெரிற்ரன்ஸ் (Multi - Level Inheritance) இற்குரிய உதாரணத்தினைப் பார்ப்போம்.

பல நிலைகளில் கிளாஸ்கள் உருவாக்கப்படுவதே மல்ரி லெவல் இன்ஹெரிற்ரன்ஸ் என அழைக்கப்படுகின்றது. இதற்குரிய தெளிவான விளக்கத்தினைக் கீழேயுள்ள உதாரணம் மூலம் பார்ப்போம்.

உதாரணமாக, முதலில் Person என்ற பேஸ் கிளாஸ் உருவாக்கப் பட்டு, பின்னர் Student என்ற டிரைவ்ட் கிளாஸ் உருவாக்கப்படுகிறது. மேலும் இந்த Student என்ற டிரைவ்ட் கிளாஸிலிருந்து SpecialStudent என்ற புதிய டிரைவ்ட் கிளாஸ் உருவாக்கப்படுகிறது.

முதலில் இதற்குரிய கிளாஸ் வரிப்படத்தைக் (Class Diagram) கீழேயுள்ளவாறு வரைய வேண்டும்.



அடுத்து, இந்தக் கிளாஸ் வரிப்படத்துக்குரிய C++ மொழிப் புறோகிராமினைப் பார்ப்போம்.

```
#include <iostream.h>
class Person
{
    private :
        char *name;
        char *address;
        char *icnum;
        int age;
    public :
        void readData();
        void display() ;
};
```

```
void Person::readData()
{
    cout << "Enter name :";
    cin>>name;
    cout << "Enter address:";
    cin>>address;
    cout<< "Enter IC number:";
    cin>>icnum;
    cout << "Enter age :";
    cin>>age;
}
void Person::display()
{
    cout<< "Name :" <<name << endl;
    cout << "Address :" << address <<endl;
    cout<< "Ic number :" <<icnum <<endl;
    cout<< "Age :" <<age << endl;
}
class Student:public Person
{
    private:
        char *indexNo;
        char *results;
    public:
        void readData();
        void display();
};
void Student::readData()
{
    Person::readData(); //To call the base class function readData()
    cout << "Enter index number :";
    cin>> indexNo;
    cout << "Enter results :" ;
    cin >>results;
}
void Student::display()
{
    Person:: display(); // To call the base class function display()
    cout << "Index No :" << indexNo << endl;
    cout << "Results :" << results <<endl;
}
```

```
class SpecialStudent : public Student
{
   private:
      char *scholarship;
      char *grade;
   public :
      void readData();
      void display() ;
};
void SpecialStudent::readData()
{
   Student::readData(); //Tocall the derived class function readData()
   cout << "Enter the shcolaship category ";
   cin >> scholarship;
   cout << "Enter grade :";
   cin >> grade ;
}
void SpecialStudent::display()
{
   Student::display(); // To call the derived class function display()
   cout<<"Scholarship categories :"<<scholarship<<endl;
   cout << "Grade"<< grade << endl;
}
void main()
{
   SpecialStudent s;
   s.readData() ;
   s. display();
}
```

மேலேயுள்ள புறோகிராமில், SpecialStudent என்ற கிளாஸானது Student என்ற கிளாஸிலிருந்தும், Student என்ற கிளாஸானது Person என்ற கிளாஸிலிருந்தும் உருவாக்கப்பட்டுள்ளது. எனவே, SpecialStudent என்ற டிரைவ்ட் கிளாஸில் Person, Student போன்ற கிளாஸிலுள்ள பண்புகள் (Attributes), ∴பங்ஷன்கள் (Functions) போன்றன காணப்படும். எனினும் Person, Student போன்ற கிளாஸ்களிலுள்ள பண்புகளை SpecialStudent இல் பயன்படுத்த முடியாது. ஏனெனில் Person, Student போன்ற கிளாஸ்களில் உள்ள பண்புகள் யாவும் பிரைவேற் (Private) என்ற பகுதிக்குள் வரையறுக்கப்பட்டுள்ளன. எனினும், பண்புகளை ∴பங்ஷின்கள் மூலம் SpecialStudent என்ற கிளாஸில் பயன்படுத்த முடியும்.

ஒரு கிளாஸிலிருந்து இன்னுமொரு கிளாஸினை உருவாக்கும் போது, அதனை public, private மற்றும் protected ஆக வரையறுக்க முடியும். உதாரணமாக, Student என்ற கிளாஸிற்கு Person என்ற கிளாஸானது private ஆக அமைய வேண்டுமாயின்,

class Student:private Person என கட்டளையை எழுத வேண்டும். இவ்வாறு private ஆக வரையறுத்தால், SpecialStudent என்ற கிளாஸில் Person என்ற கிளாஸிற்குரிய பண்புகள், ஃபங்ஷன்கள் போன்றவற்றைப் பயன்படுத்த முடியாது.

இந்த உதாரணத்தில், பல நிலைகளில் கிளாஸ்கள் வரையறுக்கப் பட்டிருப்பதால், பொதுவான இயல்புகள் மீண்டும், மீண்டும் வரையறுக்கப்படத் தேவையில்லை. எனவே, புறோகிராம்களின் வரிகள் (Codes) குறைவாகக் காணப்படும்.

அடுத்து, மல்ரிபில் இன்ஹெரிற்ரன்ஸிற்குரிய உதாரணத்தைப் பார்ப்போம்.

Father, Mother போன்ற இரு பேஸ் கிளாஸ்களிலிருந்து Son என்ற டிரைவ்ட் கிளாஸினை எவ்வாறு உருவாக்க முடியும் என்பதைப் பார்ப்போம்.

```
#include <iostream.h>
class Father
{
   public:
      Father()
      {
         cout<< "Constructor of the Father class \n";
      }
};
class Mother
{
   public:
      Mother()
      {
         cout<<"Constructor of the Mother class \n";
      }
};
class Son:public Father, public Mother
{
   public:
      Son()
      {
```

```
            cout << "Constructor of the Son class \n";
        }
};
void main()
{
    Son s;
}
```

இந்த புறோகிராமைச் செயற்படுத்திப் பார்த்தால்,

```
    Constructor of the Father class
    Constructor of the Mother class
    Constructor of the Son class
```

போன்ற வெளியீட்டைத் திரையில் காட்டும்.

ஹைரார்சிகல் இன்ஹெரிற்ரன்ஸிற்குரிய உதாரணப் புறோகிராமினை அடுத்துப் பார்ப்போம்.

பல்கலைக்கழகத்தில் உள்ளவர்களை வகைப்படுத்தும் விதமான தீர்வை, சி++ மொழிப் புறோகிராம் மூலம் எவ்வாறு எழுத முடியும் என்பதைப் பார்ப்போம்.

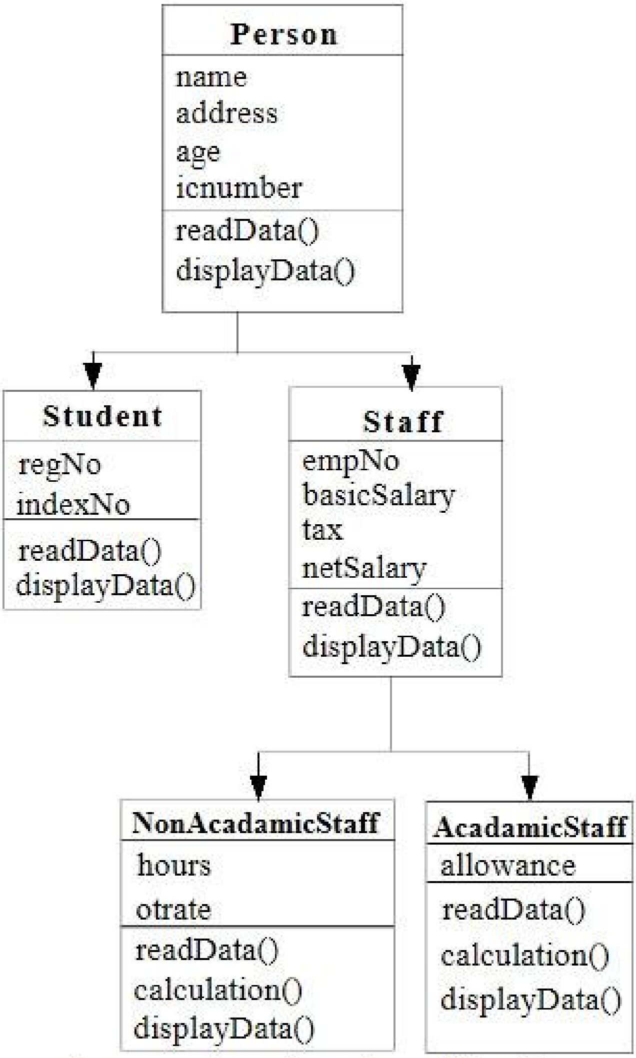
பல்கலைக்கழகத்தில் உள்ளவர்களை முதலில் Person என்ற கிளாஸாக வரையறுக்க வேண்டும். இந்த Person என்ற கிளாஸிற்குரிய பண்புகள், ∴பங்ஷன்கள் குறிப்பிடப்பட வேண்டும். உதாரணமாக, Person என்ற கிளாஸிற்கு பெயர், விலாசம், வயது, அடையாள அட்டை இலக்கம் போன்ற பண்புகளும், இந்த பண்புகளை உள்ளீடு செய்வதற்குரிய readData() என்ற ∴பங்ஷனும், பண்புகளை வெளியீடாகக் காண்பிப்பதற்குரிய displayData() என்ற ∴பங்ஷனும் வரையறுக்கப்பட வேண்டும். பின்னர், Person என்ற கிளாஸினை Student, Staff போன்ற இரு கிளாஸாகப் பிரிக்க முடியும். Student, Staff போன்ற கிளாஸ்களுக்குரிய பொதுவான பண்புகளும், ∴பங்ஷன் களும் Person என்ற பொதுவான கிளாஸில் வரையறுக்கப்படுகின்றன. எனவே Student, Staff போன்ற கிளாஸ்களில் அவைக்குரிய மேலதிக பண்புகளையும், ∴பங்ஷன்களையும் மட்டுமே வரையறுக்க வேண்டும்.

உதாரணமாக, Person என்ற பேஸ் கிளாஸ் (Base Class) இல் காணப்படுகின்ற பண்புகளும், ∴பங்ஷன்களும் எவ்வாறு Student என்ற சப் கிளாஸிற்குக் கடத்தப்படுகின்றன என்பதனைப் பார்ப்போம்.

```
class Student:public Person
{
    ..............;
    ..............;
};
```

Person என்ற பேஸ் கிளாஸில் public, protected ஆக வரையறுக்கப்பட்ட பண்புகளும் (Attributes), ∴பங்ஷன்களும் (Functions) மட்டுமே Student என்ற டிரைவ்ட் கிளாஸ் (Derived Class) இல் பயன்படுத்த முடியும். Person என்ற பேஸ் கிளாஸில், private ஆக வரையறுக்கப் பட்ட பண்புகளையும், ∴பங்ஷன்களையும் Student என்ற டிரைவ்ட் கிளாஸில் பயன்படுத்த முடியாது. பின்னர், Staff என்ற கிளாஸானது AcademicStaff, NonAcademicStaff என இரண்டு கிளாஸ்களாகப் பிரிக்கப்படுகின்றது.

இந்த உதாரணத்திற்குரிய கிளாஸ் வரிப்படத்தினை முதலில் பார்ப்போம்.



இவ்வாறு மரத்தை ஒத்த கிளாஸ் வரிப்படமே ஹைரார்சிகல் இன்ஹெரிற்ரன்ஸ் என அழைக்கப்படுகின்றது.

அடுத்து, இந்த கிளாஸ் வரிப்படத்துக்குரிய புறோகிராமினைப் பார்ப்போம்.

பேஸ் கிளாஸான Person என்ற கிளாஸிற்குரிய புறோகிராமினை முதலில் எழுத வேண்டும்.

```
#include <iostream.h>
// First create the base class Person
class Person
{
    private:
        char name[40];
        char address[50];
        int age;
        char icnum[12];
    public:
        void readData();
        void displayData();
};
void Person::readData()
{
    cout<<"Enter the name : ";
    cin>>name;
    cout<<"Enter the address : ";
    cin>>address;
    cout<<"Enter the age : ";
    cin>>age;
    cout<<"Enter the ic number : ";
    cin>>icnum;
}
void Person::displayData()
{
    cout<<"Name : "<<name<<endl;
    cout<<"Address : "<<address<<endl;
    cout<<"Age : "<<age<<endl;
    cout<<"IC Number : "<<icnum<<endl;
}
```

இந்த Person என்ற பேஸ் கிளாஸில், private என்ற பகுதிக்குள் name, address, age, icnum போன்ற பண்புகள் வரையறுக்கப்பட்டுள்ளன.

மேலும் readData(), displayData() போன்ற இரு ∴பங்ஷின்கள் public என்ற பகுதிக்குள் வரையறுக்கப்பட்டுள்ளன.

அடுத்து, Student என்ற டிரைவ்ட் கிளாஸினை Person என்ற பேஸ் கிளாஸிலிருந்து உருவாக்குவதற்குரிய கட்டளைகளை எழுத வேண்டும்.

```
// To create the derived class Student from the Person class
class Student : public Person
{
    private:
        char regNo[10];
        char indexNo[50];
    public:
        void readData();
        void displayData();
};
void Student::readData()
{
    Person::readData();
    cout<<"Enter Reg. No : ";
    cin>>regNo;
    cout<<"Enter Index No : ";
    cin>>indexNo;
}
void Student::displayData()
{
    Person::displayData();
    cout<<"Registration No:"<<regNo<<endl;
    cout<<"Index No: "<<indexNo<<endl;
}
```

இந்த Student என்ற டிரைவ்ட் கிளாஸில், Person இல் வரையறுக்கப் பட்ட பண்புகள் மற்றும் இப்பண்புகளை உள்ளீடு, வெளியீடு செய்வதற் குரிய கட்டளைகளை எழுதத் தேவையில்லை. எனவே, Student என்ற டிரைவ்ட் கிளாஸில், Person என்ற கிளாஸில் இல்லாத பண்புகளையும், செயற்பாடுகளையும் மட்டும் வரையறுத்தால் போதுமானதாகும்.

அடுத்து, Staff என்ற டிரைவ்ட் கிளாஸினை Person என்ற பேஸ் கிளாஸிலிருந்து உருவாக்குவதற்குரிய கட்டளைகளை எழுத வேண்டும்.

```
// Create derived class Staff from Person class
class Staff : public Person
{
    private:
        char empNo[40];
    protected:
         float basicSal,netSalary,tax;
    public:
        void readData();
        void displayData();
};
void Staff::readData()
{
    Person::readData();
    cout<<"Enter the Employee number : ";
    cin>>empNo;
    cout<<"Enter the basic salary : ";
    cin>>basicSal;
}
void Staff::displayData()
{
    Person::displayData();
    cout<<"Employee No : "<<empNo<<endl;
    cout<<"Basic Salary : "<<basicSal <<endl;
    tax = basicSal*0.12;
    cout<<"Tax : "<<tax<<endl;
}
```

Student என்ற டிரைவ்ட் கிளாஸ், Person என்ற பேஸ் கிளாஸிலிருந்து உருவாக்கப்பட்டது போல், Staff என்ற புதிய டிரைவ்ட் கிளாஸ், Person என்ற பேஸ் கிளாஸிலிருந்து உருவாக்கப்பட்டுள்ளது.

அடுத்து, AcademicStaff என்ற டிரைவ்ட் கிளாஸ் Staff என்ற கிளாஸிலிருந்து உருவாக்குவதற்குரிய கட்டளைகளை எழுத வேண்டும்.

```
//To create the derived class AcademicStaff from Staff class
class AcademicStaff:public Staff
{
    private:
        float allowance;
        void calculation();
    public:
        void readData();
        void displayData();
};
void AcademicStaff::readData()
{
    Staff::readData();
    cout<<"Enter Allowance : ";
    cin>>allowance;
}
void AcademicStaff::calculation()
{
    netSalary = basicSal+allowance - tax;
}
void AcademicStaff::displayData()
{
    calculation();
    Staff::displayData();
    cout<<"Employee net salary : "<<netSalary<<endl;
}
```

Staff என்ற கிளாஸிலிருந்து AcademicStaff என்ற புதிய கிளாஸ் உருவாக்கப்பட்டுள்ளது. இந்த AcademicStaff என்ற கிளாஸில், Person என்ற பேஸ் கிளாஸில் காணப்படும் பண்புகளும், ∴பங்ஷன் களும், மற்றும் Staff என்ற கிளாஸில் காணப்படும் பண்புகளும், ∴பங்ஷன்களும் காணப்படும். எனினும், private ஆக வரையறுக்கப்பட்ட பண்புகளை ஒருபோதும் AcademicStaff என்ற கிளாஸில் நேரடியாகப் பயன்படுத்த முடியாது.

அடுத்து, NonAcademicStaff என்ற டிரைவ்ட் கிளாஸினை Staff என்ற கிளாஸிலிருந்து உருவாக்குவதற்குரிய கட்டளைகளை எழுத வேண்டும்.

```
// Create derived class NonAcademicStaff  from Staff class
class NonAcademicStaff : public Staff
{
   private:
       float otRate;
       int hours;
       void calculation();
   public:
       void readData();
       void displayData();
};
void NonAcademicStaff::readData()
{
   Staff::readData();
   cout<<"Enter OT hours : ";
   cin>>hours;
}
void NonAcademicStaff::calculation()
{
    netSalary = basicSal + otRate*hours;
}
void NonAcademicStaff::displayData()
{
   calculation();
   Staff::displayData();
   cout<<"Employee net salary : "<<netSalary<<endl;
}
```

இறுதியாக மெயின் ∴பங்ஷனில், Student என்ற கிளாஸிலிருந்து p என்ற பொயின்ரர் ஒப்ஜெக்ற்றினை உருவாக்கி, இவற்றுக்குரிய பண்புகளை உள்ளீடு, வெளியீடு செய்வதற்குரிய கட்டளைகள் எழுதப்பட வேண்டும்.

இவ்வாறு, NonAcademicStaff என்ற கிளாஸிலிருந்து q என்ற ஒப்ஜெக்ற் உருவாக்கப்பட்டு readData(), printData() போன்ற ∴பங்ஷன்கள் மூலம் பண்புகள் உள்ளீடு, வெளியீடு செய்யப் பயன்படுத்தப்பட்டுள்ளது.

```
void main()
{
    // Create object p from Student class
    Student *p = new Student();
    p->readData();
    p->displayData();
    cout<<"Thanks\n"<<endl;
    // Create object q from Non AcademicStaff class
    NonAcademicStaff  *q = new NonAcademicStaff();
    q->readData();
    q->displayData();
    cout<<"Thanks\n"<<endl;
}
```

சி++ மொழியில் நான்கு வகையான இன்ஹெரிற்ரன்ஸ் முறைகள் பயன்படுத்தப்படுகின்றன எனவும், அவற்றுக்குரிய உதாரணப் புறோகிராம்களையும் பார்த்தோம்.

OOP இன் அடிப்படைத் தத்துவங்கவங்களில் மிக முக்கிய பங்கை வகிக்கும் பொலிமோபிஸம் (Polymorphism) பற்றிய தெளிவான விளக்கங்களை அடுத்துப் பார்ப்போம்.

7.4 பொலிமோபிஸம் (Polymorphism)

ஒப்ஜெக்ற் ஓரியன்ரட் புறோகிராமிங் (OOP) இன் அடிப்படைத் தத்துவங்களில் மிக முக்கிய பங்கை வகிக்கும் பொலிமோபிஸம் (Polymorphism) பற்றிய விளக்கத்தினைப் பார்ப்போம்.

பொலிமோபிஸம் என்ற சொல், கிரேக்க மொழியிலிருந்து வந்த தாகும். இதன் கருத்து பல வடிவம் (Many Shape) ஆகும். சி++ மொழியில் பொலிமோபிஸம் என்றால், குறித்தொரு பெயருடைய ∴பங்ஷன் அல்லது குறித்தொரு ஒப்பறேற்றர் (Operator), புறோகிராம் ஒன்றில் ஒன்றுக்கு மேற்பட்ட செயற்பாடுகளைக் கொண்டிருப்பதாகும்.

பொதுவாக சி++ மொழியில் பொலிமோபிஸமானது, இரு வகையாகச் செயற்படுகின்றது.

1. கொம்பைல் செய்யும் நேரத்தில் தீர்மானிக்கும் பொலிமோபிஸம் (Compile Time/ Static/ Early Binding Polymorphism)

இவற்றுக்குரிய உதாரணங்கள்:

- ∴பங்ஷன் ஓவலோடிங் (Function Overloading)
- ஒப்பறேற்றர் ஓவலோடிங் (Operator Overloading)

2. புரோகிராம் செயற்படும் போது தீர்மானிக்கும் பொலிமோபிஸம் (Run Time/ Dynamic/ Late Binding Polymorphism)

இவற்றுக்குரிய உதாரணம் :

∴பங்ஷன் ஓவரைடிங் (Function Overriding) ஆகும். சி++ மொழியில் virtual என்ற கீவேட் (Keyword) இனை ∴பங்ஷனின் பெயருக்கு முன் எழுதுவதன் மூலம், புறோகிராம் செயற்பட்டுக் கொண்டிருக்கும் போது எந்த ∴பங்ஷன் செயற்பட வேண்டும் என்பது தீர்மானிக்கப் படுகின்றது.

