C++ in Hindi

BccFalna.com
097994-55505

Kuldeep Chand

C with Class is C++. It means, without understanding Object Oriented Programming System (OOPS) Concepts properly, no one can understand C++ and any other Modern Programming Language like Java, C#, VB.Net, etc…

So, In this EBook, I have covered each and every concept related to OOPS and I have tried to Implement each OOPS Concept with easy to understand Demo C++ Program.

In this EBook, I have defined more then 350 Programs and hundreds of Code Fragement to clear each and every C++ and OOPS Concept.

So, This EBook would not only be easy to learn OOPS and C++ but also very useful if you are interested in learning Java or .NET Language like C# or VB.NET.

C++

In Hindi

Kuldeep Chand

BetaLab Computer Center
Falna

C++ Programming Language in Hindi

Lead Editors: **Kuldeep Chand**

Distributed to the book trade worldwide by Betalab Computer Center, Behind of Vidhya Jyoti School, Falna Station Dist. Pali (Raj.) Pin 306116

e-mail bccfalna@gmail.com,

or

visit http://www.bccfalna.com

For information on translations, please contact Betalab Computer Center, Behind of Vidhya Jyoti School, Falna Station Dist. Pali (Raj.) Pin 306116

Phone **097994-55505**

This book is dedicated to those

who really wants to be

a

PROFESSIONAL DEVELOPER

INDEX OF CONTENTS

Table of Contents

OOPS

&

C++

OOPS and C++

सबसे पहला सवाल तो यही है कि C++ क्यों सीखा जाए? ये आज की प्रभावशाली भाषा है। जब Programmers को बडे व जटिल प्रोग्राम बनाने होते हैं, तब Professional Programmers C++ को Choose करते हैं। कई और भी सरल व प्रभावी भाषाएं हैं, लेकिन उनकी कुछ कमियों की वजह से उनमें प्रोग्राम Development की एक सीमा है।

जैसे Visual Basic Microsoft Company की एक बहुत ही सरल भाषा है, लेकिन जब प्रोग्राम बहुत ही बडे व जटिल होते हैं, तब इस भाषा में Program Develop करना समझदारी की बात नहीं होती है। क्योंकि इस भाषा में Graphics का बहुत प्रयोग होता है, और भी कई कारण हैं, जिससे यदि इस भाषा में बडे प्रोग्राम Develop किए जाएं तो प्रोग्राम की Speed बहुत कम हो जाती है। Assembly भाषा भी काफी अच्छी है। इसमें लिखे गए प्रोग्रामों की गति काफी अच्छी होती है, लेकिन ये भाषा Hardware के Device Drivers के प्रोग्राम लिखने के लिये ज्यादा अच्छी है ना कि Application प्रोग्राम लिखने के।

इसी तरह Java Internet के लिये अच्छी है, हालांकि Java C++ से ही प्रेरित है। लेकिन एक बडे Standalone Application Development करने के लिये C++ सबसे लोकप्रिय भाषा है। ये एक बहुत ही Flexible व Best Performing Language है।

Procedural Languages

Pascal, C, Basic, Fortran जैसी पारम्परिक भाषाएं Procedural Languages के उदाहरण हैं। जिसमें प्रत्येक Statement Computer को कुछ काम करने का आदेश देता है। यानी Procedural Languages Instructions का एक समूह होता है। Procedural Languages में छोटे Programs के लिये किसी भी अन्य प्रकार के Pattern की आवश्यकता नही होती है। Programmer Instructions की List बनाता है और Computer उनके अनुसार काम करता है।

जब प्रोग्राम काफी बडे व जटिल हो जाते हैं, तब Instructions की यह List काफी परेशानी पैदा करती है। इसलिये एक बडे प्रोग्राम को छोटे–छोटे टुकडों में बांट दिया जाता है। इन छोटे–छोटे टुकडों को Functions कहा जाता है। Functions को दूसरी अन्य भाषाओं में Subroutine, Subprogram या Procedure कहा जाता है।

एक बडे प्रोग्राम को छोटे–छोटे Functions में विभाजित करने से पूरा Program Functions का एक समूह बन जाता है, जिसे **Module** कहा जाता है।

लेकिन ये Modules भी Procedural Programming के अन्तर्गत ही आते हैं क्योंकि सभी Functions में Statements की एक List होती है और सभी Functions मिल कर पूरा Program बनाते हैं, जिससे पूरा Program Instructions की एक बहुत बडी List बन जाता है।

Procedural Languages के शुरूआती दौर में इनमें ही Program Develop किए जाते थे। “C” भी एक Procedural Languages है और जब “C” भाषा का आविष्कार हुआ था, तब Programmers अन्य भाषाओं को छोड कर “C” में ही अपने Program Develop करने लगे थे।

लेकिन समय व आवश्यकता के अनुसार जब Program बडे व जटिल होने लगे, तब Programmers को इस भाषा में प्रोग्राम बनाने में दिक्कतें आने लगीं। उन्होने महसूस किया कि इस भाषा में कुछ सुधार की आवश्यकता है ताकि ये भाषा सरल व लोकप्रिय बन सके।

ये भाषा सरल बन सके इसके लिये इसका वास्तविक जीवन के अनुसार होना जरूरी था। यानी हम हमारे सामान्य जीवन में जिस प्रकार से व्यवहार करते हैं, इस भाषा का भी वैसा ही होना जरूरी था ताकि Programmers इसमें अधिक सरलता व सफलता से Program बना सकें।

भाषा वास्तविक जीवन के अनुसार हो, यही Concept Object Oriented Programming यानी OOPS का आधार बना। "C" भाषा की इन कमियों को पहचाना गया और इसमें सुधार किया गया। फलस्वरूप हमें "C" भाषा का एक नया संस्करण "C++" प्राप्त हुआ। आइयें, हम भी जानने की कोशिश करते हैं कि "C" भाषा में ऐसी कौनसी कमियां थीं, जिनमें सुधार की आवश्यकता महसूस की गई ?

Procedural Languages में काम होने का महत्व था Data का नहीं, यानी कि Keyboard से Data Input किया जाए, Data पर Processing की जाए, Errors को Check किया जाए आदि। Functions में भी इसी महत्व को जारी रखा गया। Functions कोई काम करते हैं, उसी प्रकार से जिस प्रकार से साधारण Statement करता है। Functions कोई जटिल काम भी कर सकते हैं, लेकिन इनमें भी काम के होने का ही महत्व था।

पूरे Program में Data पर कोई ध्यान नहीं दिया जाता था, जबकि पूरे प्रोग्राम का मूल आधार Data ही होता है। किसी Inventory के Program में इस बात का कोई ज्यादा महत्व नहीं होता है कि Data को किस प्रकार से Display किया जाता है या एक Function किस प्रकार से Corrupt Data को Check करता है, बल्कि इस बात का होता है कि Data क्या है और वह किस प्रकार से Program में काम आ रहा है। Procedural Program में Data को द्वितीय स्तर पर रखा गया था जबकि किसी भी Program का मूल आधार Data ही होता है।

उदाहरण के लिये, किसी Inventory के Program में किसी Data File को Memory में Load किया जाता है, तब ये File एक Global Variable की तरह होती है, जिसे कोई भी Function Use कर सकता है। ये Functions Data पर विभिन्न प्रकार के Operations करते हैं। यानी ये Data को Read करते है, Analyze करते हैं, Update करते हैं, Rearrange करते हैं, Display करते हैं और वापस Disk पर Write करते हैं। "C" में Local Variables भी होते हैं लेकिन Local Variables, महत्वपूर्ण Data के लिये इतने उपयोगी नहीं होते हैं, जो कि विभिन्न Functions द्वारा Access किए जाते हैं।

मान लो कि एक नए Programmer को Data को किसी खास तरीके से Analyze करने के लिये एक Function लिखने को कहा गया। प्रोग्राम की गूढता से अनभिज्ञ Programmer एक ऐसा Function बनाता है जो कि अचानक किसी महत्वपूर्ण Data को नष्ट कर देता है। ऐसा होना काफी आसान है क्योंकि कोई भी Function Data को Access कर सकता है, इसलिये क्योंकि Procedural Language में Data Global होता है। ये कुछ ऐसा ही है जैसे कि आप अपने Personal कागजात को Telephone Directory के पास रख दें जहां कभी भी कोई भी पहुंच सकता है, उससे छेडछाड कर सकता है और उसे नष्ट कर सकता है। इसी प्रकार से Procedural Languages में होता है जहां आपका Data Global होता है और कोई भी Function उसे Use करके खराब कर सकता है या नुकसान पहुंचा सकता है।

Procedural Languages की दूसरी कमी ये थी कि कई Functions एक साथ एक ही Data को Use कर रहे होते हैं, इसलिये Data को Store करने का तरीका काफी जटिल हो जाता है। समान Data को Use कर रहे सभी Functions को Modify किए बिना Data में किसी प्रकार का कोई परिवर्तन नहीं किया जा सकता है। उदाहरण के लिये यदि आप एक नया Data Add करते हैं तो उन सभी Functions को Modify करना होगा जो कि Data को use कर रहे हैं ताकि ये सभी Functions Add किए गए नए Data को Use कर सकें। ये पता करना कि कौन–कौन से

Function Data को Use कर रहे हैं और सभी को बिल्कुल सही तरीके से Modify करना काफी कठिन होता है।

Procedural Programs को Design करना काफी मुश्किल होता है। समस्या ये होती है कि इनका Design वास्तविक जीवन से Related नहीं होता है। जैसे कि, माना आप एक Graphics User Interface में Menus Window के लिये Code लिखना चाहते हैं, तो आपको ये तय करना मुश्किल होगा कि कौनसा Function Use किया जाए? कौनसा Data Structure Use किया जाए? आदि। इनका कोई स्पष्ठ उत्तर नहीं है।

Procedural Programs के साथ कई और परेशानियां हैं। उनमें से एक समस्या नए Data Type की है। Computer Languages में कई प्रकार के Built-in Data Types होते हैं, जैसे कि Integer, Float, Character आदि। मानलो कि आप Complex Numbers के साथ प्रक्रिया करना चाहते हैं या Two-dimensional Coordinates के साथ काम करना चाहते हैं या Date के साथ प्रक्रिया करना चाहते हैं।

Built-in Data Type इनको आसानी से Handle नहीं कर सकते हैं। इसलिए हमें हमारी आवश्यकतानुसार स्वयं के Data Type बनाने की जरूरत होती है। यानी Real World Objects को Represent करने के लिए हमें एक ऐसे तरीके की जरूरत होती है, जिससे आसानी से Real World Objects को Computer में Represent किया जा सके और जिस तरह से Real World में विभिन्न Objects आपस में Interaction करके किसी समस्या का एक उचित समाधान प्राप्त करते हैं, उसी तरह से Computer में भी किसी समस्या का समाधान प्राप्त किया जा सके।

Procedural Language में स्वयं के Data Type बना कर हम उन्हे बिल्कुल Built-in Data Type की तरह Use नहीं कर सकते हैं। Procedural Language इतने उन्नत नहीं हैं। जटिल तरीकों को अपनाए बिना हम Procedural Languages में x व y दोनों Coordinates को एक ही Variable में Store करके उस पर Processing नहीं कर सकते हैं। Procedural Languages को लिखना व Maintain करना काफी मुश्किल काम होता है।

सारांश

- कामों को करने का महत्व था, Data का नहीं। यानी Primary Attention Problem को Functions में विभाजित करने का था Data की Security का नहीं। **(Algorithm)**
- बडे Program को छोटे Programs में विभाजित किया जाता था, जिन्हें Functions कहते हैं। ज्यादातर Functions Global Data को Share करते थे, जिससे Data Insecure हो जाता था।
- Application में Data का काफी आसानी से विभिन्न Functions के बीच Transaction होता था। विभिन्न Functions Data को विभिन्न Forms में Transform करते थे।
- Top-Down Approach को Support करते थे।

The Object-Oriented Approach

Object Oriented Language के पीछे का मूलभूत विचार ये है कि एक Program में Data और उस Data पर काम करने वाले Functions को Combine करके एक Unit के रूप में ले लिया जाए। इस Unit को **Object** कहा जाता है। एक Object के Function यानी Data व Data पर काम करने के लिये लिखे गए Function को "C++" में **Member Function** कहा जाता है।

क्योंकि ये किसी Object के किसी अमुक Class से सम्बंधित होते हैं, जो कि किसी Data को Access करने का एक मात्र माध्यम होते हैं। यदि आप किसी Object के अन्दर रखे किसी Data

को Read करना चाहते हैं, तो आपको इसी Object के अन्दर लिखे उस Member Function को Call करना पडता है, जिसे उस Object के Data को Use करने के लिये ही लिखा गया है। यही एक Function होता है, जिसकी मदद से आप उस Object के Data को Read कर सकते हैं। आप सीधे ही Data के साथ किसी प्रकार की प्रक्रिया नहीं कर सकते हैं, क्योंकि Data Hidden रहता है। इसलिये किसी प्रकार से अचानक हुए परिवर्तन से Data सुरक्षित रहता है।

Data व Data को Use कर सकने वाले Function का एक साथ एक ही Unit के रूप में होना **Encapsulation** कहलाता है। Data का Hidden रहना यानी **Data Hiding** व **Encapsulation** Object Oriented Programming का मूल तथ्य या Key terms है। यदि आप किसी Data को Modify करना चाहते हैं, तो आपको पता होना चाहिए कि कौनसा Function उस Data पर काम करेगा यानी Object का वह Member Function जिसे Data के साथ लिखा गया है। कोई भी अन्य Function उस Data को Access नहीं कर सकता है। ये Processing Program को लिखना, Debug करना व Maintain करना आसान बनाती है। एक "C++" का प्रोग्राम ढेर सारे विभिन्न प्रकार के Objects का बना होता है, जो कि अपने–अपने Member Functions के द्वारा आपस में Communication करते हैं। "C++" व कई अन्य OOP Languages में Member Functions को **Methods** कहा जाता है और Data Item को **Instance Variable** कहा जाता है। किसी Object के Member Function को Call करना उस Object को **Message Send** करना कहलाता है।

हम एक उदाहरण लेते हैं। माना एक बडा प्रीति–भोज (Party) का समारोह है, जिसमें सभी मेहमान किसी Dining Table के चारों ओर बैठे हैं। किसी Table के चारों ओर बैठे लोगों को हम Functions मान सकते हैं जो कि खाना खाने का काम करते हैं और जो भी खाना Table पर रखा है, उसे Data कह सकते हैं। जब भी किसी को Table पर रखे विभिन्न प्रकार के व्यंजनों में से कुछ लेना होता है, तो वह स्वयं ही उस व्यंजन तक पहुंचता है और उसे उपयोग में ले लेता है किसी पडौसी मेहमान से कोई भी व्यंजन Pass करने को नहीं कहता। Procedural Program का भी यही तरीका होता है।

ये तरीका तब तक बहुत ठीक है जब तक कि खाना खाने वाले मेहमानों की संख्या सीमित हो यानी छः से आठ लोग अलग–अलग समूह में अलग–अलग Table पर खाना खा रहे हैं। लेकिन यदि मेहमानों की संख्या बीस या इससे भी अधिक हो तो ये तरीका ठीक नहीं कहा जा सकता है।

क्योंकि जब मेहमान अधिक होंगे तो Table भी बडा होगा और खाने के विभिन्न सामान पूरे Table पर काफी दूर–दूर होंगे। ऐसे में यदि कोई मेहमान किसी दूर रखे व्यंजन तक पहुंचना चाहता है तो हो सकता है कि उसके Shirt की Sleeves किसी दूसरे मेहमान के खाने में चली जाए या कई मेहमान एक साथ किसी व्यंजन पर हाथ बढाएं और व्यंजन Table पर गिर कर खराब हो जाए। यानी यदि मेहमानों की संख्या काफी ज्यादा हो तो एक ही Table पर भोजन करना एक परेशानी वाला काम होगा। एक बडे Procedural Program में भी यही होता है।

इस समस्या के समाधान के रूप में यदि कई छोटे–छोटे Tables हों और उन पर एक सीमित मात्रा में मेहमान हों और सबके पास उनका अपना भोजन हो तो ये एक अच्छी व्यवस्था हो सकती है। इस छोटे Table पर सभी मेहमान किसी भी व्यंजन पर आसानी से पहुंच सकते हैं और किसी प्रकार की परेशानी Create नहीं होती है। यदि कोई मेहमान किसी अन्य Table पर रखे किसी व्यंजन को लेना चाहता है तो सम्भवतया वह किसी अन्य मेहमान से उस व्यंजन को लाने के लिये कह सकता है। ये तरीका Object Oriented Programming का है जिसमें हरेक छोटी Table को एक Object कहा जा सकता है।

हरेक Object में उसका स्वयं का Data और Function होता है उसी प्रकार से जिस प्रकार से हरेक Table पर अलग – अलग मेहमान होते हैं और हरेक Table पर अपना अलग खाना होता है। Data

व Function के बीच होने वाले विभिन्न लेन–देन अधिकतर Object के अन्दर ही होते हैं लेकिन आवश्यकतानुसार ये भी सम्भव है कि किसी अन्य Object के Data को भी Use किया जा सके। इस तरह से किसी बडे Procedural Program को छोटे–छोटे Object के रूप में व्यवस्थित करके ज्यादा अच्छी तरह से Program को Maintain किया जा सकता है।

Features of Object-Oriented Languages

Objects

जब आप किसी समस्या को Object Oriented Language के रूप में बनाना चाहते हैं, तब ये तय नहीं करना होता है कि समस्या को Functions में किस प्रकार से विभाजित किया जाए बल्कि ये तय करना होता है कि समस्या को Objects में किस प्रकार से विभाजित किया जाए। साधारण सा सवाल दिमाग में आ सकता है कि ये Objects क्या होते हैं? इसका जवाब भी इतना ही साधारण है। हम जो कुछ भी सोंच सकते हैं, दुनिया की वह हर वस्तु Object है। फिर भी थोडा सा समझाने के उद्देश्य से हम यहां पर Objects के कुछ उदाहरण दे रहे हैं:

1 Physical Objects
- ❖ किसी Lift से सम्बंधित प्रोग्राम जिसमें Program का मूल बिन्दु Lift पर निर्भर करता है, इसमें Lift को एक Object कहा जा सकता है।
- ❖ किसी अर्थव्यवस्था से सम्बंधित प्रोग्राम में विश्व के सभी देशों को Object माना जा सकता है। क्योंकि इस प्रोग्राम का मूल बिन्दु विभिन्न देश होंगे, जिनकी अर्थव्यवस्था पर सारा प्रोग्राम निर्भर होगा।
- ❖ किसी Traffic Flow से सम्बंधित प्रोग्राम में विभिन्न प्रकार के वाहन Objects हो सकते हैं, क्योंकि पूरा प्रोग्राम वाहनों को केन्द्र में रख कर ही Develop किया जाएगा।
- ❖ किसी Air Traffic से सम्बंधित प्रोग्राम में विभिन्न देशों के Aircraft Object होंगे।

2 किसी Computer User Environment में कम्प्यूटर के विभिन्न अवयव जैसे कि Window, Menu, Graphics Objects (Line, Rectangle, Circle) Mouse, Keyboard, Toolbars, Command Buttons, Disks Drives, Printer आदि Objects होते हैं।

3 Human Entities में
- ❖ किसी Company के Employees
- ❖ किसी विद्यालय के विभिन्न विद्यार्थी
- ❖ विभिन्न ग्राहक
- ❖ Salesmen आदि Objects होते हैं, क्योंकि ये ही किसी प्रोग्राम की मूल इकाई होते हैं।

4 Data Storage Construct में
- ❖ विभिन्न `Customized Arrays`
- ❖ Arrays
- ❖ Stacks
- ❖ Linked Lists
- ❖ Binary Trees आदि Objects होते हैं।

5 Collection Of Data में
- ❖ Inventory
- ❖ Personal File

- ❖ Dictionary
- ❖ Longitude व Latitude की Table

6 User Defined Data Types में Time, Complex Numbers, Points Of Planes आदि Objects होंगे।

7 Computer Game में

- ❖ कोई चित्र
- ❖ Chess या Checkers के मोहरे आदि
- ❖ जानवरों के चिन्ह
- ❖ विभिन्न चिन्ह आदि Objects हो सकते हैं।

इन उदाहरणों से हम समझ सकते हैं कि किसी Program के किसी हिस्से में या पूरे प्रोग्राम में जो मूल वस्तु होती है, वह **Object** कही जा सकती है। किसी भी समस्या को Object के रूप में विभाजित करना काफी आसान है। क्योंकि दुनिया की हर वस्तु को जरूरत के अनुसार एक Object माना जा सकता है। हरेक Object की अपनी क्षमताएं होती हैं और हरेक Object कुछ ना कुछ काम कर सकता है। इसे समझने के लिये एक उदाहरण देखते हैं।

बडी इमारतों में विभिन्न मंजिलों पर आने–जाने के लिये Lifts होती हैं। इस Lift को एक Object माना जा सकता है। माना कि चौथी मंजिल पर किसी Lift में चार Passengers हैं और Passengers ने 8^{th}, 10^{th} व 15^{th} मंजिल पर जाने के लिये Button Press किया है, तो Lift में ये क्षमता होती है कि ये नीचे जा सकती है, ऊपर जा सकती है, ये इसके दरवाजों को खोल सकती है व बन्द कर सकती है, ये पता कर सकती है कि दूसरी Lifts कौनसी मंजिल पर हैं और उसे अगली किस मंजिल पर जाना है।

"C++" में एक Object के Data ये ध्यान रखते हैं कि Object में क्या–क्या क्षमताएं हैं और Object के Member Functions इस बात का ध्यान रखते हैं कि Object उन Data के साथ क्या–क्या कर सकता है। इस Lift Object में निम्न Data हो सकते हैं:

- Current_floor_number
- Number_of_passengers
- List_of_buttons_pussed

और Member Functions निम्न हो सकते हैं:

- GoDown()
- GoUp()
- OpenDoors()
- CloseDoors()
- GetInfo()
- CalculateWhereToGo()

Object Oriented Programming में किसी वस्तु की **विशेषता** व वस्तु की **क्षमता** एक साथ में होती है, ठीक उसी प्रकार से Object Oriented Program में Data व Functions एक साथ में एक Unit के रूप में होते हैं, जिसे **Object** कहा जाता है। Object के Data व उन Data की ***State*** में परिवर्तन करने वाले Functions को एक **Unit** या इकाई के रूप में Combine करने की प्रक्रिया को **Encapsulation** कहते हैं। Encapsulation की प्रक्रिया से प्राप्त होने वाला Template या Description **Class** कहलाता है।

Objects किसी Object Oriented System की Basic Run Time Entities होते हैं। Objects Memory में Space लेते हैं और हर Object का एक Memory Address होता है। जब हम किसी Object Oriented Program को Execute करते हैं, तब विभिन्न Objects एक दूसरे को Message Pass करके यानी उनके Member Functions को Call करके एक दूसरे से Communication करते हैं।

उदाहरण के लिए मान लो कि "Customer" व "Account" दो Objects हैं। अब यदि Customer को अपना Bank Balance जानना हो, तो वह Account Object को एक Message Pass करता है और अपने Bank Balance की जानकारी प्राप्त करने के लिए Request करता है। Account Object Customer Object की Request को पूरा करता है और उसे Bank Balance की जानकारी प्रदान कर देता है। इस तरह से Customer Object बिना ये जाने हुए कि Account Object में कौन–कौन से Data हैं, अपने Balance की जानकारी प्राप्त कर सकता है।

अपना Balance जानने के लिए Customer Object को केवल इतना ही ध्यान रखना होता है कि Account Object को कौनसा Message Pass करने पर यानी Account Object के किस Member Function को Call करने पर Account Object Account Balance की जानकारी प्रदान करेगा।

Classes

जब हमें एक ही Data Type के कई Data को किसी Variable में Store करना होता है, तब हम Array का प्रयोग करते हैं और जब विभिन्न Data Type के आपस में Logically Related कई Data को एक ही Variable में रखना होता है तब हम Structure का प्रयोग करते हैं। "C++" में **Class** भी यही काम करता है। यानी Class में भी हम विभिन्न Data Types के, आपस में Logically Related Variables Declare करते हैं। Structure व Class में मूलभूत अन्तर यही है कि एक Structure के Members Default रूप से **Global** या **Public** होते हैं जबकि एक Class के Members **Local** या **Private** होते हैं।

Class की दूसरी विशेषता ये है कि Class के Data Members को Access करने के Functions भी Class के अन्दर ही लिखे जाते हैं और हम इन्हीं Member Functions की सहायता से किसी Class के Object के Data Members को Access कर सकते हैं।

एक Class में Data छुपा हुआ रहता है क्योंकि केवल उस Class के Objects ही उस Class के Data को Access करने में सक्षम होते हैं। किसी अन्य Class के Objects या किसी External Functions के लिए ये Data Accessible नहीं होते हैं। इस प्रक्रिया को **Data Hiding** कहते हैं।

Data व Data को Access करने के लिए Authorized Function दोनों को एक Unit के रूप में Combined Form में Describe किया जाता है। किसी Object के Data व Data पर Perform होने वाले Operations के Functions को एक Unit के रूप में रखने की प्रक्रिया को **Encapsulation** कहा जाता है। किसी Class के अन्दर Declare किए गए वे Variables जिनमें Object के Data Stored रहते हैं, **Data Members** कहलाते हैं और उन Data के साथ प्रक्रिया करने वाले Functions **Member Functions** कहलाते हैं।

किसी Object के Data व Data पर Perform होने वाले Operations को Encapsulate करके एक Unit में रखने पर Create होने वाले Template या Description को Class कहते हैं। Class एक नए प्रकार का User Defined Data Type होता है। इस Create होने वाली Class का हम जितने

चाहें उतने Objects Create कर सकते हैं। यानी एक Class समान प्रकार के Objects के समूह की Description या Template को Represent करता है। Classes User Defined Data Type होती हैं और Built-In Data Type की तरह Behave करती हैं।

जिस प्रकार से किसी Structure को Define करने के बाद उस Structure के प्रकार के Variables Declare किए जाते हैं, उसी प्रकार से किसी Class के भी Variables Declare किए जाते हैं ताकि उस Class के Data को Use किया जा सके। "C++" में Class प्रकार से Declare किए गए Variables को **Object** कहा जाता है। जब कोई Class Define की जाती है, तब वह Memory में कोई Space Reserve नहीं करता है। एक Class प्रकार का Variable जिसे Object कहते हैं, Declare करने के बाद ही वह Memory में अपने Class की क्षमता के अनुसार Space Reserve करता है, ठीक उसी प्रकार से जिस प्रकार से किसी Structure को Define करने पर वह Memory में कोई Space Reserve नहीं करता है। जब हम Structure प्रकार के Variable Declare करते हैं तभी Memory में Space Reserve होती है।

Data Hiding, Data Abstraction and Encapsulation

किसी System की Inner Working को Hide करके उसे उपयोग करने के लिए एक Well Defined Interface तैयार करने की प्रक्रिया को हम ***Encapsulation*** कहते हैं। Create होने वाले Object को Real World में जब Use करना होता है, तब हम उसी Well Defined Interface द्वारा उस Object के Data को Access करते हैं।

Encapsulation का सबसे अच्छा उदाहरण एक Watch का लिया जा सकता है। एक Wristwatch किस प्रकार से काम करता है, Internally उस Wristwatch में क्या प्रक्रिया होती है, Wristwatch चाहे Analog हो या Digital, वह किस प्रकार से Hours, Minutes व Seconds को Manage करता है, यानी Wristwatch की Internal Working से हमें तब तक कोई मतलब नहीं होता है, जब तक कि Wristwatch सही तरीके से काम करती है। हम केवल समय जानने के लिए किसी Wristwatch को Use करते हैं।

Data व Data पर Perform होने वाले Operations के Functions को एक **Unit** के रूप में Bind करके Object की Class बनाने की प्रक्रिया को ***Encapsulation*** कहते हैं। इस Encapsulation के Concept के आधार पर बनने वाली Class के Data को केवल उसी Class में Define किए गए Member Functions ही Access कर सकते हैं।

इन Member Functions के अलावा कोई भी External Function उस Specific Class के Data को Access नहीं कर सकता है। किसी Class के अन्दर Define किए गए Member Function ही Program व Object के Data के बीच ***Interface*** प्रदान करते हैं। बिना इन Member Function के Main Function, किसी अन्य Class में Define किया गया Member Function या कोई भी अन्य User Defined External Function उस Specific Object के Data को Access नहीं कर सकता है। यानी Main Program से Object का Data पूरी तरह से ***Insulated*** या अलग होता है।

इस Insulation के कारण किसी Specific Object का Data किसी अन्य User Defined External Function या Main Program से छुपा हुआ रहता है। इसलिए इस Insulation की प्रक्रिया को ***Data Hiding*** कहते हैं।

Data Abstraction एक ऐसी प्रक्रिया होती है, जिसमें किसी समस्या से सम्बंधित जरूरी बातों को बिना जरूरी बातों से अलग किया जाता है। फिर उन जरूरी बातों को समस्या के किसी Object की Properties के रूप में Describe किया जाता है। उदाहरण के लिए मानलो कि किसी Company

के Administrator को अपनी Company के सभी Employees की Educational Information की जानकारी रखनी है। इस स्थिति में किसी Employee की विभिन्न Properties निम्नानुसार हो सकती हैं:

- **Name Of the Employee**
- Father's Name of the Employee
- Address of the Employee
- City of the Employee
- State of the Employee
- Country of the Employee
- Date Of Birth of the Employee
- **Education Of the Employee**
- Hobbies of the Employee
- Experience in the Company of the Employee
- **Extra Degree Of the Employee**
- Cast of the Employee
- Religion of the Employee

किसी Employee की और भी बहुत सी विशेषताएं हो सकती हैं, लेकिन चूंकि समस्या में Company के Administrator को अपने Employees की केवल Educational Information से ही मतलब है, शेष Information का Company के Administrator के लिए कोई उपयोग नहीं है, इसलिए Administrator की समस्या से सम्बंधित जरूरी बातें केवल निम्नानुसार ही हैं:

- **Name Of the Employee**
- **Education Of the Employee**
- **Extra Degree Of the Employee**

इनके अलावा एक Employee की जो भी Information हैं, वे इस समस्या के लिए जरूरी नहीं हैं, हालांकि किसी अन्य समस्या के लिए ये जरूरी हो सकती हैं। इस तरह से हमने एक Employee की विभिन्न विशेषताओं में से उन विशेषताओं को अलग किया जो Company के Administrator के लिए जरूरी हैं या हमारी Current समस्या से सम्बंधित हैं। इस प्रक्रिया को ही ***Abstraction*** कहते हैं, जिसमें हमने Employee की उन जरूरी बातों को उन बिना जरूरी बातों से अलग किया है, जिनकी Company के Administrator को जरूरत है।

