

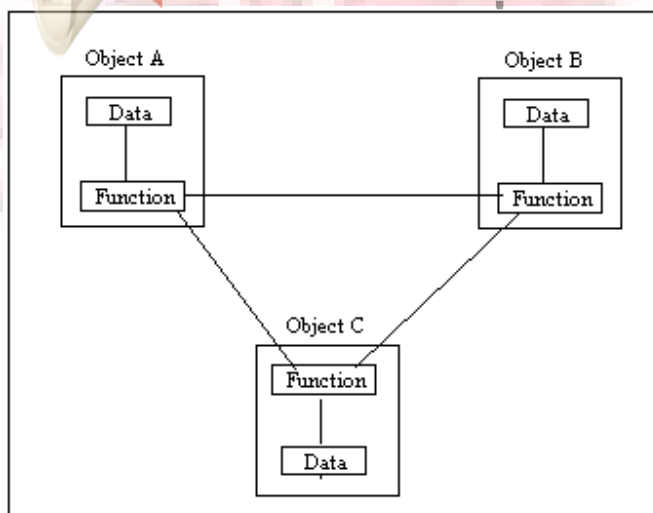
## ऑब्जेक्ट ओरिएन्टेड प्रोग्रामिंग के सिद्धांत

### 1.1 परिचय

उच्च स्तरीय भाषाओं जैसे कोबॉल फोरट्रॉन या सी में लिखे प्रोग्राम प्रोसीजर ओरिएन्टेड प्रोग्राम (Procedure Oriented Program) कहलाते हैं। इस प्रकार की प्रोग्रामिंग में प्रोग्राम द्वारा किये जाने वाले कार्यों को क्रम के रूप में देखा जाता है। इन कार्यों को करने के लिये एक से अधिक फंक्शन लिखे जाते हैं। लेकिन फंक्शन के विकास पर ध्यान केन्द्रित करते समय हम उस डाटा को भूल जाते हैं जिसे इन फंक्शन द्वारा प्रयोग किया जाना है। सामान्य तौर पर प्रोसीजर ओरिएन्टेड प्रोग्राम के गुणों (अवगुणों) को इस तरह निधारित किया जा सकता है।

- प्रोग्राम एल्गोरिदम (Algorithm) पर आधारित होते हैं।
- प्रोग्राम को छोटे-छोटे प्रोग्राम/फंक्शन में विभाजित किया जाता है। सार्वभौमिक डाटा (global data) प्रयोग करते हैं अर्थात् एक ही डाटा बहुत से प्रोग्राम द्वारा प्रयोग किया जाता है।
- डाटा स्वतंत्र रूप से एक से दूसरे फंक्शन में जा सकता है।
- प्रोग्राम डिजाइन के लिये टॉप-डाउन (Top-Down) अवधारणा का प्रयोग किया जाता है।

ऑब्जेक्ट ओरिएन्टेड प्रोग्रामिंग का विकास उक्त समस्याओं से निदान पाने के लिए किया गया है। ऑब्जेक्ट ओरिएन्टेड प्रोग्रामिंग, जिसे संक्षेप में (OOPS) भी कहा जाता है, में फंक्शन के बजाय डाटा को प्रोग्राम के विकास का महत्वपूर्ण भाग माना जाता है। इसमें डाटा, उन फंक्शन के ही साथ जुड़े होते हैं जिन्हें उसे प्रयोगकतना है। इस प्रकार डाटा बाहरी फंक्शनों में प्रयोग तथा उनके द्वारा हो सकने वाले संभावित परिवर्तनों से बचा रहता है। किसी ऑब्जेक्ट ओरिएन्टेड प्रोग्राम में डाटा तथा फंक्शन की संरचना निम्न चित्र से प्रदर्शित होता है।



इस चित्र के आधार पर ऑब्जेक्ट ओरिएन्टेड प्रोग्राम की परिभाषा कुछ इस तरह दी जा सकती है। “ऑब्जेक्ट ओरिएन्टेड प्रोग्रामिंग एक ऐसी अवधारणा है जो डाटा तथा फंक्शन के लिये इस तरह अलग-अलग मेमोरी एरिया का निर्माण कर प्रोग्राम का मॉड्यूलराइजेशन करती है जिससे उन्हें माइक्रूल की कॉपी करने के लिये टैपलेट की जरूरत प्रयोग किया जा सकता है।”