கொம்பைல் ரைம் பொலிமோபிஸம் (Compile Time Polymorphism)

புறோகிராம் ஒன்றை கொம்பைல் செய்யும் போது, எந்த ∴பங்ஷன் அல்லது எந்த ஒப்பறேற்றர் தேவை என்பது தீர்மானிக்கப்பட்டு விடும். சி++ மொழியில், இந்த கொம்பைல் ரைம் பொலிமோபிஸமானது இரு வகையான செயற்பாடுகளுக்குப் பயன்படுத்தப்படுகின்றது.

அவையாவன,

1. ∴பங்ஷன் ஓவலோடிங்
2. ஒப்பறேற்றர் ஓவலோடிங்

ஃபங்ஷன் ஓவலோடிங் (Function Overloading)

ஒரு புறோகிராமில், ஒரே பெயரில் பல ∴பங்ஷன்களை உருவாக்க முடியும். இதுவே ∴பங்ஷன் ஓவலோடிங் என அழைக்கப்படுகின்றது. இந்த ∴பங்ஷன் ஓவலோடிங்கை இரண்டு வகையாகப் பயன்படுத்த முடியும்.

1. பராமீற்றர்களின் (Parameters) எண்ணிக்கையை வேறுபடுத்துவதன் மூலம் வெவ்வேறு செயற்பாடுகளைச் செயற்படுத்த முடியும். அதாவது, ஒரு குறித்த பெயர் உடைய ∴பங்ஷனில் உள்ளீடு செய்யும் பராமீற்றர் களின் எண்ணிக்கைக்கு ஏற்றவாறு, வெவ்வேறு செயற்பாடுகளைச் செயற்படுத்தும்.

2. பராமீற்றரின் விபர இனம் (Data Type) வேறுபட்டிருந்தால், வெவ்வேறு செயற்பாடுகளைச் செயற்படுத்த முடியும். அதாவது, ஒரு குறித்த பெயர் உடைய ∴பங்ஷனில் உள்ளீடு செய்யும் பராமீற்றர்களின் விபர இனத்திற்கு ஏற்றவாறு, வெவ்வேறு செயற்பாடுகளைச் செயற் படுத்தும்.

பராமீற்றர் என்றால் என்ன?

∴பங்ஷனிற்குள் அனுப்பப்படும் மாறிகளே பராமீற்றர்கள் என அழைக்கப்படுகின்றது.

உதாரணமாக,

```
int sum(int x, int y)
{
    ......;
    ......;
}
```

இந்த sum() என்ற ∴பங்ஷனில் x, y என்னும் இரண்டு பராமீற்றர்கள் பயன்படுத்தப்பட்டுள்ளன. இந்த பராமீற்றர்களினால் ஏற்படும் நன்மை யாதெனில், main() ∴பங்ஷனில் நாம் வரையறுத்த ∴பங்ஷனை வெவ்வேறு உள்ளீடுகளைக் கொடுத்து பல தடவைகள் பயன்படுத்த முடியும்.

∴பங்ஷன் ஓவலோடிங் (Function Overloading) இன் முதலாவது வகையான பராமீற்றர்களின் எண்ணிக்கையில் வேறுபட்டிருக்கலாம் என்பதற்குரிய உதாரணத்தினை முதலில் பார்ப்போம்.

ஒரு குறித்த பெயருடைய ∴பங்ஷனானது, பராமீற்றர்களின் எண்ணிக்கைக்கு ஏற்றவாறு வெவ்வேறு செயற்பாடுகளைச் செயற் படுத்தும். அதாவது, ஒரு குறித்த பெயருடைய ∴பங்ஷனானது main()

என்ற ∴பங்ஷனில், இரண்டு பராமீற்றர்களைக் கொடுக்கும் போது ஒரு விதமாகவும், மூன்று பராமீற்றர்களைக் கொடுக்கும் போது வேறு விதமாகவும் செயற்படும். இவ்வாறு பராமீற்றர்களின் எண்ணிக்கைக்கு ஏற்றவாறு வெவ்வேறு ∴பங்ஷன்கள் செயற்படும்.

நாம் கொடுக்கும் பராமீற்றருக்கு ஏற்றவாறு ஒரே பெயருடைய ∴பங்ஷன்கள் செயற்பட வேண்டுமாயின், இவற்றுக்குரிய தனித்தனி ∴பங்ஷன்கள் எமது புறோகிராமில் எழுதப்பட்டிருக்க வேண்டும். இந்த ∴பங்ஷன்கள் ஒரே கிளாஸிலோ அல்லது ஒரே புறோகிராமிலோ இருக்க முடியும்.

உதாரணமாக, area() என்ற குறித்தொரு ∴பங்ஷன் பெயரில், ஒரு பராமீற்றரினைக் கொடுக்கும் போது வட்டத்தின் பரப்பளவையும், இரண்டு பராமீற்றர்களைக் கொடுக்கும் போது செவ்வகத்தின் பரப்பளவையும் கணிப்பதற்குரிய ∴பங்ஷன்களைப் பார்ப்போம்.

```
#include <iostream.h>
double area(double  r)
{
     return 22/7.0*r*r ;
}
double area(double x, double y)
{
     return x*y;
}
void main()
{
     double a,b,c;
     cout<<"Enter the circle's radius: ";
     cin>>a;
     cout<<"Enter the length and width of the box: ";
     cin>>b>>c;
     cout<<"Area of the circle "<<area(a)<< endl;
     cout<<"Area of the box "<<area(b,c)<<endl;
}
```

மேலேயுள்ள புறோகிராமில், area() என்ற ∴பங்ஷனுக்கு ஒரு பராமீற்றரினைக் கொடுத்தால், வட்டத்தின் பரப்பளவையும், இரண்டு பராமீற்றர்களைக் கொடுத்தால், செவ்வகத்தின் பரப்பளவையும் கணித்துத் தரும். இங்கு இரு வெவ்வேறு கணிப்பீடுகளுக்கு ஒரே பெயரான area() பயன்படுத்தப்பட்டுள்ளது.

main() என்ற ∴பங்ஷனில், நாம் கொடுக்கும் பராமீற்றர்களின் எண்ணிக்கைக்கு ஏற்ப main() என்ற ∴பங்ஷனுக்கு மேல் வரையறுக்கப் பட்டுள்ள area() என்ற ∴பங்ஷன் செயற்படும்.

∴பங்ஷன் ஓவலோடிங் (Function Overloading) இன் இரண்டாவது வகையான பராமீற்றர்களின் விபர இனங்கள் வேறுபட்டிருக்கலாம் என்பதற்குரிய உதாரணத்தினை அடுத்துப் பார்ப்போம்.

ஒரு குறித்த பெயருடைய வெவ்வேறு ∴பங்ஷன்களானது, பராமீற்றர்களின் விபர இனங்கள் வேறுபட்டிருந்தால், வெவ்வேறு செயற்பாடுகளைச் செயற்படுத்தும்.

உதாரணமாக, abs() என்ற ∴பங்ஷனிற்கு மறை முழு எண் ஒன்றை உள்ளீடு செய்யும் போது, அதனை நேர் முழு எண்ணாக மாற்றி வெளியீட்டைக் காண்பிக்கும். இவ்வாறு மறை தசம எண் ஒன்றை உள்ளீடு செய்யும் போது, அதனை நேர் தசம எண்ணாக மாற்றி வெளியீட்டைக் காண்பிக்கும்.

இதற்குரிய சி++ மொழிப் புறோகிராமினை, ∴பங்ஷன் ஓவலோடிங் முறையினைப் பயன்படுத்தி எவ்வாறு எழுதலாம் எனப் பார்ப்போம்.

உதாரணம் (1)

```
#include <iostream.h>
int abs(int n)
{
    return (n<0) ? -n : n;
}
double abs(double n)
{
    return (n<0) ? -n : n;
}
void main()
{
    cout<<abs(-3)<<endl;
    cout<<abs(-4.96)<<endl;
    cout<<abs(-7.0)<<endl;
}
```

மேலேயுள்ள புறோகிராமினைச் செயற்படுத்தினால், வெளியீடாக 3, 4.96, 7.0 போன்றவை அடுத்தடுத்த வரிகளில் காண்பிக்கப்படும்.

abs() என்ற ∴பங்ஷனிற்கு உள்ளீடாக மறை முழு எண்ணினைக் கொடுத்தால், நேர் முழு எண்ணினை விடையாகக் காண்பிக்கும். உதாரணமாக: abs(-6) எனின், 6 என்ற விடையைக் காண்பிக்கும்.

abs() என்ற ∴பங்ஷனிற்கு உள்ளீடாக மறை தசம எண்ணினைக் கொடுத்தால், நேர் தசம எண்ணினை விடையாகக் காண்பிக்கும். உதாரணமாக: abs(-6.5) எனின், 6.5 என்ற விடையைக் காண்பிக்கும்.

உதாரணம் (2)

```
#include <iostream.h>
void swap (int &a, int &b)
{
    int t = a;
        a = b;
        b = t;
}
void swap(double &a, double &b)
{
    double t = a;
           a = b;
           b =t;
}
void swap(char &a, char &b)
{
    char t = a;
         a = b;
         b = t;
}
void main()
{
    int a = 5, b = 8;
    swap(a, b);
    cout << " a = "<< a<< endl;
    cout << " b = " << b<< endl;
    char x = 'j', y = 'k';
    swap(x, y);
    cout << " x = " << x << endl;
    cout << " y = " << y << endl;
    double p = 38.5, q = 75.8;
    swap(p, q);
    cout << " p = " << p<< endl;
    cout << " q = " << q << endl;
}
```

மேலேயுள்ள புறோகிராமினைச் செயற்படுத்தினால், முதலில் a, b மாறிகளில் காணப்படும் முழு எண்களான 5, 8 போன்றன தமக்குள் ஒத்து மாற்றப்படும். எனவே, தற்பொழுது a, b மாறிகளில் காணப்படும் மதிப்புக்கள் தமக்குள் இடம்மாற்றப்பட்டு காண்பிக்கப்படும். இவ்வாறு இரண்டு எழுத்துக்கள், இரண்டு தசம எண்கள் போன்றவற்றை swap() என்ற ∴பங்ஷன் மூலம் ஒத்துமாற்றம் (Swaping) செய்ய முடியும்.

கொன்ஸ்ரக்ரர் ஓவலோடிங்

ஒரு கிளாஸில் ஒரே பெயரில் (கிளாஸின் பெயரில்) ஒன்றுக்கு மேற்பட்ட கொன்ஸ்ரக்ரர்கள் காணப்படலாம். ஆனால், ஒவ்வொரு கொன்ஸ்ரக்ரர்களும் பராமீற்றர்களில் வேறுபட்டிருக்க வேண்டும். இவற்றையே கொன்ஸ்ரக்ரர் ஓவலோடிங் (Constructor Overloading) என அழைக்கப்படுகின்றது. கிளாஸிலிருந்து ஒப்ஜெக்ற்களை உருவாக்கும் போதே கொன்ஸ்ரக்ரர்களின் பராமீற்றருக்கு அமைய வெவ்வேறு கொன்ஸ்ரக்ரர்கள் செயற்படும்.

உதாரணமாக,

```
class Student
{
    char* name;
    int marks;
    public:
        Student() //constructor with out patameter
        {
            name ="";
            marks = 0;
        }
        Student(char* ch) //constructor with one patameter
        {
            name = ch;
            marks = 0;
        }
        Student(char* ch, int m) //constructor with 2 patameters
        {
            name = ch;
            marks = m;
        }
        void display()
        {
            cout<<"Name "<<name<<endl;
            cout<<"Marks "<<marks<<endl;
        }
};
```

```
void main()
{
    Student s1= new Student(); // called first constructor
    s1.display(); // Display student's name "" & marks 0
    Student s2= new Student("siva"); // called second constructor
    s2.display(); // Display student's name siva & marks 0
    Student s3= new Student("Kumar",89); //called third constructor
    s2.display(); //Display student's name kumar & marks 89
}
```

இந்தப் புறோகிராமில் Student என்ற கிளாஸில் மூன்று கொன்ஸ்ரக்ரர்கள் வரையறுக்கப்பட்டுள்ளன. இந்த மூன்று கொன்ஸ்ரக்ரர் களில் எந்த கொன்ஸ்ரக்ரர் செயற்படுவது என்பதை மெயின் ஃபங்ஷனில் ஒப்ஜெக்ற் உருவாக்கும் போது கொடுக்கப்படும் பராமீற்றர்களின் எண்ணிக்கைக்கு அமைய செயற்படும். இவ்வாறு ஒரே பெயரில் பல கொன்ஸ்ரக்ரர்களை ஒரு கிளாஸில் எழுதிப் பயன்படுத்தும் முறையே கொன்ஸ்ரக்ரர் ஓவலோடிங் என அழைக்கப்படுகின்றது.

கொன்ஸ்ரக்ரர் ஓவலோடிங் (Constructor Overloading) முறைக்குரிய மற்றுமோர் உதாரணத்தினைப் பார்ப்போம்.

```
class Circle
{
        double r;
    public:
        Circle()  //constructor with out patameter
        {
            r = 0.0;
        }
        Circle (double r)  //constructor with patameter
        {
            this.r = r;
        }
        void display()
        {
            cout<<"Area of this circle = "<<22.0/7*r*r<<endl;
        }
};
void main()
{
    Circle c1= new Circle(); // called first constructor
    Circle c2= new Circle(7.5); // called second constructor
    c2.display(); // display area of the circle
}
```

இந்தப் புறோகிராமில் Circle என்ற கிளாஸில் இரண்டு கொன்ஸ்ரக்ரர்கள் வரையறுக்கப்பட்டுள்ளன. ஒரு பராமீற்றரும் கொடுக்காது ஒப்ஜெக்ற்றினை உருவாக்கும் போது முதலாவது கொன்ஸ்ரக்ரரும், ஒரு பராமீற்றரினை மட்டும் கொடுத்து ஒப்ஜெக்ற்றினை உருவாக்கும் போது இரண்டாவது கொன்ஸ்ரக்ரரும் செயற்படும்.

ஒப்பறேற்றர் ஓவலோடிங்
(Operator Overloading)

ஒரு குறித்த ஒப்பறேற்றரானது, ஒன்றுக்கு மேற்பட்ட செயற்பாடுகளைச் செயற்படுத்துவதையே ஒப்பறேற்றர் ஓவலோடிங் (Operator Overloading) என அழைக்கப்படுகின்றது.

உதாரணமாக "+" என்ற ஒப்பறேற்றரினை, இரண்டு முழு எண்களைக் கூட்டுவதற்கு அல்லது இரண்டு தசம எண்களைக் கூட்டுவதற்குப் பயன்படுத்த முடியும்.

இந்த "+" என்ற ஒப்பறேற்றரினை இரண்டு சிக்கல் எண்களைக் கூட்டுவதற்கோ அல்லது இரண்டு திகதிகளைக் கூட்டுவதற்கோ அல்லது இரண்டு நேரங்களைக் கூட்டுவதற்கோ நேரடியாகப் பயன்படுத்த முடியாது. ஆயினும், சி++ மொழியிலுள்ள ஒப்பறேற்றர் ஓவலோடிங் முறையினைப் பயன்படுத்தி எழுதிய புறோகிராம்களின் மூலம் அதனைச் செய்ய முடியும்.

சி++ மொழியில் int, long, double, char, float, ... போன்ற பல விபர இனங்கள் (Data types) காணப்படுகின்றன. ஆனால் திகதி, நேரம், சிக்கல் எண்கள் போன்றவற்றுக்குரிய விபர இனங்கள் சி++ மொழியில் இல்லை. எனவே, நாம் இந்த விபர இனங்களைக் கொண்ட தரவுகளை புறோகிராமில் பயன்படுத்த வேண்டுமாயின், புதிய விபர இனங்களை உருவாக்க வேண்டும். சி++ மொழியில் இந்தப் புதிய விபர இனங்களை உருவாக்க கிளாஸ் (Class) அல்லது ஸ்ரக்சர் (Structure) பெரிதும் உதவிபுரிகின்றன.

உதாரணமாக, திகதிகளுக்குரிய புதிய விபர இனத்தினை உருவாக்க வேண்டுமாயின், date என்ற கிளாஸ் அல்லது ஸ்ரக்சர் வரையறுக்கப்பட வேண்டும். date என்ற கிளாஸில் அல்லது ஸ்ரக்சரில் மூன்று பண்புகளான நாள், மாதம், வருடம் போன்றவற்றை வரையறுக்க வேண்டும்.

கிளாஸ் மூலம் எவ்வாறு date என்ற புதிய விபர இனத்தை வரையறுப்பது எனப் பார்ப்போம்.

```
class date
{
     int day, month, year ;
};
```

அடுத்து, ஸ்ரக்சர் மூலம் எவ்வாறு date என்ற புதிய விபர இனத்தை வரையறுப்பது எனப் பார்ப்போம்.

```
struct date
{
     int day, month, year ;
};
```

சிக்கல் எண்களுக்குரிய புதிய விபர இனத்தை கிளாஸ் மூலம் வரையறுக்கும் போது,

```
class complex
{
     float real, imag;
};
```

என அமையும்.

இவ்வாறு எமக்குத் தேவையான புதிய விபர இனங்களை உருவாக்க, கிளாஸ் அல்லது ஸ்ரக்சரினைப் பயன்படுத்துவது மிகவும் இலகுவானதாகும். மேலேயுள்ளவாறு date, complex போன்ற விபர இனங்களை புறோகிராமில் வரையறுத்த பின்னர், தேவையான சந்தர்ப்பங் களில் இந்தப் புதிய விபர இனங்களுக்குரிய தரவுகளைப் பயன்படுத்த முடியும்.

உதாரணமாக,

```
void main()
{
    int  x1, x2;
    date  d1, d2;
    complex  c1, c2;
}
```

main() ஃபங்ஷனில் வரையறுக்கப்பட்ட x1, x2 போன்ற முழு எண் விபர இனத்துக்குரிய மாறிகளில், முழு எண்களை மட்டும் சேமிக்க முடியும். ஆனால், date என்ற விபர இனமாக வரையறுக்கப்பட்ட d1, d2 போன்ற மாறிகளில் (இங்கு மாறி எனக் கூறுவதை விட ஒப்ஜெக்ற் எனக் கூறுவது சிறந்ததாகும்) மூன்று முழு எண்களைச் சேமிக்க

முடியும். அதாவது, d1.day, d1.month, d1.year போன்றவற்றில் மூன்று மதிப்புக்களைச் சேமிக்க முடியும். இந்த மூன்று மதிப்புக்களும் d1 என்ற ஒப்ஜெக்ற்றில் அடங்கியிருக்கும் என்பது குறிப்பிடத்தக்க தாகும்.

இறுதியாக, ஒப்பறேற்றர் ஓவலோடிங் முறையைப் பயன்படுத்தி எவ்வாறு இரண்டு சிக்கல் எண்களைக் கூட்டுதல், கழித்தல், பெருக்கல், பிரித்தல் போன்ற செய்கைகளுக்குப் பயன்படுத்த முடியும் என்பதனை அடுத்து வரும் உதாரணம் மூலம் பார்ப்போம்.

இரண்டு சிக்கல் எண்களை "+" என்ற ஒப்பறேற்றரினைப் பயன்படுத்தி நேரடியாகக் கணினியில் கூட்ட முடியாது. எனவே, இரண்டு சிக்கல் எண்களைக் கூட்ட வேண்டுமாயின், நாம் புதிதாக ஒப்பறேற்றர் ஓவலோடிங் முறையினைப் பயன்படுத்தி புறோகிராமினை எழுத வேண்டும். இவ்வாறு இரண்டு சிக்கல் எண்களைக் கழிக்க, பெருக்க, பிரிக்க வேண்டுமாயின், சி++ மொழியிலுள்ள ஒப்பறேற்றர் ஓவலோடிங் முறையைப் பயன்படுத்தி எழுதப்பட்ட கட்டளைகளுக்குப் பின்னர், நேரடியாக ஒப்பறேற்றர்களான -, *, / போன்றவற்றைப் பயன்படுத்த முடியும்.

சி++ மொழியில் ஒப்பறேற்றர் ஓவலோடிங் முறையானது, இரு வகையாகப் பயன்படுத்தப்படுகின்றன.