किसी अन्य समस्या जैसे कि Employees के Bio Data को Manage करने वाली Class को Create करते समय ये सभी Information जरूरी हो सकती हैं। उस स्थिति में कुछ अन्य बातें होंगी जो Employee के Bio Data से सम्बंधित नहीं होंगी। तब भी हमें Employee के विभिन्न Data पर Abstraction की प्रक्रिया को लागू करके जरूरी Data को बिना जरूरी Data से अलग करना होगा।

जब हम किसी समस्या के समाधान के लिए किसी Object की जरूरी बातों को बिना जरूरी बातों से अलग कर लेते हैं, यानी Object की विभिन्न Properties पर Data Abstraction की Process को लागू करने के बाद जो जरूरी Properties प्राप्त होती हैं, उन Properties को Class के **Attributes** के रूप में Define कर लिया जाता है और इन Attributes या Properties की **State** या स्थिति में परिवर्तन करने वाले Operations के Functions को Attributes के साथ एक **Unit** के रूप में **Encapsulate** कर लिया जाता है, जिससे एक Description बन जाता है। फिर इस Description को एक नाम दे दिया जाता है, जो कि किसी **Class** को Define करता है, यानी हम एक नई Class बना लेते हैं।

चूंकि हम जानते हैं कि Class OOPS में एक नए User Defined Data Type को Represent करता है और हम जो Class बनाते हैं, उस Class में **Abstraction** प्रक्रिया के बाद प्राप्त हुए Attributes होते हैं, इसलिए इस नए Create होने वाले Data Type को ***Abstract Data Type*** कहते हैं।

Inheritance

जिस प्रकार से हम वास्तविक जीवन में विभिन्न Classes को Subclasses में विभाजित करते हैं, ठीक उसी प्रकार से हम "C++" में भी Classes को विभिन्न Subclasses में विभाजित कर सकते हैं। जैसे जानवरों के समूह को हम पानी में रहने वाले जानवरों, वायु में उडने वाले जानवरों व धरती पर चलने वाले जानवरों में बांट सकते हैं। फिर हरेक जानवर को उसकी विशेषताओं के आधार पर और भागों में विभाजित कर सकते हैं। ठीक इसी प्रकार से हम "C++" में भी Classes का विभाजन कर सकते हैं, जो कि पिछली Classes की विशेषताओं के साथ अपनी भी कुछ अन्य विशेषताओं को उपयोग में लेता हैं। ये प्रक्रिया **OOPS** में ***Inheritance*** कहलाती है।

जो मुख्य Class होती है, उसे ***Base Class*** कहते हैं और इसकी अपनी विशेषताएं होती हैं। इस Class के अन्दर और विभाजन किया जाए तो वह विभाजित Class ***Derived Class*** कहलाती है, जिसमें उसकी Base Class के गुण तो होते ही हैं, साथ ही इसके अपने भी कुछ अलग गुण होते हैं जो कि Base Class में नहीं होते।

Inheritance एक ऐसा तरीका है जिससे एक Class के Objects किसी दूसरी Class के गुणों को प्राप्त कर लेते हैं। ये Hierarchical Classification को Support करता है। OOPS में Inheritance का ये Concept Reusability का Idea Provide करता है। इस Concept के आधार पर एक पहले से बनी हुई Class के गुणों को Derive करके नई Class बनाई जाती है।

Reusability

एक बार एक Class लिख कर तैयार कर लेने के बाद हम उसी Class को कई अन्य प्रोग्रामों में आवश्यकतानुसार उसी प्रकार से Use कर सकते हैं जिस प्रकार से Procedural Languages में Library Functions को **Use** करते हैं। Inheritance की विशेषता को Use करते हुए हम ऐसा भी कर सकते हैं कि किसी पहले से बनाई गई Class में परिवर्तन किये बिना ही हम उसमें एक Derived Class बनाकर Base Class के गुण भी Use कर लें और अपनी Derived Class के गुण भी उसमें जोड सकते हैं।

जैसे कि माना आपने एक Class बनाया जो कि Menu बनाने का काम करता है। आप इस Class में परिवर्तन करना नहीं चाहते हैं, लेकिन आप चाहते हैं कि उसमें Animation की Capability भी आ जाए। ऐसे में आप एक नई Class बना सकते हैं, जिसमें Base Class के सारे गुण तो होंगे ही साथ में आप Animation का गुण भी उसमें जोड सकते हैं। इस तरह से OOPS की वजह से आप किसी Class को बार–बार लिखने से बच जाते हैं और एक ही Class को कई जगह Use कर सकते हैं। ये भी OOPS की एक खास विशेषता है।

Creating New Data Types

Object की एक विशेषता ये भी है कि ये Programmers को आवश्यकतानुसार नए Data Types बनाने की भी सुविधा प्रदान करता है। एक Programmer जो भी Class Create करता है, वह

Class Computer में एक नए प्रकार के Data Type को Represent करता है और Class किसी Real World Object की विशेषताओं और क्षमताओं का Description या Specification होता है कि उस Class का Object किस तरह का है और क्या–क्या कर सकता है।

Polymorphism and Overloading

जैसाकि हमने पहले भी बताया कि एक Class "C" के Structure का ही Modified रूप है। यानी हम Structure प्रकार के Variable तो Declare कर सकते हैं, लेकिन जिस प्रकार से Built - In प्रकार के Data Type के दो Variables को हम आपस में जोड सकते हैं, ठीक उसी प्रकार से किसी Structure के दो Variables को हम नहीं जोड सकते। इसे समझने के लिये हम एक उदाहरण देखते हैं। माना एक Structure निम्नानुसार है:

```
struct Add
    {
        int num1;
        int num2;
    };

struct Add A, B, C ;
```

यदि हम `C = A + B ;` करें, तो ये एक गलत Statement होगा। किसी Class के Objects को भी हम ठीक इसी प्रकार से नहीं जोड सकते हैं, क्योंकि Class Structure का ही Modified रूप है।

इसका कारण ये है कि Compiler ये नहीं जानता है कि User द्वारा Define किए गए Variables के साथ किस प्रकार से जोड किया जाए। जबकि Built-in Data Types में जोड करने के इस तरीके को Compiler में पहले से ही निश्चित कर दिया गया है और + Operator ये जानता है कि इन Variables को कैसे जोडा जाए।

इसलिये User Defined Data Type के Variables या Object को जोडने का तरीका **Operators** को बताना पडता है। यानी + Operators को ये बताना पडता है कि किस प्रकार से User Defined Data Type के Variables या Objects को ये Operator जोडेगा।

इसके लिये हमें नए प्रकार के Operations Define करने होते हैं। इस प्रकार से Operators व Functions को अलग–अलग तरीके से इस बात पर निर्भर करते हुए कि वे किस प्रकार के Data पर Operation कर रहे हैं, Use किया जाता है और इस प्रक्रिया को **Polymorphism** कहा जाता है।

किसी Existing Operator जैसे कि +, = आदि को इस लायक बनाया जाता है कि वे User Defined Data Type के विभिन्न प्रकार के Variables या Objects पर क्रिया कर सकें। विभिन्न Operators को इस लायक बनाने की क्रिया को Operators की **Overloading** करना कहा जाता है।

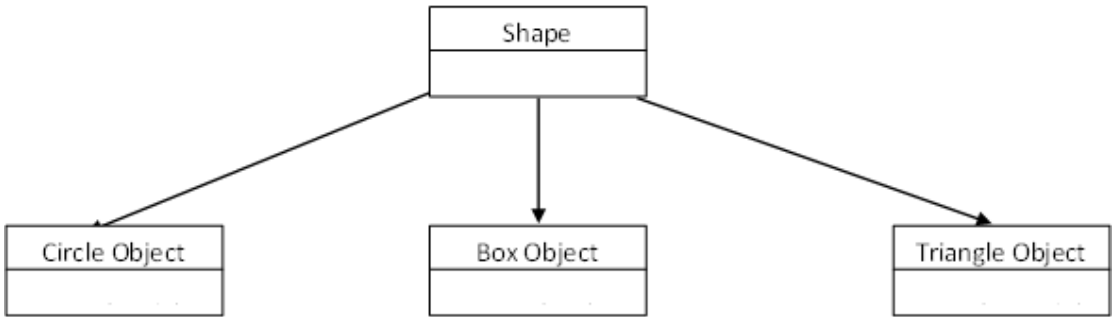
जब कई Functions के नाम समान होते हैं लेकिन उनके Arguments या Parameters की संख्या या Return Data Type या फिर Formal Arguments के Data Type में आपस में अन्तर होता है, तो इसे Functions की **Overloading** होना कहा जाता है। Overloading से एक Programmer के लिए Program लिखना व समझना आसान हो जाता है। यह भी Polymorphism का एक तरीका है।

Dynamic Binding

किसी Object के Reference में कौनसा Function Call होना चाहिए, जब ये बात Program के Compile Time में तय होती है, तो इसे **Early Binding** कहते हैं। जबकि किसी Object के Reference में किसी काम के लिए कौनसा Procedure Execute होगा, ये बात जब Program के Runtime में तय होती है, तब इसे **Late Binding** या **Dynamic Binding** कहते हैं। Polymorphism के अन्तर्गत Dynamic Binding का काम होता है। इसे समझने के लिए निम्न चित्र देखिए:

इस चित्र में हम देख सकते हैं कि Shape Class एक Base Class है जिसे Inherit करके तीन नई Classes Circle, Box व Triangle को Create किया गया है। चूंकि ये तीनों ही Classes Shape Class से Inherited हैं, इसलिए Base Class Shape का Draw() Method तीनों ही Classes में Inherited है।

अब मानलो कि हमने तीनों Derived Classes का एक–एक Object Create किया और उस Object को Draw करने के लिए Draw() Method को Call किया। ऐसे में जब हम Circle Class के Object के लिए Draw Method Call करते हैं, तब Compiler Circle Class के Draw Method को Call करके Circle Draw करता है।



जब हम Box Object Draw करने के लिए Box Class के Object के Reference में Draw() Method को Call करते हैं, तब Compiler Box Class के Draw Method को Execute करता है। इसी तरह से जब हम Triangle Class का Object Create करना चाहते हैं, तब Compiler Triangle Class के Draw Method को Execute करके Triangle Draw कर देता है। यानी एक ही नाम का Draw Method Create हो रहे Object की Class के आधार पर उसी Class के Draw Method को Execute करता है, जिस Class का Object Create किया जा रहा है।

इस प्रक्रिया को Binding Object के साथ Method की Binding होना कहते हैं। चूंकि किस Object के Reference में कौनसा Draw Method Call होगा, इसका निर्णय Compiler Program के Runtime में करता है, इसलिए इस प्रक्रिया को Late Binding या Dynamic Binding कहते हैं।

Message Passing

Object Oriented Program में Objects का पूरा एक समूह होता है। ये Objects आपस में Communication करते हैं। किसी Object Oriented Programming में निम्न तीन Concepts होते हैं:

- Abstract Data के आधार पर एक ऐसी Class Definition Create करना जो Required Object +`
  -की Properties व उसके Behaviors को Describe करे।
- Create किए गए Abstract Data Type या Class के Objects Create करना।
- Create होने वाले विभिन्न Objects के बीच Communication को Establish करना।

दो Objects आपस में उसी प्रकार से Information भेज कर व प्राप्त करके Communication करते हैं, जिस तरह से Real World में आम लोग आपस में Message Pass करके Communication करते हैं।

Message Passing के Concept से हम Computer Application के रूप में किसी समस्या को उसी प्रकार से Directly Modal कर सकते हैं, जिस तरह से उस समस्या को Real World में Describe करते हैं। Message Passing Concept में जब एक Object A किसी दूसरे Object B से Communication करना चाहता है, तब वह Object A उस दूसरे Object B को एक Message Pass करता है यानी उस दूसरे Object B के किसी Member Function को Call करता है। इस Concept में तीन बातें होती हैं:

- **Object**
- **Message** (Member Function of the object)
- **Information** (Arguments in the Member Function of the object)

उदाहरण के लिए मानलो कि *Customer* Object *Account* Object से अपने Bank Balance की जानकारी प्राप्त करना चाहता है। ऐसे में Customer Object Account Object को एक **Message** भेजता है (यानी Account Object के Reference में किसी **Member Function** को Call करता है।) और Message में ***Bank Balance*** की जानकारी प्राप्त करने के लिए Information (***Arguments***) भेजता है। यानी Customer Object Account Object से निम्नानुसार Communication करता है:

```
Account.Information(Bank_Balance);
```

जब हम इस प्रकार से Account Object से Bank Balance Return करवाते हैं, तो उस Bank Balance को Customer Object के लिए Use कर सकते हैं। इस प्रकार से एक Object (Customer) दूसरे Object (Account) से Communication करने के लिए दूसरे Object के किसी Appropriate Member Function को Call करते हुए उसे **Message** भेजता है, तथा किसी जानकारी के लिए **Arguments** के रूप में Request करता है। दूसरा Object उस Request को Message द्वारा प्राप्त करता है और Appropriate Member Function के Execution द्वारा पहले Object को उसकी Required Information प्रदान कर देता है।

OOPS एक ऐसा Concept है, जिसके आधार पर हम किसी समस्या को Design करते समय उसे विभिन्न प्रकार के Physical Objects के रूप में परिभाषित करते हैं और इन विभिन्न प्रकार के Physical Objects को Computer में किसी भी Programming Language में Logically Represent कर सकते हैं। विभिन्न प्रकार के Objects को जिस Programming Language में अच्छी तरह से Represent किया जा सकता है, उस Programming Language को हम **Object Oriented Programming Language** कह सकते हैं।

"C++" एक ऐसी ही Programming Language है, जो OOPS के सभी Concept को Computer में Implement करने में सक्षम है। यदि हम Object को सरल रूप में परिभाषित करें, तो हम ये कह

सकते हैं कि Object एक ऐसा Variable होता है, जिसमें एक से ज्यादा प्रकार के "**Basic Data Type"** के मानों को Store व Manipulate किया जाता है।

ये तय करने के लिए कि Object किस प्रकार के और कितने मानों को Manipulate करने का काम करेगा या Object किस Physical Real World Object को Logically Computer में Represent करेगा, हमें एक Specification (Modal) बनाना होता है। ये Specification या Modal जो कि Object की विशेषताओं या **Attributes** (Data Members) और उन विशेषताओं (Data Members की States) में परिवर्तन करने वाले Object के **Behaviors** (Member Functions) को Represent करता है।

इस Specification को ही **Class** कहते हैं, जो कि एक तरफ तो किसी Real World Object को Computer में Logically Represent कर रहा होता है और दूसरी तरफ वही Class किसी Object का एक ऐसा Specification होता है, जो Programming Language में उस Real World Object को Logically एक नए Data Type के रूप में Represent कर रहा होता है।

सारांश

- समस्या में Procedures के बजाए Data का महत्व है। यानी OOPS में Data को Primary Level पर रखा गया है और Data पर Perform होने वाले Operations के Functions को Secondary Level पर रखा गया है।
- Problem को Functions में विभाजित करने के बजाय Objects में विभाजित किया जाता है।
- Data Structure को इस प्रकार से Design किया गया है, जो कि Object को Characterize करते हैं।
- Data पर Perform होने वाले Operations वाले Functions को Object के Data Structure के साथ ही Combined कर दिया गया है, जिसे Encapsulation कहते हैं।
- Data को केवल Data के साथ Associate किए गए Functions ही Access कर सकते हैं, जिससे Data External Functions के लिए Hidden रहता है। इस प्रक्रिया को OOPS में Data Hiding कहते हैं।
- Objects आपस में Functions द्वारा Communication करते हैं। इस प्रक्रिया को Message Passing करना कहते हैं।
- आवश्यकता होने पर Object में नए Data व Data पर Perform होने वाले Operations को Add किया जा सकता है। इस प्रक्रिया को OOPS में Inheritance कहते हैं।
- Program Design में OOPS के Approach को Bottom-Up Approach कहते हैं।

Benefits Of OOPS

OOPS Program Designer व Program User दोनों के लिए कई सुविधाएं प्रदान करता है। OOPS द्वारा Develop किए जाने वाले Programs में निम्न विशेषताएं होती हैं:

- Inheritance का प्रयोग करके एक Programmer बार–बार एक जैसी Coding लिखने से बच जाता है। वह एक बार लिखी गई Coding को बार–बार Reuse कर पाने में सक्षम हो जाता है। Coding को Reuse करने के कारण Programmer को Program Develop करने में कम समय लगता है और Program को Maintain करना सरल हो जाता है।

- चूंकि, OOPS में Program का Data पूरी तरह से Outer World से Hide रहता है। Data को Access करने के लिए अधिकृत Member Functions ही उस Data को Access करने में सक्षम होते हैं। चूंकि, Program के Data को कोई भी Unauthorized External Function Access करने में सक्षम नहीं होता है, इसलिए Accidental Modifications से Data सुरक्षित रहता है।
- एक ही Object के कई Instances बिना किसी Interference के एक ही Program में एक साथ Exist हो सकते हैं।
- Program को Objects में विभाजित करने के कारण Program Real World Concepts को Logically ज्यादा बेहतर तरीके से Computer में Represent करने में सक्षम होता है।
- OOPS पर आधारित Application को Upgrade, Modify व Change करना काफी सरल व सुविधाजनक होता है।
- बहुत ही जटिल समस्याओं के समाधान को भी OOPS Concept के आधार पर काफी आसान तरीके से Develop व Manage किया जा सकता है।

Object Oriented Languages

OOPS के Concepts को "C" या "Pascal" जैसी Procedural Languages में भी पूरी तरह से Implement किया जा सकता है। लेकिन जब हम "C" जैसी Procedural Language में OOPS के Concepts को Implement करने की कोशिश करते हैं, तब हमें कई अन्य Compiler सम्बंधित सीमाओं का सामना करना पडता है।

जबकि "C++" जैसी OOPS को ध्यान में रख कर Design किए गए Compiler को Use करने पर हमें इस प्रकार की समस्याओं का सामना नहीं करना पडता। OOPS को Implement करने के सम्बंध में भी हम Programming Languages को दो भागों में बांट सकते हैं:

Object-Based Programming Language

Microsoft Company का Visual – Basic एक Object Based Programming Language है। इस प्रकार की Programming Languages Encapsulations व Object Identity को Support करता है। Object Based Programming Languages के मुख्य Features Encapsulation, Data Hiding व Access Mechanism, Objects का Automatically Initialize व Clear होना तथा Operator Overloading होते हैं। Object-Based Programming Languages Inheritance व Dynamic Binding को Support नहीं करते हैं। वे Languages जो Objects को Use करते हुए Programming सम्भव बनाती हैं, Object – Based Programming Languages कहलाती हैं।

Object-Oriented Programming Language

Object Oriented Programming में Object Based Programming के सभी Features Available होने के साथ ही इनमें Inheritance व Dynamic Binding की भी सुविधा होती है। "C++" एक Hybrid Language है, क्योंकि ये Procedural Programming के साथ–साथ Object Oriented Programming को भी Support करती है।

OOPS

WITH

C++

OOPS with C++

Object Oriented Programming Concept को समझे बिना हम "C++" की Capabilities का पूरा फायदा नहीं उठा सकते हैं। Class "C++" का सबसे जरूरी Concept है। Object Oriented Programming का Concept समझे बिना हम कोई Serious Program नहीं बना सकते हैं। इसलिए Class को समझना "C++" को समझने के लिए सबसे जरूरी है। Object Oriented Concept को "C++" के साथ Use करते हुए OOPS को समझाना काफी जटिल काम है। O

bject Oriented Technology का सार यही है कि Programmers अपने Program उसी प्रकार से Design कर सकें जिस तरह से Real World में किसी काम या समस्या को Design किया जाता है। हम यहां पर पहले यही समझेंगे कि Object Oriented Design क्या होता है और किस प्रकार से किसी समस्या को Object Oriented Form में Design करके Computer पर उस समस्या का समाधान प्राप्त किया जाता है। Object Oriented Programming Concept का मुख्य आधार **Object** व **Class** हैं।

Class and Objects

OOPS का Modal इस तथ्य पर आधारित है कि दुनिया की हर वह वस्तु जिसे हम Physically देख सकते हैं, छू सकते हैं या Logically अनुभव कर सकते हैं, एक Real World **Object** है और हर Object कई अन्य छोटे Objects से बना होता है। साथ ही हर Object किसी ना किसी Class का एक उदाहरण (Instance) या सदस्य होता है।

यदि एक उदाहरण देखें तो Computer एक Object है। लेकिन ये Computer स्वयं कई अन्य छोटे Objects जैसे कि Motherboard, Processor, RAM, Hard Disk, Floppy Drive, Cabinet, SMPS, Monitor, Keyboard, Mouse आदि से मिलकर बना होता है।

हम Real World में भी देखते हैं कि कई Objects के Features समान होते हैं। जैसे कि जितने भी Computers होते हैं, उन सभी में Motherboard होता है, Processor होता है, Hard Disk होती है, RAM होती है, CD ROM होता है, आदि। यानी हर Computer के Basic Features या आधारभूत अवयव समान होते हैं। इन सभी Objects के चारित्रिक गुणों या Characteristics में समरूपता है। इसलिए एक Computer किसी Computer Class का एक Instance या Object होता है।

Class एक ऐसा **Modal, Design** या **Description** होता है, जिसके आधार पर समान Characteristics वाले कई Object Create किए जा सकते हैं। यानी यदि दूसरे शब्दों में कहें तो Class एक ऐसा Modal, Design या Description होता है, जो किसी एक अमुक (**Particular**) समूह के Objects (**Entity Set**) की विशेषताओं (**Characteristics**) या **Features** को Represent करता है।

जब हम किसी Object Oriented Programming Language में Program लिखते हैं, तो Programming के समय हम किसी Object को Define नहीं करते हैं, बल्कि हम उस Object की विशेषताओं का एक Modal बना लेते हैं, जिसे Class कहते हैं और उस Modal **(Class)** के आधार पर आवश्यकतानुसार कई Objects Create कर लेते हैं। इसे एक उदाहरण द्वारा समझने की कोशिश करते हैं।

मानलो कि यदि आपसे कहा जाए कि एक Car की परिभाषा दीजिए। परिभाषा के रूप में आप क्या कहेंगे? आप Car की ऐसी विशेषताओं (**Characteristics**) का वर्णन (**Description**) करेंगे जो

उसे दुनिया की दूसरी चीजों (**Objects**) से अलग बनाती है। आप कह सकते हैं कि Car एक ऐसी चीज होती है जिसके चार Wheels होते हैं। उसमें बैठने के लिए कुछ सीटें होती हैं। Car में एक Engine होता है। ये Engine Diesel से चलता है। इसमें गति को प्रभावित करने के लिए एक Accelerator और एक Gear Box होता है। इसी तरह से कई और विशेषताएं बताई जा सकती हैं, जिनसे एक Car की पहचान हो सकती है।

आपने एक Car की जो परिभाषा बताई है, क्या वह परिभाषा कार है। नहीं! ये परिभाषा कार नहीं है बल्कि ये परिभाषा तो Car का एक विवरण या **Description** मात्र है।

मानलो कि आपको किसी Automobile Company से ये Offer आता है कि आप जिस तरह की Car चाहते हैं, Company उस तरह की Car बनाने के लिए तैयार है। आप जिस तरह की चाहेंगे, उस तरह की Car बन जाएगी। अब आपको Company को बताना है कि आपको किस तरह की Car पसन्द है। सीधी सी बात है कि Company के Engineers आपकी कल्पनाओं वाली Logical Car को तभी साकार रूप की Physical Car में बदल सकते हैं, जब आप किसी तरीके से उन Engineers को अपनी उस सपनों वाली Logical कार का विवरण दे पाऐं। मानलो कि आप अपनी Car में निम्न विशेषताएं (Characteristics) या Features चाहते हैं:

1. आपकी कार पर Yellow, Blue व White ये तीन Color होने चाहिएं।
2. Car हल्की होनी चाहिए।
3. Car में केवल दो सीटें ही होनी चाहिए।
4. Car में सामान्य कारों की तुलना में एक Extra Gear होना चाहिए ताकि जरूरत पडने पर उस Gear के प्रयोग से Car जमीन से ऊपर भी उड सके।
5. Car का Engine Latest Quality का व सबसे Highest Power का होना चाहिए।
6. कार ऐसी होनी चाहिए की वह अपने चालक को Identify कर सके ताकि आप ये तय कर सकें कि उस Car को कौन–कौन चला सकेंगे। इस विशेषता के कारण आपकी कार कोई चोरी नहीं कर सकता।
7. जब भी आप Car के पास जाएं Car आपके लिए दरवाजा खोल दे।
8. जब भी आप अपने गन्तव्य स्थान पर पहुंचे, Car के रूकते ही दरवाजा खुल जाना चाहिए।

वगेरह, वगेरह। कल्पनाओं की कमी नहीं होती है, इसलिए ऐसी हजारों खूबियों वाली Car की आप कल्पना कर सकते हैं।

ये तो आपने अपनी सपनों वाली Logical Car का **Description** दे दिया। लेकिन कई स्थानों पर ऐसी जरूरत भी पड सकती है जब आप अपनी बात को केवल Description से नहीं समझा सकते। उस स्थिति में आप क्या करेंगे। आप मानें या ना माने, आप पेन और कागज उठाएंगे और स्वयं अपनी Car का एक पूर्ण सन्तुष्टिदायक **Modal** या **Design** बनाएंगे। क्या ये Modal आपकी Car है? क्या आप इस Modal को चला सकते हैं?

नहीं! ये वास्तविक कार नहीं है बल्कि एक Logical Car या Car का Modal मात्र है। लेकिन ये Modal फालतू नहीं है बल्कि ये Modal ही सबसे महत्वपूर्ण चीज है। एक बार इस Modal के बन जाने के बाद Company के Engineers आपकी Logical Car को एक Physical Car में बदल सकते हैं और आपके सपनों वाली Car को एक Physical Modal का रूप दे सकते हैं।

मानलो कि Company के Engineers ने आपकी Car बना दी और वह Car बिल्कुल वैसी ही है जैसी आप चाहते थे। अब यदि Company बिल्कुल उसी तरह की एक और Car बनाना चाहे, तो क्या Company को शुरू से वापस सारा काम करना पडेगा। क्या उसे दुबारा आपसे Modal बनवाना पडेगा।

नहीं! Modal केवल एक ही बार बनता है, लेकिन उस Modal के आधार पर हजारों कारें बन सकती हैं और जिस Modal के आधार पर हजारों Cars बन रही हैं, हम समझ सकते हैं कि जब तक Modal Change नहीं किया जाएगा, तब तक बनने वाली सभी कारों की सभी खूबियां एक समान होंगी। किसी भी Car के किसी भी Feature या Characteristic में किसी प्रकार का कोई अन्तर नहीं होगा।

जो Logical Modal आपने बनाया है, वह Object Oriented Programming Language में एक **Class** को Represent करता है और उस Modal के आधार पर Company के Engineers ने जो Physical Car बनाई है, वह Car एक Physical **Object** को Represent करता है। ये Physical Car आपके द्वारा बनाए गए Modal (**Class**) का एक उदाहरण (Instance) मात्र है, क्योंकि यदि Company चाहे तो उसी Modal के आधार पर कई अन्य Cars बना सकती है।

जिस तरह से एक Modal के आधार पर यदि Company चाहे तो हजारों कारें बना सकती है, उसी तरह से एक Object Oriented Programming Language में एक बार एक Class Design कर लेने के बाद एक Programmer उस Class के जितने चाहे उतने Instance Create कर सकता है।

"C++" में Class Create करने के लिए **class** Keyword का प्रयोग करना पडता है। वास्तव में Class एक नए प्रकार का **User Defined Data Type** होता है। ये एक ऐसा Data Type होता है, जिसे Programmer Real World के Physical Objects को Computer Software या Program के Logical World के Object के रूप में Represent करता है।

हम एक और उदाहरण लेते हैं। पिछले उदाहरण में हमने एक Modal के आधार पर Object बनाया था। इस उदाहरण में हम एक Real World Object के आधार पर अपने Computer Program के लिए एक Logical Class बना रहे हैं। एक Real World Object के आधार पर उस Object की Logical Class बनाने के बाद, उस Class के आधार पर हम जितने चाहें उतने Objects बना सकते हैं, क्योंकि एक Object के Description के विश्लेषण से हमें Car का Modal (**Class**) मिल जाएगा।

मानलो कि हमें एक Physical Car Object को Computer में एक Logical Car के रूप में Represent करना है। अब Physical कार की ऐसी क्या विशेषताएं हैं, जो उसे किसी दूसरे Real World Object से अलग बनाती हैं, उन Characteristics को निम्नानुसार लिखा जा सकता है। यानी

1. एक Physical Car का कोई एक नाम होता है।
2. एक Physical Car का कोई ना कोई Modal Number होता है।
3. एक Physical Car के Engine का एक Unique Serial Number होता है।
4. एक Physical Car के Chassis का एक Unique Serial Number होता है।
5. एक Physical Car किसी ना किसी ईंधन से चलती है।
6. एक Physical Car का कोई Color होता है।
7. एक Physical Car की कोई Size होती है।
8. एक Physical Car में कुछ Seats होती हैं।
9. एक Physical Car में कुछ Gear होते हैं।
10. एक Physical Car का Pickup Rate होता है।
11. एक Physical Car किसी ना किसी अधिकतम Speed पर दौड सकती है।
12. एक Physical Car में चार Wheels होते हैं।

इसी तरह से हम कुछ और विशेषताएं भी बता सकते हैं, जो किसी Car को परिभाषित करती हैं। किसी एक Car के आधार पर हम Car की इतनी विशेषताएं प्राप्त कर सकते हैं। इन सभी Descriptions को निम्नानुसार भी लिखा जा सकता हैः

Description of the Car

(**Class**)

nameOfCar
modalNumberOfCar
serialNumberOfEngine
serialNumberOfChassis
fuelOfCar
colorOfCar
sizeOfCar
noOfSeatsInCar
noOfGearsInCar
pickupRateOfCar
maximumSpeedOfCar
noOfWheelsInCar

यदि हम चाहें तो इन Features में निम्नानुसार विभिन्न प्रकार के मान भी Fill कर सकते हैंः

Description of the Maruti800 Car

(**Class**)

nameOfCar	=	*Maruti800*
modalNumberOfCar	=	*March1999*
serialNumberOfEngine	=	*12325465*
serialNumberOfChassis	=	*65457898*
fuelOfCar	=	*Diesel*
colorOfCar	=	*Yellow*
sizeOfCar	=	*4.5ft*
noOfSeatsInCar	=	*6*
noOfGearsInCar	=	*5*
pickupRateOfCar	=	*60KMPH After 4 Seconds*
maximumSpeedOfCar	=	*140KMPH*
noOfWheelsInCar	=	*4*

यदि हम थोडा ध्यान दें तो समझ सकते हैं, कि ये कुछ ऐसी Common विशेषताएं या कुछ ऐसे Common Features (Characteristics) हैं, जो लगभग सभी प्रकार की कारों के Features को Represent कर सकते हैं। उदाहरण के लिए हमने ऊपर Maruti800 के विभिन्न Features के मानों को देखा। अब निम्नानुसार Features के मानों में परिवर्तन करने पर हम इन्ही Features के आधार पर Tata Indica या Tata Sumo को भी Computer में Logically Represent कर सकते हैं। यानी