एक ऑब्जेक्ट ओरिएन्टेड प्रोग्राम की निम्न विशेषताएं हो सकती हैं –

- प्रोसीजर के बजाय डाटा पर केन्द्रित होती है।
- प्रोग्राम को ऑब्जेक्ट में विभाजित किया जाता है।
- डाटा संरचना को इस प्रकार डिजाइन किया जाता है कि वे ऑब्जेक्ट परिलक्षित करते हैं।
- डाटा अपने उपयोग किये जाने वाले फंक्शनों के साथ इस प्रकार बंधा होता है कि वह बाहरी अन्य फंक्शन द्वारा प्रयोग नहीं किया जा सकता।
- नये डाटा तथा फंक्शन को कभी भी जोड़ा जा सकता है।
- प्रोग्रामिंग में बॉटम-अप (Bottom-up) अवधारणा का प्रयोग किया जाता है।

## 1.2 ऑब्जेक्ट ओरिएन्टेड प्रोग्राम की मूल अवधारणाएँ

प्रोग्राम के सिद्धांतों में ऑब्जेक्ट ओरिएन्टेड प्रोग्राम का सिद्धांत सबसे नया है। इसके अंतर्गत प्रयुक्त प्रमुख संकल्पनाएं निम्न हैं।

1. ऑब्जेक्ट (Object)
2. क्लासेस (Classes)
3. डाटा एब्स्ट्रैक्शन (Data Abstraction)
4. डाटा एन्सप्यूलेशन (Data Encapsulation)
5. इनहेरिटेन्स (Inheritance)
6. पोलिमॉर्फिज्म (Polymorphism)
7. संदेश प्रसारण (Message Passing)

**ऑब्जेक्ट** : किसी ऑब्जेक्ट ओरिएन्टेड सिस्टम में ऑब्जेक्ट ही सबसे मूल इकाई है। ऑब्जेक्ट ऐसा कोई भी डाटा हो सकता है जो प्रोग्राम द्वारा विकसित तथा निर्धारित विशिष्ट डाटा-टाइप है जो प्रोग्राम में अन्य समान डाटा टाइप की ही भांति प्रयोग होते हैं। डाटा तथा उन्हें प्रयोग करने वाले कोड्स को क्लास की मदद से उपयोगकर्ता द्वारा निर्धारित डाटा टाइप में परिवर्तित किया जाता है।

**क्लास** : ये उपयोगकर्ता द्वारा निर्धारित विशिष्ट डाटा-टाइप है जो प्रोग्राम में अन्य समान डाटा टाइप की ही भांति प्रयोग होते हैं। डाटा तथा उन्हें प्रयोग करने वाले कोड्स को क्लास की मदद से उपयोगकर्ता द्वारा निर्धारित डाटा टाइप में परिवर्तित किया जाता है।

**डाटा एन्कैप्सुलेशन** : डाटा एन्कैप्सुलेशन से अभिप्राय डाटा तथा फंक्शन को एक इकाई के रूप में इकट्ठा करने से है। यह क्लास की एक प्रमुख विशेषता है जिससे डाटा को बाहरी फंक्शनों के द्वारा प्रयुक्त होने से बचाया जा सकता है।

**डाटा एन्कैप्सुलेशन** : वह प्रक्रिया है। जिससे किसी प्रोग्राम के लिये अनुकूल डाटा टाइप का निर्माण किया जाता है। क्लास के अंतर्गत इस अवधारणा का प्रयोग किया जाता है।

**इनहेरिटेन्स** : वह प्रक्रिया जिसके द्वारा किसी क्लास के ऑब्जेक्ट किसी दूसरे क्लास के ऑब्जेक्ट के गुण धर्म धारणा कर सकते हैं। इनहेरिटेन्स कहलाती है इसका अर्थ यह है कि हम किसी पूर्व स्थित क्लास में अन्य विशेषताएं जोड़ सकते हैं

तथा इसके लिये क्लास में परिवर्तन करने की आवश्यकता नहीं है। इसमें पूर्व स्थित क्लास जिसे **base** कहते हैं से एक नयी क्लास जिसे **derived** क्लास कहते हैं का निर्माण किया जाता है। नयी क्लास में **base** तथा **derived** दोनों की विशेषताएं होती हैं।