1. பைனறி ஒப்பறேற்றர் ஓவலோடிங்
   (Binary Operator Overloading)
2. யூனறி ஒப்பறேற்றர் ஓவலோடிங்
   (Unary Operator Overloading)

பைனறி ஒப்பறேற்றர்: இரண்டு ஒப்பரன்ற்களும் (Operants), ஒரு ஒப்பறேற்றரும் (Operator) காணப்பட வேண்டும்.

உதாரணமாக: a+b, a*b, a/b, a>b, a == b, ... போன்றவற்றைக் குறிப்பிடலாம்.

யூனறி ஒப்பறேற்றர்: ஒரு ஒப்பரன்றும், ஒரு ஒப்பறேற்றரும் அல்லது ஒரு ஒப்பரன்றும், இரு ஒப்பறேற்றர்களும் காணப்படலாம்.

உதாரணமாக: a++, a--, x! போன்றவற்றைக் குறிப்பிடலாம்.

அடுத்து, ஒரு உதாரணப் புறோகிராம் மூலம் ஒப்பறேற்றர் ஓவலோடிங் (Operator Overloading) முறையினைத் தெளிவாகப் பார்ப்போம்.

ஒப்பறேற்றர் ஓவலோடிங் முறையினைப் பயன்படுத்தி, இரண்டு சிக்கல் எண்களைக் கூட்ட, கழிக்க, பெருக்க, பிரிக்க ஒப்பறேற்றர்களான +, -, *, / போன்றவற்றைப் பயன்படுத்த முடியும்.

சிக்கல் எண்களில், இரண்டு பகுதிகள் காணப்படுகின்றன. அவை யாவன உண்மைப் பகுதி (Real Part), கற்பனைப் பகுதி (Imaginary Part) போன்றவையாகும்.

உதாரணமாக 3+7i, 2-2i போன்றவற்றைக் குறிப்பிட முடியும்.

சி++ மொழியில் சிக்கல் எண்களைப் பயன்படுத்த வேண்டுமாயின், முதலில் Complex என்ற கிளாஸினை வரையறுக்க வேண்டும்.

Complex என்ற கிளாஸில், இரண்டு பண்புகளான real, imag என்பன வரையறுக்கப்பட வேண்டும். மேலும், இந்த கிளாஸில் இரண்டு பண்புகளான real, imag போன்றவற்றுக்கு ஆரம்பப் பெறுமானங்களாகப் பூச்சியத்தை கொடுப்பதற்கு Complex() என்ற கொன்ஸ்ரக்ரரும், சிக்கல் எண்களை உள்ளீடு செய்ய getData() என்ற ஃபங்ஷனும், சிக்கல் எண்களை வெளியீடாகக் காட்டுவதற்கு printData() என்ற ஃபங்ஷனும் வரையறுக்கப்பட வேண்டும். பின்னர் +, -, *, / போன்ற ஒப்பரேற்றர்களை ஓவலோடிங் செய்வதற்குரிய கட்டளைகளை எழுத வேண்டும்.

கீழேயுள்ள சி++ மொழிப் புறோகிராமானது, ஒப்பரேற்றர் ஓவலோடிங் முறைக்குரிய உதாரணமாகும்.

```
#include <iostream.h>
class Complex
{
 private:
      float real;
      float imag;
 public:
      Complex();
      void getData(float, float);
      void printData();
      Complex operator + (Complex);
      Complex operator - (Complex);
      Complex operator * (Complex);
      Complex operator / (Complex);
};
Complex::Complex()
{
      real = imag = 0.0;
}
void Complex::getData (float r, float i)
{
```

```
        real = r;
        imag = i;
    }
    void Complex::printData()
    {
        if (imag >= 0)
          cout<<real<<" + "<<imag<<"i"<<endl;
        else
          cout<<real<<imag<<"i"<<endl;
    }
    Complex Complex::operator + (Complex e)
    {
         Complex t;
         t.real = real + e.real;
         t.imag = imag + e.imag;
         return t;
    }
    Complex Complex::operator - (Complex e)
    {
        Complex t;
        t.real = real - e.real;
        t.imag = imag - e.imag;
        return t;
    }
    Complex Complex::operator * (Complex e)
    {
        Complex t;
        t.real = real*e.real - imag*e.imag;
        t.imag = real*e.imag + imag*e.real;
        return t;
    }
    Complex Complex::operator / (Complex e)
    {
         Complex t;
         float x = e.real*e.real + e.imag*e.imag;
         t.real = (e.real*e.real + e.imag*e.imag)/x;
         t.imag = (imag*e.real - real*e.imag)/x;
         return t;
    }
```

```
void main()
{
     Complex c1,c2,c3;
     float r1,i1,r2,i2;
     cout<<"Enter the complex number C1...\n";
     cout<<"Enter real part     : ";
     cin>>r1;
     cout<<"Enter imaginary part: ";
     cin>>i1;
     c1.getData(r1,i1);
     cout<<"C1 = ";
     c1.printData();
     cout<<"\nEnter the complex number C2...\n";
     cout<<"Enter real part     : ";
     cin>>r2;
     cout<<"Enter imaginary part: ";
     cin>>i2;
     c2.getData(r2,i2);
     cout<<"C2 = ";
     c2.printData();
     cout<<endl;

     c3=c1+c2; // Add two complex number
     cout<<"C1 + C2 = ";
     c3.printData();

     c3=c1-c2; // Subtract two complex number
     cout<<"C1 - C2 = ";
     c3.printData();

     c3=c1*c2; // Multiply two complex number
      cout<<"C1 * C2 = ";
     c3.printData();

     c3=c1/c2; // Divide two complex number
     cout<<"C1 / C2 = ";
     c3.printData();
}
```

அடுத்து, இரண்டு திகதிகளுக்கிடையில் உள்ள வித்தியாசம் எமக்குத் தேவைப்பட்டால், எவ்வாறு "-" என்ற ஒப்பறேற்றரினைப் பயன்படுத்த முடியும் என்பதனைப் பார்ப்போம்.

உதாரணமாக, சி++ மொழிப் புறோகிராமில் ஒருவருடைய வயதை ஆண்டு, மாதம், நாள் எனத் துல்லியமாகக் கணிக்க வேண்டுமாயின், நேரடியாக இன்றைய திகதியிலிருந்து அவருடைய பிறந்த திகதியைக் கழிப்பதன் மூலம் பெற முடியாது. எனவே, முதலில் "-" என்ற ஒப்பறேற்றருக்குரிய சி++ மொழிப் புறோகிராம் எழுதப்பட வேண்டும். பின்னர், நேரடியாக இரண்டு திகதிகளுக்கு இடையிலுள்ள வித்தியாசத்தினைக் கணிப்பிட முடியும்.

சி++ மொழியில் இவ்வாறு திகதியினைப் பயன்படுத்த வேண்டுமாயின், Date என்ற கிளாஸினை வரையறுத்து, அக்கிளாஸிலிருந்து ஒப்ஜெக்ற்களை உருவாக்க வேண்டும்.

உதாரணமாக: Date d1, d2 என Date இனத்துக்குரிய இரு ஒப்ஜெக்ற்களை வரையறுக்க முடியும்.

Date என்ற கிளாஸில், அதன் மூன்று பண்புகளான நாள் (date), மாதம் (month), வருடம் (year) போன்றவற்றை வரையறுக்க வேண்டும். மேலும், இந்த கிளாஸின் பண்புகளான day, month, year போன்றவற்றிற்குரிய ஆரம்பப் பெறுமானம் பூச்சியம் கொடுப்பதற்குரிய கொன்ஸ்ரக்ரர் (Constructor) ஒன்றை வரையறுக்க வேண்டும். மற்றும், திகதிகளை உள்ளீடு செய்வதற்குரிய getDate() என்ற ∴பங்ஷனும், திகதிகளை வெளியீடாகக் காட்டுவதற்குரிய printDate() என்ற ∴பங்ஷனும், இந்த Date என்ற கிளாஸில் வரையறுக்க வேண்டும். பின்னர், "-" என்ற ஒப்பறேற்றரினை ஓவலோடிங் செய்வதற்குரிய கட்டளைகளை எழுத வேண்டும்.

மேலே கூறப்பட்ட Date என்ற கிளாஸிற்குரிய சி++ மொழிப் புறோகிராம் கீழே எழுதப்பட்டுள்ளது.

```
#include <iostream.h>
class Date
{
   private:
      int day, month, year;
   public:
      Date();
      void getDate (int, int, int);
```

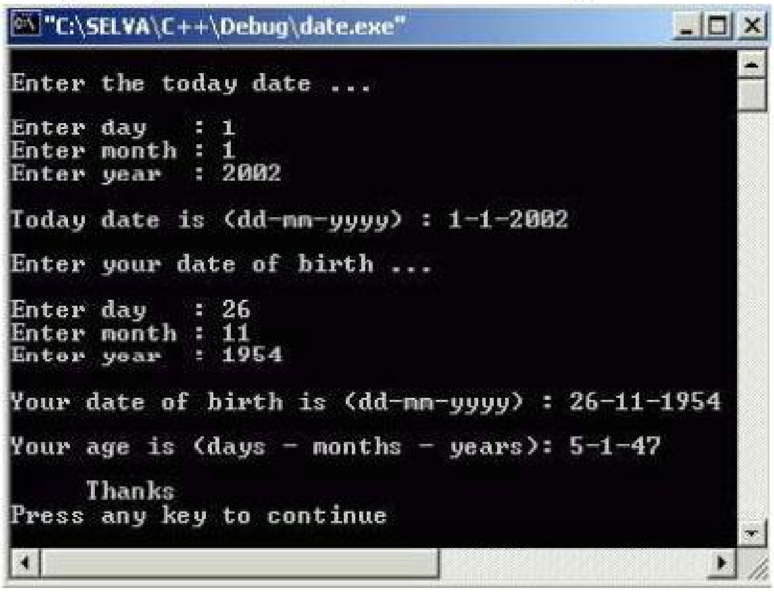
```
        void printDate();
        Date operator - (Date);
};
Date::Date()
{
    day = month = year = 0;
}
void Date::getDate(int d, int m, int y)
{
    day = d;
    month = m;
    year = y;
}
void Date::printDate()
{
    cout<<day<<"-"<<month<<"-"<<year<<endl;
}
Date Date::operator - (Date e)
{
    Date t;
    if (day >= e.day)
        t.day = day-e.day;
    else
    {
        t.day = day + 30 - e.day;
        month--;
    }
     if (month >= e.month)
        t.month = month - e.month;
    else
    {
        t.month = month + 12 - e.month;
        year--;
    }
    t.year = year - e.year;
    return t;
}
```

```
void  main()
{
    Date  d1,d2,d3;
    int da1,da2,m1,m2,y1,y2;
    cout<<"\n Enter the today date ... \n";
    cout<<"\n Enter day   : ";
    cin>>da1;
    cout<<"Enter month : ";
    cin>>m1;
    cout<<"Enter year  : ";
    cin>>y1;
    d1.getDate(da1,m1,y1);
    cout<<"\nToday date is (dd-mm-yyyy) : ";
    d1.printDate();

    cout<<"\nEnter your date of birth ...\n";
    cout<<"\nEnter day   : ";
    cin>>da2;
    cout<<"Enter month : ";
    cin>>m2;
    cout<<"Enter year  : ";
    cin>>y2;
    d2.getDate(da2,m2,y2);
    cout<<"\nYour date of birth is (dd-mm-yyyy) :";
    d2.printDate();

    d3=d1-d2;
    cout<<"\nYour age is (days - months - years):";
    d3.printDate();
    cout<<"\n Thanks \n";
}
```

இந்தப் புறோகிராமினைச் செயற்படுத்தும் போது, முதலில் இன்றைய திகதியை உள்ளீடு செய்யுமாறு கேள்வி கேட்கப்படும். பின்னர், பிறந்த திகதியை உள்ளீடு செய்யுமாறு கேள்வி கேட்கப்படும். நீங்கள் இந்த இரண்டு திகதிகளையும் உள்ளீடு செய்ததும், உண்மையான வயதினை நாள், மாதம், ஆண்டு எனத் துல்லியமாகக் கணித்துத் தரும்.

உதாரணமாக, நீங்கள் இன்றைய திகதியை 1 ஆம் திகதி, ஜனவரி மாதம், 2002 ஆம் ஆண்டு எனவும், பிறந்த திகதியை 26 ஆம் திகதி,

நவம்பர் மாதம், 1954 ஆம் ஆண்டு எனவும் உள்ளீடு செய்தால், வெளியீடானது கீழே உள்ளவாறு காண்பிக்கப்படும்.



எனவே, நீங்களும் தேவையான கிளாஸினை வரையறுத்து ஒப்பறேற்றர் ஓவலோடிங் முறையினைப் பயன்படுத்த முடியும்.

உதாரணமாக, இரண்டு தூரங்களை அல்லது நேரங்களைக் கூட்ட, கழிக்க +, - போன்ற ஒப்பறேற்றர்களை, அவற்றின் ஓவலோடிங் செய்வதற்குரிய கட்டளைகளை எழுதிய பின்னர் பயன்படுத்தலாம்.

சிக்கல் எண்களைக் கூட்ட, கழிக்க, பெருக்க மற்றும் பிரிக்க ஒப்பறேற்றர்களான +, -, *, / போன்றவற்றை எவ்வாறு பயன்படுத்தப் படுகின்றன எனவும், திகதிகளைக் கழிப்பதற்கு ஒப்பறேற்றரான '-' இனை எவ்வாறு பயன்படுத்துவது எனவும் உதாரணங்கள் மூலம் தெளிவாகப் பார்த்தோம்.

அடுத்து, பொலிமோபிஸத்தின் மற்றைய பிரிவான, புறோகிராம் செயற்படும் போது தீர்மானிக்கும் ∴பங்ஷனை எவ்வாறு வரையறுக்க முடியும் என்பதனைப் பார்ப்போம்.

ரன் ரைம் பொலிமோபிஸம் (Run Time Polymorphism)

புறோகிராம் செயற்படும் (Program Execute) போது, எந்த ∴பங்ஷன் (Function) அல்லது எந்த ஒப்ஜெக்ற் (Object) தேவை என்பதனைத் தீர்மானிப்பது ரன் ரைம் பொலிமோபிஸம் என அழைக்கப்படுகின்றது. அதாவது, புறோகிராம் இயங்குகின்ற நேரத்தில்தான் ஒப்ஜெக்ற் களுக்கிடையே தொடர்பு ஏற்படுகின்றது. எனவேதான், இதனைப் பிந்திய பிணைப்பு (Late Binding) எனவும் அழைக்கப்படுகின்றது.

சி$^{++}$ மொழியில், வேர்சுவல் (Virtual) ∴பங்ஷன் என்பது லேற் பைன்டிங் (Late Binding) ஆகத் தொழிற்படுகின்றது.

உதாரணமாக, சி$^{++}$ மொழியில் உருவாக்கப்பட்ட பேஸ் கிளாஸ் (Base Class) இல் வரையறுக்கப்பட்ட ஒரு ∴பங்ஷனானது, டிரைவ்ட் கிளாஸ் (Derived Class) இல் அதே பெயருடைய ∴பங்ஷனாக பயன்படுத்தும் சந்தர்ப்பத்தில் இந்த வேர்சுவல் ∴பங்ஷன் பயன் படுத்தப்படுகின்றது.

மேலே கூறப்பட்டதற்குரிய சி$^{++}$ மொழிப் புறோகிராமினை அடுத்துப் பார்ப்போம்.

```
class Person
{
 private:
    char name[30];
    char address[50];
public:
    virtual void getData()
    {
        name = “Siva”;
        address = “Colombo-3”;
    }
    virtual void printData()
    {
        cout<<“ Name : ”<<name<<endl;
        cout<<“ Address : ”<<address<<endl;
    }
};
class Student:public Person
{
    private:
        char indexNo[10];
     public:
        void getData()
        {
            indexNo=“s5283”;
        }
        void printData()
        {
            cout<<“Index No: ”<<indexNo<<endl;
        }
};
```

```
void main()
{
    Person *p,q;
    Student s;
    p = &q;
    p->getData();
    p->printData();
    p = &s;
    p->getData();
    p->printData();
}
```

இந்த உதாரணத்தில், virtual என்ற கீவேட் (Keyword) ஆனது பேஸ் கிளாஸிலுள்ள getData(), printData() போன்ற இரு ∴பங்ஷின் களுக்கு முன்னால் பயன்படுத்தப்பட்டுள்ளது. பின்னர், Student என்ற டிரைவ்ட் கிளாஸில் அதே பெயருடைய getData(), printData() போன்ற ∴பங்ஷின்கள் பயன்படுத்தப்பட்டுள்ளன. இச்செயற்பாடுகள் ∴பங்ஷின் ஓவரைடிங் (Overriding) என அழைக்கப்படுகின்றது. அதாவது, பேஸ் கிளாஸில் காணப்படும் வேர்சுவல் ∴பங்ஷினானது ட்ரைவ்ட் கிளாஸில் மீள்வரையறை செய்யப்படுவதையே ஓவரைடிங் என அழைக்கப் படுகின்றது. ட்ரைவ் கிளாஸிலுள்ள ஓவரைடிங் ∴பங்ஷினின் பெயர், பராமீற்றர்களின் எண்ணிக்கை, விபர இன ஒழுங்கு போன்றன பேஸ் கிளாஸிலுள்ள வேர்சுவல் (Virtual) ∴பங்ஷினை ஒத்ததாக இருக்க வேண்டும்.

மெயின் ∴பங்ஷினில், Person என்ற பேஸ் கிளாஸிலிருந்து p என்ற பொயின்ரர் ஒப்ஜெக்ற்றையும், q என்ற சாதாரண ஒப்ஜெக்ற்றையும் உருவாக்கும் விதமாக முதலாவது கட்டளை எழுதப்பட்டுள்ளது. அடுத்து, டிரைவ்ட் கிளாஸான Student என்ற கிளாஸிலிருந்து s என்ற ஒப்ஜெக்ற் உருவாக்கப்பட்டுள்ளது.

Person என்ற பேஸ் கிளாஸிலிருந்து உருவாக்கப்பட்ட p என்ற பொயின்ரர் ஒப்ஜெக்ற் மூலம், பேஸ் கிளாஸில் உள்ள getData(), printData() போன்ற ∴பங்ஷன்கள் கையாளப்படுகின்றன. இதே p என்ற பொயின்ரர் ஒப்ஜெக்ற் மூலம், டிரைவ்ட் கிளாஸிலுள்ள getData(), printData() போன்ற ∴பங்ஷின்கள் கையாளப்படுகின்றன. இந்தச் செயற்பாடுகள் யாவும் புறோகிராம் செயற்படும் போதுதான் தீர்மானிக்கப்படுகின்றது. எனவேதான், இச்செயற்பாடு ரன் ரைம் பொலிமோபிஷம் என அழைக்கப்படுகின்றது.

p என்ற பொயின்ரர் Person இனத்தில் அறிவிக்கப்பட்டுள்ளது. எனினும், இந்தப் பொயின்ரர் Student இன ஒப்ஜெக்ற்றையும் சுட்டிக் காட்டப் பயன்படுத்தப்பட்டுள்ளது. p என்ற பொயின்ரர் Person, Student போன்ற இன ஒப்ஜெக்ற்களையும் மாறி மாறிச் சுட்டிக் காட்டுவதற்குப் பயன்படுத்தப்பட்டுள்ளது.

அதாவது, p = &s;

p->getData(); ஆகும்.

ஃபங்ஷன் ஓவலோடிங், ஃபங்ஷன் ஓவரைடிங் போன்றவற்றுக்கிடையே உள்ள வித்தியாசங்கள்

புறோகிராமொன்றில், குறித்தொரு பெயருடைய ∴பங்ஷனிற்கு உள்ளீடு செய்யும் பராமீற்றர்களின் எண்ணிக்கை வேறுபட்டிருக்கலாம் அல்லது பராமீற்றர்களின் விபர இனம் வேறுபட்டிருக்கலாம், இச்செயற்பாடானது ∴பங்ஷன் ஓவலோடிங் (Function Overloading) என அழைக்கப்படுகின்றது. ∴பங்ஷன் ஓவலோடிங் முறையானது ஒரே கிளாஸில் மட்டுமே நடைபெறுகின்றது.

புறோகிராமொன்றில், குறித்தொரு பெயருடைய ∴பங்ஷனிற்கு உள்ளீடு செய்யும் பராமீற்றர்களின் எண்ணிக்கை, மற்றும் பராமீற்றர்களின் விபர இனம் போன்றன ஒரே விதமாக காணப்படுமாயின், இவ்வகைச் செயற்பாடானது ∴பங்ஷன் ஓவரைடிங் (Function Overriding) என அழைக்கப்படும். ஆனால், இங்கு காணப்படும் ∴பங்ஷன்கள் புறோகிராமில் வெவ்வேறு கிளாஸ்களில் காணப்பட வேண்டும்.

பியோர் வேர்சுவல் ஃபங்ஷன்கள் (Pure Virtual Functions):

பியோர் வேர்சுவல் ∴பங்ஷன் என்றால் என்ன? என்பதனை முதலில் பார்ப்போம்.