Description of the Tata Indica Car

(**Class**)

nameOfCar	=	*TataIndica*
modalNumberOfCar	=	*Feb2000*
serialNumberOfEngine	=	*25465454*

serialNumberOfChassis	=	*98787898*
fuelOfCar	=	*Diesel*
colorOfCar	=	*Silver*
sizeOfCar	=	*4.25ft*
noOfSeatsInCar	=	*5*
noOfGearsInCar	=	*5*
pickupRateOfCar	=	*80KMPH After 4 Seconds*
maximumSpeedOfCar	=	*160KMPH*
noOfWheelsInCar	=	*4*

हम देख सकते हैं कि Car के केवल एक Physical Modal के आधार पर हमने Computer में एक Logical Modal बनाया और उस Logical Modal की Description के आधार पर हम जितनी चाहें उतनी कारों को Logically Computer में Represent कर सकते हैं। इस Description को ही **Class** कहते हैं।

इस Car Class के आधार पर हम जितनी भी कारों को Computer में Logically Represent करेंगे, वे सभी कारें इसी Car Class के उदाहरण (Instance) या Objects होंगे, क्योंकि उन सभी कारों के Basic Features एक समान होंगे केवल उन Features के मानों में ही अन्तर होगा।

हमने ये Description किसी एक Car के आधार पर Develop किया था, लेकिन ये Description अन्य सभी कारों को भी समान प्रकार से Represent कर सकता है। Computer में भी यही किया जाता है। किसी समस्या के सबसे मूल Object का पता लगाया जाता है। फिर उस मूल Object के आधार पर समस्या से सम्बंधित जरूरी Features को लेकर एक Description तैयार किया जाता है।

इस Description के आधार पर एक Modal बनाया जाता है और उस Modal के आधार पर हम उस Class के जितने चाहें उतने Logical Objects Create कर सकते हैं और उसी तरह से उन्हें Represent कर सकते हैं जिस तरह से Real World Objects को Represent करते हैं।

यदि आपसे कहा जाए कि पांच कारों के उदाहरण दो तो आप Tata Sumo, Tata Indica, Maruti Zen, Maruti 1000, व Maruti 800 के उदाहरण दे सकते हैं। क्या इन सभी प्रकार की कारों को उपरोक्त Description के आधार पर Represent नहीं किया जा सकता है? निश्चित रूप से किया जा सकता है, क्योंकि ये सभी Objects समान Class के उदाहरण हैं। ये सभी एक Car Class के Instances या Objects हैं और इनके Basic Features समान हैं।

यदि यहां हम वापस से Class की परिभाषा को दोहराएं तो कह सकते हैं कि एक Class समान Featurs वाले Objects के एक समूह (Entity Set with Similar Features) का Logical या **Abstract Representation** होता है, जबकि उस Class का कोई Instance (उदाहरण) उस Class का **Actual Representation** या Physical Representation होता है जिसे **Object** कहते हैं।

सामान्यतया "C++" की हर Class दो मुख्य Components से बनी होती है :

1 Attributes

उदाहरण के लिए किसी Objects के एक समूह (**Entity Set**) को लीजिए। आप देखेंगे कि उन सभी में कुछ ऐसी बातें हैं, जो सभी Objects को समान रूप से प्रभावित करती हैं। ये वही विशेषताएं होती हैं जो किसी Object को किसी दूसरे Object से अलग बनाती हैं। इन विशेषताओं को **Attributes** कहा जाता है।

हर Object किसी ना किसी Class का एक Instance या सदस्य होता है। किसी Entity Set के ये Attributes ही उनकी Class के Features होते हैं। जैसे यदि हम एक Book को ही लें तो Book की अपनी एक Class हो सकती है, जिससे हम दुनिया की किसी भी Book को Describe कर सकते हैं।

अब इस Book की Class के Features वही होंगे जो किसी एक Book के Features हैं। Book की ऐसी कौनसी Properties हो सकती हैं, जो एक Book को दूसरे किसी Object से अलग बनाती हैं? इसके सवाल के जवाब में हम किसी Book को निम्न Properties द्वारा परिभाषित कर सकते हैं:

1. हर Book का एक नाम होता है।
2. हर Book का एक Author होता है।
3. हर Book किसी ना किसी Publication से प्रकाशित होती है।
4. हर Book का एक ISBN Number होता है, जिससे उस Book की Unique पहचान होती है।
5. हर Book में कुछ Pages होते हैं।
6. हर Book की कुछ कीमत होती है।

इसी तरह से Book की कई और Characteristics हो सकती हैं। ये सभी Characteristics उस Book Object की Class के Features होते हैं। इन्हीं Features को Book Class के **Attributes** भी कह सकते हैं। यदि हम इस Book की Description को Class के Modal के रूप में Describe करें तो इस Book Object की Class को निम्नानुसार Represent कर सकते हैं:

Book's Attributes Description

(Book Class)
- bookName
- bookAuthor
- bookPublication
- bookISBN
- bookPages
- bookPrice

किसी Class के **Attributes** किसी **Object** की **Appearance, State** व **Condition** के आधार पर तय होते हैं। यानी किसी Class की Description के Attributes इस तथ्य पर आधारित होते हैं कि उस Class के Object किस प्रकार के दिखाई देते हैं और उनके Objects की स्थिति (Position or Situation) क्या हो सकती है। किसी Object के इन Appearance व State की विभिन्न Description ही उस Object को दुनिया के सभी Object से अलग पहचान प्रदान करती है।

उदाहरण के लिए इस Book Class को ही लेते हैं। हम इस Book Class का एक Object Create करके उसके विभिन्न Attributes को निम्नानुसार मान प्रदान कर सकते हैं:

Book's Attributes Description

(Book Class)

bookName	=	B with OOPS through C++ in Hindi
bookAuthor	=	Kuldeep Mishra
bookPublication	=	Publication
bookISBN	=	010101010101

bookPages	=	600
bookPrice	=	300

ये सभी Fields Book की State यानी Situation बता रहे हैं। यानी Book का नाम "B with OOPS through C++ In Hindi" है, जो कि Book की एक स्थिति है। इसी तरह से Book की Price 300 है, ये भी Book की स्थिति को Represent कर रहा है।

यदि हम Car का उदाहरण लें, तो Car का Color व Car की Size Car के Appearance को Represent करते हैं। यानी ये बताते हैं कि Car दिखाई कैसी देती है। जबकि Car का नाम Car की State या Situation को Represent करता है, जो ये बताता है कि Car की स्थिति क्या है यानी Car बनाने वाली Company की Market में अच्छी साख है या कोई साधारण Company है।

Class की Common Descriptions उसकी Class के सभी Objects के लिए समान होती हैं और Object की ये Common Descriptions एक Object को दुनिया के किसी भी अन्य Object से अलग बनाती हैं। जैसे एक Car और एक Book दोनों अलग Object हैं। Car को Car Class के Attributes से पहचाना जाता है जबकि Book को Book Class के Attributes से। Class किसी भी एक Object से दूसरे Object को अलग Represent करने का एक Description होता है जबकि एक Object की उसी Class के दूसरे Objects से अलग पहचान Object के Features या Attributes को प्रदान किए जाने वाले मान पर निर्भर होती है।

यानी यदि एक Book का नाम **"C In Hindi"** है और दूसरी Book का नाम **"OOPS with C++ In Hindi"** है, तो ये दोनों ही Books **Book** Class के उदाहरण (Instance) या सदस्य हैं, लेकिन दोनों ही Books अलग Objects को Represent कर रहे हैं, क्योंकि दोनों के **bookName** Attribute के मान या Data में अन्तर है।

Class एक Derived Data Type होता है, जिसे एक Programmer दुनिया के Actual Objects को Computer में Logically Represent करने के लिए Create करता है। किसी Real World Problem को Computer द्वारा Solve करने के लिए हमें हमेंशा एक ऐसे Data Type की जरूरत होती है, जो Real World Object को Reflect कर सके। ये सुविधा "C++" में हमें Class द्वारा प्राप्त होती है।

चलिए एक उदाहरण द्वारा समझने की कोशिश करते हैं कि एक Real World Problem द्वारा किस प्रकार से एक नए Data Type को प्राप्त किया जा सकता है जिसके आधार पर Computer में Logically किसी Object को Define व Declare किया जा सकता है।

Real World में भी Data (मान या मानों का समूह [**Value** or a **Set of Values**]) कई प्रकार के होते हैं। उदाहरण के लिए मानलो कि एक School का Principal अपनी School के विभिन्न Students की जानकारी को Computer पर Maintain करना चाहता है। वह चाहता है कि उसे जब भी जरूरत हो, वह Computer का प्रयोग करके अपनी School के किसी भी Student की Basic जानकारी को प्राप्त कर ले। यदि हम किसी Student की Basic Information को देखें तो किसी भी Student की निम्न Basic Information हो सकती हैं, जिन्हें School का Principal जानना पसन्द कर सकता है:

- Student का नाम
- Student के पिता का नाम
- Student का Address
- Student की City
- Student की तहसील

- ❖ Student का जिला
- ❖ Student का State
- ❖ Student की Class
- ❖ Student के Contact Number
- ❖ Student की Date of Birth
- ❖ Student की School में Join करने की Date of Admission
- ❖ Student की Age
- ❖ Student का Serial Number
- ❖ Student की जाति

ये कुछ ऐसी जानकारियां हैं, जिन्हें School का Principal जानना पसन्द कर सकता है। किसी Student की पूरी Information को इस प्रकार से टुकडों में विभाजित किया जा सकता है और जानकारी के इन सभी हिस्सों में कोई मान (Data) Fill किया जा सकता है। किसी जानकारी के इन छोटे–छोटे हिस्सों या टुकडों को **Field** कहा जाता है। जब विभिन्न Fields मिलते हैं तब किसी एक Student के बारे में पर्याप्त जानकारी या Information प्रदान करते हैं। यानी किसी समस्या से सम्बंधित विभिन्न Fields मिलकर उस समस्या के बारे में पूरी जानकारी प्रदान करते हैं। इन Fields के समूह को **Record** कहा जाता है। यानी हर Student का एक पूरा Record होता है जिसमें उसकी जानकारी के विभिन्न हिस्से Fields के रूप में होते हैं। इसी तरह के कई Records को जब एक साथ रखा जाता है, तो उस एक साथ रखे गए विभिन्न Records के समूह को **File** कहा जाता है।

Student की कुछ और भी जानकारियां हो सकती हैं, जो एक Student को अन्य Student से अलग बनाती हैं। जैसे Student क्या खाना पसन्द करता है, किस प्रकार से रहना पसन्द करता है, उसे कौनसा TV Channel अच्छा लगता है, आदि। लेकिन School का Principal Student की इन जानकारियों को जानने में Interested नहीं है। ये जानकारियां उसके लिए फालतू हैं। यानी केवल उपरोक्त जानकारियां School के Principal के लिए जरूरी हैं। किसी समस्या की प्रकृति के अनुसार समस्या की केवल जरूरी बातों को लेना और बिना जरूरी बातों को छोड देना, इस प्रक्रिया को Object Orientation में **Abstraction** कहते हैं।

किसी Student की इन सभी जरूरी जानकारियों को यदि मान प्रदान किया जाए तो हम किसी एक Student के इन Fields को निम्नानुसार विभिन्न मान प्रदान कर सकते हैं:

1	Student का नाम	=	Bal Gopal
2	Student के पिता का नाम	=	Nandlal
3	Student का Address	=	Bedal Road
4	Student की City	=	Falna
5	Student की तहसील	=	Bali
6	Student का जिला	=	Pali
7	Student का State	=	Rajasthan
8	Student की Class	=	10^{th}
9	Student के Contact Number	=	9352768938
10	Student की Date of Birth	=	06/03/1982
11	Student की School में Join करने की Date of Admission	=	03/07/1996
12	Student की Age	=	15
13	Student का Serial Number	=	1234545
14	Student की जाति	=	Brahmin

हम देख सकते हैं कि ये सभी Fields एक दूसरे से Logically Related हैं। यदि हम चाहें तो इसे एक Structure के रूप में Define कर सकते हैं और Structure एक नए प्रकार का User Defined Data Type ही तो होता है। "C++" में इस Structure को ही Modify करके Class के रूप में परिभाषित किया गया है। इसलिए हम इसी Description के आधार पर एक Class का निर्माण कर सकते हैं।

जब किसी Object को Computer में Logically Represent करने के लिए Class की Description या Class के Modal को कागज पर तैयार कर लिया जाता है, तब उस Class को Computer में Programming Language के अनुसार Define किया जाता है। Class को Logically Computer में Define करने के बाद उस Class के Objects Create किए जा सकते हैं और उनके साथ उसी प्रकार से काम किया जाता है, जिस प्रकार से किसी Real World Object के साथ काम किया जाता है। "C++" में एक Class को Define करने का Format निम्नानुसार होता है:

```
class class_name
{
private:
    Data_Members;
    Member_Functions;

public:
    Data_Members;
    Member_Functions;
};
```

class "C++" का एक Keyword है। किसी Class को Computer में Create करने के लिए इस Keyword का प्रयोग किया जाता है। इस Keyword के बाद एक Programmer को अपनी Class का नाम देना होता है। ये वही नाम होता है, जिसके आधार पर Programmer अपने Program में उस Class के Instances या Objects Create करता है।

यदि हम हमारी Book व Car Class के सन्दर्भ में देखें तो यहां हमें Book या Car लिखना होता है। ये एक Identifier होता है, इसलिए Identifier को नाम देने के लिए जिन नियमों का "C" में पालन करना होता है, उन नियमों का पालन यहां भी करना होता है।

यानी हम केवल Small व Capital Letters का प्रयोग या Underscore चिन्ह का प्रयोग अपनी Class के नाम में कर सकते हैं। यदि हमें अंकों का प्रयोग करना हो तो वह अंक Class के नाम की शुरूआत में नहीं आ सकता।

एक Student के Record को हम Class के रूप में Describe कर सकते हैं। उपरोक्त Description किसी भी Student के उन Fields को Represent करते हैं, जिनकी School के Principal को जरूरत है। यदि हम इन Description को सामान्य Fields में Convert करें तो हमें निम्नानुसार विभिन्न Fields प्राप्त होते हैं, जिनको Combined रूप में देखने पर Principal को किसी Student के बारे में वे सभी जानकारियां प्राप्त हो सकती हैं, जिन्हें उस School का Principal जानना चाहता है:

1 studentName

2 studentFName
3 studentAddress
4 studentCity
5 studentTehsil
6 studentDistrict
7 studentState
8 studentClass
9 studentContactNumber
10 studentDateOfBirth
11 studentDateOfAdmission
12 studentAge
13 studentSerialNumber
14 studentCast

चूंकि Student एक Real World Object है और इसे Computer में Logically Represent करना है, इसलिए हमें इसे Class द्वारा Represent करना होगा। चूंकि Student की सभी Basic जानकारियों के Fields जिनकी School के Principal को जरूरत है, हमारे पास Student Class की Description के रूप में उपलब्ध है, इसलिए हम इन्हीं Attributes का प्रयोग Student Class के लिए कर सकते हैं। लेकिन इससे पहले हम किसी Student के इन सभी Fields को मान प्रदान करते हैं। ये मान निम्नानुसार प्रदान किए जा सकते हैं:

1	studentName	=	Bal Gopal
2	studentFName	=	Nandlal
3	studentAddress	=	Bedal Road
4	studentCity	=	Falna
5	studentTehsil	=	Bali
6	studentDistrict	=	Pali
7	studentState	=	Rajasthan
8	studentClass	=	10^{th}
9	studentContactNumber	=	9352768938
10	studentDateOfBirth	=	06/03/1982
11	studentDateOfAdmission	=	03/07/1996
12	studentAge	=	15
13	studentSerialNumber	=	1234545
14	studentCast	=	Brahmin

हम देख सकते हैं कि एक Student के विभिन्न Attributes को मान यानी Data प्रदान किया जा सकता है। चूंकि हम जानते हैं कि Computer में मानों "**Values / Set of Values**" को Store करने के लिए Variables का प्रयोग किया जाता है, इसलिए इस Record के आधार पर हम समझ सकते हैं कि Student के हर Attribute को Computer में Store करने के लिए हमें एक Variable की जरूरत होगी। अब Variable किस Data Type का होगा, ये हम उपरोक्त Record में Fill किए गए मानों के आधार पर तय कर सकते हैं। चलिए इस Record के विभिन्न Attributes को Class में Describe करने के लिए हर Attribute का Data Type तय करते हैं।

हमेंशा Data Type तय करते समय हमें सबसे पहले ये तय करना होता है कि किसी Field का या किसी Attribute का मान Numerical होगा या Non-Numerical, यदि मान Numerical है, तो

हमें ये देखना होता है कि उस Numerical Field के साथ किसी प्रकार की प्रक्रिया हो सकती है या नहीं। यदि प्रक्रिया हो सकती है, तब तो उस Attribute को उसकी Size के अनुसार किसी Numeric Data Type का Declare करना चाहिए अन्यथा उस Numerical दिखाई देने वाले Field या Attribute को भी String प्रकार का ही Declare करना चाहिए। चलिए, इसी आधार पर हम Student के विभिन्न Attributers का Data Type तय करते हैं।

चूंकि Student का नाम, उसके पिता का नाम, Address, जाति, City, District, तहसील, State, Class ये सभी ऐसे Fields हैं, जिनमें Non-Numerical Data Store होगा, इसलिए इन सभी को Character प्रकार के Data Type के एक One – Dimensional Array के रूप में Declare करना होगा। Admission की Date, Birth of Date व Age ये तीनों ऐसे Fields हैं, जिनके साथ ये जानने के लिए Processing की जा सकती है, कि कोई Student कितने समय से School में है और कितने दिन से School में है। ये जानकारी प्राप्त करने के लिए Current Date को Admission Date में से घटाया जा सकता है। साथ ही ये तीनों ही पूर्णांक संख्याएं हैं इसलिए इस प्रकार के Fields को Integer प्रकार का Declare करना होगा। Serial Number, Class व Contact Number ये तीन Attributes ऐसे हैं कि इनमें Store तो संख्या ही होती है, लेकिन इन संख्याओं के साथ किसी प्रकार की Calculation नहीं की जा सकती है, इसलिए यदि हम चाहें तो इन्हें String प्रकार के Variable में Store कर सकते हैं। लेकिन String प्रकार का Variable एक Character को Store करने के लिए 1 Byte लेता है। इसलिए यदि 10 अंकों के Contact Number को Store करना हो, तो हमें दस Byte की Memory इसके लिए Reserve करनी होगी, जबकि यदि हम **long** प्रकार का Variable Declare करते हैं, तो हम केवल 8 Bytes में ही दस अंकों की संख्या Store कर सकते हैं। इसलिए यदि हम चाहें तो इन्हें Long प्रकार का Declare कर सकते हैं। इस प्रकार से हम Students के सभी Attributes के Data Type तय कर सकते हैं। Data Type तय करने के बाद इन Data Type के साथ हमें इन Attributes को केवल Class के Attributes Section में लिखना होता है। यानी हम एक Student की Class को "C++" Language द्वारा निम्नानुसार Computer में Logically Describe कर सकते हैं:

Student Class Defining

```
class Student
{
    private:
        String studentName[20];
        String studentFName[20];
        String studentAddress[40];
        String studentCity[15];
        String studentTehsil[15];
        String studentDistrict[15];
        String studentState[15];
        char studentClass[1];
        long studentContactNumber;
        long studentDateOfBirth;
        long studentDateOfAdmission;
        char studentAge[1];
        String studentSerialNumber[10];
        String studentCast;

    //Behaviors of the Objects
};
```

चूंकि किसी Class के Create होने वाले सभी Instances एक अलग Object होते हैं और उनके Attributes के मान भी अलग–अलग होते हैं। जैसे **Govind** नाम के किसी Student के Record व **Shyam** नाम के किसी Student के Record, इन दोनों Objects के Record के हर Field या Attribute का मान अलग होगा। ये अलग मान Memory में तभी Store हो सकते हैं जब हर Object के इन Attributes के लिए Memory में अलग Space हो। वास्तव में होता भी ऐसा ही है।

जैसाकि हमने पहले भी कहा कि Class तो मात्र किसी Object के Attributes की एक Description होती है। उस Description के अनुसार जितने भी Objects बनते हैं, उन सभी Objects के हर Attribute को Store होने के लिए एक अलग Variable मिलता है। चूंकि हर Object किसी Class का एक Instance होता है इसलिए हर Object के इन Attributes को Represent करने वाले Variables को भी **Instance Variable** कहते हैं। इसी उदाहरण के आधार पर हम Car व Book की Class को भी Create कर सकते हैं।

हम देख सकते हैं कि हमने Student के विभिन्न Attributes को **private:** Section में Declare किया है। हमने ऐसा इसलिए किया है, ताकि किसी Object के Data को केवल वही Object Access कर सके। Outer World के लिए किसी Object के लिए ये Data Hide रहते हैं।

ये प्रक्रिया ठीक उसी प्रकार की है जिस प्रकार से आपके घर के विभिन्न सामानों को केवल आप ही उपयोग में ले सकते हैं। कोई अनजान व्यक्ति आपके घर में घुस कर आपके सामान को तब तक उपयोग में नहीं ले सकता जब तक कि आप उसे इस बात की अनुमति प्रदान ना करें।

हमने Class तो बना दी लेकिन ये Class अभी पूरी नहीं है। जिस तरह से हर Object की कुछ States या Situations होती हैं, जिनसे वह Object अन्य Objects से अलग पहचान प्राप्त करता है। इसके अलावा हर Object किसी ना किसी प्रकार का व्यवहार भी करता है। यानी हर Object में व्यवहार करने की भी कुछ (**Abilities**) क्षमताएं होती हैं।

जैसे एक पक्षी Class के Instances को देखें तो हर पक्षी के पास पंख होते हैं, हर पक्षी के चोंच होती है। ये दोनों Features तो ये बताते हैं कि कोई पक्षी किस तरह का दिखता (Appearance) है। इसके अलावा एक पक्षी कुछ काम भी कर सकता है। यानी वह दाना खा सकता है और उड भी सकता है।

यानी हर Object की अपनी कुछ स्थितियां (**States**) होती हैं और हर Object की अपनी कुछ क्षमताएं (**Abilities**) होती हैं। Object की उन क्षमताओं से Object की States में परिवर्तन होता है। किसी भी Object की स्थिति (**State and Appearance**) के मान में परिवर्तन हो सकता है।

यदि कार का ही उदाहरण लें तो कोई Car Yellow Color की भी हो सकती है और किसी दूसरी Car का Color Blue भी हो सकता है। हम किसी Car के Color को Red Color से भी Paint कर सकते हैं और White Color से भी Paint कर सकते हैं। Real World में किसी भी Object की स्थिति में परिवर्तन किया जा सकता है। उसी प्रकार से Computer में भी किसी Object के Color State या Attribute का मान Change किया जा सकता है।

हम ये भी कह सकते हैं कि Real World में एक Object में उसके Attributes के मान को Change करने की क्षमता होती है। एक Real World Object जिस Operation को Perform करके अपनी स्थिति या Appearance में परिवर्तन करता है, उस Operation को Object का **Behavior** या Object की **Ability** कहते हैं। चलिए, एक और उदाहरण देखते हैं।

हम सभी ने देखा है कि बडे शहरों में कई–कई मंजिलों की इमारतें होती हैं इसलिए अक्सर उन मंजिलों पर पहुंचने के लिए Lift का प्रयोग किया जाता है। Lift को सामान्यतया Elevator कहते हैं। Elevator भी एक Real World Physical Object है, क्योंकि हम इसे देख और छू सकते हैं। यदि हम इसकी States व Appearance यानी Characteristics को Describe करें तो इस Elevator का निम्नानुसार Description दे सकते हैं:

1. एक Elevator किसी ना किसी मंजिल (Floor) पर रूका हुआ या स्थित हो सकता है।
2. एक Elevator अधिकतम चार लोगों को वहन कर सकता है।
3. एक Elevator में कुछ Buttons होते हैं, जिनको Press करके विभिन्न Passengers किसी ना किसी Floor पर पहुंच सकते हैं।
4. एक Elevator में एक दरवाजा हो सकता है जो Open व Close होता है।

किसी Elevator में ये चारों गुण हो सकते हैं जो उस Elevator की स्थिति व Appearance को Describe कर रहे हैं। लेकिन ये Elevator Passengers की इच्छानुसार कुछ Operations भी Perform कर सकता है, क्योंकि इन Operations को Perform करने की Ability भी एक Elevator Object में होती है। Elevator की इन Abilities को निम्नानुसार Describe किया जा सकता है:

1. एक Elevator किसी भी समय किसी ना किसी Floor पर स्थित होगा, इसलिए वह अपनी Current State में परिवर्तन करने के लिए अपनी मंजिल (Floor) से ऊपर की मंजिल (Floor) पर जा सकता है।
2. एक Elevator किसी भी समय किसी ना किसी Floor पर स्थित होगा, इसलिए वह अपनी Current State में परिवर्तन करने के लिए अपनी मंजिल (Floor) से नीचे की मंजिल (Floor) पर आ सकता है।
3. Passengers Elevator में प्रवेश कर सकें और Elevator से बाहर निकल सकें, इसके लिए एक Elevator का दरवाजा (Door) Open या Close हो सकता है।
4. जब कोई Passenger किसी मंजिल पर जाने के लिए उस मंजिल का कोई Number Press करता है, तब Elevator ये Calculate कर सकता है कि उसे किस जगह जाना है।
5. एक Elevator ये भी पता लगाता है कि उस Building के अन्य Elevators किस स्थिति (Floor) पर हैं।

इस Elevator Object की States या Characteristics व Object द्वारा उन Characteristics पर Perform किए जा सकने वाले Operations को हम निम्नानुसार Describe कर सकते हैं:

```
//Characteristics (Attributes or States)
    1  noOfCurrentFloor
    2  noOfPassengers
    3  listOfButtonsPushed

//Performable Operations (Behaviors or Abilities)
    1  GoDown
    2  GoUp
    3  OpenDoors
    4  CloseDoors
    5  GetOtherElevatorsInfo
    6  CalculateWhereToGoNext
```

इस उदाहरण से हम समझ सकते हैं कि एक Real World Physical Object में भी वास्तव में दो Components होते हैं। पहला Component Object की स्थिति बताता है और दूसरा Component Object की क्षमता बताता है। जब एक Real World Physical Object के भी दो Component होते हैं तो उस Object को Represent करने वाली Logical Class में भी इन दोनों Components का Description होना चाहिए, ताकि Computer का Logical Object Real World के Physical Object को पूरी तरह से Represent करे।

एक और उदाहरण लेते हैं, Real World के Class व Objects की परिभाषा के अनुसार दुनिया की हर चीज को उसकी Class बना कर, उस Class के Instance यानी Object के रूप में परिभाषित किया जा सकता हैं। इसी आधार पर आप भी एक Real World Physical **Object** हैं और किसी एक समय पर आप भी किसी ना किसी एक Class के Instance होते हैं।

उदाहरण के लिए मान लीजिए कि आप एक Student हैं और 10^{th} Class में पढते हैं इसलिए आप भी एक Student का उदाहरण (Instance) हैं क्योंकि आपको किसी कक्षा (Class) का एक विद्यार्थी (Object) कहा जा सकता है। आप जैसे ही कई और Students होंगे जो आपके साथ पढते होंगे।

मानलो कि 10^{th} Class में पढने वाले सभी Students को मुख्य रूप से 6 Subjects पढने होते हैं। सभी Students को सभी प्रकार के Exams देने होते हैं। सभी Students समान समय पर Class में जाते हैं और एक समय में सभी Students समान Subject पढते हैं। जब एक कक्षा के सभी Students उस कक्षा में जो भी काम करते हैं, वे सभी काम सभी को एक साथ करने होते है तो इस आधार पर हम ये कह सकते हैं कि आप सभी Student एक ही कक्षा (Class) के विद्यार्थी(Objects) हैं, क्योंकि आप सभी एक ही कक्षा की सभी स्थितियों (States) को एक साथ वहन करते हैं।

सभी Students को एक समूह के रूप में देखा जाए तो सभी Students Objects का एक ऐसा समूह (**Entity Set**) है, जो कई समान Features को Share करते हैं। चूंकि आप 10^{th} कक्षा के सभी नियमों (**Descriptions**) का पालन करते हैं, इसलिए आप 10^{th} कक्षा (**Class**) के Students (**Objects**) हैं।

इस उदाहरण में हमने देखा कि एक Student उसकी कक्षा का एक विद्यार्थी होता है। ये Description वास्तव में कक्षा का Description है और कक्षा के Description के आधार पर बनी Class का Instance विद्यार्थी होता है। यदि कक्षा के Description को प्रदर्शित करने वाले Attributes (Instance Variable) के मान को Change किया जाए, तो Student का पूरा समूह Change हो जाता है, ना कि केवल एक Student Change होता है।

यानी यदि 10^{th} कक्षा की Description Change करके 11^{th} कक्षा की Description ले लिया जाए, तो उस कक्षा के सभी विद्यार्थी बदल जाते हैं, क्योंकि वास्तव में पूरी कक्षा ही बदल जाती है। लेकिन यदि हम चाहते हैं कि हम Particular किसी विद्यार्थी की Description बनाएं तो हमें कक्षा की Class बनाने के बजाय Student की Class बनानी होगी।

जब किसी Student के Class की Description तय करनी होती है या एक Student को Represent करने वाला Modal (Class) Create करना होता है, तब हमारा मुख्य Object Student होता है। इसलिए हमें केवल Student के Attributes व States का ही पता लगाना होता है।

अब यदि देखा जाए किस एक Student के समूह (**Entity Set**) की ऐसी कौनसी Characteristics होती हैं, जो उसे दुनियां के सभी Objects से अलग बनाते हैं, तो हम एक Student की निम्न Attributes या States प्राप्त कर सकते हैं:

1. Student Entity Set के हर Entity का एक Serial Number होता है।
2. Student Entity Set के हर Entity का एक नाम होता है।
3. Student Entity Set के हर Entity के पिता का नाम होता है।
4. Student Entity Set के हर Entity का एक Address होता है।
5. Student Entity Set के हर Entity की एक City होती है।
6. Student Entity Set के हर Entity की एक District होती है।
7. Student Entity Set के हर Entity का एक State होता है।
8. Student Entity Set के हर Entity के City का एक Pin Code Number होता है।
9. Student Entity Set के हर Entity की कोई जाति होती है।
10. Student Entity Set के हर Entity का कोई रंग होता है।
11. Student Entity Set के हर Entity की कोई कक्षा होती है।
12. Student Entity Set के हर Entity के कद की एक लम्बाई होती है। आदि

इसी तरह से हम एक Student की कई और Attributes व States का पता लगा सकते हैं। चलिए, Real World में हम देखते हैं कि हर Object अपना काम खुद करता है।

मानलो कि आप एक Student हैं। क्या आपको भूख लगने पर आपका खाना कोई और खा सकता है। क्या आपको प्यास लगने पर आपके लिए कोई और पानी पी सकता है। क्या ऐसा होता है कि School जाते समय आपकी Uniform कोई और पहने और School आप चले जाएं। क्या ऐसा होता है कि आपके बदले कोई और आपका Exam Fight करे और मिलने वाला Certificate आपके नाम का हो और आप को प्राप्त हो जाए।

नहीं! Real World में ऐसा नहीं होता। इसीलिए Computer में भी यदि ऐसा होता है, तो उस Programming Language को Object Oriented नहीं कहा जा सकता। "C++" में भी ऐसा नहीं होता है। किसी Class के Description के आधार पर जो Object Create किया जाता है, उस हर Object का अपना स्वयं का Attribute या States को Represent करने वाला स्वतंत्र Variable होता है, जिसमें केवल उसी Object के Attributes या States के मान होते हैं।

चलिए, अब Object के Behavior या Ability पर थोडा और विचार करते हैं। हम सभी इन्सान हैं। हमारी एक Physical आकृति, रंग, रूप, लम्बाई आदि है, इसलिए हम सभी Human Being Class के Instances या Objects हैं। यदि हम एक Human Being Class का Description देना चाहें, तो निम्न Description दे सकते हैं:

1. हर Human Being का एक नाम होता है।
2. हर Human Being Object के पास उसका स्वयं का दिमाग होता है।
3. हर Human Being के दो हाथ होते हैं।
4. हर Human Being के दो पैर होते हैं।
5. हर Human Being के एक सिर होता है।
6. हर Human Being के सिर पर बाल होते हैं।
7. हर Human Being कान होते हैं
8. हर Human Being के दो आंखें होती हैं।
9. हर Human Being के चेहरे पर एक मुंह होता है।
10. हर Human Being के दोनों हाथों में पांच–पांच अंगुलियां होती हैं। आदि

इन सभी States व Attributes के अलावा एक Human Being में कुछ Abilities भी होती हैं। कुछ Abilities को निम्नानुसार Describe किया जा सकता है:

1. हर Human Being अपना नाम Change कर सकता है।

```
2  हर Human Being Object अपने दिमाग से सोंचता है और निर्णय लेता है।
3  हर Human Being अपने हाथों का प्रयोग करके किसी प्रकार का इशारा कर सकता है।
4  हर Human Being अपने पैरों से चल सकता है।
5  हर Human Being भाग सकता है।
6  हर Human Being अपने सिर पर टोपी रख सकता है।
7  हर Human Being अपने सिर पर पगडी बांध सकता है।
8  हर Human Being अपने सिर के बिखरे हुए बालों को कंघे से संवार सकता है।
9  हर Human Being अपने बाल कटवा सकता है।
10 हर Human Being अपने बाल बडे कर सकता है।
11 हर Human Being अपने कानों से सुन सकता है।
12 हर Human Being अपनी आंखों से देख सकता है।
13 हर Human Being अपनी आंखों को बन्द करके सो सकता है।
14 हर Human Being अपनी आंखों से इशारे कर सकता है।
15 हर Human Being अपने मुंह से बोल सकता है।
16 हर Human Being अपने मुंह से खाना खा सकता है।
17 हर Human Being अपने हाथों की अंगुलियों में कोई चीज पकड सकता है।
18 हर Human Being अपने हाथों की अंगुलियों से कोई चीज फेंक सकता है।
```

यहां हमने कुछ ऐसे काम बताए जो सभी Human Being Class के Instances यानी इन्सान Objects कर सकते हैं। ऐसे और भी हजारों उदाहरण दिए जा सकते हैं। क्या आपको Human Being Object के States व Behavior में किसी प्रकार का कोई सम्बंध पता चला ?