**पॉलीमॉर्फिज्म** : यह ऑब्जेक्ट ओरिएन्टेड प्रोग्राम की एक प्रमुख विशेषता है जिसके द्वारा एक ऑपरेटर प्रयुक्त डाटा के प्रकार पर निर्भर करता है। निम्न चित्र में यह प्रदर्शित है कि कैसे एक ही फंक्शन को विभिन्न आर्गुमेंट को संभालने के लिये प्रयोग किया जा सकता है। इसका अर्थ ऐसा ही है जैसे किसी एक शब्द का अलग-अलग संदर्भों में अलग-अलग अर्थ है।

**संदेश प्रसारण** : एक ऑब्जेक्ट ओरिएन्टेड प्रोग्राम बहुत से ऑब्जेक्ट से मिलकर बना होता है जो आपस में सूचनाओं का आदान-प्रदान कर सकते हैं इसके अंतर्गत ऑब्जेक्ट का नाम, फंक्शन तथा भेजे जाने वाली सूचना संग्रहित हैं।

### 1.3 ऑब्जेक्ट ओरिएन्टेड प्रोग्रामिंग के प्रयोग

वर्तमान प्रोग्रामिंग में OOPS सबसे ज्यादा प्रचलित शब्द है। इसका सबसे ज्यादा प्रयोग ए बेहतर यूजर इंटरफेस, जैसे विंडो, तैयार करने के लिये होता है। निम्न क्षेत्रों में OOPS का सर्वाधिक प्रयोग किया जा सकता है।

- रियल-टाइम सिस्टम
- सिमुलेशन (Simulation) तथा मॉडलिंग (Modeling)
- ऑब्जेक्ट ओरिएन्टेड डाटाबेस
- हायपर टेक्स्ट, हायपर मीडिया तथा एक्सपर्ट टेक्स्ट (Expert Text)
- समानांतर (Parallel) प्रोग्रामिंग
- ऑफिस स्वचालन सिस्टम
- (CIM/CAM/CAD) सिस्टम

यह विश्वास किया जाता है कि oops सिस्टम की विशेषताओं का प्रयोग न केवल सॉफ्टवेयर की क्वालिटी सुधारने के लिये हो सकता है बल्कि उनकी उत्पादकता बढ़ाने के लिये भी हो सकता है।

## 2.1 परिचय

सी++ एक ऑब्जेक्ट ओरिएन्टेड प्रोग्रामिंग भाषा है जो सी का ही एक विकसित रूप है। इसका आविष्कार AT&T बेल लेबोरेटरीज में Bjarne Stroustrup द्वारा किया गया। चूंकि यह सी का ही एक विस्तृत रूप है अतः सी के अधिकतर प्रोग्राम C++ में भी चलाये जा सकते हैं।

चूंकि सी++ एक ऑब्जेक्ट ओरिएन्टेड भाषा है अतः बड़े प्रोग्राम को संभालने में यह विशेष उपयोगी है। यह मुख्यतः एडिट, कंपाइलर, डाटाबेस, संचार तंत्र तथा उपयोगी यूजर इंटरफेस निर्माण करने के लिये प्रयोग हो सकती है।

इसके जरिये विशिष्ट ऑब्जेक्ट ओरिएन्टेड लायब्रेरी तैयार की जा सकती है जिन्हें बाद में अन्य प्रोग्राम तंत्र द्वारा प्रयोग किया जा सकता है।

## 2.2 C++ प्रोग्राम

सी की ही तरह C++ प्रोग्राम भी फंक्शनों का एक संग्रह है। सभी प्रोग्राम का क्रियान्वन `main()` फंक्शन से प्रारंभ होता है। यह एक **free-form** भाषा है। सी की ही तरह सभी आदेशों की समाप्ति एक सेमी कॉलन (;) से होती है। C++ में टिप्पणियों के लिये **double slash**(//) का प्रयोग होता है। कोई भी टिप्पणी एक (//) के बाद लिखी जाती है तथा टिप्पणी की समाप्ति के पश्चात् पुनः एक(//) प्रयुक्त होता है। हालांकि सी का पारंपरिक रूप /\* भी प्रयोग किया जा सकता है। सामान्य तौर पर एक C++ प्रोग्राम में निम्न खण्ड होते हैं।