பேஸ் கிளாஸில் வரையறுக்கப்பட்ட ஒரு ∴பங்ஷன், டிரைவ்ட் கிளாஸில் அதே பெயர், பராமீற்றர்களின் எண்ணிக்கை, விபர இனங்கள் போன்றன ஒன்றாக உடைய மற்றுமொரு ∴பங்ஷனை வரையறுக்க முடியும். இதற்கு பேஸ் கிளாஸிலுள்ள ∴பங்ஷனிற்கு முன்னால், virtual என்ற சி++ கீவேட் (Keyword) பயன்படுத்தலாம் எனப் பார்த்தோம். பேஸ் கிளாஸில் காணப்படும் வேர்சுவல் ∴பங்ஷனிற்குரிய கட்டளை களை எழுதாது, ∴பங்ஷனை மட்டும் வரையறுக்கும் சந்தர்ப்பத்தில், பியோர் வேர்சுவல் ∴பங்ஷன் தேவைப்படுகிறது.

டிரைவ்ட் கிளாஸில் கட்டாயமாக ஓவரைடிங் ∴பங்ஷன் இருக்க வேண்டும் என முன்கூட்டியே கொம்பைலர் எமக்கு அறிவுறுத்த வேண்டும் என்பதற்காகவே, இந்த பியோர் வேர்சுவல் ∴பங்ஷன் எழுதப்படுகிறது. பியோர் வேர்சுவல் ∴பங்ஷன் காணப்படும் கிளாஸானது,

முற்றுப்பெறாத கிளாஸாகும். எனவே, இந்த கிளாஸிற்குக் கட்டாயமாக டிரைவ்ட் கிளாஸ் வரையறுக்கப்பட வேண்டும்.

பியோர் வேர்சுவல் ∴பங்ஷன் (Pure Virtual Function) இனை எவ்வாறு வரையறுப்பது என அடுத்துப் பார்ப்போம்.

virtual void calculation()=0; என பியோர் வேர்சுவல் ∴பங்ஷனை கிளாஸில் வரையறுக்க முடியும்.

ஒரு உதாரணப் புறோகிராம் மூலம், எவ்வாறு பியோர் வேர்சுவல் ∴பங்ஷன் பயன்படுத்தப்படுகிறது எனப் பார்ப்போம்.

```
#include <iostream.h>
class Person
{
     private:
          char name[30];
     public:
          void getData()
          {
               cout<<"Enter name : ";
               cin>>name;
          }
     void printData()
     {
          cout<<" Name : "<<name<<endl;
     }
     virtual void calculation()=0;
};
class Staff:public Person
{
     private:
          double salary, netSalary;
          int othours;
     public:
          void getData()
          {
               Person::getData();
               cout<<"Enter salary : ";
               cin>>salary;
          }
```

```
            void calculation()
            {
                netSalary=salary+othours*150;
             }
            void printData()
            {
                Person::printData();
                cout<<"Net Salary : "<<netsalary<<endl;
            }
    };
```

மேலேயுள்ள புறோகிராமில், Person என்ற பேஸ் கிளாஸில் calculation() என்ற ∴பங்ஷனானது பியோர் வேர்சுவல் ∴பங்ஷனாக வரையறுக்கப்பட்டுள்ளது. ஏனெனில், Person என்ற பேஸ் கிளாஸில் எமக்கு கணிப்பீடு தேவைப்படவில்லை. ஆனால், டிரைவ்ட் கிளாஸில் நிச்சயமாகக் கணிப்பீடு தேவைப்படும் என்பதால், முன்கூட்டியே அக் கிளாஸிற்குரிய calculation() என்ற ∴பங்ஷனானது, பேஸ் கிளாஸில் அறிவிக்கப்பட்டுள்ளது.

7.5 அப்ஸ்ரக்ற் கிளாஸ்கள் (Abstract Classes)

பியோர் வேர்ச்சுவல் ∴பங்ஷன் வரையறுக்கப்பட்ட கிளாஸானது, அப்ஸ்ரக்ற் கிளாஸ் என அழைக்கப்படும். அப்ஸ்ரக்ற் கிளாஸிலிருந்து ஒருபோதும் ஒப்ஜெக்ற்களை உருவாக்க முடியாது. அப்ஸ்ரக்ற் கிளாஸிலிருந்து டிரைவ்ட் கிளாஸ்களை உருவாக்கிய பின்னர், டிரைவ்ட் கிளாஸ்களிலிருந்து ஒப்ஜெக்ற்களை உருவாக்க முடியும். மேலேயுள்ள உதாரணப் புறோகிராமில், Person என்பது அப்ஸ்ரக்ற் கிளாஸாக தொழிற்படும். எனவே, மேலேயுள்ள புறோகிராமில் Person என்ற பேஸ் கிளாஸிலிருந்து ஒருபோதும் ஒப்ஜெக்ற்களை உருவாக்க முடியாது.

7.6 ஃபிரண்ட் ஃபங்ஷன்களும், ஃபிரண்ட் கிளாஸ்களும் (Friend Functions and Friend Classes)

ஃபிரண்ட் ஃபங்ஷன்கள் (Friend Functions):

சி++ மொழியில் ஒரு கிளாஸினை உருவாக்கும் போது, அதனுள் பல மெம்பர் ∴பங்ஷன்களை வரையறுக்க முடியும். அந்தக் கிளாஸிற் குரிய மெம்பர் ∴பங்ஷன் இல்லாமல், கிளாஸிற்கு வெளியில் வரையறுக் கப்பட்டுள்ள ∴ங்ஷனை friend என்ற சி++ மொழி கீவேட் (Keyword) மூலம், அந்தக் கிளாஸின் மெம்பர் ∴பங்ஷன்களைப் போல் பிரைவேட் டேற்றா மெம்பர்களையும் friend ∴பங்ஷன் கையாள முடியும்.

உதாரணமாக,

```
class Mycalss
{
    privte:
        int a, b;
    public:
        void set_ab(int i, int j);
        friend int sum(Myclass x);
};
void Myclass::set_ab(int i, int j)
{
    a = i;
    b = j;
}
// sum() is not a member function of  Myclass
// It can be directly access a and b
int sum(Myclass x)
{
    return x.a+x.b;  // a & b are private member of Myclass
}
void main()
{
    Myclass n;
    n.set_ab(7,9);
    cout<<sum(n)<<endl; // 16
}
```

மேலேயுள்ள புறோகிராமில், sum() என்ற ∴பங்ஷன் friend ∴பங்ஷனாகும். அதாவது, இந்த ∴பங்ஷனானது கிளாஸின் மெம்பர் ∴பங்ஷனில்லாத ஒரு வெளி ∴பங்ஷனாகும். எனினும் Mycalss இல் உள்ள private டேற்றாவை இந்த sum() என்ற friend ∴பங்ஷன் கையாள முடியும். இதுவே friend ∴பங்ஷனின் சிறப்பாகும். இந்த sum() என்ற friend ∴பங்ஷனை, மெயின் ∴பங்ஷனில் அழைக்கும் போது ஒப்ஜெக்ற்றினைக் குறிப்பிடத் தேவையில்லை. அதாவது, sum() ∴பங்ஷனை நேரடியாக மெயின் ∴பங்ஷனில் அழைக்க முடியும். இவ்வாறு, friend ∴பங்ஷனைக் கிளாஸில் பயன்படுத்த முடியும்.

ஃபிரண்ட் கிளாஸ்கள் (Friend Classes):

friend ∴பங்ஷன் போன்றே friend கிளாஸ் என்றால், ஒரு கிளாஸில் இன்னுமொரு கிளாஸினை இன்ஹெரிற் (Inherit) செய்யாது, பயன்படுத்த முடியும். அதாவது, ஒரு கிளாஸில் இன்ஹெரிற் செய்யாத

friend கிளாஸினைப் பயன்படுத்த முடியும். இந்த friend கிளாஸானது, அழைக்கப்பட்ட கிளாஸிலுள்ள private டேற்றாவைக் கையாள முடியும்.

உதாரணமாக,

```
class TwoValues
{
    privte:
        int a, b;
    public:
        TwoValues(int i, int j)
        {
            a = i;
            b = j;
        }
        friend class Max;
};
// Max is not a inherit class
// It can be directly access a and b
// Class Max has access to the private variable a & b
class Max
{
    public:
        Max(TwoValues x)
        {
            return (x.a>x.b) ? x.a : x.b;
        }
}
void main()
{
    TwoValues ob(30,56);
    Max m;
    cout<<m.Max(ob)<<endl;
}
```

மேலேயுள்ள புறோகிராமில், Max என்ற கிளாஸ் friend கிளாஸ் ஆகும். அதாவது, இந்த கிளாஸானது TwoValues இலுள்ள private டேற்றாவைக் கையாள முடியும். இதுவே friend கிளாஸின் சிறப்பம்சமாகும்.

8. ரெம்ப்லேற்கள் (Templates)

எமக்குத் தேவையான செயற்பாட்டினை முன்கூட்டியே தயார் நிலையில் உருவாக்கி வைத்திருக்கும் அச்சினையே ரெம்ப்லேற் என அழைக்கப்படுகிறது. சி++ மொழி ரீதியாகக் கூறுவோமாயின், வெவ்வேறு விபர இனங்களை (Data Type) உடைய உள்ளீடுகளைக் குறித்தொரு ரெம்ப்லேற் ∴பங்ஷனிற்குக் கொடுப்பதன் மூலம், விபர இனங்களுக்கு ஏற்ப வெவ்வேறு செயற்பாடுகளை செயற்படுத்தப் பயன்படும் கட்டமைப்பே ரெம்ப்லேற் என அழைக்கப்படுகிறது.

சி++ மொழியில் இரு வகையான ரெம்ப்லேற்களை வரையறுக்கப்பட முடியும்.

அவையாவன,

(1) ரெம்ப்லேற் ∴பங்ஷன்கள் (Template Functions)
(2) ரெம்ப்லேற் கிளாஸ்கள் (Template Classes)

8.1 ரெம்ப்லேற் ∴பங்ஷன்கள் (Template Functions)

ஒரு குறித்த செயற்பாட்டினை நிறைவேற்றுவதற்காக ∴பங்ஷன்கள் எழுதப்படுகின்றன. இதே செயற்பாட்டினை ஒத்த இன்னுமொரு செயற்பாட்டினை நிறைவேற்ற வேறொரு ∴பங்ஷன் எழுதப்பட வேண்டியுள்ளது எனக் கருதுக. இந்த இரு செயற்பாட்டிற்கும் ஒரே பெயருடைய ∴பங்ஷனை சி மொழியில் பயன்படுத்த முடியாது. ஆனால், சி++ மொழியில் ஒத்த செயற்பாட்டினை உடைய இரு ∴பங்ஷன்களுக்கு ஒரே பெயரினை கொடுக்க முடியும். இச்செயற்பாடானது, சி++ மொழியில் ∴பங்ஷன் ஓவலோடிங் (Function Overloading) என அழைக்கப் படுகின்றது என முன்னைய அத்தியாயத்தில் பார்த்தோம்.

உதாரணமாக, ஒரு பட்டியலில் முழு எண்கள், தசம எண்கள், எழுத்துக்கள், எழுத்துக்கோவைகள் (Strings) போன்ற ஏதாவதொரு விபர இனம் அடங்கியிருக்கலாம். இந்தப் பட்டியலில் எமக்குத் தேவையான உறுப்பினை கண்டுபிடிக்க வேண்டுமாயின், search() என்ற குறித்த ஒரு ∴பங்ஷன் பெயரில், வெவ்வேறு விபர இனங்களுக்கு ஏற்ப பல ∴பங்ஷன்களை எழுத வேண்டும்.

சி++ மொழிப் புறோகிராம் ரீதியாக எவ்வாறு வெவ்வேறு விபர இனங்களுக்குரிய search() என்ற ∴பங்ஷன் எழுதப்படுகிறது எனப் பார்ப்போம்.

search() என்ற ∴பங்ஷனிற்கு உள்ளீடாக மூன்று பராமீற்றர்கள் (Parameters) கொடுக்கப்பட்டுள்ளன எனக் கருதுக. அவையாவன தேடப்படும் மூலகம், தேடப்படும் மூலகம் காணப்படும் பட்டியல்,

பட்டியலில் காணப்படும் மூலகங்களின் எண்ணிக்கை போன்றனவாகும். தேடப்படும் மூலகம், தேடப்படும் மூலகம் காணப்படும் பட்டியல் போன்றன முழு எண்கள், தசம எண்கள், எழுத்துக்கள், எழுத்துக் கோவைகள் (Strings) போன்ற ஏதாவதொரு விபர இனத்தினைக் கொண்டிருக்கலாம். எனவே, ஒவ்வொரு இனங்களுக்குமுரிய தனித் தனி ∴பங்ஷன்களை எழுதுவதன் மூலம், இச்செயற்பாட்டினை நிகழ்த்த முடியும். சி++ மொழியில் காணப்படும் ∴பங்ஷன் ஓவலோடிங் (Function Overloading) முறைக்கு அமைய குறித்தொரு ∴பங்ஷன் பெயரினைப் பயன்படுத்தி search() ∴பங்ஷன்களை எழுத முடியும்.

வெவ்வேறு விபர இனங்களுக்குரிய search() என்ற ∴பங்ஷன்களைப் புறோகிராம் ரீதியாக கீழே எழுதப்பட்டுள்ளன.

```
#include <iostream.h>
void search(int *list, int x, int n)
{
    int i = 0, found = 0;
    while (i<n && found == 0)
    {
        if ( x == list[i] )
            found = 1;
        else
            i++;
    }
     if (found == 1)
        cout<<"Element" <<x<<" found at position "<<i+1;
    else
        cout<<"Element "<<x<<" is not found !!! \n";
}
void search(double *list, double x, int n)
{
     int i = 0, found = 0;
    while (i<n && found == 0)
    {
        if (x == list[i])
            found = 1;
        else
             i++;
    }
```

```
    if (found == 1)
        cout<<"Element " <<x<<" found at position "<<i+1;
    else
        cout<<"Very sorry element x is not found !!! \n";
}
void search (char *list, char x, int n)
{
    int i = 0,found = 0;
    while (i<n && found == 0)
    {
     if (x == list[i])
        found = 1;
    else
         i++;
    }
    if (found == 1)
        cout<<"Element " <<x<<"found  at position "<<i+1;
    else
        cout<<"Very sorry element x is not found !!! \n";
}
void main()
{
    int list1[]={23, 44, 56, 77, 54, 677, 88, 55};
    int x1;
    cout<<"Please enter the search element : ";
    cin>>x1;
    search (list1,x1,8);
    cout<<"\n\n";
    double list2[]={23.2, 67.5, 77.6, 54.7, 88.45};
    double x2;
    cout<<"Please enter the search element : ";
    cin>>x2;
    search (list2,x2,5);
    cout<<"\n\n";
    char list3[]="sivakumar";
    char x3;
    cout<<"Please enter the search element : ";
```

```
        cin>>x3;
        search (list3,x3,9);
        cout<<"\n";
    }
```

இந்தப் புறோகிராமினைச் செயற்படுத்திப் பார்த்தால், கீழேயுள்ளவாறு வெளியீடானது தோன்றும்.

```
"C:\Selva\C++\Debug\search.exe"
Please enter the search element : 56
Element 56 is find at position 3


Please enter the search element : 98.78
Element 98.78 is not found !!!


Please enter the search element : a
Element a is find at position 4

Press any key to continue
```

மேலேயுள்ள புறோகிராமில், மூன்று முறை search() என்ற ∴பங்ஷன் எழுதப்பட்டுள்ளது. ஏனெனில், விபர இனங்களான முழு எண்கள், தசம எண்கள், எழுத்துக்கள் அடங்கிய பட்டியலில் மூலகங்களைத் தேட வேண்டும் என்பதனாலாகும்.

ஒத்த செயற்பாட்டிற்குரிய ∴பங்ஷன்களை மீண்டும், மீண்டும் எழுதுவதனைத் தடுப்பதற்கே, சி++ மொழியில் ரெம்ப்லேற் ∴பங்ஷன் என்ற அமைப்பு உருவாக்கப்பட்டுள்ளது. எனவே, மேலே கூறப்பட்ட ஒத்த செயற்பாடுகளுக்கு, ஒரேயொரு ரெம்ப்லேற் ∴பங்ஷன் வரையறுப்பதன் மூலம் எமது தேவையைப் பூர்த்தி செய்யலாம்.

ரெம்ப்லேற் ∴பங்ஷன் முறையினைப் பயன்படுத்தி எவ்வாறு வெவ்வேறு விபர இனங்களுக்குரிய search() என்ற ∴பங்ஷன் எழுதப் படுகின்றது எனப் பார்ப்போம்.

```
#include <iostream.h>
template <class T>
void search (T *list, T x, int n)
{
     int i = 0, found = 0;
    while (i<n && found == 0)
    {
```

```
        if (x == list[i])
            found = 1;
        else
            i ++;
    }
        if (found == 1)
            cout<<"Element " <<x<<" is find at position "<<i+1;
        else
            cout<<"Element "<<x<<" is not found !!!\n";
    }
```

ரெம்ப்லேற் ∴பங்ஷன் ஒன்றை வரையறுக்க வேண்டுமாயின், முதலில் template என்ற சி++ மொழிக்குரிய கீவேட் (Keyword) இனை எழுத வேண்டும். பின்னர் <,> போன்றவற்றுக்கிடையில் எமக்குத் தேவையான தரவுகளுக்குரிய மாறியின் பெயரினை, class என்ற சி++ மொழிக் கீவேட் (Keyword) இற்கு அடுத்தால்போல் எழுத வேண்டும். கொடுக்கப்படும் தரவுகளில், எல்லா ஒத்த செயற்பாட்டிற்கும் பொதுவான விபர இனம் இருந்தால், class எழுதத் தேவையில்லை. மாறாக, அந்தத் தரவுக்குரிய விபர இனத்தையும், மாறியின் பெயரையும் குறிப்பிட்டால் போதுமாகும். பின்னர், ரெம்ப்லேற் ∴பங்ஷனிற்குரிய கட்டளைகளை எழுத வேண்டும்.

உதாரணமாக,

```
template <class T>
T search (T t, T* x,  int n)
{
    ..........;
    ..........;
}
```

இங்கு T என்பது ரைப் பராமீற்றர் (Type Parameter) ஆகும்.

ரெம்ப்லேற் ∴பங்ஷனில் ஒன்றுக்கு மேற்பட்ட விபர இன பராமீற்றர் களைக் குறிப்பிடலாம்.

உதாரணமாக, template <class A, class B, ...> ஆகும்.

ஒரேயொரு பராமீற்றரினை ∴பங்ஷனிற்கு அனுப்ப வேண்டுமாயின், template <class X> என ரெம்ப்லேற்றினை எழுத வேண்டும். புறோகிராம் செயற்படும் போது, ∴பங்ஷனிற்கு உள்ளீடு செய்யும் விபர இனத்திற்கு அமைய X செயற்படும்.

```
template <class T>
T search (T x)
```

இந்த இரு கட்டளைகளுக்குமிடையில் வேறெந்த கட்டளைகளையும் எழுதக்கூடாது. அதாவது, மேலே எழுதப்பட்ட இரு கட்டளைகளும்

அடுத்தடுத்த வரிகளில் எழுதப்பட வேண்டும் என்பது குறிப்பிடத்தக்க விடயமாகும்.

உதாரணமாக,

```
template <class T>
int a,b;
T search (T x)
```
என எழுதுவது தவறாகும்.

ரெம்ப்லேற்றுக்குரிய மற்றுமொரு உதாரணத்தினை அடுத்துப் பார்ப்போம்.