हर Object के Attributes व Objects के Operations में एक सम्बंध होता है और सम्बंध बस यही है कि एक Object कुछ Operations Perform करके अपने ही किसी Attribute की State को Change करता है। हम इस Human Being उदाहरण को ही लें, तो हर Human Being Object के बाल होते हैं और हर Human Being Object किसी प्रकार का Operation Perform करके अपने बालों की स्थिति में परिवर्तन कर सकता है। बालों की स्थिति में परिवर्तन करना Object की एक Ability है। इसी तरह से हर Human Being Object के पैर होते हैं। ये पैर Human Being Object का Attribute है। इस Attribute की State को Change करने के लिए Object उस पर चलने का Operation Perform कर सकता है। Object के चलने से Object के पैर की स्थिति (State) में परिवर्तन होता है। हर Human Being Object के पास अपना मुंह होता है जो कि Human Being Object का एक Attribute है। यदि Human Being खाना खाता है, तो उसका मुंह चलता है यानी मुंह की स्थिति (State) में परिवर्तन होता है। यानी यदि हम सारांश में कहें तो कह सकते हैं कि हर Object में किसी भी प्रकार का Operation Perform करके अपने स्वयं के Attributes की स्थिति (**States**) में परिवर्तन करने की क्षमता (**Ability**) होती है।

एक Object केवल स्वयं ही अपने Attributes की स्थिति (State) में परिवर्तन करने के लिए, किसी प्रकार का Operation Perform कर सकता है। यानी किसी Object A का कोई Operation किसी Object B के किसी Attributes की स्थिति (States) में परिवर्तन नहीं कर सकता है। एक Object A के विभिन्न Attributes की States को Change करने का अधिकार केवल उसी Object A के पास ही होता है।

इस बात को यदि Human Being के उदाहरण से समझें तो कोई Human Being Object स्वयं ही ये निर्णय कर सकता है कि उसे उसके किसी Attribute की स्थिति में परिवर्तन करने के लिए कोई Operation Perform करना है या नहीं। यानी किसी Human Being को अपने बाल कटवाने हैं या नहीं, इस बात का निर्णय केवल वह Human Being ही ले सकता है। ऐसा नहीं हो सकता कि उस Human Being के बदले कोई दूसरा Human Being ये निर्णय करे और बिना उसकी Permission लिए उसके बाल काट दे। यानी एक ही Class के विभिन्न Objects भी एक दूसरे पर

बिना Permission लिए किसी भी प्रकार का Operation Perform करके किसी दूसरे Object के Attributes की स्थिति में परिवर्तन नहीं कर सकते हैं।

इस पूरे Discussion का यदि सारांश निकालें तो हम निम्न सारांश प्राप्त कर सकते हैं :

1. दुनिया के हर Object को उसके समूह के आधार पर एक Class के रूप में Describe किया जा सकता है।
2. एक बार किसी एक Object के आधार पर Class की Description बना लेने के बाद, उस Class के जितने चाहें उतने Instances Create कर सकते हैं। इन Instances को ही Object कहा जाता है।
3. हर Object के दो Components Attributes की **States** व Object की **Ability** होते हैं, जिनके आधार पर उस Object की Class का Description तैयार किया जाता है।
4. किसी Objects के Attributes की States व उन Attributes की States में Object द्वारा किए जाने वाले परिवर्तनों की Description, इन दोनों को एक साथ मिलाकर एक Entity के रूप में Use करना **Encapsulation** कहलाता है। इस Entity को ही **Object** कहते हैं और Encapsulation की ये प्रक्रिया Class पर Perform की जाती है।
5. एक Object के Attributes की स्थिति (State) के मान (Data) में केवल वह Object ही किसी Operation द्वारा परिवर्तन कर सकता है, अन्य किसी भी Object को ये अधिकार नहीं होता है, कि वह किसी दूसरे Object के Attributes की State में अपने किसी Operation द्वारा परिवर्तन करें। इस स्थिति में किसी Object के Data को केवल वह स्वयं ही Access कर सकता है। यानी एक Object का Data केवल उसी Object के लिए उपलब्ध रहता है, दूसरे Objects के लिए वह Data Hide रहता है। इसलिए Data के इस प्रकार से किसी Object के अन्दर छुपे हुए रहने के तरीके (Structure) को **Data Hiding** कहते हैं।

चलिए, अब Student की Class की Description को पूरा करते हैं। चूंकि हमें Student के सभी Attributes पता हैं जिनकी जरूरत School के Principal को है। लेकिन एक School में तो कई Students होंगे। साथ ही School में नए Students का Admission भी होता होगा और किसी पुराने Student को School से Transfer भी किया जाता होगा या निकाला जाता होगा। कई बार Student की Information को Update भी करना पडता होगा।

जैसे मानलो कि Student के पिता ने अपना घर बदल लिया है, तो Computer में Student के Address को भी Change करना होगा। School में तो कई Students होते हैं, इसलिए किसी विशेष Student को खोजना भी होगा ताकि School के Principal को किसी Student की जानकारी प्राप्त करने के लिए सभी Students के Records को ना Check करना पडे। इसी तरह से यदि सभी Students के Records को एक क्रम में Store किया जाए, तो Students के Records को Manage करना सरल होगा। इस तरह से हम किसी Student के Records को Manage करने के लिए निम्नानुसार Operations की Description तैयार कर सकते हैं:

```
//Ability Description (Operation Description)
    addNewStudent
    removeStudent
    updateStudent
    searchStudent
    sortStudent
```

Computer में किसी Object के विभिन्न Logical Operations को जो कि एक Object अपने Attributes के States को Change करने के लिए Perform करता है, उन सभी Operations को

Methods से Represent किया जाता है। Methods के बारे में हम आगे चर्चा करेंगे, लेकिन यदि Student Class के Operations को यदि Methods के रूप में लिखना हो, तो हम "C++" में Methods या "Member Function" को निम्नानुसार Describe करते हैं:

```
//Ability Description (Operation Description)
Return_Data_Type addNewStudent(Arguments)
{
    //Code Statements ;
}

Return_Data_Type removeStudent(Arguments)
{
    //Code Statements ;
}

Return_Data_Type updateStudent(Arguments)
{
    //Code Statements ;
}

Return_Data_Type searchStudent(Arguments)
{
    //Code Statements ;
}

Return_Data_Type sortStudent(Arguments)
{
    //Code Statements ;
}
```

यदि हम Student Class की पूरी Description "C++" Syntax के अनुसार लिखना चाहें, तो ये Class Description निम्नानुसार होगाः

Student Class Defining

```
class Student
{
    //Attributes Description
    private:
        String studentName[20];
        String studentFName[20];
        String studentAddress[40];
        String studentCity[15];
        String studentTehsil [15];
        String studentDistrict[15];
        String studentState[15];
        byte studentClass;
        long studentContactNumber;
        long studentDateOfBirth;
        long studentDateOfAdmission;
        byte studentAge;
```

```
        String studentSerialNumber[10];
        String studentCast;

    //Ability Description (Operation Description)
    public:
        Return_Data_Type addNewStudent(Arguments)
        {
            //Code Statements ;
        }

        Return_Data_Type removeStudent(Arguments)
        {
            //Code Statements ;
        }

        Return_Data_Type updateStudent(Arguments)
        {
            //Code Statements ;
        }

        Return_Data_Type searchStudent(Arguments)
        {
            //Code Statements ;
        }

        Return_Data_Type sortStudent(Arguments)
        {
            //Code Statements ;
        }
};
```

अब हमारी Student Class पूरी तरह से तैयार है। हमें केवल इसके Operations को Define करना है। Operations को Define करने के बाद इस Class के Objects Create किए जा सकते हैं और उनके साथ Interaction किया जा सकता है। यदि हमें इस Class के Objects Create करने हों, तो हम उसी प्रकार के Syntax का प्रयोग करते हैं, जिस प्रकार के Syntax का प्रयोग करके हम कोई Integer प्रकार का Variable Create करते हैं।

हम ऐसा इसलिए कर सकते हैं, क्योंकि Class के रूप में हम एक नए प्रकार का Data Type ही Develop करते हैं। यदि सरल शब्दों में कहें तो हमने यहां एक Student प्रकार का Data Type Create किया है और हम इस Data Type के कई Variables Create कर सकते हैं, जिन्हें Class का Instance या Object कहते हैं।

इस Class में हम देख सकते हैं कि हमने Object के विभिन्न Attributes को **private** Section में रखा है जबकि विभिन्न Member Functions को **public** Section में। ऐसा इसलिए किया है ताकि किसी Object के Data हमेंशा सुरक्षित रहें और किसी भी अन्य Object का Member Function या कोई Function किसी दूसरे Object के Data को Access ना कर सके।

जबकि Member Functions को public इसलिए रखा है ताकि वह Object स्वयं अपने Data को कहीं से भी किसी भी स्थान से Access कर सके। इससे पहले कि हम Object Oriented

Programming के Concept को आगे बढाएं, हम पहले C++ का Program बनाने की प्रक्रिया को समझ लेते हैं और C++ Program की Coding के लिए जरूरी Elements को जान लेते हैं।

Basic C++ Data Types

हम हमारे जीवन में विभिन्न प्रकार के Data Use करते हैं। यदि ध्यान से देखा जाए, तो मुख्य रूप से कुल तीन प्रकार के ही Data हो सकते हैं, जिन्हें हम Use करते हैं। या तो हम पूर्णांक संख्याओं के रूप में किसी Data को Represent करते हैं, या किसी दसमलव वाली संख्या के रूप में अथवा Character के रूप में।

ठीक इसी Concept को ही Computer Programming Language में भी Use किया गया है। C++ का Compiler भी हमे इन तीन प्रकार के Basic Data Type प्रदान करता है। इनमें एक Data Type Character को Represent करता है। दूसरा Integer प्रकार का Data Type है, जिसे तीन अलग–अलग प्रकार के Data Type में विभाजित किया गया है और इसी तरह से दसमलव वाली संख्याओं को Represent करने वाला float प्रकार का Data Type है, जिसे तीन अन्य अन्य Real Numbers या Floating Point Numbers को Represent करने वाले Data Type में विभाजित किया गया है। इन्हें C++ में और भी कई भागों में विभाजित किया गया है, ताकि Data को विभिन्न तरीकों से Represent किया जा सके। इसे निम्नानुसार दर्शाया जा सकता है:

Type Name	Used to Store	Examples of Values Stored
char	Characters	'a', 'B', '$', '3', '?'
short	Small whole numbers	, 30,000, -222
int	Normal-sized whole numbers	same as short or same as long)
long	Large whole numbers	,000,000,000, -123,456,789
float	Small real numbers	.7, 199.99, -16.2, 0.000125
double	Large real numbers	,553.393.95,47, -0.048512934
long double	Extra-large real numbers	,123,456,789,012,345.666

चलिए, अब हरेक को समझते हैं। जब हमें Memory में Characters को Store करने के लिए Space बनाना होता है, तब हम निम्न Statement द्वारा Character प्रकार का Variable Declare करते हैं:

```
char character1;
```

ये Statement Memory में एक Character Store करने के लिए 1 Byte की Memory Reserve करता है और उसका नाम Character1 रख देता है। अब यदि हमें इस Variable में कोई Character Store करना हो, तो हम निम्न Statement लिखते हैं :

```
character1 = 'A';
```

Character को हमेंशा Single Quote में लिखते हैं। जैसे 'A', 'x', '%' आदि।

Assignment Operator (=)

= के चिन्ह के Right Side में जो मान होता है उसे Compiler = चिन्ह के Left Side के Variable में Store कर देता है। जैसाकि ऊपर बताए गए Statement में character1 में A को

Assign किया गया है। ये Statement Execute होने के बाद Variable Character1 में 'A' Store हो जाता है। = चिन्ह को Assignment Operator कहा जाता है क्योंकि ये Operator इसके Right Side के मान को Left Side के Variable में Store कर देता है। Computer की Memory में वास्तव में सभी Characters Integer के रूप में Store होते हैं जिसे Computer समझता है। ASCII Codes का प्रयोग Characters को Integer में Convert करने के लिए होता है। जैसे 'A' का ASCII Code 65 है, 'B' का 66 आदि।

Escape Sequences

C++ में कुछ Special Characters होते हैं जो विशेष काम करते हैं। इन्हें Back Slash के बाद लिखा जाता है। इन्हें Escape Sequence Characters कहा जाता है। ये Special Characters निम्नानुसार हैं:

Escape Sequence	Character Represented
'\n'	इसे New Line Character Constant कहते हैं। ये हमारे Program में नई Line देता है।
'\t'	ये Tab Space प्रदान करता है।
'\b'	ये Backspace देता है।
'\r'	Carriage Return यानी Cursor को Line की शुरूआत में रख देता है। जब भी हम Enter Key Press करते हैं, ये Escape Character Generate होता है।

Character प्रकार के Variables का प्रयोग अक्सर छोटी संख्याएं Store करने के लिए भी होता है। जैसे

```
character1 = 60;
```

चूंकि Character प्रकार के Variable की Size केवल 1 byte या 8 Bit की होती है, इसलिए यदि संख्या बिना चिन्ह की यानी unsigned हो तो इसमें अधिकतम –128 से +127 तक की संख्या ही Store हो सकती है। जब हमें बहुत ही छोटे मान के साथ काम करना होता है, तब हम इस प्रकार के Data Type का प्रयोग करते हैं।

Integers

जब हमें किसी Variable में ऐसी संख्या Store करनी हो जो पूर्णांक हो, तो हम Integer प्रकार के Data Type का Variable Declare करते हैं। इस प्रकार का Variable Memory में 2 Byte या 16 Bit की Space लेता है इसलिए ये अधिकतम -32768 से 32767 तक का मान Store कर सकता है। इस प्रकार के Data Type का प्रयोग छोटे पूर्णांकों को Memory में Store करने के लिए होता है।

Type Name	Size	Range
char	1 byte (8 bits)	-128 to 127
short	2 bytes (16 bits)	-32,768 to 32,767
int	Same as short on 16-bit systems,	Same as long on 32-bit systems
long	4 bytes (32 bits)	-2,147,483,648 to 2,147,483,647

short प्रकार का Variables हमेंशा Memory में 2 Byte की Space Reserve करता है और ये –32768 से 32767 तक के मान को Store कर सकता है। Long प्रकार का Variable हमेंशा Memory में 4 byte की Space लेता है और ये –2147483648 से 2147483647 तक के मान को Store कर सकता है।

16-Bit Computers में int व short प्रकार का Variable 2 Byte Reserve करता है जबकि 32-Bit Computers में ये 4 Byte Reserve करते हैं और 16-Bit Systems की तुलना में अधिक मान को Memory में Store कर सकते हैं। DOS व Windows 3.1 16-Bit Systems हैं। Windows 95/98 Unix, Linux आदि 32-Bit Systems हैं।

Integer प्रकार के मान को Store करने के लिए int प्रकार के Variable का प्रयोग सबसे ज्यादा होता है, चाहे हम 16-Bit OS Use करें या 32-Bit OS लेकिन यदि हम 32-Bit OS पर कम Space Use करने के लिए int प्रकार के Variable को 2 byte का रखना चाहते हैं, तो हम short का प्रयोग कर सकते हैं। इसी तरह यदि हमें 16-Bit OS पर 4 Byte का Integer Store करना हो तो हमें long प्रकार का Variable लेना होगा।

```
int KeloMeters;                 // Variable Declare किया गया है।
long CentiMeters;

KeloMeters = 1024;              // Variables को मान Assign किया गया है।
CentiMeters = 1405836L;
```

यहां हमने Centimeters को मान Assign करने के बाद L का प्रयोग किया है, क्योंकि 16-Bit System में Integer 2 Byte का होने की वजह से ये मान Fit नहीं हो सकता। 16-Bit System में इस मान को मान्यता देने के लिए यहां L का प्रयोग करना जरूरी है।

Unsigned Integers

जब हमें किसी Variable में केवल पूर्णांक मान ही Store करने होते हैं साथ ही हमें Variable में केवल Positive मान ही Store करने होते हैं, तब हम Variable को Unsigned प्रकार का Declare करते हैं। इससे Variable की Data Store करने की Range दुगुनी हो जाती है। यानी Negative व Positive Limit की जोड के बराबर हो जाती है।

Type Name	Size	Range
uunsigned char	1 byte (8 bits)	0 to 255
unsigned short	2 bytes (16 bits)	0 to 65,535
unsigned int or unsigned	Same as unsigned short on 16-bit systems Same as unsigned long on 32-bit systems	
unsigned long	4 bytes (32 bits)	0 to 4,294,967,295

unsigned Keyword के बिना हम जो भी Variable Declare करते हैं, By default वे सभी signed Variables होते हैं। हम चाहें तो Variable के साथ signed Keyword का प्रयोग कर सकते हैं, लेकिन ऐसा करना जरूरी नहीं है।

Floating Point

जब हमें हमारे Program में ऐसे Variables Declare करने होते हैं जो दसमलव वाली संख्याओं को Store करने के लिए उपयोग में आने वाली हों, तो हम ऐसे Variables को Float प्रकार के Data Type का Declare करते हैं। हम इन दसमलव वाले मानों को घातांक के रूप में भी Express कर सकते हैं।

जैसे 101.456 को 1.01456e2 भी लिख सकते हैं जहां e के बाद का अंक 2 ये बताता है कि यदि इस मान को बिना घातांक वाली संख्या में बदलना हो तो जहां वर्तमान में दसमलव है, वहां से दसमलव हटा कर दो स्थान Right में दसमलव लगा दिया जाए। आगे की सारणी में Floating Point Data Type की Range दशार्यी गई है:

Type Name	Size	Range	Precision
float	4 bytes (32 bits)	10e-38 to 10e38	5 digits
double	8 bytes (64 bits)	10e-308 to 10e308	15 digits
long double	10 bytes (80 bits)	10e-4932 to 10e4932	19 digits

निम्न उदाहरण से इनके प्रयोग को बताया गया है जहां हम देखते हैं कि हम किसी प्रकार के Data Type के Variable को दसमलव के बाद कितने अंकों तक मान Assign कर सकते हैं।

```
float pi_float;
double pi_double;
long double pi_long_double;

pi_float = 3.1415;
pi_double = 3.14159265358979;
pi_long_double = 3.141592653589793238;
```

हम जब पहली बार किसी Data Type का Variable Create करते हैं, उसी समय हम Variable को मान प्रदान कर सकते हैं। यानी हमने उपर जो 6 Statements लिखें हैं उसके स्थान पर हम केवल तीन Statements से भी अपना काम कर सकते हैं। यानी निम्नानुसार Declaration भी कर सकते हैं:

```
float pi_float = 3.1415;
double pi_double = 3.14159265358979;
long double pi_long_double = 3.141592653589793238;
```

C++ के IDE में हम जिनते चाहें उतने Spaces का प्रयोग कर सकते हैं। C++ का Compiler इससे प्रभावित नहीं होता है। जैसे:

```
float pi_float              = 3.1415;
double pi_double            = 3.14159265358979;
long double pi_long_double      = 3.141592653589793238;
```

इसी तरह हम जितने चाहें उतने Tabs, New Lines आदि का प्रयोग कर सकते हैं। C++ का Compiler इन्हें Ignore कर देता है।

Comments

Program के Flow को ध्यान में रखने के लिए और Program की Debugging को सरल बनाने के लिए हम हमारे Program में जगह–जगह Comments देते रहते हैं। C++ का Compiler इन Comments को Ignore कर देता है। ये केवल हमारे या Program के Developer की सुविधा के लिए होता है। C++ में Comment को दो तरीकों से लिखा जा सकता है। जब हमें केवल एक Line का Comment देना होता है, तब हम // का प्रयोग करते हैं और जब हमें कई Lines का Comment देना होता है तब हम Comment को /* */ के बीच में लिखते हैं। जैसेः

```
// these variables are declared and initialized at the same time
float pi_float = 3.1415;                                  // 5-digit precision
double pi_double = 3.14159265358979;                      // 15-digit precision
long double pi_long_double = 3.141592653589793238;        // 19-digit precision

/* The program is Related
to a Constant number of PI
for different Data Types of
Floating Point Variables */
```

String Constants

जो Text "" के बीच लिखा जाता है उसे String Constant कहते हैं। जैसे :

```
"Hello World"
```

C++ में हम जिस तरह से किसी Variable में Store मान को Screen पर Print करते हैं उसी तरह से किसी String को भी Print कर सकते हैं। जैसे :

```
Cout << "Enter first Number ";

float Total = 12.2;
cout << Total;
```

हम एक cout का प्रयोग करते हुए कई Variables के मान को Screen पर Print कर सकते हैं। जैसे :

```
float feet = 123.33;
cout << "The Size of Feet is << feet << "Feets" ;
```

इसका Output निम्नानुसार होगाः

```
The Size of Feet is 123.33 Feet
```

इसी काम को हम तीन अलग–अलग Statements लिख कर निम्नानुसार कर सकते हैं:

```
cout << "The Size of is"
     << feet
     << Feet"
```

Formatting Output

हम हमारे किसी Program के Output को विभिन्न प्रकार के Format द्वार अच्छी तरीके से Screen पर Print कर सकते हैं। C++ में इस काम से सम्बंधित बहुत सारे तरीके हैं। C++ में Output में New line को प्रदर्शित करने के लिए हमें एक Statement देना पडता है।

Escape Sequences

Output Formatting का सबसे सरल तरीका है Strings के बीच में Escape Sequence Characters का प्रयोग करना। जैसे:

```
cout << "\n Kuldeep Mishra" ;
cout << "\nFalna";
```

यहां नई लाईन प्राप्त करने के लिए हम '\n' Character का प्रयोग करते हैं। ये Statement Output में निम्नानुसार Print होगा:

```
Kuldeep Mishra
Falna
```

हम String में जिस स्थान पर '\n' का प्रयोग करते हैं, उसी स्थान पर Output में New Line आ जाती है और '\n' के बाद लिखा String New Line में Print होता है। अब यदि हमें निम्नानुसार Format Output में Print करना हो तो हम एक अन्य Character '\t' का प्रयोग कर सकते हैं :

```
Kuldeep    Rahul    Rohit
Mohit      Raja     Rani
```

Statements :

```
cout << "\t Kuldeep \t Rahul \t Rohit\n"'
cout << "\tMohit \t Raja \t Rani" ;
```

The endl Manipulator

C++ में हम New Line प्राप्त करने के लिए एक अन्य Object जिसे Manipulator कहते हैं, का प्रयोग भी कर सकते हैं। endl Manipulator वही काम करता है जो '\n' Character करता है। हम इसका प्रयोग निम्नानुसार कर सकते हैं:

```
cout << "\t Kuldeep \t Rahul \t Rohit" << endl ;
```

```
cout << "\tMohit \t Raja \t Rani" ;
```

इससे पहले कि हम OOPS को समझते हुए C++ में Classes व Objects Create करके Programming करें, हम सबसे पहले कुछ छोटे–छोटे Programs द्वारा C++ Programming के Basics को समझ लेते हैं।

Anatomy of a C++ Program

हमने C Language के बारे में जितना कुछ सीखा है, उसमें से ज्यादातर बातें C++ Language पर भी समान रूप से लागू होती हैं। C व C++ दोनों Languages में लिखे गए ज्यादातर Programs लगभग समान होते हैं। इनके Syntax भी समान हैं और विभिन्न प्रकार के Data Types, Loops, Pointers, Structures, Data Types व Conditional Statements भी समान ही होते हैं। C++ में हम जो भी Program बनाते हैं, उन सभी Programs की Source File को हमें **.CPP** Extension से Save करना जरूरी होता है। यदि हम C++ में एक Hello World Program बनाना चाहें, तो हम ये Program निम्नानुसार बना सकते हैं:

Program

```
#include <conio.h>
#include <iostream.h>

void main()
{
    cout << "Welcome to C with Class programming language!\n";
    getch();
}
```

Output

```
Welcome to C with Class programming language!
```

C Language में विभिन्न प्रकार के I/O Operations के लिए जिस प्रकार से हम stdio.h नाम की Header File को अपने Source Program में Include करते हैं, उसी तरह से C++ के Program में I/O की सुविधा प्राप्त करने के लिए हमें iostream.h नाम की Header File को Include करना पडता है। जब इस Program को Compile करके Run किया जाता है, तब Output में हमें एक Text Line Display होती है। चलिए, इस Program के हर Statement को थोडा विस्तार से समझने की कोशिश करते हैं।

1 //

"C++" में Comment देने के लिये हम // का प्रयोग करते हैं। इस Comment की विशेषता ये है कि // के बाद लिखे गए सारे Characters Comment बन जाते हैं और Compiler Program को Compile करते समय // के बाद लिखे Statement को Ignore कर देता है। // का असर केवल एक Line में ही होता है।

अगली लाइन में लिखे Statements पर // का कोई असर नहीं पडता है और अगली पंक्ति में लिखे Statements Compile हो जाते हैं। "C++" में जब हमें केवल एक ही पंक्ति में कोई Comment लिखना होता है तब हम // का प्रयोग करते हैं। जब हमें एक से अधिक Lines का Comment देना होता है, तब हम "C" के /* ..... */ का भी प्रयोग कर सकते हैं या फिर हर Line के आगे // का प्रयोग करते हैं।

2 #include <iostream.h>

जिस प्रकार से "C" में Input/Output की सुविधा प्राप्त करने के लिये हम stdio.h नाम की Header File को अपने Program में include Keyword द्वारा Include करते हैं, ठीक उसी प्रकार से "C++" में Input/Output की सुविधा प्राप्त करने के लिये हमें iostream.h नाम की Header File को हमारे Program में Include करना पडता है।

3 void main()

यह main() Function है। हमने "C" में किसी भी Program में main() Function से पहले void Keyword का प्रयोग नहीं किया है। लेकिन "C++" में हम ऐसा कर रहे हैं, क्योंकि अलग–अलग प्रकार के Compilers को इस प्रकार से बनाया गया है कि किसी Compiler को ये बताना पडता है कि main() Function कोई मान Return कर रहा है या नहीं। जबकि कई Compilers को ये नहीं बताना पडता कि main() Function कोई मान Return कर रहा है या नहीं।

जैसे कि Turbo C++ में व ANSI C++ में Compiler को ये बताना पडता है कि main() Function कोई मान Return कर रहा है या नहीं जबकि Borland C++ के Compiler को ये नहीं बताना पडता।

चूंकि यदि हम Turbo C++ के IDE में काम कर रहे हैं, तो main() Function के पहले **void** Keyword का प्रयोग करके Compiler को ये बताया गया है कि ये main() Function कोई मान Return नहीं कर रहा है। यदि हम void main() के स्थान पर केवल main() लिखते हैं, तो Program Compilation के समय हमें एक Warning **(Function should return a value)** प्राप्त होती है।

यदि हम **void** Keyword का प्रयोग ना करें तो हमें main() Function की Body को बन्द करने से पहले **return 0** लिखना पडता है। **return 0** Statement भी Compiler को यही बताता है कि ये main() Function किसी प्रकार का कोई मान Return नहीं कर रहा है। "C" की तरह ही "C++" में भी हर Function की Body लिखने से पहले Opening bracket व Function का अन्त करते समय Closing bracket का प्रयोग किया जाता है।

4 cout

cout (Character Out) iostream.h header File का एक Object है। यह Object वही काम करता है जो काम "C" में printf() Function करता है यानी Characters के समूह या String को Screen पर Print करता है।

5 << (Put to Operator)