Include Files
Class Declaration
Class Functions Definitions
Main Functions Program

## 2.3 टोकन (Token)

किसी प्रोग्राम में सबसे छोटी इकाई टोकन कहलाती है सी++ में निम्न टोकन होते हैं—

- की-वर्ड (Key-word)
- आइडेन्टीफायर (Identifier)
- स्थिरांक (String)
- ऑपरेटर (Operator)

**की-वर्ड :** की-वर्ड का अर्थ सी++ भाषा के विशिष्ट शब्द से है। ये विशिष्ट शब्द वे हैं जिन्हें प्रोग्रामिंग वरियेबल की तरह प्रयोग नहीं किया जा सकता है। इस शब्दों के विशिष्ट अर्थ कंपाइलर द्वारा ही प्रयोग किये जा सकते हैं।

निम्न तालिका में सी++ के विशिष्ट शब्द प्रदर्शित हैं।

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

**आइडेन्टीफायर :** इसका अर्थ उन वेरियेबल, फंक्शन या अरे के नामों से है जो प्रोग्रामर अपनी सुविधा या आवश्यकतानुसार प्रोग्राम में प्रयोग करते हैं। निम्न नियम सी++ में प्रयुक्त होते हैं।

- वेरियेबल फंक्शन या अरे नामों के लिये सिर्फ अक्षर, अंक या एक विशिष्ट चिन्ह **underscore** ही प्रयोग किया जा सकता है।
- नाम का प्रारंभ सिर्फ छोटे अक्षर से ही हो सकता है।
- अंग्रेजी के बड़. और छोटे अक्षरों में लिखे नाम अलग-अलग प्रयुक्त होते हैं।

- किसी विशिष्ट की-बोर्ड को वेरियेबल नाम की जगह प्रयुक्त नहीं किया जा सकता है।
- सी तथा सी++ में एक विशिष्ट फर्क है कि सी में खासकर ANSI C में वेरियेबल नाम के लिये अधिकतम 32 कैरेक्टर ही प्रयोग किये जा सकते हैं जबकि सी++ में ऐसी कोई सीमा निर्धारित नहीं है।

## 2.4 प्रमुख डाटा प्रकार

जैसे पहले कहा गया है कि सी++ में एक वेरियेबल, फंक्शन या अरे प्रयोग किये जाते हैं जो विभिन्न प्रकार का डाटा संभाज सकते हैं।

सी++ में प्रयुक्त डाटा, उनके लिये आवश्यक जगह तथा उनके अंतर्गत प्रयुक्त किये जा सकने वाला डाटा की अधिकतम सीमा निम्न तालिका में प्रदर्शित है।

Type	Byte	Range
char	1	-128 to 127
unsigned char	1	0 to 255
signed char	1	-128 to 127
int	2	-32768 to 32767
unsigned int	2	0 to 65535
signed int	2	-32768 to 32767
short int	2	-32768 to 32767
unsigned short int	2	0 to 65535
signed short int	2	-32768 to 32767
long int	4	-2147483648 to 2147483647
signed long int	4	-2147483648 to 2147483647
unsigned long int	4	0 to 4294967295
float	4	3.4E - 38 to 3.4E + 38
double	8	1.4E - 308 to 1.7E + 308
long double	10	3.4E - 4932 to 1.1E + 4932

सी भाषा में सभी वेरियेबल प्रोग्राम के प्रारम्भ में ही परिभाषित किये जाते हैं जबकि सी++ में इन वेरियेबल को उसके वास्तविक प्रयोग से पहले कहीं भी परिभाषित किया जा सकता है।

### संदर्भ - वेरियेबल (Reference Variable)

सी ++ में वेरियेबल का एक नया प्रकार संदर्भ – वेरियेबल भी प्रयोग किया जाता है। यह संदर्भ वेरियेबल किसी पहले से प्रयुक्त वेरियेबल के लिये एक अन्य वैकल्पिक नाम उपलब्ध करता है। जैसे यदि एक वेरियेबल का नाम

sum है तथा हमने एक अन्य वैकल्पिक वेरियेबल **TOTAL** तैयार किया है इन दोनों वेरियेबल को अदल-बदल कर प्रयोग कर सकते हैं। एक संदर्भ वेरियेबल निर्धारित करने के लिये सामान्य प्रारूप है-