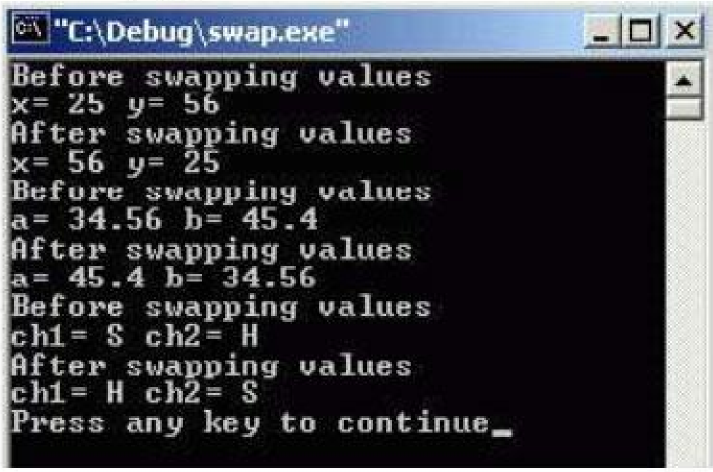
இரண்டு முழு எண்களை இடமாற்றம் (Swapping) செய்ய வேண்டுமாயின், swap() என்ற ஃபங்ஷனை எழுத வேண்டும். இதே புறோகிராமில், இரண்டு தசம எண்களை இடமாற்றம் செய்ய வேண்டுமாயின், swap() என்ற ஒரே பெயரில் வேறொரு ஃபங்ஷனை எழுத வேண்டும். இவ்வாறு எழுத்துக்கள் (Characters), எழுத்துக் கோவைகள் (Strings) போன்ற விபர இனங்கள் உடைய தரவுகளை இடமாற்றம் செய்ய வேண்டுமாயின், swap() என்ற ஒரே பெயரிலான வெவ்வேறு ஃபங்ஷன்களை எழுத வேண்டும். இவ்வாறு ஒத்த செயற்பாடுகளை மீண்டும், மீண்டும் எழுதுவதைத் தவிர்ப்பதற்கு சி++ மொழியில் ரெம்ப்லேற் ஃபங்ஷன் எழுதப்படுகின்றது என முதல் உதாரணத்திலேயே குறிப்பிட்டிருந்தோம். ஒரே ஒரு ரெம்ப்லேற் ஃபங்ஷன் மூலம் மேலே கூறப்பட்ட செயற்பாட்டினை மிக இலகுவாக எழுத முடியும்.

swap() என்ற ரெம்ப்லேற் ஃபங்ஷனிற்குரிய புறோகிராம் கீழே எழுதப்பட்டுள்ளது.

```
#include <iostream.h>
template <class T>
void swap(T &x, T &y)
{
    T temp = x;
    x = y;
    y = temp;
}
void main()
{
    int x = 25,y = 56;
    cout<<"Before swapping values "<<endl;
    cout<<"x = "<<x<<" y = "<<y<<endl;
    swap(x,y);
```

```
        cout<<"After swapping values "<<endl;
        cout<<"x= "<<x<<" y= "<<y<<endl;
        double a=34.56,b=45.4;
        cout<<"Before swapping values "<<endl;
        cout<<"a= "<<a<<" b= "<<b<<endl;
        swap (a,b);
        cout<<"After swapping values "<<endl;
        cout<<"a= "<<a<<" b= "<<b<<endl;
        char ch1='S',ch2='H';
        cout<<"Before swapping values"<<endl;
        cout<<"ch1= "<<ch1<<" ch2= "<<ch2<<endl;
        swap (ch1,ch2);
        cout<<"After swapping values"<<endl;
        cout<<"ch1= "<<ch1<<" ch2= "<<ch2<<endl;
    }
```

இந்தப் புறோகிராமினைச் செயற்படுத்திப் பார்த்தால், வெளியீடனாது கீழேயுள்ளவாறு தோற்றமளிக்கும்.



8.2 ரெம்ப்லேற் கிளாஸ்கள் (Template Classes)

வெவ்வேறு விபர இனங்களைக் கொண்ட தரவுகளையும், ஒரே விதமான செயற்பாடுகளையும் உடைய வெவ்வேறு கிளாஸ்களுக்குப் பதிலாக ஒரேயொரு ரெம்ப்லேற் கிளாஸினை உருவாக்கிப் புறோகிராமில் தேவையான போது பயன்படுத்த முடியும். இதையே ரெம்ப்லேற் கிளாஸ் என அழைக்கப்படுகின்றது.

ரெம்ப்லேற் கிளாஸ்கள் எவ்வாறு, எதற்காக உருவாக்கப்படுகின்றன எனப் பார்ப்போம்.

ஒத்த செயற்பாடுகளைக் கொண்ட ∴பங்ஷின்களை உருவாக்க ரெம்ப்லேற் ∴பங்ஷின் வரையறுக்கப்படுவது போன்று, ஒத்த கிளாஸ் களை உருவாக்குவதற்கு ரெம்ப்லேற் கிளாஸ் வரையறுக்கப்படுகின்றது.

வெவ்வேறு விபர இனங்களைக் கொண்ட தரவுகளையும், ஒரே விதமான செயற்பாடுகளையும் உடைய வெவ்வேறு கிளாஸ்களுக்குப் பதிலாக ஒரேயொரு ரெம்ப்லேற் கிளாஸினை உருவாக்கி எமது புறோகிராமில் பயன்படுத்த முடியும்.

உதாரணமாக முழு எண்கள் கொண்ட உறுப்புக்களையும், இந்த உறுப்புக்களைக் கையாளும் ∴பங்ஷின்களையும் கொண்ட ஒரு கிளாஸினை உருவாக்கிக் கொள்ள முடியும். ஆனால், இந்த கிளாஸினைப் பயன்படுத்தி, ஒருபோதும் தசம எண்கள், எழுத்துக்கள், எழுத்துக்கோவைகள் போன்றவற்றைப் பயன்படுத்த முடியாது. எனவே, வெவ்வேறு தரவினங்களுக்குரிய கிளாஸ்களை மீண்டும், மீண்டும் உருவாக்க வேண்டும். இந்தப் பிரச்சினையினைத் தீர்வுசெய்யும் முகமாக சி++ மொழியில் ரெம்ப்லேற் கிளாஸினை உருவாக்க முடியும்.

இதற்குரிய உதாரணப் புறோகிராமினை அடுத்துப் பார்ப்போம்.

```
# include <iostream.h>
<template L>
class List
{
    private:
        int size ;
        L* data;
    public:
        List (int n)
        {
            size = n;
            data = new L[size];
        }
        ~List()
        {
            delete [] data ;
        }
};
```

```
void main()
{
      List <int> X(50);
      List <double> Y(50);
      List <char> Z(100);
      List <float> A(30);
}
```

மேலேயுள்ளவாறு, ஒரேயொரு ரெம்ப்லேற் கிளாஸினை மட்டும் உருவாக்கி, எந்தவொரு விபர இனங்களுக்குமுரிய ஒப்ஜெக்றகளைக் கிளாஸிலிருந்து உருவாக்க முடியும்.

main() என்ற ∴பங்ஷனில், List <int>X (50) என்ற கட்டளை மூலம் முழு எண் விபர இனமுடைய 50 முழு எண்களை கொண்ட ஒப்ஜெக்றகளை உருவாக்க முடியும்.

இவ்வாறு List <double> Y(50), List <char> Z(100) போன்றன முறையே தசம எண்கள், எழுத்துக்கள் கொண்ட ஒப்ஜெக்றகளை உருவாக்க முடியும்.

9. ஃபைல்கள் (Files)

புறோகிராமானது இயங்கும் போது மதிப்புக்களைத் தற்காலிகமாகச் சேமித்துவைக்கப் பயன்படும் இடமே மாறிகள் (Variables) என அழைக்கப்படுகின்றது. ஒரே வகையான விபர இனங்களை உடைய பல தரவுகளைச் சேமிப்பதற்காக அறேயினைப் பயன்படுத்த முடியும். எனினும், அறேயில் வெவ்வேறு வகையான தரவுகளைச் சேமிக்க முடியாது. எனவே, இந்தப் பிரச்சினையினைத் நிவர்த்தி செய்யும் முகமாக சி++ மொழியில் ஸ்ரக்சர் (Structure) அல்லது கிளாஸ் (Class) இனைப் பயன்படுத்த முடியும். இவை யாவும் தற்காலிகமான நினைவகத்தில் தான் சேமிக்கப்படுகின்றது. எனவே, நிரந்தரமாகத் தரவுகளை ஒரு ரெக்ஸ்ற் ∴பைலில் சேமிப்பதற்கு சி++ மொழியில் எவ்வாறு கட்டளைகள் எழுதுவது எனப் பார்ப்போம்.

சி++ மொழியில் தரவுகளை உள்ளீடு செய்வதற்கு cin என்ற ஸ்ரீமும், தரவுகளை வெளியீடாகக் கணினித்திரையில் காண்பிப்பதற்கு cout என்ற ஸ்ரீமும் பயன்படுத்தப்படுகின்றது. cin, cout போன்ற இரு கட்டளைகளையும், iostream.h என்ற ஹெடர் ∴பைல் (Header file) இனைப் புகுத்தினால் மட்டுமே பயன்படுத்த முடியும். எனவேதான், அனேகமான சி++ மொழிப் புறோகிராமின் தொடக்கக் கட்டளையாக iostream.h என்ற ஹெடர் ∴பைல் புகுத்தப்படுகிறது.

பொதுவாக, கீபோர்ட் (Keyboard) அல்லது மவுஸ் (Mouse) அல்லது ∴பைலின் மூலமே கணினிக்குத் தகவல்களை உள்ளீடு செய்யப் படுகின்றன.

சி++ மொழிக் கட்டளையினைப் பயன்படுத்தி எவ்வாறு ஒரு ரெக்ஸ்ற் ∴பைலினை உருவாக்கப்படுகிறது எனப் பார்ப்போம்.

சி++ மொழியில் உள்ளீடு, வெளியீடு செய்வதற்குப் பயன்படுத்தப் படும் கட்டளைகளான cin, cout போன்றன iostream.h என்ற ஹெடர் ∴பைலினைப் புகுத்தினால் மட்டுமே செயற்படுத்த முடியும் என்பதனை நாம் அறிவோம். இவ்வாறு ரெக்ஸற் ∴பைல் (Text File) ஒன்றில் தரவுகளை எழுத, வெளியீடாகக் காண்பிப்பதற்கு fstream.h என்ற ஹெடர் ∴பைலினைப் புகுத்த வேண்டும். இந்த fstream.h என்ற ஹெடர் ∴பைலில் cout, cin போன்றனவும் காணப்படுவது குறிப்பிடத்தக்க விடயமாகும். எனவே, fstream.h என்ற ஹெடர் ∴பைலினைப் புகுத்துவோமாயின், iostream.h என்ற ஹெடர் ∴பைலினைப் புகுத்த வேண்டிய அவசியமில்லை.

fstream.h என்ற ஹெடர் ∴பைலில் ofsteam (output file stream), ifstream (input file stream), ios (input and output stream) போன்ற கிளாஸ்கள் உள்ளன. இதன் விளக்கத்தினைப் பின்னர் பார்ப்போம்.

ரெக்ஸ்ற் ∴பைல் ஒன்றில் தகவல்களை எழுத வேண்டுமாயின், முதலில் ஒரு ∴பைல் ஒப்ஜெக்ற்றினை உருவாக்க வேண்டும். ofstream என்ற கிளாஸிலிருந்து அந்த ∴பைல் ஒப்ஜெக்ற் உருவாக்கப்படுகின்றது.

உதாரணமாக:

```
ofstream f ;
```

இங்கு f என்பது, ofstream என்ற கிளாஸிலிருந்து உருவாக்கப்பட்ட ஒரு ஒப்ஜெக்ற் ஆகும். f என்பது ஒப்ஜெக்ற் பெயர் என்பதால், நாம் என்ன பெயர் வேண்டுமானாலும் கொடுக்க முடியும்.

அடுத்து, ரெக்ஸ்ற் பைலின் பெயரினை எவ்வாறு குறிப்பிட முடியும் எனப் பார்ப்போம்.

ofstream f("exam.txt") என்ற கட்டளையானது, exam.txt என்ற ரெக்ஸ்ற் ∴பைல் ஒன்றில் தரவுகளை உள்ளீடு செய்யும் விதமாக திறந்து வைக்கப் பயன்படுத்தப்படுகிறது. அதாவது, ofstream என்ற கிளாஸின் கொன்ஸ்ரக்ரர் (Constructor) ∴பங்ஷனில், exam.txt என்ற ஸ்றிங் (String) பராமீற்றராகக் கொடுக்கப்பட்டுள்ளது.

மேலே எழுதப்பட்ட கட்டளைக்குப் பதிலாக,

```
ofstream f ;
f.open ("exam.txt");
```

எனவும் எழுத முடியும். இங்கு open என்பது ofstream என்ற கிளாஸின் மெம்பர் ∴பங்ஷனாகும்.

அடுத்து, ரெக்ஸ்ற் ∴பைல் ஒன்றில் எவ்வாறு எழுத்துக்கோவைகளை உள்ளீடு செய்யலாம் என்பதற்குரிய மிக இலகுவான உதாரணப் புறோகிராமினைப் பார்ப்போம்.

```
#include <fstream.h>
void main()
{
      ofstream f ("exam.txt");
      f<< "Welcome to Jaffna";
      f.close();
}
```

இந்தப் புறோகிராமில் காணப்படும் முதல் கட்டளையானது, ofstream என்ற கிளாஸ் காணப்படும் ஹெடர் ∴பைலான ofstream.h புகுத்தப் பட்டுள்ளது. பின்னர், main() என்ற ∴பங்ஷனிற்குள் மூன்று கட்டளைகள்

எழுதப்பட்டுள்ளன. இதில் முதல் கட்டளையானது, ofstream என்ற கிளாஸிலிருந்து f என்ற ஒப்ஜெக்ற் exam.txt என்ற பராமீற்றருடன் உருவாக்கப்பட்டுள்ளது. இரண்டாவது கட்டளையானது, exam.txt என்ற ரெக்ஸ்ற் ∴பைல் ஒன்றில் எழுத்துக்கோவைகள் (String) எழுதுவதற்குப் பயன்படுத்தப்பட்டுள்ளது. இறுதிக் கட்டளையான f.close() ஆனது திறக்கபட்ட ∴பைலினை மூடுவதற்குப் பயன்படுத்தப்பட்டுள்ளது.

சி++ மொழிப் புறோகிராம் மூலம், ∴பைலிலுள்ள தகவல்களை எவ்வாறு படிப்பது என அடுத்துப் பார்ப்போம்.

ஒவ்வொரு எழுத்தாக ஒரு ∴பைலிலுள்ள தகவல்களை படிப்பதற்கு ifstream (input file stream) என்ற கிளாஸ் பயன்படுத்தப்படுகிறது. எனவே, ஒரு ∴பைலில் உள்ள தகவல்களை படிப்பதற்கு ifstream என்ற கிளாஸிலிருந்து ஒரு ஒப்ஜெக்ற் உருவாக்க வேண்டும். இங்கு ∴பைலின் பெயரினைப் பராமீற்றராகக் கொடுக்க வேண்டும்.

உதாரணமாக, ifstream f (“jaffna.txt”) என வரையறுக்க முடியும். இங்கு jaffna.txt என்பது ஏற்கனவே உள்ள ஒரு ரெக்ஸ்ற் ∴பைலின் பெயராகும். மேலேயுள்ள கட்டளையானது jaffna.txt என்ற ரெக்ஸ்ற் ∴பைல் திறக்கப்பட்டுப் படிப்பதற்குத் தயாராக வைத்திருக்கும்.

இந்த ரெக்ஸ்ற் ∴பைலிலுள்ள தகவல்களை, ஒவ்வொரு எழுத்தாக படிக்க வேண்டுமாயின், char விபர இன மாறி வரையறுக்கப்பட வேண்டும். get() என்ற ∴பங்ஷன் மூலம் ஒவ்வொரு எழுத்தாக ரெக்ஸ்ற் ∴பைலிலிருந்து எழுத்துக்களைக் கையாள முடியும்.

∴பைலிலுள்ள எல்லா எழுத்துக்களையும் படிக்க வேண்டுமாயின், while லூப்பானது எமக்குத் தேவைப்படுகிறது. இவற்றுக்குரிய சி++ மொழிப் புறோகிராமினை அடுத்துப் பார்ப்போம்.

```
#include <fstream.h>
void main()
{
    char ch;
    ifstream f("jaffna.txt");
    cout<<"The context of jaffna.txt file is as below \n";
    while (f )
    {
        f.get(ch);
        cout<<ch;
    }
    f.close();
}
```

இங்கு jaffna.txt என்பது, மேலேயுள்ள புறோகிராம் சேமிக்கப்பட்ட அதே ∴போல்டரில் காணப்படும் ரெக்ஸ்ற் ∴பைலின் பெயராகும். f என்பது ifstream என்ற கிளாஸிலிருந்து உருவாக்கப்பட்ட ஒப்ஜெக்ற் ஆகும்.

மேலேயுள்ள புறோகிராமினைச் செயற்படுத்திப் பார்த்தால், வெளியீடாக jaffna.txt என்ற ரெக்ஸ்ற் ∴பைலிலுள்ள தகவல்கள் அனைத்தையும் திரையில் காட்டும்.

இதுவரை ரெக்ஸ்ற் ∴பைலிலிருந்து தரவுகளை எவ்வாறு படிப்பது எனவும், தரவுகள் எவ்வாறு உள்ளீடு செய்யப்படுகின்றது எனவும் பார்த்தோம்.

இனி, எவ்வாறு றெக்கோட்களை (Records) ∴பைலில் உள்ளீடு செய்வது எனவும், எவ்வாறு றெக்கோட்களை ∴பைலிலிருந்து எடுத்துப் படிப்பது எனவும் பார்ப்போம்.

பல்கலைக்கழக மாணவர்களுக்குரிய தகவல் ∴பைல் (Data File) இல் காணப்படும் ஒவ்வொரு றெக்கோட்களும் மாணவரின் பெயர், சுட்டிலக்கம், பெறுபேறுகள் போன்ற விபரங்களைக் கொண்டிருக்கும். இத்தகவல் ∴பைலிலிருந்து, ஒவ்வொரு றெக்கோட்களாகப் படித்துக் கணிப்பீட்டுக்குப் பயன்படுத்த முடியும்.

இதற்குரிய புறோகிராமினை எழுத வேண்டுமாயின், முதலில் ஒரு மாணவருக்குரிய கிளாஸினை வரையறுக்க வேண்டும். இந்தக் கிளாஸில் தரவுகள், தரவுகளைக் கையாளும் ∴பங்ஷன்கள் போன்றன வரையறுக்கப்பட வேண்டும். இந்தக் கிளாஸிலிருந்து ஒப்ஜெக்ற்கள் உருவாக்கப்பட்டு, அவற்றினை ஒரு ∴பைலில் அப்படியே சேமிக்க வேண்டும்.

உதாரணமாக, Student என்ற கிளாஸில் name, indexNo, result போன்ற டேற்றாவையும், இந்த டேற்றாவை உள்ளீடு செய்ய getData() என்ற ∴பங்ஷனும் வரையறுக்க வேண்டும்.

```
#include <fstream.h>
class Student
{
   private: // this statement is optional
       char name[50];
       char indexNo[10];
       char result;
   public:
       void getData()
```

```
        {
            cout<<"Enter name : ";
            cin>>name;
            cout<<"Enter index number : ";
            cin>>indexNo;
            cout<<"Enter result : ";
            cin>>result;
        }
};
void main()
{
    Student s;
    s.getData();
    ofstream f("stud.dat");
    f.write((char*)&s, sizeof (s));
    f.close();
}
```

மேலேயுள்ள புறோகிராமில், முதலில் Student என்ற கிளாஸானது வரையறுக்கப்பட்டுள்ளது. பின்னர், மெயின் ∴பங்ஷனில் s என்ற ஒரு ஒப்ஜெக்றானது Student என்ற கிளாஸிலிருந்து உருவாக்கப்பட்டுள்ளது. அடுத்து, ஒரு மாணவருக்குரிய தரவினை உள்ளீடு செய்யும் ∴பங்ஷன் மூலம் ஒரு மாணவனுக்குரிய தரவு உள்ளீடு செய்யப்பட்டுள்ளது. பின்னர், stud.dat என்ற ∴பைலில் இந்த றெக்கோட் உள்ளீடு செய்யப் படுகிறது. இங்கு stud என்ற ∴பைலானது ரெக்ஸ்ற் ∴பைல் (text file) ஆக வரையறுக்காது, டற் (DAT) ∴பைலாக வரையறுத்ததன் நோக்கம், இந்த டற் (DAT) ∴பைலினை நேரடியாகத் திறந்து படிக்க முடியாமல் தடுப்பதற்காகும்.

அடுத்து, இந்த stud.dat என்ற ∴பைலிலிருந்து எவ்வாறு சி++ மொழிப் புறோகிராம் மூலம் தரவுகளைப் படிப்பது எனப் பார்ப்போம்.

```
#include <fstream.h>
class Student
{
    private:
        char name[50];
        char indexNo[10];
        char result;
    public:
```

```
        void printData()
        {
            cout<<"Name : "<<name<<endl;
            cout<<"Index No : "<<indexNo<<endl;
            cout<<"Result : "<<result<<endl;
        }
    };
    void main()
    {
        Student  s;
        s.printData();
        ifstream f("stud.dat");
        f.read((char*)&s, sizeof(s));
        f.close ();
    }
```

மேலேயுள்ள இரண்டு உதாரணப் புறோகிராம்களிலும், ஒரேயொரு றெக்கோட்டினை மட்டுமே எழுத, வெளிக்காட்ட முடியும். எனவே, அடுத்து எவ்வாறு ஒரு ∴பைலில் பல றெக்கோட்களை எழுத, வெளிக்காட்ட முடியும் என்பதற்குரிய புறோகிராம்களைப் பார்ப்போம்.