किसी Massage को Screen पर भेजने के लिये "C++" में **<<** Operator काम में लिया जाता है। इसे **Put To Operator** कहा जाता है। यह Operator कुछ Data जैसे कि String या Message को किसी I/O Stream के Object में भेजने का काम करता है। इसे समझने के लिए निम्न Program देखिए:

Program

```
#include <iostream.h>
#include <conio.h>

void main()
```

```
{
    cout << "Gopal " << "Nandlal " << "Madhav " << "Krishna \n";
    getch();
}
```

Output

```
Gopal Nandlal Madhav Krishna
```

इसी तरह से यदि हम चाहें दो Character Pointers को भी इसी cout Object और Put To Operator के प्रयोग द्वारा Output में Print कर सकते हैं। Pointers के बारे में हम आगे विस्तार से पढेंगे।

Program

```
#include <iostream.h>
#include <conio.h>

void main()
{
    char *fName = "Rahul", *lname = "Sharma";
    cout << fName << " " << lname << endl;
    getch();
}
```

Output:

```
Rahul Sharma
```

इस Program में हमने cout Object के साथ एक नए शब्द endl का प्रयोग किया है। इसे C++ Programming में endl Manipulator कहते हैं। ये Operator वही काम करता है, जो काम C Language में "\n" Character करता है। यानी ये Manipulator एक नई Line प्रदान करता है। हम cout Object के साथ << Operator को निम्नानुसार भी Use कर सकते हैं:

```
cout    << fName
        << " "
        << lname
        << endl;
```

यदि हम इस प्रकार से भी इस Statement को प्रयोग करें, तो भी Compiler हमें किसी प्रकार की कोई Error प्रदान नहीं करता है। जिस तरह से हमने Character To Pointer व String Constants को एक ही **cout** Object व **Put To Operator** द्वारा Screen पर Display करवाया है, उसी तरह से हम विभिन्न प्रकार के Basic Data Type के मानों को भी Screen पर Print करवा सकते हैं। हमें ऐसा करने के लिए C Language की तरह किसी Control String का प्रयोग करने की जरूरत नहीं होती है।

Program

```
#include <iostream.h>
#include <conio.h>

void main()
{
      cout << "String   = " << "\"All Primary Data Types\"" << endl;
```

```
        cout << "Integer   = " << 123 << endl;
        cout << "Float     = " << 123.456 << endl;
        cout << "Character = " << "\'A\'" << endl;
        cout << "Double    = " << 12345678.0987654 << endl;
        cout << "Long      = " << 1234567987878786668 << endl;
        getch();
    }
```

Output

```
String    = "All Primary Data Types"
Integer   = 123
Float     = 123.456
Character = 'A'
Double    = 1.23457e+07
Long      = 1234567987878786668
```

इस Program के Output द्वारा हम समझ सकते हैं, कि केवल एक ही cout Object व Put To Operator की मदद से हम विभिन्न प्रकार के Basic Data Type के मानों को Screen पर बिना किसी Control String का प्रयोग किए ही Print कर सकते हैं। जबकि यही Program यदि हमें C Language में बनाना हो, तो हमें अलग–अलग Data Type के मानों को Screen पर Print करवाने के लिए ***printf***() Function में विभिन्न प्रकार के Control Strings को प्रयोग करना पडेगा। C Programming व C++ Programming के I/O का ये सबसे बडा अन्तर है। यानी C++ में I/O Operations को काफी सरल कर दिया गया है।

कई बार जब हम Program को Compile करते हैं, तो Compiler हमें कई प्रकार की Warnings प्रदान करता है। हालांकि यदि हम इन Warnings पर कोई ध्यान ना दें, तो भी Program Execute होता है। लेकिन जहां तक हो सके, हमें Compiler द्वारा दी जाने वाली Warnings को Clear कर देना चाहिए। यदि हम ऐसा नहीं करते हैं, तो कई बार हमें हमारा Required परिणाम प्राप्त नहीं होता है, जिसका कारण कोई ऐसी Warning होता है, जिसे हम Ignore कर देते हैं।

यदि हम विभिन्न प्रकार के Primary Data Type के Variables द्वारा Occupy किए जाने वाले Variables की अधिकतम व न्यूनतम मान का पता लगाना चाहें और ये जानना चाहें, कि विभिन्न प्रकार के Variables Memory में कितनी संख्या तक का मान Store कर सकते हैं, तो हम ये जानने के लिए ***climit.h*** Header File को Use कर सकते हैं।

इस Header File में विभिन्न प्रकार के Data Types द्वारा Store किए जा सकने वाले मान की Range को विभिन्न प्रकार के Constants द्वारा Define कर दिया गया है। कई बार इस File का नाम climits.h के स्थान पर **limits.h** भी होता है। इसे हम निम्न Program में निम्नानुसार Use कर सकते हैं:

Program

```
#include <iostream.h>
#include <conio.h>
#include <limits.h>

void main()
{
    cout << "Minimum short         :  " << SHRT_MIN << endl;
```

```
    cout << "Maximum short         :  " << SHRT_MAX << endl;
    cout << "Minimum unsigned short  :  " << 0 << endl;
    cout << "Maximum unsigned short  :  " << USHRT_MAX << endl;
    cout << "Minimum int           :  " << INT_MIN << endl;
    cout << "Maximum int           :  " << INT_MAX << endl;
    cout << "Minimum unsigned int    :  " << 0 << endl;
    cout << "Maximum unsigned int    :  " << UINT_MAX << endl;
    cout << "Minimum long          :  " << LONG_MIN << endl;
    cout << "Maximum long          :  " << LONG_MAX << endl;
    cout << "Minimum unsigned long   :  " << 0 << endl;
    cout << "Maximum unsigned long   :  " << ULONG_MAX << endl;
    getch();
}
```

Output

```
Minimum short             :   -32768
Maximum short             :   32767
Minimum unsigned short    :   0
Maximum unsigned short    :   65535
Minimum int               :   -2147483648
Maximum int               :   2147483647
Minimum unsigned int      :   0
Maximum unsigned int      :   4294967295
Minimum long              :   -2147483648
Maximum long              :   2147483647
Minimum unsigned long     :   0
Maximum unsigned long     :   4294967295
```

इसी तरह से कई और Definitions ***limits.h*** नाम की Header File में दिए गए हैं, जिन्हें इस Header File को Open करके देखा जा सकता है। इसी तरह से यदि हम विभिन्न प्रकार के Floats Data Types की Range का पता लगाना चाहें, तो हमे cfloats.h नाम की Header File को अपने Program में Use करना होता है। इस File को भी कई Compiler में केवल floats.h नाम से सम्बोधित किया जाता है।

यदि हम चाहें तो विभिन्न प्रकार के Floats की Range को देखने के लिए इस Header File को Open कर सकते हैं। यदि हम ये जानना चाहें कि विभिन्न प्रकार के Data Type के Variable Memory में कितने Bytes की Space Reserve करते हैं, तो इस बात का पता लगाने के लिए हम ***sizeof***() Operator का प्रयोग कर सकते हैं।

सामान्यतया Integer Data Type के अलावा सभी Data Types सभी प्रकार के Computers में समान Memory Occupy करते हैं, जबकि Integer Memory में Compiler के Register की Size के बराबर Space Reserve करता है।

यदि हम 16-Bit Compiler में 16-Bit Processor पर Program Develop करते या Run करते हैं, तो Integer 16-Bit System में 2-Bytes का होता है जबकि 32-Bit System में Integer की Size 4-Bytes होती है। हम जिस Compiler को Use कर रहे हैं, उस Compiler द्वारा हर प्रकार के Basic Data Type द्वारा Occupy की जा रही Memory का पता हम निम्न Program द्वारा लगा सकते हैं:

Program

```
#include <iostream.h>
#include <conio.h>

void main()
{
    cout << "char       :  " << sizeof(char) << " Bytes\n";
    cout << "short      :  " << sizeof(short) << " Bytes\n";
    cout << "int        :  " << sizeof(int) << " Bytes\n";
    cout << "long       :  " << sizeof(long) << " Bytes\n\n";

    cout << "signed char  :  " << sizeof(signed char) << " Bytes\n";
    cout << "signed short :  " << sizeof(signed short) << " Bytes\n";
    cout << "signed int   :  " << sizeof(signed int) << " Bytes\n";
    cout << "signed long  :  " << sizeof(signed long) << " Bytes\n\n";

    cout << "unsigned char :  " << sizeof(unsigned char) << " Bytes \n";
    cout << "unsigned short:  " << sizeof(unsigned short) << " Bytes\n";
    cout << "unsigned int  :  " << sizeof(unsigned int) << " Bytes\n";
    cout << "unsigned long :  " << sizeof(unsigned long) << " Bytes\n\n";

    cout << "float      :  " << sizeof(float) << " Bytes \n";
    cout << "double     :  " << sizeof(double) << " Bytes\n";
    cout << "long double  :  " << sizeof(long double) << " Bytes\n";

    getch();
}
```

Output

```
// 16 - Bit Compiler's Output
char           :   1 Bytes
short          :   2 Bytes
int            :   2 Bytes
long           :   4 Bytes

signed char    :   1 Bytes
signed short   :   2 Bytes
signed int     :   2 Bytes
signed long    :   4 Bytes

unsigned char  :   1 Bytes
unsigned short:   2 Bytes
unsigned int   :   2 Bytes
unsigned long  :   4 Bytes

float          :   4 Bytes
double         :   8 Bytes
long double    :   10 Bytes

// 32 - Bit Compiler's Output
char           :   1 Bytes
short          :   2 Bytes
int            :   4 Bytes
long           :   4 Bytes
```

```
signed char    :   1 Bytes
signed short   :   2 Bytes
signed int     :   4 Bytes
signed long    :   4 Bytes

unsigned char  :   1 Bytes
unsigned short:    2 Bytes
unsigned int   :   4 Bytes
unsigned long  :   4 Bytes

float          :   4 Bytes
double         :   8 Bytes
long double    :   10 Bytes
```

Real Numbers को C++ ***float***, ***double*** व ***long double*** तीन तरह से Support करता है। ज्यादातर Computers पर double Data Type float Data Type की तुलना में दुगना Bytes लेता है। यानी ज्यादातर Computers पर float 4 – Bytes की Space लेता है जबकि double Data Type 8 – Bytes की Space का प्रयोग करता है। जब कि long double 8, 10, 12 या 16 Bytes का प्रयोग Data को Store करने के लिए करता है।

जिस प्रकार के Data को Represent करने यानी Memory में Store करने व Memory से Access करने के लिए हम float प्रकार के Variable का प्रयोग करते हैं, उस प्रकार के Data को Real Numbers कहते हैं। ज्यादातर Computers पर Real Numbers जैसे कि 123.45 सबसे पहले निम्नानुसार Binary Form में Convert होते हैं:

$$123.45 = 1111011.01110011_2 * 2^7$$

जब Real Number Binary Form में Convert हो जाता है, तब Point के बाद की संख्या दसमलव के बाद की Binary को Represent करती है और Point से पहले की Binary संख्या के Left Hand Side के अंकों को Represent करती है। अब Binary Digits के Point को Float किया जाता है, ताकि सभी Bits Point के Right Side में आ जाएं। ऐसा करने पर Point को सात Bit Left में Sift करना पडता है, जिससे घातांक 2^7 प्राप्त होता है। अब हमें जो Binary Number प्राप्त होता है, वह निम्नानुसार होता है:

$$123.45 = 0.111101101110011_2 * 2^7$$

अब इस Number के सामान्य Number के Binary Bits व उसके घातांक के मान 7 दोनों को Memory में Internally अलग–अलग Store किया जाता है। 32 – Bit Float Real Number की Binary Digits को 23 – Bits Segment में Store होती है और घातांक का मान 8 – Bit में Store किया जाता है। एक Bit द्वारा संख्या का Sign तय किया जाता है। इसी तरह से जब हम double प्रकार के Variable में मान Store करते हैं, तब 64 – Bit double में संख्या 52 – Bit Segment में Store होती है और घातांक को 11 – Bit में Store किया जाता है।

निम्न Program द्वारा हम ये पता लगा सकते हैं, कि हमारा Computer Float व Double प्रकार के Variables में कितने Bits में किसी Real Number के मान व उसके घातांक को Store कर रहा है। इस बात का पता लगाने के लिए हमें float.h नाम की Header File को अपनी Source File में Include करना जरूरी होता है। क्योंकि Float के सम्बंध में जितने भी Definitions हैं, उन्हें float.h नाम की Header File में Define किया गया है।

Program

```
#include <iostream.h>
#include <conio.h>
#include <float.h>

void main()
{
    // Prints the storage size of fundamental types:
    int fbits = 8 * sizeof(float);                    // Each byte contains 8-bits
    cout    << "Float Uses " << fbits << "bits: \n\t"
    << (FLT_MANT_DIG – 1) << " bits for its mantissa. \n\t"
    << (fbits - FLT_MANT_DIG) << "bits for its exponent.\n\t"
    << 1 << " bit for its sign\n"
    << "to obtain: " << FLT_DIG << " sig. digits\n"
    << "with minimum value: " << FLT_MIN << endl
    << "and maximum value: " << FLT_MAX << endl;
    getch();
}
```

Output

```
Float Uses 32bits:
    23 bits for its mantissa.
    8 bits for its exponent.
    1 bit for its sign
to obtain: 6 sig. digits
with minimum value: 1.17549e-38
and maximum value: 3.40282e+38
```

इसी तरह से यदि हम double प्रकार के Data Type की Range जानना चाहें, तो इसी Program को निम्नानुसार Modify कर सकते हैं:

Program

```
#include <iostream.h>
#include <conio.h>
#include <float.h>
void main()
{
    // Prints the storage size of fundamental types:
    int fbits = 8 * sizeof(double);                              // Each byte contains 8-bits
    cout    << "Float Uses " << fbits << "bits: \n\t"
            << (DBL_MANT_DIG – 1) << " bits for its mantissa. \n\t"
            << (fbits - FLT_MANT_DIG) << "bits for its exponent.\n\t"
            << 1 << " bit for its sign\n"
            << "to obtain: " << FLT_DIG << " sig. digits\n"
            << "with minimum value: " << FLT_MIN << endl
            << "and maximum value: " << FLT_MAX << endl;
}
```

Output

```
Float Uses 64bits:
```

```
    52 bits for its mantissa.
    40bits for its exponent.
    1 bit for its sign
to obtain: 6 sig. digits
with minimum value: 1.17549e-38
and maximum value: 3.40282e+38
```

Type Conversion (Type Casting)

Computer में कई बार कई स्थानों पर जरूरत के आधार पर विभिन्न प्रकार के Type Conversions होते रहते हैं। कुछ Conversions Compiler स्वंय ही कर लेता है और कई बार हमें हमारी जरूरत के आधार पर कुछ Conversions करने होते हैं। जैसे मानलो कि हमें किसी Float प्रकार की संख्या को Integer में Convert करना पड सकता है या किसी Integer प्रकार की संख्या को Character में Convert करना पड सकता है।

Code Segment 01

```
int n = 20;
float PI = 3.1415;

n = PI + n;             // The Value is automatically converted to 20.0
cout << PI – 2;         // The value 2 is automatically converted to 2.0
```

इस Code Segment में जब Integer प्रकार के Variable में PI के Float प्रकार के मान को जोडना होता है, तो Compiler स्वयं ही n के मान 20 को Float प्रकार के मान 20.0 में Convert कर लेने के बाद PI के मान से जोडता है और Float प्रकार का मान ही Return करता है। लेकिन Assignment Operator के Left में n एक Integer प्रकार का Variable है, इसलिए n में Return होने वाले Float प्रकार के मान का केवल Integer Part ही Store होता है, दसमलव के बाद वाले मान Compiler Ignore कर देता है।

इसी तरह से जब हम PI के मान में से Integer के मान 2 को घटाना चाहते हैं, तब Compiler स्वयं ही Integer मान 2 को Float प्रकार के मान 2.0 में Convert करने के बाद PI के मान में से घटाता है और Return होने वाले मान को Output में Screen पर Print करता है।

हम देख सकते हैं कि Integer प्रकार का मान Float प्रकार के मान में Automatically Convert होता है, लेकिन Float प्रकार का मान Integer प्रकार के मान में Automatically Convert नहीं होता है, बल्कि हमें स्वयं को Float प्रकार के मान को Integer प्रकार के मान में Convert करना होता है।

ऐसा करने के लिए हम जिस Process को Use करते हैं, उसे Type Casting करना या Type Conversion करना कहते हैं, क्योंकि इस प्रक्रिया में एक Data Type को दूसरे Data Type में Convert किया जाता है। मानलो कि ConversionType ***CV*** वह Type है, जिसमें Value ***V*** को Convert करके ConvertedType ***CT*** में Store करना है, तो हमें इस Conversion को Perform करने के लिए निम्नानुसार Statement लिखना होगाः

```
CT = (CV)V
```

मानलो कि V एक Float प्रकार का मान है, जिसे Integer प्रकार के मान में Convert करके एक Integer प्रकार के Variable में Store करना है। इस Type Casting को Perform करने के लिए हमें निम्नानुसार Statement लिखना होगाः

Code Segment 02

```
int CT;
float V = 123.699;

CT = (int)V;

cout << CT;        // Value of CT would be 123 After Type Casting
```

उपरोक्त Code Segment Float प्रकार के मान को Truncate करके केवल Integer Part को ही Return करता है, जो कि CT में Store हो जाता है। एक बात ध्यान रखें कि CT में Store होने वाला मान V के दसमलव वाले भाग को Round-Off नहीं करता है, बल्कि इस तरह Type Casting करने के कारण Compiler दसमलव वाले भाग को छोड देता है। इसी Manual Type Conversion के Syntax को एक दूसरे तरीके से भी लिख सकते हैं। ये तरीका C++ में तो Valid है लेकिन C में Valid नहीं है। यानी हम इसी Code Segment को निम्नानुसार भी लिख सकते हैंः

Code Segment 02

```
int CT;
float V = 123.699;

CT = int(V);

cout << CT;        // Value of CT would be 123 After Type Casting
```

हमने इसी उदाहरण में दो बातें देखी हैं। पहली ये कि Integer प्रकार का मान Float प्रकार के मान में Convert हो रहा है। सामान्यतया जब किसी छोटी Size के Data Type से बडी Size के Data Type में मान Convert होता है, तो वह Conversion Compiler द्वारा Automatically होता है, जैसा कि पहले Code Segment में हुआ है। इसलिए इस प्रकार की Type Casting को ***Automatic Type Casting*** कहते हैं।

जबकि दूसरे Code Segment द्वारा हमने Float प्रकार के बडे Size के मान को छोटे Size के Integer प्रकार के मान में Convert करने के लिए C++ के **Casting Concept** को Use किया है। इसलिए इस प्रकार की Casting को ***Manual Type Casting*** कहते हैं। Compiler Automatic Type Casting को निम्न क्रम में करता हैः

```
// Automatic Type Conversion Flow by the Compiler
char => short => int => long => float => double
```

इस Flow से हम समझ सकते हैं कि Compiler Lower Size से Higher Size वाले Data Type में Convert होता है। यानी char प्रकार के मान को short में, short को int में int को long में long को float में व float को double में Convert करता है। यानी किसी Expression को Perform करने से पहले Compiler Expression के सभी Elements को सबसे Highest Data Type के मान में Convert कर देता है। उदाहरण के लिए निम्न Expression को देखिएः

```
int x = 23320;
float y = 2.4
short s = 1221

long l;

l = x + y – s;      // Expression
```

इस Expression में Assignment Operator के Right Side में सबसे बडा Data Type float है, इसलिए Integer प्रकार के Variable x का मान व short प्रकार के Variable s का मान दोनों मान float प्रकार के मान में Convert होने के बाद Calculate होंगे और Return होने वाला Result Float प्रकार का Return होगा, लेकिन Assignment Operator के Left Side में long प्रकार का Variable है, इसलिए Long प्रकार के Variable l में Return होने वाला Float प्रकार का मान Long प्रकार में Convert होने के बाद Store होगा।

Input From The Keyboard

चलिए, अब हम देखते हैं कि User जो मान Keyboard से Input करता है, उसे C++ में किस प्रकार से Memory Variables में Store किया जा सकता है। निम्न Statement देखिए

```
int number ;
cin >> number ;
```

cin Object Keyboard को Represent करता है और (>>) Operator “get from” Keyboard से Input किए जाने वाले मान को इसके Right Side में स्थित Variable **number** में Store कर देता है। सामान्यतया Input लेने से पहले हम User को Prompt करते हैं कि Program को किस प्रकार का मान चाहिए, इसके लिए हम cout का प्रयोग करते हैं। जैसेः

```
int age;
cout << “Enter your age: “ ‘
cin >> age;
```

ये Instructions Output Screen निम्नानुसार Run होते हैंः

```
Enter your age : 60
```

जहां User 60 Enter करता है।

हम **get from** Operator का प्रयोग एक ही Statements में कई बार करके कई Input प्राप्त कर सकते हैं, ठीक उसी तरह से जिस तरह से हमने पिछले Program में cout Object के “put to” Operator को Use करके एक ही Statement से कई मानों को Output में Print किया है।

```
int age;
float height;
cout << “Enter your age and height:”;
cin >> age >> height;
```

यहां User को हर Variable को मान प्रदान करने के बाद Enter Key, Space Key या Tab Key Press करना होता है।

Numerical Overflow

ज्यादातर Computers में Long Integer प्रकार का Variable हमें अधिकतम 4294967296 तक के मान को Store करने की सुविधा प्रदान करता है, जो कि हालांकि काफी बडे मान को Store करने में सक्षम है लेकिन फिर भी ये एक सीमित संख्या है। Mathematical Calculations में हमें कई बार अनन्त तक को Store करना होता है।

इस स्थिति में जब हमारा मान Long Integer की Limit से भी Over हो जाता है, तो इस प्रक्रिया को ***Numeric Overflow*** कहते हैं। मानलो कि यदि हम किसी Program में 1989000 को 500 से गुणा करवाते हैं तो हमें निम्नानुसार Output प्राप्त होता है:

Program

```
#include <iostream.h>
#include <conio.h>

void main()
{
    long n = 198900;
    cout << "N = " << n << endl;

    n = n * 500;
    cout << "N = " << n << endl;

    n = n * 500;
    cout << "N = " << n << endl;

    getch();
}
```

Output

```
N = 198900
N = 99450000
N = -1814607552
```

इस Program के Output में हम देख सकते हैं कि तीसरी बार जब संख्या को गुणा किया जाता है, तो मान Long Data Type के Variable की Range से Over हो जाता है, इसलिए हमें प्राप्त होने वाला मान Minus में प्राप्त होता है। जबकि इसी Program में यदि हम Float का प्रयोग करते हैं, तो कई Advance Compilers में Floating Point के Overflow होने पर एक गलत मान के स्थान पर हमें Output में "Infinity" Display होता है।

Round-Off Errors

कई बार हम Floating Point Number पर Calculation Perform करते हैं और उन्हें Condition Testing में Use कर लेते हैं। वास्तव में हमें ऐसा नहीं करना चाहिए। क्योंकि Floating Point Values हमेंशा समान मान Return नहीं करते हैं। इसे समझने के लिए निम्न Program देखिए:

Program

```
#include <iostream.h>
#include <conio.h>

void main()
{
    float nf = 1.0/3;
    float nd = 0.333333;
    bool flag;

    flag = (nf==nd);
    cout << "Flag = " << flag << endl;
    cout << "NF = " << nf << endl;
    cout << "ND = " << nd << endl;

    nf = 0.333333;

    flag = (nf==nd);
    cout << "\nFlag = " << flag << endl;
    cout << "NF = " << nf << endl;
    cout << "NS = " << nd << endl;
}
```

Output

```
Flag = 0
NF = 0.333333
ND = 0.333333

Flag = 1
NF = 0.333333
ND = 0.333333
```

इस Program के Output में हम देख सकते हैं कि 1.0/3 करने पर भी 0.333333 ही आता है। लेकिन जब हम 1.0/3 को 0.333333 से Compare करते हैं, तब प्राप्त होने वाले Result व NF में 1.0/3 के स्थान पर 0.333333 कर देने पर प्राप्त होने वाले Result में क्या अन्तर है। जहां पहला Statement हमें False Return करता है, वहीं दूसरा Statement हमें True Return कर रहा है। इसीलिए कभी भी दो Floating Point Values को तब तक आपस में Compare नहीं करवाना चाहिए जब तक कि हम Result के प्रति पूरी तरह से आश्वस्त ना हों।

Statement Block

{ } (Curly Braces) के बीच लिखे जाने वाले सारे Statements को Statements का एक **Block** कहा जाता है। ये बात हमेंशा ध्यान रखें कि एक Block के Statements दूसरे Block के Statements से बिल्कुल ही Separated होते हैं। हम जिस Block में जिन Statements को

Use करते हैं या जिन Variables को Define करते हैं, वे Variables केवल उसी Block के लिए ही उपयोगी होते हैं।

उन्हीं Variables को किसी दूसरे Block में Directly Access नहीं किया जा सकता है। Block के साथ ही हर Object या Variable का एक Scope होता है। Scope वह स्थिति होती है, जहां से कोई Variable Create होने के बाद Use होने की अवस्था में होता है। इसे समझने के लिए हम एक Program देखते हैं।

Program

```
#include <iostream.h>
#include <conio.h>

void main()
{       // Start Block of main Function
    m = 10;          // Error: m is not in this block until now.

    int x = 30;
    {  // Inner Block Started
        int y = 30;
        int x = 50;

        cout << "Value of x of current block " << x << endl;
    } // Inner Block Ended

    { // Inner Block Started
        // OK: x have been created before. So this is in the scope in this block
        cout << " X in Inner Block of outer Block" << x << endl;
        cout << " Y of Other Block " << y << endl;
    } // Inner Block Ended

        cout << "Value of Y " << y << endl;
        cout << "Value of Y of Inner Block " << y << endl; // Error:

    int m;
}       // End of the main Function Block
```

इस Program को जब हम Compile करने की कोशिश करते हैं, तो हमें निम्न Errors प्राप्त होती हैं:

Errors:

```
Info :Compiling C:\BC5\BIN\blocknscope.cpp

Error:  blocknscope.cpp(9,7):Undefined symbol 'm'
Warn :  blocknscope.cpp(17,5):'y' is assigned a value that is never used
Error:  blocknscope.cpp(22,36):Undefined symbol 'y'
Error:  blocknscope.cpp(28,10):Multiple declaration for 'm'
Warn :  blocknscope.cpp(29,2):'m' is declared but never used
```

यहां सबसे पहली Error ये आ रही है कि हमने m नाम के Variable को Define नहीं किया है, जबकि Program के अन्तिम Statement में हमने m को Define किया है। ये Error इसलिए आ रही है, क्योंकि Compiler Program को उपर से नीचे की तरफ Execute करता है। इस स्थिति में उसे सबसे पहले मिलने वाला Statement m = 10; होता है। चूंकि अभी m नाम का Variable Memory में Create नहीं हुआ है, इसलिए अभी तक ये Variable Main Function के Block के Scope में नहीं है।

अतः हम इस Variable को अभी Access नहीं कर सकते, जब तक कि ये Define ना हो जाए और ये Variable Program के अन्तिम Statement से Define होता है। यानी Program के अन्तिम Statement से इस Variable का Scope चालू होता है।

यदि इस Statement को Program की शुरूआत में लिख दें, तो Program के शुरू होते ही ये Statement Execute होगा और m नाम का Variable Main Function के Block में Memory में Create हो जाएगा और इस Variable का Scope शुरू हो जाएगा। इस Error को सही करके Program को फिर से Compile किया जाए, तो अब हमें निम्न Errors प्राप्त होती हैं:

```
Warn :  blocknscope.cpp(18,5):'y' is assigned a value that is never used
Error:  blocknscope.cpp(23,36):Undefined symbol 'y'
Warn :  blocknscope.cpp(29,2):'m' is assigned a value that is never used
```

यहां y नाम के Variable को Compiler Undefined मानता है, जबकि हमने y नाम के Variable को Main Block के अन्दर एक और Block Create करके उसमें Declare किया है। Compiler ऐसा इसलिए करता है क्योंकि जब हम किसी Block में कोई Variable Create करते हैं, तो वह Variable केवल उस Block तक के लिए ही उपयोगी होता है। यानी Variable केवल उस Block तक ही सीमित रहता है।

जैसे ही Program Control Block के Statements को Execute करके Block से बाहर निकलता है, Block में Created Variable y का Scope भी समाप्त हो जाता है यानी Variable y केवल Block में ही Accessible है। अतः इस Error को हटाने के लिए हमें Main Functions के Block में y नाम के Variable को Create करना होगा। ये दोनों परिवर्तन करने पर हमारा Program निम्नानुसार हो जाता है:

Program

```
#include <iostream.h>
#include <conio.h>

void main()
{       // Start Block of main Function
    int m;
    int y = 20;
    m = 10;          // Error: m is not in this block until now.

    int x = 30;
    { // Inner Block Started
        int y = 30;
        int x = 50;
```

```
        cout << "Value of x of current block " << x << endl;
    } // Inner Block Ended

    { // Inner Block Started
        // OK: x have been created before. So this is in the scope in this block
        cout << " X in Inner Block of outer Block" << x << endl;
        cout << " Y of Other Block " << y << endl;
    } // Inner Block Ended

        cout << "Value of Y " << y << endl;
        cout << "Value of Y of Inner Block " << y << endl; // Error:
        getch();
}       // End of the main Function Block
```

Output

```
Value of x of current block 50
 X in Inner Block of outer Block 30
 Y of Other Block 20
Value of Y 20
Value of Y of Inner Block 20
```

इस Program के Output में हम देख सकते हैं कि हमारे पास x व y नाम के दो Variables हैं। पहला Variable Main Functions के Block में है जबकि दूसरा Main Function के अन्दर Define किए गए दूसरे Block में हैं। जब Program को Run किया जाता है, तब Program के Inner Block में लिखे गए cout Statement का Execution होने पर हमें Output में x का मान 30 नहीं बल्कि 50 प्राप्त होता है। जबकि हमने Main Function के Block में x का मान 30 दिया है।

साथ ही जब हम Inner Block के y का मान Print करना चाहते हैं, तब भी हमें Main Function के ही Block के y का मान Output में प्राप्त होता है। ऐसा इसलिए होता है क्योंकि Compiler जब Inner Most Block में किसी Variable के मान को Print करना चाहता है, तब वह उस Block में ही सबसे पहले वांछित Variable को खोजता है।

चूंकि x नाम का Variable Compiler को Inner Block में ही मिल जाता है, इसलिए Compiler Main Function के Block के x के मान को Output में Print नहीं करता, बल्कि Inner Block के x के मान को ही Output में Print कर देता है। जबकि जब हम y का मान Main Functions के Block में Print करना चाहते हैं, तब Compiler Inner Block के Variable y के मान को Print नहीं करता बल्कि दोनों ही बार Main Function के Block के y के मान को ही Print कर देता है।