**Data type & reference-name=Variable name**

जहां **Variable name** पहले से निर्धारित किसी वेरियेबल का नाम है जबकि **reference-name** वह नया नाम है जो हम उस वेरियेबल को देना चाहते हैं। जैसे –

Float total = 100;

Float & sum = total;

यहां **sum** तथा **total** दोनों ही वेरियेबल का मान 100 हो जाएगा तथा किसी एक में ही परिवर्तन करने पर दूसरे का मान स्वतः परिवर्तित हो जाएगा।

संदर्भ - वेरियेबल को प्रयोग करने से पहले प्रारंभिक मान देना आवश्यक है। इस प्रकार के वेरियेबल को सबसे अधिक प्रयोग किसी फंक्शन में आर्गुमेंट प्रविष्ट कराने के लिये होता है।

## 2.5 सी++ में ऑपरेटर

सी भाषा में प्रयुक्त सभी ऑपरेटर सी++ में भी वैसे ही प्रयोग किया जा सकते हैं। इसके अलावा सी++ में कुछ नये ऑपरेटर भी प्रयुक्त होते हैं। हम पहले इन नये ऑपरेटर की कुछ चर्चा करते हैं।

. <<

Insertion put to operator

>>

Extraction get from operator

::

Scope resolution operator

::\*

Pointer- to-member declarator

-> \*

Pointer-to-member operator

. \*

Pointer-to-member operator

Delete

Memory release operator

Ndl                      Line feed operator

New                      Memory allocation operator

Stew                      Field width operator

इसके अलावा सी ++ में कई पूर्व निधारित (Built-in) ऑपरेटर को एक से अधिक अर्थ भी प्रदान किये जा सकते हैं। इसे ऑपरेटर ओवरलोडिंग (Operator overloading) कहते हैं।

**Scope resolution operator :** हमने सी भाषा के दौरान यह देखा कि जो वेरियेबल जिस ब्लॉक के अंतर्गत परिभाषित किया गया है वह वही प्रयोग किया जा सकता है। किसी अंदर के ब्लॉक से किसी वेरियेबल को सार्वभौमिक (global) मान नहीं प्रदान किया जा सकता है न ही ऐसे किसी वेरियेबल को प्रयोग किया जा सकता है। इसके लिये सी++ में एक महत्वपूर्ण ऑपरेटर :: प्रयुक्त होता है। इसका सामान्य प्रारूप है।

:: Variable name

जैसे :: Count

यह आदेश count के ग्लोबल प्रारूप का प्रयोग करेगा।

**Member deferencing operator :** जैसा कि हम जानते हैं सी++ के जरिये हम कोई क्लास परिभाषित कर सकते हैं जिसमें कई प्रकार का डाटा तथा फंक्शन होता है। इस क्लास के सदस्य (class members) को पाइन्टर के जरिये प्राप्त कर सकते हैं। इसके लिये निम्न तीन ऑपरेटर प्रयुक्त होते हैं।

::\*                      किसी क्लास में सदस्य के लिये पॉइंटर के निधारित करना।

.\*                      ऑपरेटर नाम तथा पॉइंटर के जरिये मेम्बर को पढ़ना।

->\*                      किसी ऑब्जेक्ट के लिये तथा किसी सदस्य के लिये पाइन्टर के जरिये उस सदस्य का मान पढ़ना।

**मेमोरी प्रबंधक ऑपरेटर :** मेमोरी के आवंटन या मेमोरी को स्वतंत्र कराने के लिये C++ के अन्तर्गत दो विशिष्ट ऑपरेटर New तथा delete प्रयुक्त होते हैं। new ऑपरेटर के जरिये किसी प्रकार के ऑब्जेक्ट का निर्माण किया जा सकता है। इसका सामान्य प्रारूप है।

Pointer-Variable=new data type:



जहां **pointer-variable** किसी निर्धारित प्रकार वाला पॉइन्टर हैं। यहाँ **new** ऑपरेटर उस निर्धारित प्रकार वाले डाटा के लिये समुचित मेमोरी निर्धारित करेगा। उदाहारण स्वरूप-

```
p=new int;  
q=new float;
```