முதலில் ஒரு ∴பைலில் எவ்வாறு பல றெக்கோட்களை உள்ளீடு செய்ய முடியும் என்பதற்குரிய புறோகிராமினைப் பார்ப்போம்.

```
    #include <fstream.h>
    class Student
    {
        private:
            char name[50];
            char indexNo[10];
            char result;
        public:
            void getData()
            {
                cout<<"Enter name : ";
                cin>>name;
                cout<<"Enter index number : ";
                cin>>indexNo;
                cout<<"Enter result : ";
                cin>>result;
            }
    };
```

```
void main()
{
    Student s;
    char ch;
    ofstream f ("stud.dat");
    do
    {
        s.getData();
        f.write((char*)&s,sizeof(s));
        cout<<"Sussessfully add this record \n";
        cout<<"Do you add more record (y/n) ? ";
        cin>>ch;
    }
    while (ch == 'Y' || ch == 'y');
    f.close ();
}
```

இந்தப் புறோகிராமினைச் செயற்படுத்திப் பார்த்தால், முதலில் ஒரு றெக்கோட்டினைச் சேர்க்கும். பின்னர், மீண்டும் மேலும் றெக்கோட் சேர்க்க வேண்டுமா என கேள்வி கேட்கும். இந்தக் கேள்விக்கு "Y" அல்லது "y" இனை அழுத்தினால் மட்டுமே, அடுத்த றெக்கோட்டினைச் சேர்க்க அனுமதிக்கும். மாறாக, "Y" அல்லது "y" தவிர்ந்த மற்றைய எந்த எழுத்தினை அழுத்தினாலும் அடுத்த றெக்கோட்டினைச் சேர்க்காது.

ஆனால், இந்த புறோகிராமில் ஒரு குறைபாடுள்ளது. அதாவது, ஏற்கனவே றெக்கோட்கள் உள்ள .:.பைலில், புதிதாக மேலும் பல றெக்கோட்களை சேர்க்கும் போது முன்னைய றெக்கோட்கள் யாவும் அழிக்கப்பட்ட பின்னரே அந்தப் புதிய றெக்கோட்கள் சேர்க்கப்படும்.

ofstream f("stud.dat") என்ற கட்டளையில் கொடுக்கப்பட்ட stud.dat என்ற .:.பைலில் ஏற்கனவே பல றெக்கோட்கள் காணப்பட்டால், அந்த stud.dat திறக்கப்பட்டு, அதில் காணப்படும் சகல றெக்கோட்களையும் அழித்த பின்னர், புதிதாக உள்ளீடு செய்யும் றெக்கோட்கள் மட்டுமே சேர்க்கப்படும்.

fstream.h என்ற ஹெடர் .:.பைலில் ofstream, ifstream, ios போன்ற பல கிளாஸ்கள் காணப்படுகிறது என முன்னர் குறிப்பிட்டிருந்தோம். இதன் விளக்கத்தினை இங்கு சற்றுப் பார்ப்போம்.

ios என்ற கிளாஸானது ofstream, ifstream போன்ற கிளாஸ்களின் சுப்பர் கிளாஸ் (Super class) ஆகும். இந்த ios என்ற கிளாஸில் app

என்ற டேற்ரா மெம்பரினைப் பயன்படுத்தி புதிதாக றெக்கோட்களைச் சேர்க்க முடியும்.

அதாவது, ofstream (“stud.dat”, ios :: app) என்ற கட்டளை மூலம், ஏற்கனவே றெக்கோட்கள் காணப்படும் ∴பைலில் புதிதாக பல றெக்கோட்களைச் சேர்க்க முடியும். மேலேயுள்ள புறோகிராமின் குறைபாட்டினை அகற்றிய புறோகிராமினை அடுத்துப் பார்ப்போம்.

```
#include <fstream.h>
class Student
{
    private:
        char name[50];
        char indexNo[50];
        char result;
    public:
        void getData()
        {
            cout<<"Enter name : ";
            cin>>name;
            cout<<"Enter index number : ";
            cin>>indexNo;
            cout<<"Enter result : ";
            cin>>result;
        }
};
void main()
{
    Student s;
    char ch;
    ofstream f("stud.dat",ios :: app);
    do
    {
        s.getData();
        f.write((char*)&s,sizeof(s));
        cout<<"Sussessfully add this record\n";
        cout<<"Do you add more record (y/n)?";
        cin>>ch;
    }
    while (ch == 'Y' || ch == 'y');
```

```
    f.close();
}
```

இந்தப் புறோகிராமின் மூலம் ஏற்கனவே பல றெக்கோட்கள் காணப்படும் ஃபைலில், புதிய பல றெக்கோட்களை இணைக்க முடியும்.

அடுத்து, ஒன்றுக்கு மேற்பட்ட றெக்கோட்களை ஒரு ஃபைலிலிருந்து படித்து, அனைத்து றெக்கோட்களையும் திரையில் காட்டுவதற்குரிய புறோகிராமினைப் பார்ப்போம்.

```
#include <fstream.h>
class Student
{
    private:
        char name[50];
        char indexNo[50];
        char result;
    public:
        void printData()
        {
            cout<<"Name : "<<name<<endl;
            cout<<"Index No : "<<indexNo<<endl;
            cout<<"Result : "<<result<<endl;
        }
};
void main()
{
    Student s;
    char ch;
    ifstream f("stud.dat");
    f.read((char*)&s,sizeof(s));
    while (!f.eof())
    {
        s.printData();
        f.read((char*)&s,sizeof(s));
    }
    f.close();
}
```

10. பிழைகளும், பிழையேந்திகளும் (Errors and Exception Handling)

கணினி உயர்நிலை புறோகிராம்களை எழுதிச் செயற்படுத்தும் போது ஏற்படக்கூடிய பிழைகளை மூன்று பிரிவுகளாக வகைப்படுத்த முடியும்.

அவையாவன,

(1) கட்டளை அமைப்புப் பிழைகள் (Syntax Errors) / கொம்பைல் நேரப் பிழைகள் (Compile Time Errors)

(2) தர்க்க ரீதியான பிழைகள் (Logical Errors)

(3) இயக்க நேரப் பிழைகள் (Run Time Errors)

கட்டளை அமைப்புப் பிழைகள் (Syntax Errors)

இந்தக் கட்டளை அமைப்புப் பிழைகள் (Syntax Errors) புறோகிராம் களைக் கொம்பைல் (Compile) செய்யும் போதே கண்டுபிடிக்க முடியும். இந்தப் பிழைகளை நீக்கினால் மட்டுமே புறோகிராம்களைச் செயற்படுத்த முடியும்.

நாம் எழுதும் புறோகிராம்களில் இந்தக் கட்டளை அமைப்புப் பிழைகளைத் தவிர்க்க வேண்டுமாயின், இந்தக் கணினி உயர்நிலை மொழியிலுள்ள கட்டளை அமைப்பில் நன்கு தேர்ச்சி பெற்றுருக்க வேண்டும். இவ்வகைப் பிழைகளைப் புறோகிராமர்களே கண்டுபிடித்துத் திருத்த முடியும்.

உதாரணமாக, while என்ற சி++ மொழி கீவேட் சொல்லினை whlie எனத் தவறுதலாக எழுதினால் அல்லது கட்டளைகளின் முடிவில் அரைப்புள்ளி (; - Semicolon) குறிப்பிடாது விட்டால் அல்லது மாறியினை அறிவிக்காது பயன்படுத்தினால், புறோகிராமினைக் கொம்பைல் செய்யும் போதே பிழையினைச் சுட்டிக் காட்டும்.

தர்க்க ரீதியான பிழைகள் (Logical Errors)

இத்தகைய பிழைகளை ஒருபோதும் கொம்பைலர் (Compiler) சுட்டிக் காட்டாது. இந்த லொஜிக்கல் வகைப் பிழைகளினால், புறோகிராமில் எதிர்பார்த்த முடிவினைப் பெற முடியாது. இவ்வகைப் பிழைகளைப் புறோகிராமர்கள் கண்டுபிடித்துத் திருத்துவதும் கடினமான செயலாகும். இவ்வகைப் பிழைகள் உள்ள புறோகிராமினை இயக்கிப் பார்த்தால், நன்றாகவே தொடர்ந்து செயற்படும். ஆனால், எதிர்பார்த்த முடிவினைக் காண்பிக்காது.

உதாரணமாக, 500 என்ற பெறுமானம் விடையாகக் காண்பிப்பதற்குப் பதிலாக 5000 என்ற பெறுமானம் விடையாகக் காண்பிக்கப்படும்.

இவ்வகைப் பிழைகளை, பல உள்ளீடுகள் கொடுத்து எதிர்பார்த்த வெளியீடுகள் காண்பிக்கப்படுகின்றனவா எனப் பலமுறை சோதனை செய்வதன் மூலமே களைய முடியும். சில சமயம் புறோகிராம்களில் இவ்வகைப் பிழைகள் பல காலமாகக் கண்டுபிடிக்கப்படாமல் மறைந்து காணப்பட முடியும். அதாவது, சில உள்ளீடுகளுக்கு மட்டுமே பிழையான வெளியீடுகளைக் காண்பிக்கும் லொஜிக்கல் பிழைகளாக இருக்க முடியும்.

இயக்க நேரப் பிழைகள் (Run Time Errors)

புறோகிராம்கள் செயற்படும் போது ஏற்படும் பிழைகளை இயக்க நேரப் பிழைகள் என அழைக்கப்படுகின்றது. இந்தப் பிழைகளை ஒருபோதும் கொம்பைலர் (Compiler) கண்டுகொள்ளாது. புறோகிராம் களைச் சிக்கலின்றி கொம்பைல் செய்து, இயக்கவும் முடியும். சில உள்ளீடுகளுக்கு மட்டும் இயங்கிக் கொண்டிருந்த புறோகிராம் இடை நடுவே நின்றுவிடும். இவ்வாறு, இடைநடுவே நின்றுவிடாமல் மீளுவதற்கே இயக்க நேரப் பிழைகளைக் கையாளும் கட்டளைகள் பயன்படுத்தப் படுகின்றன.

இயக்க நேரத்தில் ஏற்படக்கூடிய பிழைகள் (Run time errors) சில கீழே தரப்பட்டுள்ளது.

- புறோகிராமில் ஒரு இலக்கத்தினைப் பூச்சியத்தால் வகுக்கும் கட்டளை செயற்படும் சந்தர்ப்பத்தில், செயற்பட்டுக் கொண்டிருந்த புறோகிராம் அப்படியே இடைநடுவில் நின்றுவிடும்.
- test.txt என்ற ஒரு ஃபைலினைத் திறந்து அதிலுள்ள தகவல்களை படிப்பதற்கு எழுதப்பட்ட புறோகிராம் செயற்படும் சந்தர்ப்பத்தில், test.txt என்ற ஃபைல் காணப்படாதிருந்தால், செயற்பட்டுக் கொண்டிருந்த புறோகிராம் அப்படியே இடைநடுவில் நின்றுவிடும்.
- முழு எண் விபர இனமாக வரையறுக்கப்பட்ட மாறியில், எழுத்து அல்லது தசம தான பெறுமானத்தினை உள்ளீடாகக் கொடுக்கும்போது, செயற்பட்டுக் கொண்டிருந்த புறோகிராம் அப்படியே இடைநடுவில் நின்றுவிடும்.
- அறேக்களில் வரையறுக்கப்பட்ட உறுப்புக்களின் எண்ணிக்கையை விடக் கூடிய எண்ணிக்கையான உறுப்புக்களை உள்ளீடாகக் கொடுக்கும் போது, செயற்பட்டுக் கொண்டிருந்த புறோகிராம் அப்படியே இடைநடுவில் நின்றுவிடும்.

இயக்க நேரப் பிழைகளை விதிவிலக்குகள் (Exceptions) என அழைக்கப்படுகின்றது. புறோகிராம்களில் ஏற்படக்கூடிய இயக்க நேரப்

பிழைகளைக் கண்டறிந்து, அதனைச் சரி செய்யும் முறையினை எக்ஸெப்ஸன் கன்ட்லிங் (Exception Handling) அல்லது எரர் கன்ட்லிங் (Error Handling) என அழைக்கப்படுகின்றது.

இயக்க நேரப் பிழைகளை எதிர்கொள்வதற்குகென்றே தனியான செயற்பாடுகள் உள்ளன. அவையாவன try, catch மற்றும் throw ஆகும். இக்கட்டளைகளை Exception Handling கட்டளைகள் என அழைக்கப்படுகின்றன.

இயக்க நேரப் பிழைகளை ஒருபோதும் புறோகிராமர் தடுக்க முடியாது. ஆனால், அவற்றை முறைப்படி கையாள முடியும். அதாவது, புறோகிராம் இடைநடுவே நிற்பதைத் தடுப்பதற்கு Exception Handling இனைப் பயன்படுத்தி முறைப்படி முடித்து வைக்க முடியும்.

மேலும், இயக்க நேரப் பிழைகளை எதிர்கொண்டு, பிழைகளைச் சுட்டிக் காட்டும் அறிவிப்புக்களை அறிவித்துவிட்டு, புறோகிராமினைத் தொடர்ந்து செயற்படுத்த, இந்த Exception Handling கட்டளைகள் பயன்படுத்தப்படுகின்றன.

இயக்க நேரப் பிழைகளைக் கையாளும் கட்டளைகளான try, catch, throw போன்றவற்றில், இயக்க நேரப் பிழைகள் ஏற்பட வாய்ப்புக் கள் உள்ள கட்டளைகளை try இற்குள் வரையறுக்க வேண்டும். இயக்க நேரப் பிழைகள் ஏற்பட்டால், அவற்றைக் கண்டுபிடித்து அறிவிப்பதற்கு throw என்ற கட்டளையையும், கண்டுபிடித்த பிழைகளைக் கையாள catch என்ற கட்டளையையும் வரையறுக்க வேண்டும்.

உதாரணமாக,

```
try
{
     .....;
     .....;
     throw ....;
     ....;
}
catch (...)
{
     ....;
     ....;
}
```

catch இற்கு ஆர்கியூமென்ற் (Argument) ஆக, அப்பிழையை எதிர் கொள்ளும் விபர இனத்தினைக் குறிப்பிட வேண்டும். இயக்க நேரப் பிழைகள் எதுவும் ஏற்படாத சந்தர்ப்பத்தில் catch இற்குள் எழுதப்பட்ட கட்டளைகள் எதுவும் செயற்படாது.

உதாரணமாக, இரண்டு எண்களை உள்ளீடாகக் கொடுத்து, அவற்றில் முதல் எண்ணை இரண்டாம் எண்ணால் வகுக்கும் போது கிடைக்கும் விடையினைத் திரையில் காட்டுவதற்குரிய புறோகிராமினைக் கருதுக.

```
#include <iostream.h>
void main()
{
    int a,b;
    cout<<"Enter two integer numbers : ";
    cin>>a>>b;
    cout<<a<<"/"<<b<<" = "<<a/b<<endl;
    cin.get();
}
```

இந்தப் புறோகிராமினை எழுதிக் கொம்பைல் செய்தால், எவ்வித பிழையுமின்றி ஒழுங்கான முறையில் கொம்பைல் செய்யப்படும். பின்னர், புறோகிராமினைச் செயற்படுத்தியவுடன் முதலில் இரண்டு இலக்கங்களை உள்ளீடு செய்யுமாறு கேள்வி கேட்கப்படும்.

உதாரணமாக 14, 3 என இரு முழு எண்களை உள்ளீடு செய்தால், விடையாக 14/3 = 4 எனத் திரையில் காணபிக்கும். உள்ளீடாக 4, 0 என இரு முழு எண்களைக் கொடுத்தால், இயங்கிக் கொண்டிருந்த புறோகிராம் இடைநடுவில் நின்றுவிடும்.

இவ்வாறான சந்தர்ப்பம் ஏற்படும் என முன்கூட்டியே புறோகிராமர் அறிந்து, இதற்குரிய நடவடிக்கை எடுத்திருக்க வேண்டும்.

இந்தப் புறோகிராமில், இரண்டு வகையான இயக்க நேரப் பிழைகள் ஏற்பட வாய்ப்புக்கள் உண்டு.

அவையாவன,

(1) இரண்டாம் இலக்கம் பூச்சியமாக உள்ளீடு செய்யும் போது ஏற்படும் பிழையாகும்.

(2) உள்ளீடு செய்யும் முழு எண்களுக்குப் பதிலாக எழுத்து அல்லது தசம எண் அல்லது எழுத்துக்கோவைகளை உள்ளீடு செய்யும் போது ஏற்படும் பிழையாகும்.

முதலாவதாகக் கூறப்பட்ட இயக்க நேரப் பிழை (Run Time Error) இனைக் கையாளுவதற்குரிய கட்டளைகளுடன், மேலேயுள்ள புறோகிராமினை, மீள எழுதிய புறோகிராம் கீழே தரப்பட்டுள்ளது.

```
#include <iostream.h>
void main()
{
```

```
        int a,b;
        cout<<"Enter the two numbers : ";
        cin>>a>>b;
        try
        {
            if (b == 0)
                throw b;
            else
                cout<<"a/b = "<<a/b<<endl;
        }
        catch(int)
        {
            cout<<"Cannot divided by zero! \n";
        }
        cin.get();
    }
```

புறோகிராம் ஒன்றில், ஒன்றுக்கு மேற்பட்ட இயக்க நேரப் பிழைகள் காணப்பட முடியும். இவ்வாறான சந்தர்ப்பத்தில், ஒரு try கட்டளைக்கு அடுத்து எத்தனை catch கட்டளைகளும் காணப்படலாம். ஒவ்வொரு இயக்க நேரப் பிழைக்கும், பிழையினைக் கையாளும் வகையில் கட்டளைகளை ஒவ்வொரு catch கட்டளைக்குள்ளும் எழுத வேண்டும்.

முழு எண், தசம எண், எழுத்து, எழுத்துக்கோவை போன்றவற்றை இனங்கண்டு செயற்படும் catch கட்டளைக்குரிய உதாரணப் புறோகிராமினைப் பார்ப்போம்.

```
    void main()
    {
        int x = 20;
        float y = 3.5;
        char ch ='A';
        char* name = "selva";
        int n;
        cout<<"Enter the option : ";
        cin>>n;
        try
        {
            switch(n)
            {
```

```
            case 1: throw x; break;
            case 2: throw y; break;
            case 3: throw ch; break;
            default: throw name;
        }
    }
    catch (int)
    {
        cout<<"Caught a Integer \n";
    }
    catch (double)
    {
        cout<<"Caught a Float \n";
    }
    catch (char)
    {
        cout<<"Caught a Character \n";
    }
    catch (char*)
    {
        cout<<"Caught a String \n";
    }
}
```

இந்தப் புறோகிராமினைச் செயற்படுத்தி, உள்ளீடாக 1 இனைக் கொடுத்தால், முழு எண்ணினைப் பராமீற்றராகக் கொண்ட catch இற்குரிய கட்டளைகள் செயற்படும். இவ்வாறு, உள்ளீடாக 2 இனைக் கொடுத்தால், தசம எண்ணினைப் பராமீற்றராகக் கொண்ட catch இற்குரிய கட்டளைகளும், உள்ளீடாக 3 இனைக் கொடுத்தால், எழுத்தினைப் பராமீற்றராகக் கொண்ட catch இற்குரிய கட்டளைகளும் செயற்படும். வேறெந்த முழு எண்ணை உள்ளீடு செய்தாலும், எழுத்துக்கோவையினைப் பராமீற்றராகக் கொண்ட catch இற்குரிய கட்டளைகள் செயற்படும். இறுதியாக எழுதப்பட்ட catch (char*) கட்டளைக்குப் பதிலாக, catch (...) என்ற மூன்று புள்ளிகள் உடைய கட்டளையினை எழுத முடியும்.

இயக்க நேரப் பிழைகளைக் கையாளுவதற்கு முதலில் try என்ற கட்டளை அமைப்பினையும், கண்டுபிடிக்கப்பட்ட பிழையினை எறிவதற்கு throw என்ற கட்டளையினையும், எறியப்பட்ட பிழையினைக் கையாளுவதற்கு catch என்ற கட்டளை அமைப்பினையும் பயன்படுத்தப்படுகின்றது.

இந்த அத்தியாயத்தில் இதுவரை எழுதிய உதாரணப் புறோகிராம் கள் அனைத்தும் ஒரு ∴பங்ஷனில் இடம்பெற்றுள்ளன. எனினும், இயக்க நேரப் பிழைகளைக் கையாளும் கட்டளைகளான try, throw, catch போன்றவற்றை ஒரு ∴பங்ஷனில் எழுதி, அவற்றை மெயின் ∴பங்ஷனில் அழைக்க முடியும்.