इसका कारण ये है कि Compiler कभी भी जिस Block में होता है, उसमें और उससे बाहर की तरफ ही किसी भी Variable को खोजता है। किसी Block के अन्दर यदि दूसरा Block हो, तो उस Inner Block में Compiler किसी Variable को नहीं खोजता है। इसका कारण ये होता है, कि जैसे ही Compiler Inner Most Block को Execute करके Block से बाहर आता है, Inner Block और उसके सभी Variables Memory से Destroy हो जाते हैं।

इसी कारण से Compiler Inner Most Block में Variable को नहीं खोजता है, साथ ही जो भी Statement हम किसी Program में Use करते हैं, उसमें से Compiler केवल उन्हीं Statements

के बारे में जानता है, जहां तक के Statements को Compiler ने Execute कर लिया है। Compiler ने जिन Statements को Execute नहीं किया होता है, Compiler उन Statements के बारे में तब तक कुछ नहीं जानता है, जब तक कि Compiler उन्हें Execute नहीं कर देता है।

इस तरह से Compiler जब किसी Current Statement पर होता है, तब दो स्थितियों में से किसी एक स्थिति में होता है। यदि Compiler किसी Inner Block को Execute कर चुका है, तो उसमें Define किए गए किसी Variable के बारे में नहीं जानता है, क्योंकि Block से बाहर आते ही Block के सभी Variables Destroy हो चुके होते हैं, जबकि यदि Compiler जिस Statement पर है उस Statement से बाद में किसी Statement या Block में कोई Variable Define किया गया है, तो उसके बारे में भी कुछ नहीं जानता है, क्योंकि अभी तक Compiler ने उन Statements को Execute ही नहीं किया है।

I/O Streams

हमने **cout** व **cin** Objects को Use करने की जो तकनीक इस अध्याय में प्रदर्शित की है, इन्हें I/O Streams कहा जाता है। Stream Data के Flow का एक साधारण तरीका होता है। हम जब भी कोई Program बनाते हैं, हमें हमारे Program में Input व Output की C++ द्वारा प्रदान की जाने वाली सुविधा को प्राप्त करने के लिए, IOSTREAM.H नाम की एक Header File को Include करना पडता है।

C Language में हम Input Output के लिए printf() व scanf() Functions का प्रयोग करते हैं, जिसमें हमें विभिन्न प्रकार के Data Type के Variables को Access करने के लिए विभिन्न प्रकार के Control Strings का प्रयोग करना पडता है।

यदि हम गलती से कहीं पर गलत Control String का प्रयोग कर देते हैं, तो हमारा Program हमें कभी सही परिणाम प्रदान नहीं करता। C++ में इस प्रकार से होने वाली Mistakes को Avoid करने के लिए इसमें नए तरीके के Input व Output Approach को Add किया गया है।

इसके प्रयोग से हम किसी भी प्रकार के Data Type के साथ बिना किसी प्रकार के Control Strings का प्रयोग किए प्रक्रिया कर सकते हैं। ये एक बहुत ही उपयोगी तकनीक है जिसमें Data Type Control String के प्रयोग से होने वाली Mismatching से होने वाली Mistakes की सम्भावना नहीं रहती है।

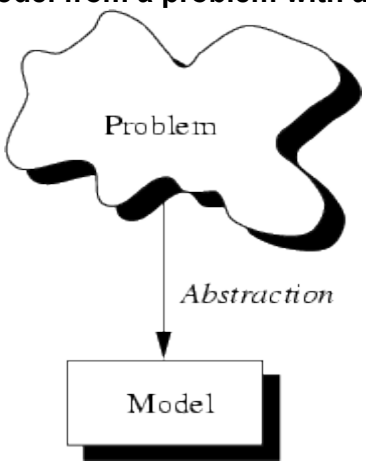
Abstract Data Types

एक Programmer जब किसी Program को लिखना चाहता है तो सबसे पहली चीज जो उसे Handle करनी होती है वह होती है समस्या। सामान्यतया हम एक Real Life Problem को Face कर रहे होते हैं और हम चाहते हैं कि उस Real Life Problem को Solve करने के लिए एक Computer Program बना लिया जाए। लेकिन Real Life Problems की कई बातें होती हैं। कुछ बातें समस्या के समाधान के लिए जरूरी होती हैं तो कुछ बातें उस समस्या के समाधान के लिए फालतू होती हैं।

हमें जो भी Program बनाना होता है, उस "Real Life" से जुडी हुई समस्या को Computer में Data के रूप में Feed करना होता है, ताकि Computer उस Data पर Processing कर सके। समस्या को Data के रूप में विभाजित करने के लिए हमें समस्या को समझना होता है, ताकि समस्या की उन आवश्यक बातों को अनावश्यक बातों से अलग किया जा सके, जो उस समस्या के समाधान के लिए उपयोगी हो सकती हैं।

समस्या को अच्छी तरह से समझने के बाद सारांश के रूप में जो आवश्यक Details हमें प्राप्त होती हैं, उन्हें Computer में Data के रूप में इस्तेमाल किया जा सकता है। किसी "Real Life" Problem को समझ कर उसकी उन आवश्यक बातों को जो कि समस्या के समाधान के लिए उपयोगी हो सकती हैं, ले लेना और शेष अनावश्यक बातों को छोड देना, इस प्रक्रिया को **Abstraction** कहा जाता है। इसे निम्न चित्र में दर्शाया गया है:

Create a model from a problem with abstraction.



किसी बिखरी हुई समस्या को सुलझाने के लिए उसे चित्र में बताए **Model** Box के अनुसार व्यवस्थित कर लेना समस्या का **Abstraction** कहलाता है। यहां ऊपर बताए गए चित्र में **Model** Box समस्या का Abstract View दर्शा रहा है जहां **Model** Box में केवल समस्या से सम्बंधित Details ही होती हैं जो समस्या के समाधान को प्रभावित करती हैं। शेष अनावश्यक Details को हटा दिया गया होता है। समस्या का Abstraction करने के बाद हमारे पास समस्या से सम्बंधित केवल उपयोगी जानकारी ही होती है और समस्या से सम्बंधित इन उपयोगी जानकारियों से हमें समस्या की विशेषताओं को परिभाषित करना होता है। इन विशेषताओं में हमें निम्न दो बातें तय करनी होती हैं:

1. वे Data कौन से हैं, जो कि समस्या में प्रभावित हो रहे हैं। यानी किस Data पर प्रक्रिया हो रही है। और
2. वे प्रक्रियाएं कौनसी हैं जो Data पर Operations करती हैं और Data की स्थिति को परिवर्तित करती हैं।

इस प्रक्रिया को समझने के लिए हम एक उदाहरण लेते हैं। माना किसी Company का Head आता है और आपसे एक ऐसा Program बनाने के लिए कहता है, जिससे वह अपने Employees के बारे में सारी जानकारी रख सके और जब जिस Employee के बारे में चाहे उसकी Detail प्राप्त कर सके।

इस समस्या में सबसे पहले हमारा काम इस समस्या के मूल उद्देश्य को जानना है। इस समस्या में Administrator आपसे अपने Employees का Bio-Data जानना चाहता है। यानी इस समस्या में Employee प्रभावित हो रहा है इसलिए एक Employee के बारे में जो भी जानकारियां जरूरी हो सकती हैं, वे सभी Details Computer के लिए Data होंगी। चूंकि Employee एक "Real Person" है इसलिए हर Employee की कई विशेषताएं (Characteristics) हो सकती हैं जैसे

- name
- size
- date of birth,

- shape
- social number
- room number
- hair color
- hobbies

निश्चित रूप से Employee की इन सभी Qualities की Administrator को कोई जरूरत नहीं होगी। इनमें से केवल कुछ ही बातें Administrator की समस्या से सम्बंधित होंगी, जिन्हें वह जानना चाहेगा। परिणामस्वरूप समस्या के समाधान के लिए हम Employee का एक Model बनाते हैं।

इस Model में Employee की केवल उन विशेषताओं (Properties) को ही लिया जाता है, जिसकी Administrator को जरूरत हो सकती है। जैसे Employee का नाम, उसकी जन्म तारीख और उसकी Qualification, इन Properties को Model (Employee) का **Data** कहते हैं।

यानी हम एक Employee की विभिन्न Properties में से केवल उन Properties को ही समस्या में उपयोगी मान रहे हैं जिनकी जानकारी Properties के लिए उपयोगी हो सकती है। यानी एक Employee की केवल उन बातों को ले रहे हैं जिनकी Administrator को जरूरत है शेष को छोड रहे हैं। इसी प्रक्रिया को **Abstraction** कहते हैं जिसमें समस्या से सम्बंधित जरूरी चीजों को अन्य बिना जरूरत की चीजों से अलग किया जाता है।

हालांकि इस समस्या के लिए Employee की जिन Properties को छोडा जा रहा है, हो सकता है कि किसी अन्य समस्या के समाधान में उन्हीं Properties की जरूरत हो। तो उस किसी दूसरी समस्या के **Abstraction** में प्राप्त होने वाली जरूरी बातें इस समस्या की जरूरी बातों से अलग होंगी।

इस उदाहरण से हम समझ सकते हैं कि किस प्रकार से एक Abstract Model का प्रयोग करके एक "Real World Object" यानी एक व्यक्ति को Describe कर सकते हैं और फिर उस Real World Object को Computer में Logically Represent कर सकते हैं।

हालांकि अभी भी ये विवरण पूर्ण नहीं है। यहां कुछ Operations भी परिभाषित किए जाने चाहिएं, जिससे Administrator उस Abstract Employee को Handle कर सके। उदाहरण के लिए यहां एक ऐसा Operation होना चाहिए, जिससे जब कोई नया व्यक्ति Company में आए तो Program में भी नए Employee को Create किया जा सके। यानी हमें ऐसे Operations को भी तय करना होगा जिनको Abstract Employee पर Perform किया जा सके।

हमें ये भी तय करना होगा कि Company के Employees के Data को केवल वे ही Operations Access कर सकें जिन्हें ऐसा करने के लिए Define किया गया है। ऐसा करने पर हमेंशा ये निश्चित रहेगा कि Data Elements उनकी सही स्थिति में हैं। उदाहरण के लिए हम ये Check कर सकते हैं कि क्या Input की गई Date Valid है या नहीं।

सारांश में कहें तो Abstraction एक ऐसा तरीका है जिससे जटिल "Real Life" Problem को उनके **Data** और (Data के मान को Change करने वाले) **Operations** द्वारा एक Well Definedz Unit या Entity के रूप में व्यवस्थित किया जा सकता है, जिसमें समस्या के Data व Data पर काम करने वाले Operations सम्मिलित रूप से एक साथ Combined रहते हैं।

जब किसी समस्या से प्रभावित होने वाले विभिन्न Data व उन Data पर काम करने वाले विभिन्न Operations को एक इकाई के रूप में परिभाषित किया जाता है, तो इस प्रक्रिया को **Encapsulation** कहते हैं और इस Unit को Entity या **Object** कहा जाता है।

C++ में हम जो Class Develop करते हैं वह एक Abstract Data Type या नए प्रकार का Data Type होता है। जिस तरह से यदि हमें Integer प्रकार के मान को Computer में Manage करना हो तो हम **int** Class के Data Type का एक Instance या Object Create करते हैं और यदि हमें किसी नाम को Computer में Store करके Manage करना हो तो हम String Class का एक Object Create करते हैं, उसी तरह से C++ में जो Class बनाई जाती है, वह Class भी एक नए प्रकार का Data Type होता है और हम इस Class के भी Objects Create करके उसे Computer में Store व Manage कर सकते हैं।

यहां आपके दिमाग में एक सवाल आ सकता है, कि जब हमारे पास Basic प्रकार के Data Type हैं तो हमें नए Data Type बनाने की क्या जरूरत है?

इसका जवाब ये है कि Basic प्रकार के Data Type के Objects की एक परेशानी ये है कि हम Computer में कोई भी मान स्वतंत्र रूप से Store नहीं करते हैं। हर मान का किसी दूसरे मान से सम्बंध होता है। यदि हमें केवल एक संख्या को Computer में Store करना हो, तो हम int या float या double प्रकार का Object Create करके मान को Computer में Store कर सकते हैं।

यदि हमें एक ही Data Type के कई मान Computer में Store करके Manage करने हों तो हम Array का प्रयोग कर सकते हैं, लेकिन जब हमें कई प्रकार के मानों को एक समूह के रूप में Computer में Store व Manage करना हो, तब हमें एक ऐसे तरीके की जरूरत पडती है, जिससे हम विभिन्न प्रकार के Data Types का प्रयोग कर सकें और उन्हें एक साथ एक ही Object के सन्दर्भ में Use कर सकें।

उदाहरण के लिए यदि हमें केवल एक Student के Roll Number को Computer में Store करना हो, तो हम एक int प्रकार का Variable Declare कर सकते हैं। यदि हमें दस Students के Roll Number Store करने हों तो हम int प्रकार का एक Array Create कर सकते हैं, लेकिन यदि हमें एक Student के नाम, Address, Contact Number, Class, City, State आदि कई प्रकार के Data को Compute में Store करना हो, तो हमारे पास एक ऐसा तरीका होना चाहिए, जिससे हम एक ही समय पर आवश्यकतानुसार int, float, double, char सभी प्रकार के Data के समूह के साथ प्रक्रिया कर सकें।

यदि ऐसा तरीका ना हो तो हम ये किस तरह से ये तय करेंगे कि हम जिस Student के नाम को Access कर रहे हैं, उसके पिता का वही नाम Computer Return करेगा जो हमने Feed किया था। ऐसा तभी हो सकता है, जब हर Student के Record के सभी Fields एक दूसरे से Related हों और सभी Fields एक साथ Access होते हों। Class हमें यही सुविधा प्रदान करता है। एक Class के द्वारा हम Real World Object को एक नए Data Type के रूप में Computer में Logically Describe कर लेते हैं और उस Description के आधार पर नए Data Type के Objects Create करके उसी प्रकार से प्रक्रिया करते हैं, जिस तरह से उन Objects के साथ Real World में प्रक्रिया की जाती है।

"C++" की सबसे बडी विशेषता इसकी Class व Objects की नई अवधारणा ही है। Class के बारे में हमने पिछले अध्याय में थोडा सा बताने की कोशिश की थी। अब हम Class को अच्छी तरह से समझेंगे। Class वास्तव में Structure का ही विस्तृत रूप है। Class को Define करने के लिए class Keyword का प्रयोग किया जाता है। Class को Define करने का Format निम्नानुसार होता है:

```
Class class_name
{
```

```
    private:
    Data_Members;
    Member_Functions;

    Public:
    Data_Members;
    Member_Functions;
};
```

जिस तरह से हम Structure को New Data Type Define करने में Use करते हैं, उसी प्रकार से class को भी New Data Type बनाने के लिए Use किया जाता है। लेकिन जहां Structure प्रकार के Data Type के Variables को Declare करने के लिए हमें Structure के नाम के साथ struct Key word का प्रयोग करना पडता है, वहीं class प्रकार के Data Type के Variables को Declare करने के लिए हमें Class के नाम के साथ class Keyword का प्रयोग नहीं करना पडता है। साथ ही Class प्रकार के Variables को Variables के बजाय Object के नाम से पहचाना जाता है।

Structure के Members Default रूप से Global या Public होते हैं जबकि class के Members Default रूप से Local या Private होते हैं। class व Structure में दूसरा अन्तर ये है कि Structures में Structure के Data को कोई भी Function Program में कहीं भी Use कर सकता है। जबकि Class के Data Members को Use करने के Functions Class के अन्दर ही Declare किए जाते है और हम केवल इन्ही Member Function के द्वारा Class के Data Members को Use कर सकते हैं।

किसी Class में जिन Data Members व Member Function को private Area में लिखा जाता है, उन Data Members व Member Functions को केवल उसी Class द्वारा Access किया जा सकता है। जबकि जिन Data Members व Member Functions को public Area में लिखा जाता है, उन Data Members व Member Functions को पूरे Program में कहीं भी उपयोग में लाया जा सकता है। ठीक उसी तरह से जैसे Structure के Members को प्रोग्राम में कहीं भी उपयोग में लिया जा सकता है।

Class को अच्छी तरह से समझने के लिए हम सबसे पहले इसे एक Structure को Modify करके Class में बदलेंगे और फिर क्रम से Class की विशेषताओं को समझते हुए इसका प्रयोग करेंगे। हमने "C" में book_bank का एक उदाहरण लेकर Structure को समझाया था। यही उदाहरण हम Class को समझाने के लिए ले रहे हैं।

```
struct book_bank
{
    char title[20];
    char author[30];
    int page;
    float price;
} book1;
```

यहां एक Structure बनाया गया है जिसका एक Variable book1 है। यदि हम किसी book का title, author का नाम, book के पृष्ठों की संख्या और book की कीमत Input करके उसे Display करना चाहें तो हमें निम्न प्रकार का एक main() Function लिखना पडेगाः

Program

```
#include <iostream.h>
#include <conio.h>

struct book_bank
{
    char title[20];
    char author[30];
    int page;
    float price;

    void Input(void)
    {
        cout<<"Enter Book's Title \t";
        cin>>title;
        cout<<"\nEnter Book's Author \t";
        cin>>author;
        cout<<"\nEnter Book's Page \t";
        cin>>page;
        cout<<"\nEnter Book's Price \t";
        cin>>price;
    }

    void Display(void)
    {
        cout<<"Book Name \t :"<<title;
        cout<<"\nBook Author \t :"<<author;
        cout<<"\nBook Page \t :"<<page;
        cout<<"\nBook Price \t :"<<price;
    }
};

void main(void)
{
    book_bank book1;
    clrscr();
    book1.Input();
    book1.Display();
    getch();
}
```

ये प्रोग्राम देख कर शायद आप Confuse हो जाएं कि किस प्रकार से Structure को Use किया गया है। इसलिए पहले हम इसी Program को समझते हैं। हम जानते हैं कि एक Structure विभिन्न प्रकार के Data Type के, आपस में Logically Related Data का समूह होता है। यानी हम एक Structure में विभिन्न प्रकार (Data Type) के Data रख सकते हैं।

जब हम विभिन्न प्रकार के Data Type के Members किसी Structure में रख सकते हैं, तो उसी Structure में हम विभिन्न प्रकार के Functions भी रख सकते हैं और जिस तरह से किसी Structure प्रकार के Variable Declare करके, हम उन Variables द्वारा, Structure के विभिन्न

Members को Access करते हैं, उसी प्रकार से इन Functions को Dot Operator या Arrow Operator का प्रयोग करके Access कर सकते हैं।

मान लो कि हमें किसी Book का नाम इस Structure में रखना है, तो हम book_bank प्रकार का एक Structure प्रकार का Variable Declare करते हैं। माना हमने book_bank book; Statement के प्रयोग से एक book_bank प्रकार का Variable Declare किया और इसी Variable के साथ Dot Operator का प्रयोग करके हमने Structure में स्थित Member title में book.title = "Ramayan" Statement द्वारा नाम भेज दिया है। यानी

```
book_bank book;
book.title = "Ramayan";
```

इस Statement से Structure के title नाम के Member में "Ramayan" Store हो जाता है। यदि हम ठीक इसी प्रक्रिया को अपनाते हुए किसी Structure में Declare व Define किए गए Function को Use करना चाहें, तो उसे भी Access कर सकते है।

इस Program के Structure में हमने दो Function को Structure के Members के रूप में लिखा है, साथ ही उन्हें Define भी कर दिया है। फिर हमने book1 नाम का एक Structure प्रकार का Variable Declare किया है और book1.Input() Statement द्वारा Structure book_bank के एक Member, जो कि एक Function है, को Call किया है। ये Statement Compiler को ये बताता है कि Input नाम का एक Function है, जिसे book_bank Structure में Define किया गया है और ये Function उस Structure का Member Function है, इसीलिए इसे Dot Operator के साथ Use किया जा रहा है।

जैसे ही Program को Run किया जाता है, Variable Declare होने के बाद Program Control जब book1.Input() Statement को Execute करता है, तो Program Control सीधे ही Structure में पहुंच जाता है और Input() Function के Statements का Execution करना शुरू कर देता है।

ये Function एक Statement Block {} में है, और यहीं पर सभी Members Declare किए गए हैं, इसलिए Function को ये नही बताना पडता है कि हम किसी Structure के Members को मान दे रहे हैं, क्योंकि Function स्वयं उसी Structure में है। Input() Function मान प्राप्त करके Structure में रख देता है और Display() Function Structure में Store मानों को Print कर देता है।

अब इसी Program को यदि class के रूप में बनाया जाए, तो ये Program निम्नानुसार बनेगा जो कि बिल्कुल उसी प्रकार से काम करेगा जिस प्रकार से ये Program कर रहा है। चूंकि जैसाकि हमने पहले ही बताया कि class के Members Default रूप से Private होते हैं, इसलिये यदि इसी Program में केवल struct Keyword के स्थान पर class Keyword को Replace कर दें और Class में public Keyword का प्रयोग किया जाए, तो ये Structure एक Class में बदल जाएगा। इसी Program को नीचे दिखाया गया है।

Program

```
#include <iostream.h>
#include <conio.h>

class book_bank
{
```

```
    public:
    char title[20];
    char author[30];
    int page;
    float price;

    void Input(void)
    {
        cout<<"Enter Book's Title \t";
        cin>>title;
        cout<<"\nEnter Book's Author \t";
        cin>>author;
        cout<<"\nEnter Book's Page \t";
        cin>>page;
        cout<<"\nEnter Book's Price \t";
        cin>>price;
    }

    void Display(void)
    {
        cout<<"Book Name \t :"<<title;
        cout<<"\nBook Author \t :"<<author;
        cout<<"\nBook Page \t :"<<page;
        cout<<"\nBook Price \t :"<<price;
    }
};

void main(void)
{
    book_bank book1;
    clrscr();
    book1.Input();
    book1.Display();
    getch();
}
```

जिस तरह से हम किसी Structure के Members को Directly Access कर सकते हैं, उसी प्रकार से किसी Class में public Area में Declare किए गए Members को भी Use कर सकते हैं। जैसे माना हम किसी Book का नाम व उसकी कीमत को इस Class में Store करना चाहें, तो हम निम्न Statement के अनुसार ये कर सकते हैं:

```
book1.title = “Ramayana”;
book1.price = 1230;
```

साधारण रूप से Class के अन्दर किए जाने वाले Declarations को Header Files में लिख कर रख दिया जाता है और उस Header File को Program में Include कर लिया जाता है, लेकिन हम चाहें तो हम Class में किए जाने वाले Declarations को Main Program File में भी रख सकते हैं, जैसाकि अभी बनाए गए Program में किया गया है। Program के Member Functions को दो जगह Define किया जा सकता है। या तो जहां Member Function को Declare किया

जाता है, वहीं पर उसे Define कर दिया जाए। जैसाकि ऊपर के Program में किया गया है। इसे **Inside The Class Definition** कहा जाता है।

दूसरे तरीके में Class के अन्दर केवल Member Function को Declare किया जाता है और Member Function को Class के बाहर Define किया जाता है। इस तरह के Definition को **Outside The Class Definition** कहा जाता है। इस प्रक्रिया को नीचे के Program में Use किया गया है।

Program

```
#include <iostream.h>
#include <conio.h>

class book_bank
{
    public:
    char title[20];
    char author[30];
    int page;
    float price;
    void Display(void);
    void Input(void);
};

void main(void)
{
        book_bank book1;
        clrscr();
        book1.Input();
        book1.Display();
        getch();
}

void book_bank::Display(void)
{
    cout<<"Book Name \t :"<<title;
    cout<<"\nBook Author \t :"<<author;
    cout<<"\nBook Page \t :"<<page;
    cout<<"\nBook Price \t :"<<price;
}

void book_bank::Input(void)
{
    cout<<"Enter Book's Title \t";
    cin>>title;
    cout<<"\nEnter Book's Author \t";
    cin>>author;
    cout<<"\nEnter Book's Page \t";
    cin>>page;
    cout<<"\nEnter Book's Price \t";
    cin>>price;
}
```

इस Program में Member Functions को Class के बाहर Define किया गया है। जब किसी Member Function को Class के बाहर Define किया जाता है, तब हमें Compiler को ये बताना जरूरी होता है कि जिस Function को Define किया जा रहा है, वह Function किस Class में Declare किया गया है।

ये बताने के लिए हमें उस Class को Member Function के साथ Scope Resolution Operator द्वारा जोडना पडता है। जैसे कि void book_bank::Input(void) statement Compiler को बताता है कि जिस Function को Define किया जा रहा है, वह Function book_bank Class का एक Member Function है। जब भी हम "C++" में कोई Function Define करते हैं, तब Compiler को ये भी बताना पडता है कि वह Function किस प्रकार के Argument लेगा और किस प्रकार का मान Return करेगा।

चूंकि ये Function किसी प्रकार का कोई मान Return नहीं कर रहा है इसलिए Function की Return Value void Declare की है तथा ये Function किसी प्रकार का कोई Argument ले भी नहीं रहा है, इसलिए Function के कोष्ठक में भी void का प्रयोग किया है।

अभी हमने जो प्रोग्राम बताया है उस Program में class book_bank के सभी Members को public रखा है। चूंकि, OOP में Data को ज्यादातर Hide रखा जाता है, इसलिए Data Members के Variables को private Area में लिखा जाता है। जब हम Data Members को Private Area में लिखते हैं, तब हमें ऐसे Member Function लिखने होते हैं जो कि public हों, क्योंकि केवल class के वे Member Function ही उस Data के साथ प्रक्रिया कर सकते हैं।

जैसे माना हमें किसी Book की कीमत व पृष्ठों की संख्या Object में Initialize करनी है और हम पिछले Program में यदि book1.price = 123; और book1.page = 123; Statement Use करें, तो Program काम करेगा और Object में Price व Page Initialise हो जाएंगे।

लेकिन यदि Class book_bank के Data Members को private Area में Declare किया जाए, तो ये Statement Illegal होंगे। क्योंकि Class की ये विशेषता है कि Private Area में Declare किए गए Data Members को केवल उसी class के Member Function ही Access कर सकते हैं।

Object

"C" मे हमने पढा है कि किसी Structure प्रकार के Variables User Defined Data Type होते हैं। "C" में इसे Structure प्रकार के Variables कहते हैं। ठीक इसी प्रकार से हम "C++" में भी Structures को User Defined Data Type के Variables Declare करने के लिए Use कर सकते हैं। "C++" में User Defined Data Type के Variable को **Object** कहा जाता है। यानी Structure, Class या Union प्रकार के Variables को Object कहा जाता है।

यदि ये कहा जाए कि अमुक Structure प्रकार का Variable या अमुक Class प्रकार का Variable, तो ये बोलने में काफी बडा शब्द बन जाता है। इस बडे नाम को ही छोटे रूप में अमुक Structure, Class या Union का Object कहा जाता है।

Object Oriented Programming System (OOPS) एक ऐसा तरीका है जिसमें Develop किया जाने वाला Program उसी तरह किसी समस्या को हल करता है जिस तरह से हम हमारे वास्तविक जीवन में उस समस्या को हल करते हैं। Object Oriented Programming System को ठीक से

समझने के लिए हम एक सामान्य सी समस्या को लेते हैं और उसे Object Oriented Programming System (OOPS) के सिद्धांतों का प्रयोग करते हुए हल करते हैं। इस तरीके से हम Object Oriented Programming System (OOPS) को ज्यादा अच्छी तरह से और प्रायोगिक तौर पर समझ सकते हैं।

Book Stall as Object

किसी भी समस्या का समाधान Computer द्वारा प्राप्त करने के लिए हम Softwares या Programs बनाते हैं। ये Program बनाने से पहले हमें समस्या को अच्छी तरह से समझना होता है। मानलो कि हमारे पास समस्या ये है कि एक Book Stall का मालिक जिसका नाम RadhaKrishna है, हमारे पास आता है और कहता है कि वह अपने Book Stalls को Computer द्वारा Manage करना चाहता है। इसलिए उसे एक ऐसे Software की जरूरत है जिससे उसकी जरूरत पूरी हो जाए।

चूंकि उसकी जरूरत क्या है ये जाने बिना हम उसके लायक Software नहीं बना सकते हैं और Software Develop करने के लिए ये जरूरी है कि वह अपने Book Stall को वर्तमान में किस प्रकार से Handle करता है, ये जाना जाए।

जिस तरह से वह वर्तमान में वास्तविक तौर पर अपने Book Stalls को Handle करता है, Computer में भी हम उसे उसी तरह से Represent कर सकते हैं, क्योंकि हमारे पास Object Oriented Programming Language "**C++**" है। जब उससे ये पता किया कि उसके Book Stall पर क्या होता है, तो उसने बताया कि उसके 6 Book Stalls हैं और हर Book Stall पर केवल दो ही Items Books व Magazines बेचे जाते हैं।

वास्तव में इस Program का Object पूरे Book Stall को Modal नहीं करेगा बल्कि केवल उन जरूरी Data को Modal करेगा जिनकी एक Book Stall को बिना रूकावट के चलाने के लिए जरूरत होती है। RadhaKrishna के Book Stalls पर हर रोज कुछ पुस्तकें व Magazines बेंची जाती हैं। यदि पुस्तकें या Magazines समाप्त होने वाली हों, तो नई पुस्तकें खरीदी जाती हैं। बस इतना ही काम होता है।

RadhaKrishna अपने Book Stall की स्थिति (**State of Attributes**) बताता है कि पूरे शहर में अलग–अलग स्थानों पर उसके 6 Book Stalls हैं और उसने उन Book Stalls को Operate करने के लिए 6 लोगों को Hire कर रखा है। वह स्वयं एक Central Office में रहता है जो कि लगभग उसके सभी Book Stalls से मध्य में है। इस व्यवस्था को हम निम्न चित्र में देख सकते हैं।

RadhaKrishna ने अपने सभी Book Stalls पर एक Phone लगा रखा है ताकि सभी Book Stall के Operators से Touch या Connected रह सके। उसने अपने सभी Book Stall को एक Number दे रखा है, जिससे वह सभी Book Stalls को पहचानता है। ये तो RadhaKrishna के Book Stall की स्थिति (**State**) है।

अपने Book Stall की स्थिति बताने के बाद अब वह बताता है कि वह अपने Book Stalls को Handle कैसे करता है? वह कहता है कि हर रोज जिस समय कोई Operator Book Stall Open करता है, तो वह तुरन्त कुल Books व Magazines की संख्या यानी Current Stock RadhaKrishna को Phone करके बताता है। यानी उसके Book Stall के Stock की स्थिति (**State of Stock**) क्या है।

साथ ही जब भी किसी Book Stall से कोई Item बिकता है, तो उस Book Stall का Operator अपने Book Stall का Number बताता है और RadhaKrishna को ये **Message** देता है कि

उसके Stall से एक Book या एक Magazine बिक चुकी है। (हालांकि वास्तव में ऐसा नहीं हो सकता है कि हर Book के बिकने पर Appropriator को Message किया जाए और Book Stall पर केवल एक ही Book व Magazine बेची जाती हो)

इस तरीके से RadhaKrishna को उसके हर Book Stall के Stock का Input व Output पता चलता रहता है, जिसे वह Note करता रहता है और इस Node के आधार पर उसे पता रहता है कि किस Book Stall पर किसी समय कितना Stock उपलब्ध है। इस जानकारी के आधार पर RadhaKrishna समय रहते ये तय कर पाता है कि उसे Supplier को और Books व Magazines का Order कब देना है।

इस पूरी जानकारी के आधार पर हम कह सकते हैं कि RadhaKrishna को अपने हर Book Stall से तीन प्रकार से Interactions की जरूरत होती है और इन्हीं तीन तरीकों का Interaction उसे उसके Program से चाहिए ताकि वह अपने Program से निम्न तीन काम कर सके–

1. वह अपने Program में हर रोज कुल Books व Magazines की संख्या Enter कर सके।
2. हर Book या Magazine के Sale होने का Record रख सके। यानी जब भी किसी Book Stall से कोई Book या Magazine बिके, तो वह उस Book Stall के Stock को कम कर सके।
3. वह Program से ये जान सके कि किसी Book Stall पर किसी समय कुल कितना Stock शेष है।

Step 1
पहले Step को पूरा करते हुए हर रोज सुबह RadhaKrishna अपने Program द्वारा निम्नानुसार हर Book Stall के Books की संख्या या Stock को Computer में Enter कर सकता हैः

```
Stall number: 1                      <--user enters stand number
Number of Books in hand: 34          <--user enters quantity
Number of Magazines in hand: 103     <--user enters quantity

Stall number: 2                      <--user enters stand number
Number of Books in hand: 40          <--user enters quantity
Number of Magazines in hand: 13      <--user enters quantity

Stall number: 3                      <--user enters stand number
Number of Books in hand: 49          <--user enters quantity
Number of Magazines in hand: 30      <--user enters quantity

Stall number: 4                      <--user enters stand number
Number of Books in hand: 90          <--user enters quantity
Number of Magazines in hand: 100     <--user enters quantity

Stall number: 5                      <--user enters stand number
Number of Books in hand: 140         <--user enters quantity
Number of Magazines in hand: 65      <--user enters quantity

Stall number: 6                      <--user enters stand number
Number of Books in hand: 20          <--user enters quantity
Number of Magazines in hand: 60      <--user enters quantity
```

यानी Computer एक Message देता है, और RadhaKrishna उस Message के अनुसार Data Input करता है।

Step 2
दूसरी स्थिति प्राप्त करने के लिए जब Program में Book या Magazines के Selling की Entry करनी हो, तब निम्नानुसार केवल एक Message आना चाहिए, जहां RadhaKrishna उस Book Stall का Number Input करेगा जहां से Book Sell हुई है।

```
Enter stall number: 3          <--user enters stand number
```

इसके बाद निम्नानुसार एक और Message आएगा जो RadhaKrishna से पूछेगा कि Book Sell हुई है या Magazine, RadhaKrishna जो भी Option Select करेगा, Program Automatically उस Item के Stock की संख्या को एक कम कर देगा।

```
Press 0 for Book and 1 for Magazine: 1          <--user enters stand number
```

Step3
तीसरी स्थिति प्राप्त करने के लिए RadhaKrishna को केवल किसी भी Stall का Number Input करना होगा। उस Stall के Stock की जानकारी निम्नानुसार प्राप्त हो जाएगी–

```
Enter stall number: 4          <--user enters stand number
Books in hand = 30             <--program displays data
Magazines in hand = 130        <--program displays data
```

हमने पूरे Description के आधार पर ये तय किया कि RadhaKrishna की समस्या का समाधान किस प्रकार से प्राप्त किया जा सकता है और उस Description के आधार पर Interaction यानी User Interface भी तय करने की कोशिश की कि RadhaKrishna अपने Program से किस प्रकार Interface कर सकेगा या किस प्रकार से अपने Program से Interact करेगा। अब हमें ये सोंचना है कि RadhaKrishna को इस प्रकार का Interaction प्राप्त हो, इसका Design Object Oriented Programming Language "C++" में किस प्रकार से बनाया जाए?