உதாரணமாக,

```
#include <iostream.h>
void divide(int x, int y)
{
    try
    {
        if (x == 0 && y == 0)
            throw 1;
        else if (y==0)
            throw 1.0;
        else
            cout<<"x/y = "<<x/y<<endl;
    }
    catch(int)
    {
        cout<<"Zero divide by zero is indetermined! \n";
    }
    catch(double)
    {
        cout<<"Cannot divided by zero! \n";
    }
}
void main()
{
    divide(40,20); // print 2
    divide(0,20);  // print 0
    divide(20,0); // catch the first error
    divide(0,0);  // catch the second error
}
```

இந்தப் புறோகிராமினைச் செயற்படுத்திப் பார்த்தால், வெளியீடானது கீழேயுள்ளவாறு காண்பிக்கப்படும்.

```
x/y = 2
x/y = 0
Cannot divided by zero!
Zero divide by zero is indetermined!
```

பிழைகளைக் கண்டறிந்த பின்னர், அதை எறிவதற்கு throw என்ற கட்டளை பயன்படுத்தப்படுகின்றது. இந்த throw என்ற கட்டளையினை மூன்று வகையாகப் புறோகிராம்களில் பயன்படுத்த முடியும்.

```
throw (exception);
throw exception; // without brackets
throw; // rethrowing an exception
```

எறியப்பட்ட இயக்க நேர பிழையினை catch என்ற கட்டளை மூலம் ஏந்திப் பிடிக்கப்படுகின்றது.

உதாரணமாக,

```
catch (type argument)
{
    // statements
}
```

பிடிக்கப்பட்ட இயக்க நேரப் பிழைகளை மீண்டும் எறிய முடியுமா? என அடுத்துப் பார்ப்போம்.

try இற்குள் எழுதப்பட்ட கட்டளைகளில் ஏற்பட்ட இயக்க நேரப் பிழைகள் throw என்ற கட்டளை மூலம் catch இற்குள் எறியப் படுகின்றது. எறியப்பட்ட பிழைகள் catch இற்குள் பிடித்துக் கையாளப்படுகின்றது. மீண்டும் இந்தக் catch இற்குள் இருந்து throw என்ற கட்டளை மூலம் மற்றுமொரு catch இற்குள் எறியப்படுகின்றது.

உதாரணமாக,

```
try
{
    throw "Exception thrown!";
}
catch (char*)
{
    cout<<"Caught1 ";
    throw;
}
catch (double)
{
    cout<<"Caught2 ";
}
```

இவ்வாறு இயக்க நேரப் பிழைகளை சி++ மொழியில் கையாள முடியும்.

11. நாமே உருவாக்கிக் கொள்ளும் ஹெடர் ஃபைல்கள் (User defined Header files)

சி++ மொழிக்குரிய கொம்பைலரினை உருவாக்கும் போதே, பல ஹெடர் ஃபைல்கள் இணைக்கப்பட்டு வெளியிடப்பட்டுள்ளன எனவும், அந்த ஹெடர் ஃபைல்கள் ஒவ்வொன்றும் பலதரப்பட்ட கட்டளைகளையும், ஃபங்ஷன்களையும் தன்னகத்தே கொண்டுள்ளன எனவும், அவ்வாறான ஹெடர் ஃபைல்களை எமது தேவைக்கு ஏற்றால்போல், எம்மால் உருவாக்க முடியும் எனவும் ஏற்கனவே முன்னைய அத்தியாயத்தில் கூறப்பட்டது நீங்கள் அறிவீர்கள்.

சி++ மொழிப் புறோகிராம்களை எழுதும்போது முதலில் iostream.h என்ற ஹெடர் ஃபைலினை அழைத்த பின்னர்தான் cout, cin போன்ற கட்டளைகளை புறோகிராமில் பயன்படுத்த முடியும். இந்த iostream.h என்ற ஹெடர் ஃபைல், சி++ கொம்பைலரில் இணைத்து வெளியிடப் பட்டுள்ளது. இவ்வாறு math.h, fstream.h, string.h, conio.h, ... போன்ற பல ஹெடர் ஃபைல்களையும் சி++ கொம்பைலரில் இணைத்து வெளியிட்டுள்ளது. இவ்வாறான ஹெடர் ஃபைல்களை எமது தேவைக்கு ஏற்றால்போல் எம்மால் உருவாக்க முடியும்.

உதாரணமாக, புறோகிராம்களில் அதிகம் தேவைப்படும் ஃபங்ஷன் களை ஒரு ஹெடர் ஃபைலில் எழுதிச் சேமித்து வைத்த பின்னர், அந்த ஹெடர் ஃபைலினை எமது புறோகிராம்களில் அழைத்து, அதில் காணப்படும் ஃபங்ஷன்களைப் பயன்படுத்த முடியும்.

சி++ மொழியில் எவ்வாறு ஹெடர் ஃபைல்கள் உருவாக்கப்படுகிறது என்பதனைப் பார்ப்போம்.

கோட்டினை வரையும் ஃபங்ஷன் ஒன்றை ஹெடர் ஃபைலில் எழுதி, இந்த ஃபங்ஷனை எவ்வாறு எமது புறோகிராமில் அழைப்பது என்பதனைப் புறோகிராம் ரீதியாகப் பார்ப்போம்.

```
#include <iostream.h>
void line()
{
    for (int i=1; i<=30; i++)
        cout<<"-";
    cout<<endl;
}
```

மேலேயுள்ள புறோகிராமினை ஹெடர் ஃபைலாக, சப் ஃபோல்டர் (Sub folder) include இனுள் அல்லது நாம் சேமிக்கும் சி++ மொழி

∴பைல்கள் உள்ள ∴போல்டரில் சேமிக்கவும். அதாவது, இந்தப் புறோகிராமினைச் சேமிக்கும் போது, ∴பைலின் பெயருடன் .h என்ற துணைப் பெயர் (Extension) ஐயும் இணைத்து, சப் ∴போல்டரான include இனுள் அல்லது நாம் சேமிக்கும் சி++ மொழி ∴பைல்கள் உள்ள ∴போல்டரில் சேமிக்கவும்.

உதாரணமாக, இந்த ஹெடர் ∴பைலை drawing.h என்ற பெயரில் சேமிக்கவும். இந்த ஹெடர் ∴பைலினை சப் ∴போல்டரான include இனுள் சேமித்திருந்தால், இந்த ஹெடர் ∴பைலினை புறோகிராமில் அழைக்கும் போது <, > போன்ற குறியீடுகளுக்கு இடையிலேயே குறிப்பிட வேண்டும். நாம் சேமிக்கும் சி++ மொழி ∴பைல்கள் உள்ள ∴போல்டரில் இந்த ஹெடர் ∴பைலினைச் சேமித்தால், “, ” போன்ற இரு மேற்கோல்குறிகளுக்கு இடையில் குறிப்பிட வேண்டும்.

மேலே எழுதப்பட்ட drawing.h என்ற ஹெடர் ∴பைலினை எமது புறோகிராமில் பயன்படுத்த வேண்டுமாயின், முதலில் இந்த drawing.h என்ற ஹெடர் ∴பைலினை எமது புறோகிராமில் அழைக்க வேண்டும். அதாவது, #include <drawing.h> அல்லது #include “drawing.h” என்ற கட்டளையினை முதலில் எழுத வேண்டும். பின்னர், இந்த ஹெடர் ∴பைலிலுள்ள line() என்ற ∴பங்ஷினை எமது புறோகிராமில் பயன்படுத்த முடியும்.

உதாரணமாக,

```
#include <iostream.h>
#include <drawing.h>
void main()
{
    cout<<"Welcome to my first header file \n";
    line();
    cout<<"Thank You, bye....";
    line();
    cin.get();
}
```

இவ்வாறு, நாம் எழுதிய ஹெடர் ∴பைலிலுள்ள ∴பங்ஷின்களைப் புறோகிராமில் பயன்படுத்த முடியும்.

நாம் எழுதும் ஹெடர் ∴பைல்கள் கொம்பைல் செய்யாது சேமிக்கப்படுகிறது என்பதால், இந்த ஹெடர் ∴பைல்களை மிகவும் அவதானமாக எழுத வேண்டும். இதனால்தான் சிலர் மெயின் ∴பங்ஷினிற்குள் ஹெடர் ∴பைல்களுக்குரிய கட்டளைகள் எழுதப்பட்ட

பின்னர், கொம்பைல் செய்து கட்டளை அமைப்பு பிழைகள் (Syntax Errors) எதுவும் இல்லை எனத் தெரிந்த பின்னர், மெயின் என்ற கட்டளை வரி அகற்றப்பட்டு ஹெடர் ∴பைலாகச் சேமிக்கின்றார்கள்.

நாம் எழுதும் ஹெடர் ∴பைல்களில், ஒன்றுக்கு மேற்பட்ட பல ∴பங்ஷன்களை உருவாக்க முடியும்.

உதாரணமாக, square(), cube(), fact(), power(), max(), min() போன்ற ∴பங்ஷன்களை mathematics என்ற ஹெடர் ∴பைலில் உருவாக்கி, ஒரு புறோகிராமில் எவ்வாறு இந்த ∴பங்ஷன்களைப் பயன்படுத்தப்படுகின்றன எனப் பார்ப்போம்.

```
double square(double n)
{
    return  n*n;
}
double cube(double n)
{
    return  n*n*n;
}
int fact(int n)
{
    int f = 1;
    if (n == 0)
        return 1;
    else
    {
        for (int i =1; i <= n; i++)
            f *= i;
        return f;
    }
}
int power(int n, int m)
{
    int p = 1;
    for (int i =1; i <= m; i++)
        p *= n;
    return p;
}
```

```
double max(double n, double m)
{
    return (n>m) ? n : m;
}
double min(double n, double m)
{
    return (n<m) ? n : m;
}
```

இந்தப் புரோகிராமினை எழுதி, ஹெடர் ஃபைலாகச் சேமிக்கவும். உதாரணமாக, mathematics.h என்ற பெயரில் நாம் சேமிக்கும் சி++ மொழி ஃபைல்கள் உள்ள ஃபோல்டரில் சேமிக்கவும்.

மேலே எழுதப்பட்ட mathematics.h என்ற ஹெடர் ஃபைலிலுள்ள ஃபங்ஷன்களை புரோகிராமில் பயன்படுத்த வேண்டுமாயின், முதலில் இந்த mathematics.h என்ற ஹெடர் ஃபைலினை புரோகிராமில் அழைக்க வேண்டும்.

உதாரணமாக,

```
#include "mathematics.h"
#include <iostream.h>
void main()
{
    double x = square(5);  // x = 25.0
    cout<<x<<endl;
    double y = cube(10);   // y = 1000.0
    cout<<y<<endl;
    double m1 = max(12,34);  // m1 = 34
    cout<<m1<<endl;
    double m2 = min(132,234);  // m2 = 132
    cout<<m2<<endl;
    int f = fact(5);  // f =  120
    cout<<f<<endl;
    int p = power(3,4);  // p = 81
    cout<<p<<endl;
}
```

இவ்வாறு ஹெடர் ஃபைல்களை எமது தேவைக்கு ஏற்றால்போல் எம்மால் உருவாக்கிப் புரோகிராம்களில் பயன்படுத்த முடியும்.

Appendix A

உதாரணப் புறோகிராம்களும், வெளியீடுகளும்

உதாரணம் (1):

1 தொடக்கம் நாம் உள்ளீடு செய்யும் நேர் முழு எண் வரையுள்ள எண்களின் கூட்டுத்தொகையினைக் கணிப்பதற்குரிய புறோகிராம் கீழே தரப்பட்டுள்ளது.

உதாரணமாக, 1 + 2 + 3 + ------- + n = ? இங்கு n என்பது நாம் உள்ளீடு செய்யும் நேர் முழு எண் ஆகும்.

```
#include <iostream.h>
void main()
{
    int n;
    long sum = 0;
    cout<<"Enter a positive number: ";
    cin>>n;
    for (int i = 1; i <= n; i++)
        sum += i;
    cout<<"Sum of 1 to "<<n<<"numbers = "<<sum<<endl;
    cin.get();
}
```

வெளியீடுகள் (Outputs) :

Enter a positive number : 50 (50 உள்ளீடு செய்யப்படுகிறது)

Sum of 1 to 50 numbers = 1275

உதாரணம் (2):

1 தொடக்கம் நாம் உள்ளீடு செய்யும் நேர் முழு எண் வரையுள்ள எண்களின் வர்க்கங்களின் கூட்டுத்தொகையினைக் கணிப்பதற்குரிய புறோகிராம் கீழே தரப்பட்டுள்ளது.

உதாரணமாக, 1 + 4 + 9 + ------- + n*n = ? இங்கு n என்பது நாம் உள்ளீடு செய்யும் நேர் முழு எண் ஆகும்.

```
#include <iostream.h>
void main()
{
    int n;
    long sum = 0;
    cout<<"Enter a positive number: ";
    cin>>n;
```

```
    for (int i = 1; i <= n; i++)
        sum += i*i;
    cout<<"Sum of square of 1 to "<<n<<"= "<<sum;
    cin.get();
}
```

வெளியீடுகள் (Outputs) :

Enter a positive number : 20 (20 உள்ளீடு செய்யப்படுகிறது)

Sum of square of 1 to 20 = 2870

உதாரணம் (3):

ஒரு இருபடிச் சமன்பாடுக்குத் தீர்வு உண்டா அல்லது தீர்வு இல்லையா என்பதற்குரிய புறோகிராம் கீழே தரப்பட்டுள்ளது.

உதாரணமாக, $aX^2 + bX + c = 0$ என்ற இருபடிச் சமன்பாட்டில்,

$b^2 - 4*a*c >= 0$ எனின், தீர்வுகள் உண்டு.

$b^2 - 4*a*c < 0$ எனின், உண்மைத் தீர்வுகள் இல்லை.

```
#include <iostream.h>
void main()
{
    int  a, b, c;
    double delta;
    cout<<"aX² + bX +c = 0 "<<endl;
    cout<<"Enter the values for a, b and c : ";
    cin>>a>>b>>c;
    delta = b*b - 4*a*c;
    if (delta >= 0 )
        cout<<"There are two real roots "<<endl;
    else
        cout<<"There are no real roots"<<endl;
    cin.get();
}
```

வெளியீடுகள் (Outputs) :

$aX^2 + bX + c = 0$

Enter the values for a, b and c :2 5 3

(2, 5, 3 உள்ளீடு செய்யப்படுகிறது)

There are two real roots

உதாரணம் (4):

e^x இற்குரிய பெறுமானத்தினைக் கணிக்கும் புறோகிராம் கீழே தரப்பட்டுள்ளது.

உதாரணமாக, $e^x = 1 + x + x^2/2! + x^3/3! + \text{------} + x^n/n! = ?$

இங்கு n, x என்பது முறையே நாம் உள்ளீடு செய்யும் நேர் முழு எண், நேர் தசம எண்கள் ஆகும்.

```
#include <iostream.h>
int fact(int n)
{
    int f = 1;
    if (n == 0)
        return 1;
    else
    {
        for (int i =1; i <= n; i++)
            f *= i;
        return f;
    }
}
int power(int n, int m)
{
    int p = 1;
    for (int i =1; i <= m; i++)
        p *= n;
    return p;
}
void main()
{
    int  n;
    double x, tot = 0;
    cout<<" e to the power x "<<endl;
    cout<<"Enter the x and n values : ";
    cin>>x>>n;
    for (int i = 0; i <= n; i++)
        tot += power(x,i)/fact(i);
    cout<<" e to the power "<<x<<" = "<<tot<<endl;
    cin.get();
}
```

வெளியீடுகள் (Outputs) :

e to the power x

Enter the x and n values : 2.5 9 (2.5, 9 உள்ளீடு செய்யப்படுகிறது)

e to the power 9 = 6

உதாரணம் (5):

கோசைன் (cos) இற்குரிய பெறுமானத்தினைக் கணிக்கும் புறோகிராம் கீழே தரப்பட்டுள்ளது.

உதாரணமாக, $\cos(x) = 1 - x^2/2! + x^4/4! - ---+(-1)^n * x^{2n}/(2n)! = ?$

இங்கு n, x என்பது முறையே நாம் உள்ளீடு செய்யும் நேர் முழு எண், நேர் தசம எண்கள் ஆகும்.

```
#include <iostream.h>
int fact(int n)
{
    int f = 1;
    if  (n == 0)
        return 1;
    else
    {
        for (int i =1; i <= n; i++)
            f *= i;
        return f;
    }
}
double power(int n, int m)
{
    double p = 1;
    for (int i =1; i <= m; i++)
        p *= n;
    return p;
}
void main()
{
    int  n;
    double x, tot = 0;
    cout<<" cos(x) "<<endl;
    cout<<"Enter the x and n values : ";
    cin>>n;x;
    for (int i = 0; i <= n; i += 2)
        tot += power(-1,i)*(power(x,i)/fact(i));
    cout<<"cos("<<x<<") = "<<tot<<endl;
    cin.get();
}
```

வெளியீடுகள் (Outputs) :

e to the power x

Enter the x and n values : 30 10 (30, 10 உள்ளீடு செய்யப்படுகிறது)

cox (30) = 0.5

உதாரணம் (6):

எமது நாட்டில் பயன்படுத்தப்படும் அடையாள அட்டை (Identity Card) இலக்கத்தினை உள்ளீடு செய்தால், அந்த அடையாள அட்டைக் குரியவரின் பிறந்த தினத்தினைக் கணிப்பதற்குரிய புறோகிராம் கீழே தரப்பட்டுள்ளது.

உதாரணமாக, 3856312910V என்ற அடையாள அட்டை (Identity Card) இலக்கத்தில், இறுதியாகவுள்ள எழுத்துத் தவிர்ந்த மற்றைய இலக்கத்தினை உள்ளீடு செய்தால், 38 ஆம் ஆண்டு, மார்ச் மாதம், 3 ஆம் திகதி எனத் திரையில் காண்பிக்கும்.

```
#include <iostream.h>
void main()
{
    long n;
    cout<<"Enter your national IC number : ";
    cin>>n;
    int year = n/10000000; // year calculation
    n = n%10000000; // calculate remainder
    int a = n/10000;
    char *sex;
    int days;
    if (a>500)
    {
        sex="Female";
        a-=500;// female's IC number
    }
    else
        sex="Male";
    char *month;
    if (a<=31)
    {
        month ="January";
        days = a;
    }
```

```
else if (a<=60)
{
        month ="February";
        days = a-31;
}
else if (a<=91)
{
        month ="March";
        days = a-60;
}
else if (a<=121)
{
        month ="April";
        days = a-91;
}
else if (a<=152)
{
        month ="May";
        days = a-121;
}
else if (a<=182)
{
        month ="June";
        days = a-152;
}
else if (a<=213)
{
        month ="July";
        days = a-182;
}
else if (a<=244)
{
        month ="August";
        days = a-213;
}
else if (a<=274)
{
        month ="September";
        days = a-244;
}
```

```
        else if (a<=305)
        {
                month ="October";
                days = a-274;
        }
        else if (a<=335)
        {
                month ="November";
                days = a-305;
        }
        else if (a<=366)
        {
                month ="December";
                days = a-335;
        }
        cout<<"Your date of birth is ";
        cout<<days<<" "<<month<<" 19"<<year;
        cout<<" ("<<sex<<")"<<endl;
        cin.get();
    }
```

வெளியீடுகள் (Outputs) :

```
        Enter your national IC number : 543312343
         (543312343 உள்ளீடு செய்யப்படுகிறது)
        Your date of birth is 26 November 1954 (Male)
```

உதாரணம் (7):

ஆண்டினை உள்ளீடு செய்தால், அந்த ஆண்டிற்குரிய நாள் காட்டியினை காணப்பிப்பதற்குரிய புறோகிராம் கீழே தரப்பட்டுள்ளது.

உதாரணமாக, 1983 ஆம் ஆண்டினை உள்ளீடு செய்தால், இந்த ஆண்டுக்குரிய நாள்காட்டியினைக் காண்பிக்கும்.

```
    // Calendar Program
    #include <iostream.h>
    void main()
    {
        int m = 1; // month
        int y; // year
        cout<<"Enter the year : ";
        cin>>y;
        int day = y + (y/4) - (y/100) + (y/400);
        if ((y%4 == 0) && ((y%100 != 0) || (y%400 == 0)))
            day = day - 1;
```

```
while (m<=12)
{
    switch (m)
    {
        case 1 : cout<<"January - "<<y; break;
        case 2 : cout<<"February - "<<y; break;
        case 3 : cout<<"March - "<<y; break;
        case 4 : cout<<"April - "<<y; break;
        case 5 : cout<<"May - "<<y; break;
        case 6 : cout<<"June - "<<y; break;
        case 7 : cout<<"July - "<<y; break;
        case 8 : cout<<"August - "<<y; break;
        case 9 : cout<<"September - "<<y; break;
        case 10 : cout<<"October - "<<y; break;
        case 11 : cout<<"November - "<<y; break;
        case 12 : cout<<"December - "<<y; break;
    } // End of swich statement
    int i=1;
    cout<<"\nSun Mon Tue Wed Thu Fri Sat"<<endl;
    while (i<=(day % 7))
    {
        cout<<"    "; // give 4 space
        i++;
    }
    int a = 0;
    if((m==1)||(m==3)||(m==5)||(m==7)||(m==8)||(m==10)||(m==12))
        a = 31;
    else if ((m==4) || (m==6) || (m==9) || (m==11))
        a = 30;
    else if ((y%4==0) && ((y%100 !=0) || (y%400==0)))
        a = 29;
    else
        a = 28;
    i = 1;
    while (i <= a)
    {
```

```
            if (i<10)
                cout<<i<<"   "; // give 3 space
            else
                cout<<i<<"  ";  // give 2 space
            if ((i+day)%7 ==0)
                cout<<endl;
            i++;
        } // end of while loop
        cout<<"Press enter key display next month calendar \n";
        cin.get();
        day +=a;
        m++;
        } //  end of the first while loop
        cout<<"\n End of this year calendar";
        cin.get();
    } // End of main function
```

வெளியீடுகள் (Outputs) :

Enter the year : 1983 (1983 உள்ளீடு செய்யப்படுகிறது)

1983 ஆம் ஆண்டுக்குரிய நாள்காட்டியினைக் காண்பிக்கும்.

உதாரணம் (8):

உள்ளீடாகக் கொடுக்கப்பட்ட 10 இலக்கங்களில் மிகச்சிறிய, மிகப்பெரிய இலக்கத்தினைக் கண்டுபிடிப்பதற்குரிய புறோகிராம் கீழே தரப்பட்டுள்ளது.

```
// calculate the maximum and minimum number
#include <iostream.h>
void main()
{
    const int LEN=10;
    int num[LEN];
    cout<<"Input the "<<LEN<<" integer numbers :"<<endl;
    for (int i=0;i<LEN; i++)
    {
        cout<<"Enter the numer "<<i+1<<": ";
        cin>>num[i];
    }
    int max=num[0], min=num[0];
```

```
    for (i=1; i<LEN; i++)
    {
        if (num[i]>max)
            max=num[i];
        if (min>num[i])
            min=num[i];
    }
    cout<<"Maximum number = "<<max<<endl;
    cout<<"Minimum number = "<<min<<endl;
}
```

வெளியீடுகள் (Outputs) :

```
Input  the 10 integer numbers :
Enter the number 1 : 134
Enter the number 2 : 324
Enter the number 3 : 1334
Enter the number 4 : 3234
Enter the number 5 : 314
Enter the number 6 : 64
Enter the number 7 : 3564
Enter the number 8 : 3894
Enter the number 9 : 6734
Enter the number 10: 3784
```

(134, 324, 1334, 3234, 314, 64, 3564, 3894, 6734, 3784 உள்ளீடு செய்யப்படுகிறது)

```
Maximum number = 6734
Minimum number = 64
```

Appendix B

அஸ்க்கீ அட்டவணை (ASCII Table)

ASCII Characters	decimal	ASCII Characters	decimal
(NUL)	00	#	35
☺ (SOH)	01	$	36
☻ (STX)	02	%	37
♥ (ETX)	03	&	38
♦ (EOT)	04	'	39
♣ (ENQ)	05	(	40
♠ (ACK)	06	)	41
• (BEL)	07	*	42
● (BS)	08	+	43
(HT)	09	,	44
(LF)	10	-	45
♂ (VT)	11	.	46
♀ (FF)	12	/	47
♪ (CR)	13	0	48
♫ (SO)	14	1	49
☼ (SI)	15	2	50
► (DLE)	16	3	51
◄ (DC1)	17	4	52
↕ (DC2)	18	5	53
‼ (DC3)	19	6	54
¶ (DC4)	20	7	55
§ (NAK)	21	8	56
▬ (SYN)	22	9	57
↨ (ETB)	23	:	58
↑ (CAN)	24	;	59
↓ (EM)	25	<	60
→ (SUB)	26	=	61
← (ESC)	27	>	62
(cursor right) (FS)	28	?	63
(cursor left) (GS)	29	@	64
(cursor up) (RS)	30	A	65
(cursor down)(US)	31	B	66
(SP)	32	C	67
!	33	D	68
"	34	E	69

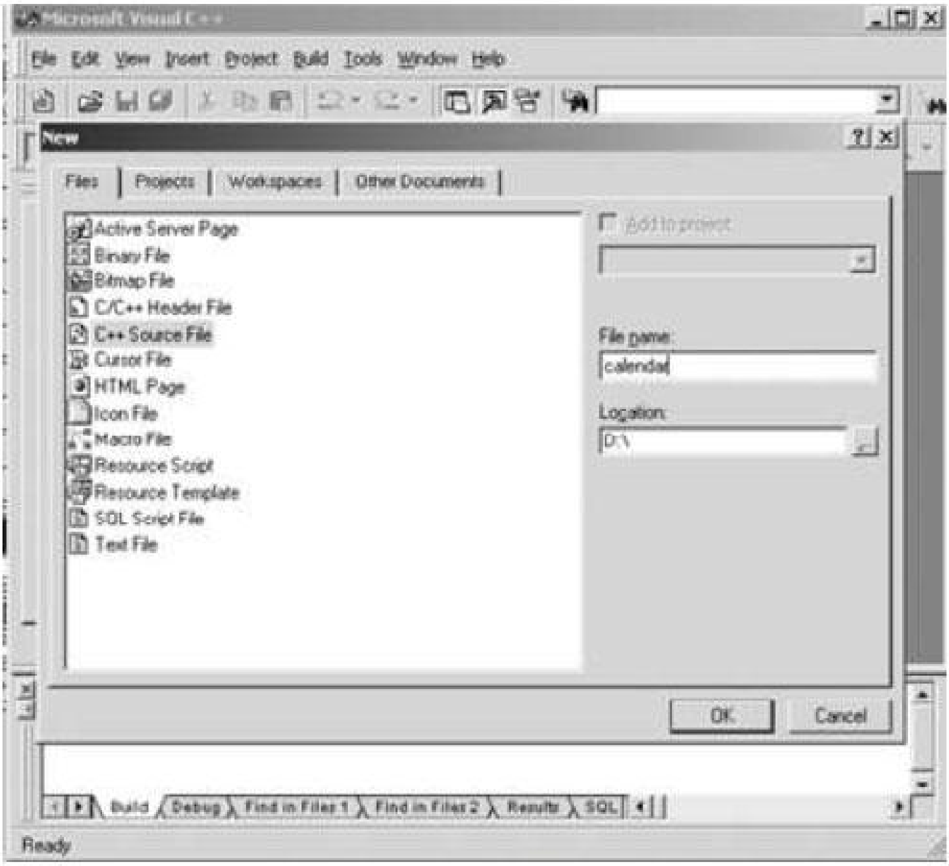
ASCII Characters	Decimal	ASCII Characters	Decimal	ASCII Characters	Decimal
F	70	i	105	î	140
G	71	j	106	ì	141
H	72	k	107	Ä	142
I	73	l	108	Å	143
J	74	m	109	É	144
K	75	n	110	æ	145
L	76	o	111	Æ	146
M	77	p	112	Ô	147
N	78	q	113	ö	148
0	79	r	114	Ò	149
P	80	s	115	û	150
Q	81	t	116	ù	151
R	82	u	117	ÿ	152
S	83	v	118	Ö	153
T	84	w	119	ü	154
U	85	x	120	¢	155
V	86	y	121	£	156
W	87	z	122	¥	157
X	88	{	123	₧	158
Y	89	|	124	ƒ	159
Z	90	}	125	á	160
[	91	~	126	í	161
\	92	(DEL)	127	ó	162
]	93	Ç	128	Ú	163
^	94	Ü	129	^	164
_	95	é	130	Ñ	165
`	96	â	131	ª	166
a	97	ä	132	º	167
b	98	à	133	¿	168
c	99	å	134	⌐	169
d	100	ç	135	¬	170
e	101	ê	136	½	171
f	102	ë	137	¼	172
g	103	è	138	¡	173
h	104	ï	139	«	174

ASCII Characters	Decimal	ASCII Characters	Decimal	ASCII Characters	Decimal
»	175	╦	203	µ	230
░	176	╠	204	γ	231
▒	177	═	205	ϕ	232
▓	178	╬	206	θ	233
│	179	╧	207	Ω	234
┤	180	╨	208	δ	235
╡	181	╤	209	∞	236
╢	182	╥	210	Ø	237
╖	183	╙	211	∈	238
╕	184	╘	212	∩	239
╣	185	╒	213	≡	240
║	186	╓	214	±	241
╗	187	╫	215	≥	242
╝	188	╪	216	≤	243
╜	189	┘	217	⌠	244
╛	190	┌	218	⌡	245
┐	191	█	219	÷	246
└	192	▄	220	≈	247
┴	193	▌	221	°	248
┬	194	▐	222	•	249
├	195	▀	223	·	250
─	196	α	224	√	251
┼	197	β	225	η	252
╞	198	Γ	226	²	253
╟	199	π	227	▪	254
╚	200	Σ	228	(SP)	255
╔	201	σ	229		
╩	202				

Appendix C

விசுவல் சி++ எடிற்ரில், எவ்வாறு சி++ மொழிப் புறோகிராம்களைச் செயற்படுத்துவது எனப் பார்ப்போம்.

முதலில், விசுவல் சி++ இனைச் செயற்படுத்த வேண்டும். அங்கு மெயின் மெனுவிலுள்ள File இல் New... இனைத் தெரிவு செய்தால், படம்-1 இல் உள்ளது போல் பொப்-அப் வின்டோ (Pop up Window) தோன்றும். அதில், File இல் C++ Source File இனைத் தெரிவு செய்து,

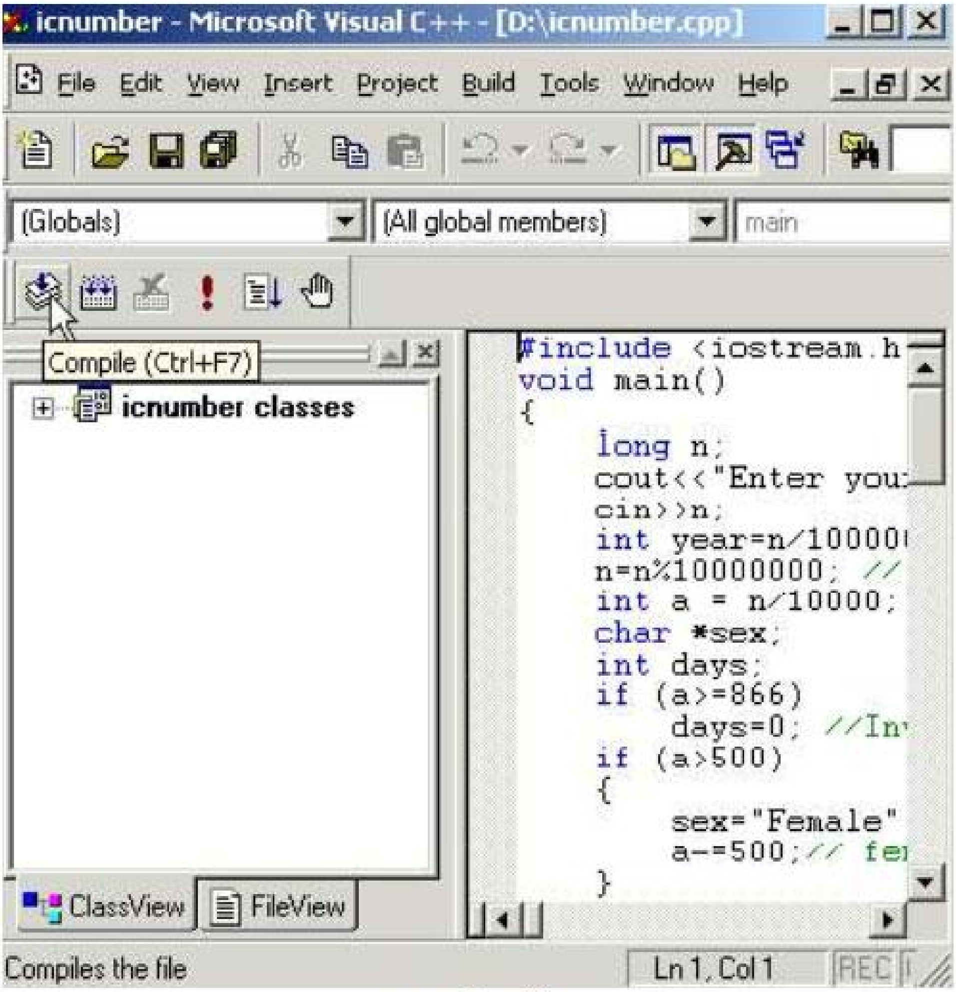


படம் - 1

File name என்ற பகுதியில் சி++ மொழிப் புறோகிராமிற்குரிய பெயரினைக் கொடுக்க வேண்டும். அந்த சி++ மொழிப் புரோகிராம் சேமிக்கும் இடத்தினை Location என்ற பகுதியில் தெரிவு செய்ய வேண்டும். பின்னர், Ok இனை அழுத்த வேண்டும்.

படம்-2 இல் உள்ளது போல் எடிற்ரர் தோன்றும். அந்த எடிற்ரில் சி++ மொழிப் புறோகிராமினை எழுத வேண்டும்.

எழுதப்பட்ட புரோகிராமினைக் கொம்பைல் செய்ய வேண்டும். விசுவல் சி++ எடிற்ரரில் கொம்பைல் செய்ய வேண்டுமாயின், படம்-2 இல் காட்டியது போல் ரூல் பாரில் (Tool Bar) உள்ள Compile இனை அல்லது Ctrl உடன் F7 என்ற கீயினை அழுத்த வேண்டும்.

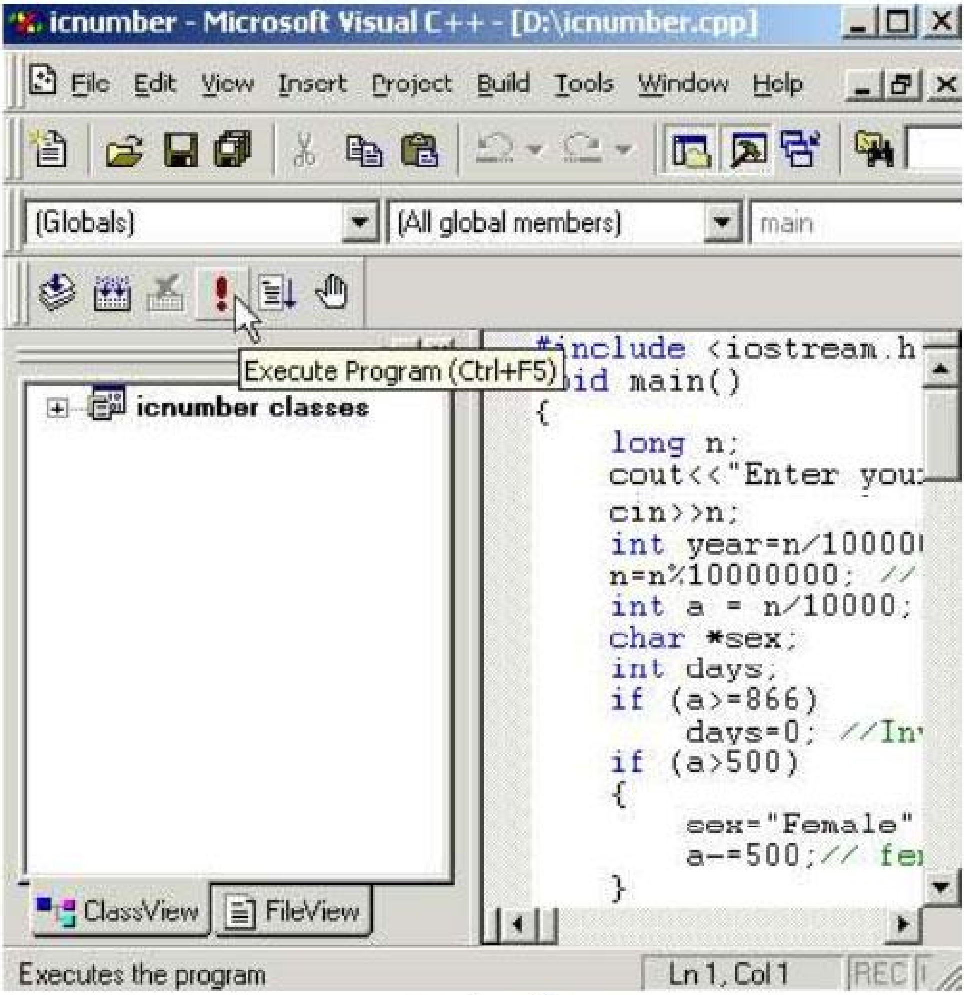


படம் - 2

எழுதப்பட்ட புரோகிராமில், ஏதாவது கட்டளை அமைப்புப் பிழைகள் (Syntax Errors) காணப்பட்டால், அவற்றினைத் தெளிவான முறையில் காண்பிக்கும்.

எழுதப்பட்ட புரோகிராமில், கட்டளை அமைப்புப் பிழைகள் எதுவுமில்லையென்றால், அந்தப் புரோகிராமினைச் செயற்படுத்த முடியும். கொம்பைல் செய்யப்பட்ட புரோகிராமினை விசுவல் சி++

எடிற்றரில் செயற்படுத்த வேண்டுமாயின், படம் - 3 இல் காட்டியது போல் ரூல் பாரில் (Tool Bar) உள்ள Execute Program இனை அல்லது Ctrl உடன் F5 என்ற கீயினை அழுத்த வேண்டும்.



படம் - 3

மேலே கூறப்பட்ட முறையினைப் பயன்படுத்துவதன் மூலம், விசுவல் சி++ எடிற்றரில் சி++ மொழிப் புறோகிராம்களைச் செயற்படுத்த முடியும்.

Index

A

abstract class 168
arithmetic operators 37
arrays 82
ASCII tables 211
assignment Operators 36

B

bitwise operators 42
break 61

C

case 50
casting 25
char 22
classes 106
class diagram 107
comments 13
compiler 5
compiler time polymorphism 147
constants 19
constructors 118
continue 62
control statements 46

D

data types 21
destructors 120
delete 98
do ... while 60
default 23
define 24
double 22
dynamic arrays 100

E

early binding 147
encapsulation 125
enum 102
else 46
exception 190

F

file stream 180
float 22
for loop 55
friend classes ... 169
friend functions 168
functions 65
function overloading 148
function overriding ... 165

G

global variable 27
goto 63

H

header files 197
hierarchical inheritance 131

I

if 46
if ... else 49
inline function 77
int 22
iteration statements 55

J

jump statements 61

K

keywords 20

L

late binding 163
left shift 43
library functions 66
local variable 26
logical operators 41
long 22

M

macro functions 78
multi-level inheritance 129
multiple inheritance 130

N

new 98

O

objects 112
operator overloading 154
operators 36
overloading 148
overriding 165

P

parameters 148
pointers 96
polymorphism 147
private 109
protected 109
public 109
pure virtual functions 166

R

recursive functions 75
relational operators 40
right shift 43
runtime polymorphism 163
return 61

S

scope resolution operator 27
selection statements 46
short 22
signed 24
sizeof 29
static 81
string 68
single inheritance 126
structures 91
switch 50

T

templates 171
template classes 177
template functions 171
ternary operator 45
this 124
translators 5
typedef 29

U

unary 156
union 102
unsigned 23
user defined functions 170

V

variables 19
virtual 164
void 73

W

while 58

நன்றியுரை

"நன்றி மறப்பது நன்றன்று"

"தகவல் தொழில்நுட்பக் கல்வியினை ஆங்கில அறிவுள்ளவர்கள் மட்டுமே கற்க முடியும்" என்ற கருத்தினைக் களைய வேண்டும் என்பதன் ஓர் முயற்சியாகவே இந்த சி++ மொழி நூலைத் தமிழில் எழுதியுள்ளேன்.

இந்நூலை வெளியிடுவதில் பல நடைமுறைச் சிக்கல்கள் இருந்த போதும், தமிழ் பேசும் மாணவர்களுக்குப் பயன்பட வேண்டும் என்ற ஒரே நோக்கத்திற்காக வெளியிட்டுள்ளேன். இந்நூலை வெளியிடுவதற்கு உதவி செய்த அனைவருக்கும் எனது இதயம் கனிந்த நன்றிகள். குறிப்பாக, இந்நூலினைச் சரவை பார்த்து உதவி செய்த கொழும்புப் பல்கலைக்கழக மாணவனான தி. ஜெகநாதனுக்கு எனது நன்றிகள்.

எனது கணினி அறிவினை எழுத்து மூலமாக வெளிக் கொண்டு வர உதவிய இலங்கைத் தேசிய கணினிச் சஞ்சிகைகளான **"Computer Today"** இற்கும், **"Computer World"** இற்கும் எனது நன்றிகள்.

மேலும் இந்த சி++ மொழி நூலினை எனது சொந்த மண்ணில் வெளியிட உதவி செய்த யாழ் நகரிலுள்ள பிரபல்யமான கணினிக் கல்விநிறுவனமான **IIS** இற்கும் எனது நன்றிகள்.

- ஆசிரியர் -

www.nselva.com

ISBN 955-97996-0-6

9 789559 799603

Rs. 250/=