एक Object Oriented Programmer किसी भी समस्या के समाधान के लिए हमेंशा सबसे पहले ये जानना चाहता है कि समस्या में ऐसा कौनसा Object है, जो उस समस्या के Program का सबसे प्रमुख Object बन सकता है। कई बार किसी समस्या का मुख्य Object पता करना काफी मुश्किल काम होता है। फिर भी, हमारे पास कई Similar Real World Objects हों तो वे सभी हमारे Program के मुख्य Object के Candidate हो सकते हैं।

RadhaKrishna की समस्या में कई Objects जैसे कि Books, Book Stall और Book Stalls को Operate करने वाले Operators हैं और सभी Objects अलग–अलग Category के हैं। इनमें से किसे मुख्य Object माना जाए?

चूंकि हमें Book के बारे में जानकारी नहीं रखनी है कि किस नाम की कितनी Books हैं या किसी Book का Writer कौन है। इसलिए Book तो वह मुख्य Object हो ही नहीं सकता। इसी तरह से हमें Book Stalls को Operate करने वाले Operators की भी जानकारी नहीं रखनी है कि किस Book Stall का Operator कौन है। कहां रहता है। कितना पढा–लिखा है, आदि। वास्तव में

RadhaKrishna को उसके सभी Book Stalls के Stock की जानकारी रखनी है। इस स्थिति में Book Stall ही हमारा मुख्य Object है। कई बार एक Programmer गलत Object को मुख्य Object मान लेता है।

शुरूआत में ज्यादातर Programmers के साथ ऐसा ही होता है। लेकिन समय के साथ जैसे–जैसे वे अधिक अनुभवी होते जाते हैं, उन्हें Right Object की पहचान हो जाती है।

समस्या का मुख्य Object पता चल जाने के बाद अब एक Programmer को ये तय करना होता है कि उस Object की Characteristics क्या है। चूंकि हमारा Object तो Book Stall है लेकिन इस Book Stall की Characteristics में से हमें केवल उन Characteristics को ही Abstract करके प्राप्त करना है जो RadhaKrishna की समस्या से सम्बंधित हैं।

चूंकि इन Book Stalls की अलग–अलग Size हो सकती है, ये शहर के अलग–अलग हिस्सों में स्थित हो सकते हैं, इन पर अलग–अलग Operators हो सकते हैं, लेकिन इन सभी Attributes का RadhaKrishna की समस्या से कोई सम्बंध नहीं है।

RadhaKrishna की समस्या से सम्बंधित तो केवल उन Book Stall पर स्थित Books व Magazines की संख्या हैं, जो Book Stall के Stock को Represent करती हैं। इसलिए Book Stall का मुख्य Attribute Books व Magazines हैं। जब कोई Item Sell होता है, तब Item के Stock की संख्या (**State**) में ही अन्तर पडता है।

इसलिए एक Real World Physical Object **Book Stall** का मुख्य Attribute जो कि समस्या से सम्बंधित है, वह है किसी Book Stall Object के कुल Books व Magazines की संख्या। ये Book व Magazine ही है जो हर Book Stall का मुख्य Data है।

हालांकि विभिन्न 6 Book Stall Objects के Books व Magazines की संख्या में अन्तर हो सकता है। फिर भी हर Book Stall पर इसी की संख्या रूपी Data (मान) की स्थिति (**State**) में परिवर्तन आता है। यानी हमारे Book Stall का मुख्य Attribute जो कि Abstraction से हमें प्राप्त होता है वह है:

- **Book**
- **Magazine**

अब चूंकि हमने Book Stall Object के समस्या से सम्बंधित Attribute को तो प्राप्त कर लिया लेकिन साथ ही ये Object अपने Attribute की State में परिवर्तन करने के लिए कुछ **Operations** भी करता है।

यानी जब Stock को Input किया जाता है, तब Object उसके Data यानी Book की Counting की स्थिति में परिवर्तन करता है और उसे Increase करता है और जब कोई Book Sell होती है, तब ये Object अपने Data की स्थिति में परिवर्तन करके उसके मान को Decrease भी करता है।

साथ ही जब User किसी Book Stall के कुल Book की संख्या जानना चाहता है, तब ये Object अपने Data की Current Position को भी Display करता है। यानी ये Object RadhaKrishna की समस्या के अनुसार तीन Operations द्वारा अपने Data या Attribute के मान से Interact करता है।

"C++" में किसी Object के इन Operations की Description को Methods (Member Functions) द्वारा Define किया जाता है। यानी "C++" में यदि इन Operations को Define करें, तो निम्नानुसार इन Operations को Describe कर सकते हैं:

```
inputData()
soldOneBook()
soldOneMagagine()
displayStock()
```

जब किसी Program में एक ही प्रकार के कई Objects होते हैं, तब सभी Objects को अलग से Describe करने के तरीके को अच्छा नहीं कहा जा सकता है। जब इस प्रकार की स्थिति होती है तब समान प्रकार के इन सभी Objects को Represent करने के लिए एक Common Specification या Modal बना लेना ज्यादा बेहतर होता है।

इस Specification को हम एक Blueprint या एक Modal कह सकते हैं। एक बार किसी Object का एक Modal या Specification Design कर लेने के बाद हम इस Blue Print का प्रयोग जितने Objects को Create करने के लिए करना चाहें, कर सकते हैं।

Object Oriented Programming में Objects Creation के इस Modal या Specification को **Class** कहते हैं। RadhaKrishna की समस्या से हमने किसी एक Book Stall Object के दोनों जरूरी Components **Attributes** व **Ability** को प्राप्त कर लिए हैं। इन दोनों के आधार पर हम निम्नानुसार एक Real World Book Stall Object का Computer में एक Logical Modal या Description बना सकते हैं:

```
Class BookStall {
    //Attribute of Object (State)
    private:
        int booksInHand;                //Instance Data
        int MagazinesInHand;            //Instance Data

    //Operations of Object (Behavior)
    public:
        void inputData(void)            //Method for Initializing Stock of Books
        {
        }

        void soldOneBook(void)          //Method for Adjusting Data (Book Counts)
        {
        }

        void soldOneMagazine(void)      //Method for Adjusting Data (Book Counts)
        {
        }

        void displayStock(void)         //Displays Current Stock of the Books
        {
        }
};
```

हम देख सकते हैं कि ये Class Specification दो भागों में विभाजित रहती है। पहले भाग में Object का Data है और दूसरे भाग में उस Data पर Perform हो सकने वाले Operations जिन्हें "C++" में Methods कहते हैं। "C++" में इसी प्रकार से किसी समस्या का समाधान प्राप्त करने के लिए Class Define की जाती है।

एक बात ध्यान रखें कि Specification लिख लेने से Object Create नहीं हो जाता है। Specification तो मात्र एक Blueprint होता है। जिस प्रकार से किसी चीज का एक नक्शा बना देने से वह चीज नहीं बन जाती है।

उसी तरह से किसी Object का Specification तैयार कर देने से Object Create नहीं हो जाता है। Class मात्र एक Specification है, इसलिए एक Class का Description Memory में तब तक कोई Space नहीं लेता है, जब तक कि हम उस Class के Instance के रूप में कोई Object Create ना करें।

हमने Book व Magazines की संख्या Store करने के लिए एक Integer प्रकार का Variable Instance Variable के रूप में Declare किया है। चूंकि Book व Magazines ऐसे Objects है, जो पूर्णांक संख्या में ही Represent हो सकते हैं। इसलिए इनकी संख्या को Store करने के लिए हमने int Keyword का प्रयोग किया है। इस Class में Object अपने Data पर चार Operations कर सकता है, इसलिए हमने Class में चार Methods Define किए हैं।

एक बात हमेंशा ध्यान रखें कि "C++" के भी हर Statement का अन्त Semicolon से होता है लेकिन Function के कोष्ठक के बाद Semicolon नहीं लगाया जाता है। चूंकि Semicolon का प्रयोग Compiler को Statement के अन्त के लिए कहता है। यानी Compiler को जहां भी Semicolon मिलता है, Compiler समझता है, कि वहीं पर किसी एक Statement का अन्त हुआ है, इसलिए यदि Method के कोष्ठक के बाद Semicolon लगा दिया जाए, तो Compiler इसे Method का अन्त मानेगा जबकि ये तो Method या Member Function के Definition की शुरूआत है।

Variable Declarations

हमने Class Specification में ये तय किया है कि किसी भी Object में दो Integer प्रकार के Variable होंगे। चूंकि यहां हम Books व Magazines की संख्या Store करेंगे, इसलिए इन्हें Integer प्रकार का Declare किया गया है। यानी

```
int BooksInHand;
int MagazinesInHand;
```

ध्यान दें कि ये Declaration किसी Variable को कोई मान प्रदान नहीं कर रहा है। यहां ये Declaration केवल ये बताता है कि BooksInHand और MagazinesInHand नाम के दो Integer प्रकार के Variable होंगे जो Memory में अपने प्रकार के अनुसार कुछ जगह Reserve कर लेंगे और उन Reserve Memory Cells का नाम BooksInHand व MagazinesInHand रख देंगे।

Member Functions (Methods)

Class Specification में चार Functions हैं : InputData(), SoldOneBook(), SoldOneMagazine(), DisplayData(), ये Functions C++ के Compiler को बताते हैं कि ये नाम Functions के हैं ना कि किसी Variable के। ये Functions भी अपनी Size के अनुसार Memory में उसी तरह Space Reserve करते हैं, जिस तरह Variables करते हैं।

Functions के आगे Use किए गए Keyword void का मतलब होता है कि Function किसी प्रकार का कोई मान Return नहीं कर रहा है। जब Function किसी प्रकार का मान Return कर रहा होता है तब void के स्थान पर उस Data Type का उल्लेख किया जाता है। जैसे यदि कोई Function ineger प्रकार का मान Return करता है, तो Function निम्नानुसार लिखा जाता है:

```
int largest()
{
}
```

हम Function के कोष्ठक में Arguments के रूप में अन्य Parameters भी दे सकते हैं। जैसे

```
int largest( int a = 10, int b = 15)
{
}
```

इस Function में दो Integer प्रकार के मान Argument के रूप में भेजे जा रहे हैं और उन दोनों मानों में से जो मान बडा है, उसे Return Value के रूप में Return करवाया जा रहा है।

Public and Private

Public और Private का Central Idea ये है कि एक Object के कुछ भाग को Program Statements Access कर सकें जबकि कुछ भाग को केवल Object स्वयं ही Access करे। Class के अन्दर Define किए गए public: व private: Section इस Design को Specify करने के लिए Use किए जाते हैं। हमारी Class में सभी Data **public** Section में हैं और सभी Member Functions **private** Section में।

हम हमेंशा चाहते हैं कि हमारे Program के Data हमेंशा सुरक्षित रहें, इसलिए हम हमारे Data को Private Section में रखते हैं और Object को ये बताने के लिए कि उसे Data के साथ क्या करना है, Member Functions को Public Section में लिखते हैं। Data को Access करने के लिए हम Directly Data को कोई मान ना तो भेज सकते हैं ना ही प्राप्त कर सकते हैं। Data को Access करने के लिए हम Member Functions का प्रयोग करते हैं। इसीलिए Member Functions को Public Area में लिखा जाता है।

Member Functions

इसी अध्याय की शुरूआत में हमने देखा है कि एक Object के दो भाग होते हैं। पहला Object के Data और दूसरा Data पर काम करने वाले Functions, Data के बारे में हमने अभी देखा है। अब हम Functions के बारे में समझने की कोशिश करते हैं।

साधारणतया एक Program किसी Object के member Functions को Call करके Object को कुछ करने के लिए कहता है। किसी Object के Member Function को Call करने को दूसरे शब्दों में Object को **Message Send** करना भी कहते हैं।

हमारे Book Stall Program में चार Member Functions हैं : InputData(), SoldOneBook(), SoldOneMagazine() और DisplayStock() शुरूआत में हमने इन Member Functions के कोष्ठक को खाली रखा था। अब हम इनमें Statements लिखेंगे और देखेंगे कि ये क्या कर सकते हैं।

Data को Initialize करना

हर दिन की शुरूआत में RadhaKrishna अपने Book Stall के Program में Books व Magazines की संख्या को Initialize करना चाहता है। इस काम को करने के लिए हम InputData() Function को Use करेंगे।

इसमें Program RadhaKrishna को Books व Magazines की संख्या को Initialize करने के लिए Prompt करेगा या सूचना देगा। इस काम के लिए हम *cout* व *cin* statement का प्रयोग करेंगे। ये Function निम्नानुसार दिखाई देगाः

```
void InputData()
    {
    cout << "Enter Books on Hand : " ;
    cin >> BooksInHand ;
    cout << "Enter Magazines On Hand : " ;
    cin >> MagazinesInHand ;
    }
```

जब RadhaKrishna Books व Magazines की संख्या Input करने के लिए Program का प्रयोग करेगा, तब निम्नानुसार वह Program से Interact होगाः

```
    Enter Books on Hand :  60
    Enter Magazines On Hand : 40
```

जहां Books व Magazines की संख्या RadhaKrishna Input कर रहा है।

Selling को Record करना

जब किसी Book Stall से एक Book Sell होती है, तो उस Book Stall का Operator RadhaKrishna को Call करके बता देता है कि उसकी Book Stall से एक Book Sell हो गई है। जब RadhaKrishna को ये पता चलता है तब RadhaKrishna उस Book Stall के Book Stack में से एक Book को Less करना चाहता है। इस काम को करने के लिए हमें SoldOneBook() Member Function में निम्न Statements लिखने होंगेः

```
void SoldOneBook()
{
    BooksInHand = BooksInHand – 1 ;    // Variable esa ls ,d de dj nsrk gSA
}
```

इसी तरह जब एक Magazine Sell होती है तब भी RadhaKrishna Program को बता देता है कि एक Magazine Sell हुई है और चाहता है कि Program स्वयं ही Current Stock में से एक Magazine Less कर दे। ऐसा करने के लिए हमें निम्न Code SoldOneMagazine() Member Function में लिखने होंगेः

```
void SoldOneBook()
{
    MagazinesInHand = MagazinesInHand – 1 ;  // Variable esa ls ,d de dj nsrk gSA
}
```

Data को Display करना

ध्यान दें कि BookStall Object Current Stock को BooksInHand व MagazinesInHand Variables में Store करके रखता है। हम cout Statement का प्रयोग करके यदि इन Variables के मानों को Screen पर Print कर दें तो ये RadhaKrishna को Current Stock की जानकारी देगा। ऐसा करने के लिए हमें DisplayStock() Functions में निम्न Coding लिखनी होगीः

```
void DisplayStock()
{
    cout << “Books In Hand = “ << BooksInHand << endl ;
    cout << “Magazines In Hand = “ << MagazinesInHand << endl ;
}
```

इस Function का Output निम्नानुसार होगाः

```
Books In Hand = 60
Magazines In Hand = 40
```

Arithmetic Operators

SoldOneBook() Function में हमने एक मान में से दूसरा मान घटाने के लिएं Subtraction Operator (-) का प्रयोग किया है। C++ में इसी तरह के पांच Operators हैं जिनका अलग–अलग जगह पर गणितीय प्रक्रियाओं को सम्पन्न करने के लिए प्रयोग किया जाता है।

Operator	Purpose
+	Addition
–	Subtraction
*	Multiplication
/	Division
%	Remainder

पहले के चार Operators वही काम करते है जिनके लिए इनका प्रयोग सामान्य गणितीय प्रक्रियाओं में होता है। पांचवा Operator % Reminder Operator है। जब हम किसी Integer संख्या में Integer संख्या का भी भाग देते हैं, तो इस Operator का प्रयोग करके हम शेषफल ज्ञात कर सकते हैं। जैसे :

```
int num = 10, reminder = 0 ;
reminder = num % 3 ;
```

इस Statement के Execute होने पर **reminder** नाम के variable में शेषफल 1 Store हो जाएगा। यानी जब हमें किसी गणितीय प्रक्रिया में किसी संख्या में भाग देने के बाद उसका भागफल किसी Variable में Store करना होता है तब हम / Operator का प्रयोग करते हैं लेकिन जब हमें

उसी Expression का शेषफल किसी Variable में Store करना होता है, तब हम % Operator का प्रयोग करते हैं। हम किसी भी प्रकार की गणितीय प्रक्रिया में इनमें से किसी भी Operator का जिस तरह चाहें उस तरह से प्रयोग कर सकते हैं। जैसे :

```
c = (f-32) * 5 / 9;
```

इस Expression में Fahrenheit के तापमान को Celsius में बदला गया है। ध्यान दें कि हमने यहां पर कोष्ठक का प्रयोग किया है, क्योंकि कोष्ठक में स्थित Expression पहले Calculate होती है और अन्य Expression बाद में। C++ में भी सभी गणितीय प्रक्रियाएं प्राथमिकता क्रम के अनुसार ही Execute होती हैं। जैसे + और / का क्रम + व – Operator से पहले आता है, इसलिए यदि किसी Expression में गुणा व जोड दोनों हो रहे हैं, तो पहले गुणा होगा फिर जोड होगा। जैसे :

```
10 * 5 + 20
```

इस Expression का मान 70 होगा ना कि 250

C++ में हम किसी भी प्रकार के Data Type के Variable की Calculation कर सकते हैं। जैसे ऊपर बताए गए Expression में c का मान Float में होगा और f का मान int में होगा, फिर भी हम इनकी Calculation करवा सकते हैं। C++ हमें किसी प्रकार का कोई Error नहीं देगा बल्कि Calculation से पहले C++ स्वयं ही इन्हें सही Data Type में Convert कर लेता है और फिर Calculation करता है।

Increment और Decrement Operators

Programming में हमें अक्सर किसी Variable में से या तो एक घटाना पडता है या एक जोडना पडता है, जैसाकि हमने SoldOneBook() व SoldOneMagazine() Functions में देखा है। C++ में हमें इस काम के लिए दो नए Operators मिलते हैं, जिन्हें Increment या Decrement Operators कहते हैं।

Increment Operator किसी Variable के मान में एक जोड देता है और Decrement Operator किसी Variable के मान में से एक कम कर देता है। इन Operators का प्रयोग करके हम SoldOneBook() व SoldOneMagazine() Functions को निम्नानुसार लिख सकते हैं:

```
void SoldOneBook()
{
    --BooksInHand ;
}

void SoldOneBook()
{
    --MagazinesInHand ;
}
```

Decrement Operator में दो Minus के चिन्ह होते हैं। यदि BooksInHand Variable का मान 60 हो तो इस Statement के Execute होने पर BooksInHand Variable का मान 59 रह जाता है। इसी तरह से Increment Operator में दो . के चिन्हों का प्रयोग होता है। इन

Increment व Decrement Operators को Unary Operators कहते हैं क्योंकि ये केवल एक ही Variable के साथ प्रयोग होते हैं।

प्राथमिकता क्रम में इन्हें अन्य Arithmetic Operators से पहले का क्रम प्राप्त है। यानी यदि किसी Expression में Increment व Subtraction दोनों Operators का प्रयोग हुआ है, तो पहले Variable का मान Increment होगा फिर Subtraction का Operator काम करेगा।

चलिए, अब हम देखते हैं कि हम अपने Program में किस प्रकार से इन Member Functions का प्रयोग Object से कुछ काम करवाने के लिए करते हैं।

Specifying a Class

अब एक Complete Class Specification के सारे अवयव हमारे पास हैं। हमें पता है कि हमारे Data को Store करने के लिए Basic Data Type के Variable कैसे Declare करने हैं, उन Data से Interact करने के लिए Member Functions कैसे Create करने हैं और कुछ उपयोगी काम करने के लिए Keyboard से Input कैसे लेना है और Screen पर Output कैसे Print करना है। अब हम निम्नानुसार Book Stall Class की पूरी Specification बना सकते हैं:

//Creating First Class in C++

```
class BookStall                 // एक Class जिसका नाम BookStall है।
{                               // Class Specification की शुरूआत
    private:
        int BooksInHand;    // Variables जिसमें Books व Magazines की संख्या Store होगी।
        int MagazinesInHand;

    public:
        void InputData()     // Variables में Data को Input करने के लिए Member Function
        {
            cout << "Enter Books on Hand : " ;
            cin >> BooksInHand ;
            cout << "Enter Magazines On Hand : " ;
            cin >> MagazinesInHand ;
        }

        void SoldOneBook()          // Book के संख्या को Adjust करने के लिए
        {
            --BooksInHand ;         //  यहां Function की Body
        }

        void SoldOneMagazine()     // Magazines की संख्या को Adjust करने के लिए
        {
            --MagazinesInHand ;    // यहां Function की Body
        }

        void DisplayStock()       // Stock को Display करने के लिए
        {
            cout << "Books In Hand = " << BooksInHand << endl ;
```

```
        cout << "Magazines In Hand = " << MagazinesInHand << endl ;
    }
};                          // Class Specification का अन्त करने के लिए
```

अब हम देख सकते हैं कि सभी Member Function Data पर Operations करने के लिए किस तरह से आपस में अलग–अलग तरीके से Related हैं। InputData() Member Function Keyboard से Data Input में ले कर Variables में Store करने का काम कर है। DisplayStock() Function Variables के Data को Screen पर Display करने का काम करता है।

Object Create करना और **Object** के साथ प्रक्रिया करना

अभी तक हमने सीखा कि किस प्रकार से Data और Data पर काम करने वाले Member Functions द्वारा एक Class Specification बनाते हैं। हालांकि Class Specification का उद्देश्य एक Blueprint या नक्शा बनाना होता है। जिसके आधार पर हम Object Create कर सकते हैं। चलिए, समझने की कोशिश करते हैं कि Object किस प्रकार से Create करते हैं और Create किए गए Objects के साथ हम किस प्रकार से Interact कर सकते हैं।

Class Specification द्वारा **Objects** बनाना

हम ठीक उसी प्रकार से Objects Create कर सकते हैं जिस प्रकार से हम किसी Basic Data Type का Variable Create करते हैं। C++ में Objects ठीक उसी प्रकार से Handle किए जाते हैं जिस प्रकार से Variables को और Class को उसी प्रकार से Handle किया जाता है जिस प्रकार से Data Types को।

यानी हम सरल भाषा में कहें तो Class एक नए प्रकार का Data Type होता है और Objects उस Class प्रकार के Data Type के Variables होते हैं। निम्न Statement BookStall का एक Object stall1 Create करता है:

```
BookStall stall1;
```

जब Program का ये Statement Execute होता है, तब सबसे पहले Program BookStall Class की Specification को Program में खोजता है जो कि हमें पहले ही बनानी होती है। ये तय करता है कि इस Class का Object memory में कितनी Byte Reserve करेगा और Object को उतनी Memory Assign कर देता है।

फिर Program Object को Assign की गई Memory को एक नाम Stall1 दे देता है। ये सब काम बिल्कुल उसी तरह से होता है जिस तरह से एक Basic Data Type का Variable Create करते समय होता है। फिर भी, Stall1 Object सामान्य Variable से अधिक जटिल है क्योंकि इसमें विभिन्न प्रकार के Data व Member Functions हैं। एक बार Class की Specification बनाने के बाद हम उस Class के जितने चाहें उतने Objects Create कर सकते हैं। जैसे :

```
BookStall Stall1 ;
BookStall Stall2 ;
BookStall Stall3 ;
```

हम निम्नानुसार भी Object Create कर सकते हैं :

```
BookStall Stall1, Stall2, Stall3 ;
```

Class की Specification बनाना एक जटिल काम होता है लेकिन Class के Object बनाना बिल्कुल सरल है।

Object को **Message** भेजना

एक बार Object Create हो जाए, तब हमें इसके साथ Interaction करना होता है। Object के साथ Interaction करने के लिए हम उन Member Functions का प्रयोग करते हैं, जिन्हें Class Specification में Describe या परिभाषित किया है। हम Object के Member Function को Call करते हैं। इसे ऐसा भी कह सकते हैं कि हम Object को एक **Message Send** करते हैं।

इस काम के लिए एक विशेष Syntax का प्रयोग किया जाता है। जब हम किसी Object को Message Send कर रहे होते हैं तब हमें दो बाते ध्यान में रखनी होती हैं। पहली ये कि हम किस Object के साथ Communicate कर रहे हैं या किस Object को Message Send कर रहे हैं और दूसरी ये कि हम Object को क्या Message Send कर रहे हैं।

इसी बात को हम दूसरे शब्दों में कहें तो ऐसा भी कह सकते हैं कि हम किस Object के Member Function को Call कर रहे हैं। इस तरह हमारे Syntax के दो भाग हो जाते हैं : Object का नाम और Member Function का नाम।

हम निम्नानुसार Syntax द्वारा stall1 Object को Data Display करने का Message Send कर रहे हैं :

```
stall1.DisplayStock() ;
```

Object का नाम व Function का नाम एक Dot से Connected रहता है। इसे Dot Operator या Class Member Access Operator कहते हैं। इस तरीके से हम किसी विशेष Object के Member Function या Data को Access कर सकते हैं। जब ये Statement Execute होता है तब निम्नानुसार Output Display होता है :

```
Books In Hand = 60
Magazines In Hand = 40
```

हम इसी तरह का Message दूसरे Object को भी भेज सकते हैं। जैसे

```
stall2.DisplayStock() ;
```

ये Syntax दूसरे Object के Data को Output में Display करेगा। जैसे

```
Books In Hand = 6
Magazines In Hand = 140
```

इसी तरह हम तीसरे Object में User से Data Input करने के लिए Message Send कर सकते हैं। जैसे

```
stall3.InputData() ;
```

जब ये Statement Execute होगा, तब Program User से निम्नानुसार Interact करेगाः

```
Enter Books In Hand : 65
Enter Magazines in hand : 100
```

एक बार stall3 Object को मान Initialize करने के बाद हम इस Stall से Book व Magazines को Sell कर सकते हैं। Selling की सूचना Object को देने के लिए हम SoldOneBook() या SoldOneMagazine() Member Function का प्रयोग कर सकते हैं। जैसेः

```
stall3.SoldOneBook() ;
```

अब यदि Stock देखना हो तो हम stall3 को Stock Display करने का Message Send करते हैं। यानीः

```
stall3.DisplaySock() ;
```

Program हमें निम्नानुसार Stock Display कर देता हैः

```
Enter Books In Hand : 64
Enter Magazines in hand : 100
```

सारांश

हमने देखा कि एक C++ Program का आधारभूत अवयव Objects हैं जो कि किसी Class Specification के अनुसार बनाए जाते हैं। किसी Program को Design करने का मतलब है कि हम ये तय करें कि Program में Object क्या होगा और वह किस प्रकार से काम करेगा। किसी भी Object की दो बातें मुख्य होती हैं :

- **Data of Object**
- **Member Functions**

एक Object के Data को Program के अन्य भागों से सामान्यतया Private रखते हैं, ताकि Program का कोई भी अन्य भाग Data को Directly Access ना कर सके। यदि Data को Access करना है तो उसके साथ लिखे गए Member Functions का उपयोग करना पडता है। इस तरीके की वजह से Data Accidentally Modify या Corrupt नहीं होता।

एक बार Class Specify करने के बाद और उस Class के Objects Create करने के बाद, Program Object को Message Send करता है। यानी Object को कुछ काम करने का Order देने के लिए उसके Member Function को Call करता है। चलिए, अब हम Book Stall का पूरा Program Access करते हैं:

Program

```
// Book Stall inventory database program
#include <iostream.h>
class BookStall
{
    private:
        int BooksInHand;
        int MagazinesInHand;
```

```
    public:
        void InputData()
        {
            cout << "Enter Books on Hand : " ;
            cin >> BooksInHand ;
            cout << "Enter Magazines On Hand : " ;
            cin >> MagazinesInHand ;
        }

        void SoldOneBook()
        {
            --BooksInHand  ;
        }

        void SoldOneMagazine()
        {
            --MagazinesInHand ;
        }

        void DisplayStock()
        {
            cout << "Books In Hand = " << BooksInHand << endl ;
            cout << "Magazines In Hand = " << MagazinesInHand << endl ;

        }
    };

void main()
{
    BookStall stall1;           // BookStand Object Create fd;k
    BookStall stall2;

    // set initial data
    cout << "\nInitialize data for stall 1";
    stall1.InputData();
    cout << "\nInitialize data for stall 2";
    stall2.InputData();

    // record some sales
    cout << "\nSelling 2 Books from stall1";
    stall1.SoldOneBook();
    stall1.SoldOneBook();
    cout << "\nSelling 3 Magazines from stall2";
    stall2.SoldOneMagazine();
    stall2.SoldOneMagazine();
    stall2.SoldOneMagazine();
    cout << endl;

    // display current data
    cout << "\nStock in hand, Stall1";
```

```
        stall1.DisplayStock();
        cout << "\nStack in hand, Stall2";
        stall2.DisplayStock();
    }
```

हम Class Specification करना जान चुके हैं और Object Create करके उसके साथ प्रक्रिया करने वाले Statement लिखना भी सीख चुके हैं। इस Program में हमने IOSTREAM.H Header File को Include किया है। इसका प्रयोग किए बिना हम cin व cout Objects, << व >> Operators व endl Manipulator का प्रयोग करके Input व Output की सुविधा प्राप्त नहीं कर सकते।

इनकी और अन्य बहुत सी चीजों की Specifications इस Header File में है जो कि हमारे Compiler के साथ आती है। ये एक Text File होती है ठीक उसी तरह जिस तरह हमारी C++ की Source File होती है।

इस Header File को हमारे Program में Insert करने के लिए हमें इसके नाम के पहले एक Preprocessor Directive का प्रयोग करना पडता है। इसे हम #include Keyword के साथ निम्नानुसार लिख कर Include करते हैः

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

ये Directive Compilation के समय इस Text File के सारे Source Codes को हमारे Program में Insert कर देता है। Preprocessor Directives सीधे ही Compiler के लिए दी गई Instructions होती हैं। जैसे Beta = 60; Microprocessor के लिए एक Instruction है, जिसे Compiler Machine Language में Translate करके Microprocessor पर भेजता है और Micro Processor उसे समझता है। Preprocessor Directive हमेंशा # चिन्ह से शुरू होता है।

जैसे ही Compiler को #include Preprocessor Directive मिलता है, ये IOSTERAM.H नाम की File को खोजना शुरू कर देता है। हमारे Compiler के साथ कुछ Sub Directories भी होती हैं जिनमें include नाम की एक Directory होती है। यदि ये File Include नाम के Folder में ना हो, तो हमें Compiler को इस फाईल का पाथ देना जरूरी होता है, अन्यथा हमारा Program Compile नहीं होता। Compiler को जैसे ही ये File मिल जाती है, Compiler इस File को वहां पर Insert कर देता है, जहां पर हमने #include Preprocessor Directive का प्रयोग किया है।

#include Directive को Specify करने के दो तरीके हैं। सही तरीका Use करने पर Program की Speed अच्छी होती है। इसमें हम header File को <> के बीच में लिखते हैं। ऐसा करने पर Compiler Header File को Include नाम के Sub Folder में ही खोजता है। दूसरे तरीके में हम Header File या अन्य किसी Source File को “” के बीच में लिखते हैं। इस तरह लिखने पर Compiler header File को वहां खोजता है जहां हमारी Source File होती है।

C++ की Library में कई प्रकार के कामों के लिए अलग–अलग Header Files हैं। जैसे गणितीय कामों के लिए हम MATH.H नाम की Header File को अपने Program में Insert कर सकते हैं और Data Conversion से सम्बंधित कामों के लिए हम STDLIB.H नाम की Header File को।

Main Function

हर चीज की एक शुरूआत होती है और हमारा Program जहां से शुरू होता है, उसे main() Function कहते हैं। ये किसी Class का Member Function नहीं होता है। ये एक Standalone

Function है जिससे Program का Control Operating System से Transfer होता है। main() Statement किसी भी Program का सबसे पहले Execute होने वाला Statement है। Program के अन्य सभी Statements main() Function के अन्दर ही लिखे जाते हैं।

Loops

हम Loops का प्रयोग किए बिना एक उपयोगी प्रोग्राम नहीं बना सकते हैं। बिना Loop का प्रोग्राम केवल एक बार ही हमारा वांछित काम करता है, जबकि Loop का प्रयोग करने से हमारा प्रोग्राम कई बार हमारा वांछित काम करता है। C++ में तीन तरह के Loops का प्रयोग किया जा सकता है। ये तय करने के लिए कि प्रोग्राम कितनी बार परिणाम प्रदान करेगा, C++ का Loops Condition या Expression के True या False होने को Check करता है। True या False का मान ये तय करता है कि Loop चलेगा या नहीं।

True and False Values

Loops और Decision Making Statements True या False के आधार पर ये तय करते हैं कि प्रोग्राम किस प्रकार से Run होगा और परिणाम देगा। C++ में 0 False को और कोई भी अन्य मान (Any nonzero value) True को Represent करता है। कई भाषाओं में एक विशेष प्रकार का Boolean Data Type होता है जो केवल True या False को ही Hold करता है, लेकिन C++ में इन मानों को किसी भी Integer Data Type में Store किया जा सकता है।

कई बार किसी एक Variable के मान को ये तय करने के लिए कि किसी Statement Block या Statement का Execution Condition के अनुसार कितनी बार होगा, प्रयोग कर लिया जाता है। भिन्न–भिन्न प्रकार की Conditions को Check करने के लिए हमें हमारे Loop में Conditional Operators का प्रयोग करना पडता है। जब तक Condition True होती है, Loop Execute होता है। जैसे ही Condition False हो जाती है, Loop terminate हो जाता है।

Relational Operators

Relational Operators दो मानों की तुलना करते हैं और True या False Return करते हैं। C++ में 6 Conditional Operators होते हैं।

Symbol	Meaning	Example
==	equal to	a == b
!=	not equal to	a != b
<	less than	a < b
>	greater than	a > b
<=	less than or equal to	a <= b
>=	greater than or equal to	a >= b

ऊपर बताई गई विभिन्न स्थितियों में a और b के मान पर ये निर्भर करता है कि Condition True होगी या False, जैसे यदि a = 1 व b = 4 है तो a= = b False Return करेगा क्योंकि a व b का मान बराबर नहीं है। लेकिन a!=b True Return करेगा, क्योंकि a b के बराबर नहीं है ये बात सही है।

हम Characters की तुलना भी उसी तरह से कर सकते हैं जिस तरह से numbers की तुलना करते हैं। जैसे यदि 'A' == 65 True Return करेगा क्योंकि 'A' की ASCII Value Integer 65 के बराबर होती है।

while Loops

while loop हमें ये सुविधा देता है कि Program किसी Statement Block को तब तक Execute करता रहता है जब तक कि Condition Change नहीं होती। यानी Condition True से False या False से True नहीं होती। निम्न Statements को देखिए,

```
char answer;
while ( answer != 'y')
{
   cout << "Enter a Character : ";
        cin >> answer ;
}
```

इस Loop में जब तक User 'y' Press नहीं करता है, तब तक Program User से Data लेता रहता है। इसका Output निम्नानुसार प्राप्त होता है:

```
Enter a Character : q
Enter a Character : d
Enter a Character : g
Enter a Character : f
Enter a Character : h
Enter a Character : j
Enter a Character : y
```

एक while Loop में एक Keyword while का और एक कोष्ठक का प्रयोग होता है जिसमें हम Expression या Condition देते हैं। लेकिन इस कोष्ठक के बाद Semicolon का प्रयोग नहीं किया जाता है क्योंकि Semicolon का प्रयोग Statement के अन्त को दर्शाने के लिए होता है और हम while Loop की शुरूआत ही इस प्रकार के Syntax से करते हैं। यदि हमारे Loop की Body में केवल एक ही Statement हो तो हमें Curly Braces का प्रयोग करने की जरूरत नहीं रहती है। जैसे–

```
while(A < 100)
   A = A * 3;     <--One-statement loop body, so no braces
```

यदि Condition सही होती है तो Program Control while Loop के Statement Block को Execute करता है। यानी पहले Condition Check होती है। लेकिन जब हमारे Program में हमें ऐसी जरूरत होती है कि Condition सही हो या ना हो लेकिन कम से कम एक बार तो Loop Execute होना ही चाहिए, तब हम do while Loop का प्रयोग करते हैं।

इस Loop में Program Control do while Loop के Statement Block को Execute करता है। उसके बाद Condition को Check करता है। यदि Condition True होती है, तो Program Control वापस Loop के Statement Block को Execute करता है अन्यथा अगले Statement पर चला जाता है। जैसे–

```
char answer;
 do
{
```

```
        cout << "Enter a Character : ";
        cin >> answer ;
    } while ( answer != 'y');
```

for Loops

जब हमें हमारे किसी Program में पता होता है कि किसी Statement को कितनी बार Execute करना है, तब हम for Loop का प्रयोग करते हैं।

Logical Operators

जब हमें कई Conditions के अनुसार True या False को ज्ञात करना होता है और Statements Execute करने होते हैं, तब हम Relational Operators के साथ Logical Operators का भी आवश्यकतानुसार प्रयोग करते हैं। Logical Operators मुख्यतः तीन होते हैं:

Logical Operator	Meaning	Example
&&	AND	x>0 && x<10
||	OR	x==3 || x<1
!	NOT	!x

Precedence

Operators	Operator	Types Precedence
* / %	Multiplicative	Higher
+ -	Additive	
< > <= >= = = !=	Relational	
&& ||	Logical	
=	Assignment	Lower

Reusability

अब हम Loops और Decision Making Statements द्वारा Book Stall program को Modify करते हैं, जिससे हम जितनी बार चाहें उतनी बार Program में Data Input कर सकें और Stock को Check कर सकें।

Program

```
//Class Description
// Book Stall inventory database program

#include <iostream.h>
#include <conio.h>
#include <stdlib.h>

class BookStall
{
    private:
    int BooksInHand;
    int MagazinesInHand;

    public:
```

```
    void InputData()
    {
        cout << "\n\tEnter Books on Hand : " ;
        cin >> BooksInHand ;
        cout << "\tEnter Magazines On Hand : " ;
        cin >> MagazinesInHand ;
    }

    void SoldOneBook()
    {
        --BooksInHand  ;
    }

    void SoldOneMagazine()
    {
        --MagazinesInHand ;
    }

    void DisplayStock()
    {
        cout << "\t\tBooks In Hand = " << BooksInHand << endl ;
        cout << "\t\tMagazines In Hand = " << MagazinesInHand << endl;
    }
};

//Class Implementation
void main()
{
    char answer;
    BookStall stall1;
    BookStall stall2;

    // set initial data
    cout << "\nInitialize data for stall 1";
    stall1.InputData();
    cout << "\nInitialize data for stall 2";
    stall2.InputData();

while (answer != 'x')
{
    cout << "\n\tPress       1       For Selling 1 Books from stall1";
    cout << "\n\tPress       2       For Selling 1 Magazine from stall1";
    cout << "\n\tPress       3       For Selling 1 Books from stall2";
    cout << "\n\tPress       4       For Selling 1 Magazines from stall 2";
    cout << "\n\tPress       5       For Display Your Current Stack";
    cout << "\n\tPress       Any other Key For Exit";

    cout << "\n\nEnter Your Choice . . . " ;
    cin >> answer;

        if(answer == '1')
```

```
            {
                clrscr();
                stall1.SoldOneBook();
                cout << "\n\tOne Book Have Been Sold From Stall1\n\n" ;
            }

            else
            if(answer == '2')
            {
                clrscr();
                stall1.SoldOneMagazine();
                cout << "\n\tOne Magazine Have Been Sold From Stall1\n\n";
            }

            else
            if(answer == '3')
            {
                clrscr();
                stall2.SoldOneBook();
                cout << "\n\tOne Book Have Been Sold From Stall2\n\n";
            }

            else
            if(answer == '4')
            {
                clrscr();
                stall2.SoldOneBook();
                cout << "\n\tOne Magazine Have Been Sold From Stall2\n\n";
            }

            else
            if(answer == '5')
            {
                clrscr();
                cout << "\nStock of Stall1" << endl;
                stall1.DisplayStock();
                cout << "\nStock of Stall2" << endl;
                stall2.DisplayStock();
            }

            else
            {
                clrscr();
                exit(1);
            }
        }
    }
```

हम देख सकते हैं कि हमने main() Program में ही दो बार परिवर्तन करके नया प्रोग्राम बनाया है। यानी main() प्रोग्राम की Functionality में ही परिवर्तन किया है। Class में हमने बिना किसी प्रकार का कोई परिवर्तन किए ज्यों का त्यों ही उपयोग में ले लिया है। यानी हम एक ही Class को

अलग–अलग तरीकों से अलग–अलग प्रोग्रामों में प्रयोग में ले सकते हैं। इस प्रक्रिया को **Reusability** कहा जाता है। Object Oriented Programming System (OOPS) की एक बहुत ही प्रभावशाली प्रक्रिया है।

इसमें हमें Class में किसी प्रकार का परिवर्तन करने की कोई चिन्ता नहीं होती। हमें हमारा ध्यान केवल इस बात पर लगाना होता है कि main() प्रोग्राम किस प्रकार से बनाना है या User Program के Objects को किस प्रकार से Use करेगा।

चलिए, काफी हद तक हमने "C++" के Class व उसके Instance यानी Objects को तथा सामान्य Programming को समझा। Class Concept के अलावा जो भी कुछ Use किया जा रहा है, वह ज्यादातर "C" Language का हिस्सा ही है। जैसे Decision Making के विभिन्न Statements, विभिन्न प्रकार के Looping Statements आदि सभी "C++" में भी उसी प्रकार से Use किए जाते हैं जिस प्रकार से "C" में। अब हम "C++" के Class की कुछ और विशेषताओं को समझने की कोशिश करते हैं।

जैसाकि हमने पहले भी कहा था कि "C++" में हम Real World Objects को Represent करने वाले New Data Types भी Create कर सकते हैं और उन्हें उसी प्रकार से Use कर सकते हैं जिस प्रकार से किसी सामान्य Basic Data Type को Use करते हैं। अब हम इसी को समझने की कोशिश करेंगे कि हम किस प्रकार से "C++" में नए Data Type बना सकते हैं और उन्हें उसी प्रकार से Use कर सकते हैं जिस तरह से Basic Data Types को Use करते हैं। हम यहां Time को एक नए Data Type के रूप में Create करेंगे।

किसी भी Time के हमेंशा दो भाग होते हैं। पहले भाग को Hours व दूसरे भाग को Minutes से Represent किया जाता है। हम इस नए प्रकार के Data Type को **TTime** नाम से Represent करेंगे।

हमें नए प्रकार के Data Types की क्या जरूरत है? क्या int, float, char प्रकार के Data Types को प्रयोग से हम सभी प्रकार के काम नहीं कर सकते हैं? इस सवाल का जवाब है कि हालांकि हम इन Basic Data Types से अपनी सभी Requirements को पूरा कर सकते हैं लेकिन जब किसी Real World Object या किसी Complex Number को Represent करना होता है, तब इन Basic Data Types का प्रयोग करना व उन्हें Maintain करना काफी मुश्किल हो जाता है।

उदाहरण के लिए मान लो कि आपको दो Time Values Time1 व Time2 को जोडना है। क्या "C" हमें ये सुविधा प्रदान करता है कि हम निम्नानुसार Statements द्वारा दो Time Values को जोडकर तीसरा Time Value प्राप्त कर सकें–

```
Time1 = Time2 + Time3
```

नहीं ! हम बिल्कुल Basic Data Types की तरह किसी User Defined Data Type को नहीं जोड सकते हैं। यदि हम इसे जोडना चाहें तो हमें इसे दो भागों Minutes व Hours में विभाजित करना होगा और उन्हें जोडकर निम्नानुसार Resulting Time प्राप्त करना होगाः

```
Hours1 = Hours2 + Hourse3
Minutes1 = Minutes2 + Minutes3
```

हम देख सकते हैं कि अब हमें दो बार जोड करने पर हमारा Resultant Time प्राप्त होता है। साथ ही हमें ये भी ध्यान रखना होता है कि जब Minutes की Total 59 से अधिक हो जाए, तब Hours में 1 बढाना है क्योंकि Minutes का मान 59 से अधिक नहीं हो सकता।

How to Get this Ebook in PDF Format

ये पुस्तक केवल **PDF Format Ebook** के रूप में ही Available है और आप इस पुस्तक को केवल हमारी **Official Website** (http://www.bccfalna.com/) से ही खरीद सकते हैं। इसलिए यदि आपको ये पुस्तक पसन्द आ रही है और आप इसे PDF Format Ebook के रूप में खरीदना चाहते हों, तो आप इस पुस्तक को Online खरीदने के लिए निम्नानुसार दिए गए **3 Simple Steps** Follow कर सकते हैं:

Select Purchasing EBooks

सबसे पहले **http://www.bccfalna.com/how-to-pay/** Link पर Click कीजिए। जैसे ही आप इस Link पर Click करेंगे, आप हमारी Website के निम्नानुसार **Order Page** पर पहुंच जाऐंगे :

इस Page पर आपको उन पुस्तकों को Select करना है, जिन्हें आप खरीदना चाहते हैं। आप जैसे–जैसे पुस्तकें Select करते जाऐंगे, आपको उनका **Total Amount, Discount** व **Total Payable Amount** उपरोक्त चित्रानुसार दिखाई देने लगेगा, जहां Total Payable Amount ही वह Amount है, जो आपको अपनी Selected EBooks को खरीदने के लिए **Pay** करना होगा।

पुस्तकें Select करने के बाद इसी Page पर दिखाई देने वाले "**Order Details**" Form में आपको निम्न चित्रानुसार अपना *Name*, *Email Address* व *Mobile Number* Specify करके "**Order Now**" पर Click करते हुए उपरोक्त Selected EBooks का Order Place करना होगा:

चूंकि ये सारी पुस्तकें Physical Books नहीं बल्कि **PDF Format Ebooks** हैं। इसलिए ये पुस्तकें आपको आपके Email पर ही भेजी जाऐंगी, जिन्हें आप अपने Email के माध्यम से अपने Computer पर Download करके अपने PDF Supported *Computer, Mobile, Smart Phone, Tablet PC, Net-Book, Notebook* या *Laptop* जैसी किसी भी Device के माध्यम से पढ सकते हैं अथवा यदि आप चाहें, तो अपने Printer द्वारा इन पुस्तकों का Hard Copy Printout निकाल सकते हैं।

इसलिए जरूरी है कि उपरोक्त "**Order Details**" Form पर आप जो **Email Address** व **Mobile Number** Specify करते हैं, वह Working और एकदम सही हो। क्योंकि किसी भी तरह की परेशानी की स्थिति में हम आपको आपके **Mobile Number** पर ही Contact करते हैं।

Pay "Total Payable Amount"

जैसे ही आप "**Order Now**" Button पर Click करेंगे, आपको एक Email मिलेगा, जिसमें आप द्वारा Order की गई EBooks की Details होगी। Selected पुस्तकों का Order Place करने के बाद अब आपको "**Total Payable Amount**" का Payment करना होगा।

यदि आपके पास **Net-Banking** या **Mobile-Banking** की सुविधा है, तो आप Payment करने के लिए अपने Account में Login करके निम्न में से किसी भी Bank A/c में Payment Deposit कर सकते हैं:

भारतीय स्टेट बैंक
State Bank of India
With you - all the way

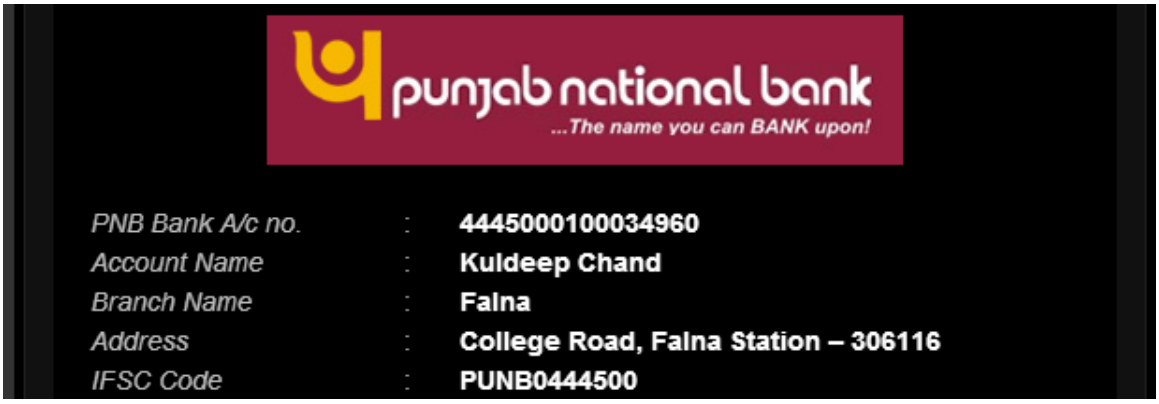
SBI Bank A/c no.	:	**31154882587**
Account Name	:	**Namita Mishra**
Branch Name	:	**Falna**
Address	:	**Near Railway Crossing, Falna Station – 306116**
IFSC Code	:	**SBIN0007868**

बैंक ऑफ़ बड़ौदा
Bank of Baroda
India's International Bank

BOB Bank A/c no.	:	**35260100003212**
Account Name	:	**Namita Sharma**
Branch Name	:	**Falna**
Address	:	**Sanderao Road, Falna, Dist. Pali (Raj.)- Pin-306116**
IFSC Code	:	**BARB0FALNAX**

स्टेट बैंक ऑफ बीकानेर एण्ड जयपुर
State Bank of Bikaner and Jaipur
The Bank with a vision

SBBJ Bank A/c no.	:	**61089986732**
Account Name	:	**Kuldeep Chand Mishra**
Branch Name	:	**Bali**
Address	:	**Sr. Secondary School Road, Bali- 306701**
IFSC Code	:	**SBBJ0010193**

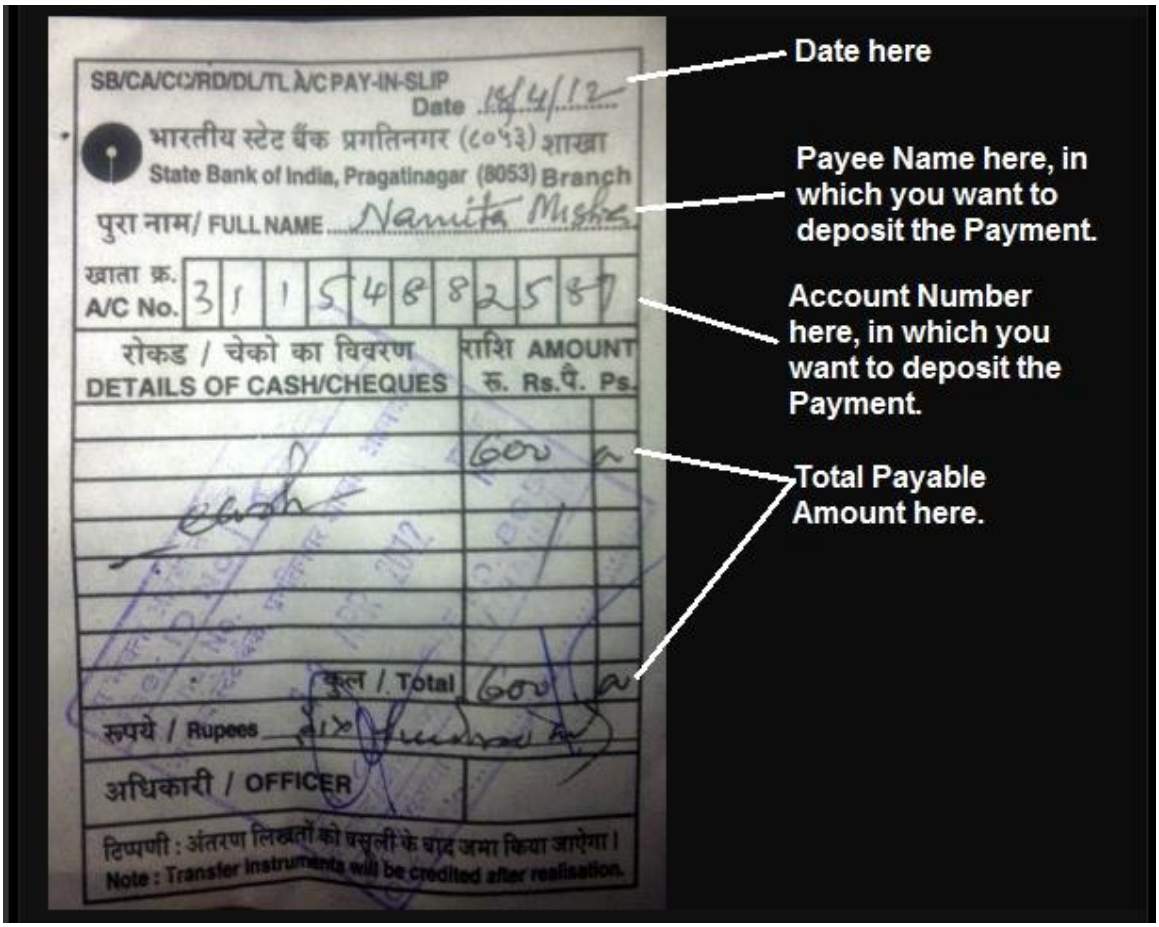


जब आप Net-Banking के माध्यम से Payment करना चाहते हैं, तो आपको लगभग **8 से 24 घण्टे पहले** हमारे उस Account को **Beneficiary** के रूप में अपने Bank A/c से Link करना पडता है, जिसमें आप Payment Deposit करना चाहते हैं।

यदि आपके पास Net-Banking या Mobile-Banking की सुविधा नहीं है, तो आप हमारे किसी भी Bank A/c में *Total Payable Amount*, **Direct Deposit** भी कर सकते हैं।

जब आप **Direct Deposit** करना चाहते हैं, तब आपको आपके किसी भी नजदीकी Bank Branch में जाकर एक *Payment Deposit Slip* Fill-Up करना होता है, जिसमें आपको हमारे किसी भी Bank A/c की Information को Fill करना होता है, **जबकि Payment Deposit करवाने के लिए उसी Bank में आपका स्वयं का Account होना जरूरी नहीं है।**

उदाहरण के लिए यदि आप हमारे SBI Bank A/c में अपनी Selected पुस्तकों का *Total Payable Amount* Pay करने के लिए Bank में जाकर Direct Deposit करना चाहते हैं, तो आप जो **Payment Deposit Slip** Fill-Up करेंगे, वह निम्न चित्रानुसार करना होता है:



इस चित्र द्वारा आप समझ सकते हैं कि Payment, Direct Deposit करने के लिए आपको हमारे किसी Bank A/c की Information को *Payment Deposit Slip* में Specify करना होता है, इसलिए उस Bank में आपका स्वयं का Bank A/c होना जरूरी नहीं होता।

Net-Banking, Mobile-Banking व **Direct Deposit** के अलावा किसी अन्य माध्यम से भी आप Payment कर सकते हैं। उदाहरण के लिए कुछ Banks अपनी ATM Machine द्वारा Direct Payment Transfer करने की सुविधा Provide करते हैं।

यदि आपके Bank का ATM Machine इस तरह से Payment Transfer करने की सुविधा देता है, तो आपको Bank में जाकर Payment Deposit Slip के माध्यम से Payment करने की जरूरत नहीं होती, बल्कि आप Bank के ATM Machine से भी Directly हमारे किसी भी Bank A/c में **Total Payable Amount** Transfer कर सकते हैं। इसी तरह से यदि आप चाहें, तो हमारे किसी भी Bank A/c में Check द्वारा भी Amount Direct Deposit कर सकते हैं।

यानी आप किसी भी तरीके से हमारे किसी भी Bank A/c में *Total Payable Amount* Deposit कर सकते हैं। लेकिन हम **Money-Order, Demand-Draft** या **Check** जैसे Manual माध्यमों से Payment Accept नहीं करते, क्योंकि इस तरह का Payment Clear होने में बहुत समय लगता है। जबकि Direct Deposit या Mobile अथवा Net-Banking के माध्यम से तुरन्त Payment Transfer हो जाता है, जिससे हम आपको आपकी Purchased EBooks **20** से **30 Minute** के दरम्यान आपके Order में Specified **Email** पर Send कर देते हैं।

Confirm the Payment

जब आप अपनी Selected पुस्तकों को खरीदने के लिए उपरोक्तानुसार किसी भी तरीके से *"Total Payable Amount"* हमारे किसी भी Bank A/c में Deposit कर देते हैं, तो Payment Deposit करते ही आपको हमें उसी Mobile Number से एक **Call/Miss Call/SMS** करना होता है, जिसे आपने Order Place करते समय "**Order Form**" में Specify किया था।

इसी Mobile Number के माध्यम से हमें पता चलता है कि आपने किन पुस्तकों के लिए Order किया है और उनका *Total Payable Amount* कितना है। साथ ही हमें ये भी पता चल जाता है कि आप द्वारा Purchase की जा रही पुस्तकें किस Email Address पर Send करनी है।

आपके *Total Payable Amount* को हम Net-Banking के माध्यम से अपने Bank A/c में Check करते हैं और यदि आपका *Total Payable Amount* हमारे किसी भी Bank A/c में Deposit हुआ होता है, तो हम आपको **30 Minute** के दरम्यान आपकी Ordered EBooks आपके Email पर Send कर देते हैं, जिसे आप अगले दिन 12AM तक Download कर सकते हैं।

यदि अभी भी आपको कोई बात ठीक से समझ में न आ रही हो या किसी भी तरह का Confusion हो, तो आप **097994-55505** पर **Call/Miss Call/SMS** कर सकते हैं। यथा सम्भव तुरन्त आपको Callback किया जाएगा और आपकी समस्या या Confusion का Best Possible Solution करने की कोशिश की जाएगी।

उम्मीद है, इस पुस्तक के Sample Chapters का Demo भी आपको पसन्द आया होगा और हमें पूरा विश्वास है कि पूरी पुस्तक आपको और भी ज्यादा पसन्द आएगी।