जहां **p** एक इंटीजर प्रकार का पॉइन्टर है जबकि **q** फ्लोट प्रकार का वेरियेबल है। यहां यह ध्यान रखना आवश्यक है कि **p** तथा **q** को पहले से ही परिभाषित किया जाना आवश्यक हैं।

ऑब्जेक्ट निर्माण के साथ साथ न्यू ऑपरेटर का प्रयोग मेमोरी को प्रारंभिक मान देने (**initialization**) के लिये भी किया जा सकता है। इसका सामान्य प्रारूप हैं।

**Pointer-Variable =new data type (value);**

जहां कि **value** कोई प्रारंभिक मान है। जैसे

```
int P*= new int (25)
```

**new** का प्रयोग किसी विशिष्ट डाटा-प्रकार के लिये मेमोरी निर्मित करने के लिये भी हो सकता है। इसका सामान्य प्रारूप है-

**Pointer-Variable =new data type (size);**

जहाँ कि **size** उस अरे में अवयवों की संख्या है। जैसे

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

सह आदेश 10 इंटीजर मानों के लिये अरे का निर्माण करेगा।

**new** के विपरीत **delete** का प्रयोग किसी अनावश्यक डाटा ऑब्जेक्ट को हटाने के लिये होता है इसका सामान्य प्रारूप है -

**delete Pointer-variable**

जैसे **delete p;** या **delete q;**

इसका प्रयोग निम्न प्रकार से भी सकता है

delete (size) Pointer-variable

जहां (size) अरे में अवयवों की संख्या है।

इसी प्रकार delete[] p;

यह आदेश संपूर्ण अरे को जो p द्वारा इंगित (pointed) है, मिटा देगा।

**Manipulators:** ये वे ऑपरेटर हैं जिनका प्रयोग डाटा के प्रदर्शन के प्रकार को नियंत्रित करता है। इसके लिये सबसे ज्यादा प्रचलित manipulator endl तथा setw है। endl के प्रयोग से आउटपुट में एक blank line जोड़ी जा सकती है। इसका प्रभाव “\n” जैसा ही है।

Setw का प्रयोग प्रत्येक आउटपुट फील्ड के लिये एक निश्चित चौड़ाई निर्धारित करने के लिये होता है। setw(5) जैसे प्रत्येक फील्ड को पांच कैरेक्टर में प्रिंट करेगा।

किसी एक समीकरण में एक से अधिक ऑपरेटर प्रयुक्त किये जा सकते हैं। ऐसी स्थिति में उनके क्रियान्वयन का क्रम निर्धारित होता है निम्न तालिका में ऑपरेटर का क्रम प्रदर्शित है।

operator	Associativity
::	left to right
-> . ( ) [ ] postfix ++ postfix--	left to right
Prefix ++ prefix -- ~ / unary + unary -	right to left
Unary * unary & (type) size of new delete	
-> **	left to right
* /%	left to right
+ -	left to right
<<>>	left to right
< <= > >=	left to right
== /=	left to right
&	left to right
^	left to right
	left to right
&&	left to right

	left to right
?:	left to right
= *//= %= += -=	right to left
<< = >> =&= ^ /=	left to right

1. sequence structure (straight line)
2. selection structure (branching)
3. loop structure (iteration or repetition)



## इनपुट एवं आउटपुट प्रक्रियाएं

### परिचय

प्रत्येक प्रोग्राम के क्रियान्वयन के लिये कुछ इनपुट की आवश्यकता पड़ती है जबकि प्रत्येक प्रोग्राम, क्रियान्वयन के पश्चात् कुछ आउटपुट प्रस्तुत करता है। अतः सह जानना आवश्यक है कि इनपुट डाटा कैसे उपलब्ध किता जा सकता है। पुर्व में हम >> तथा << ऑपरेटर के प्रयोग पढ चुके हैं। सी++ के अंतर्गत कई समृद्ध इनपुट/आउटपुट व्यजक उपलब्ध हैं। सी++ अंतर्गत stream तथा stream classes की परिकल्पना प्रयोग की जाती है।

### अनफार्मेटड इनपुट / आउटपुट प्रक्रियाएं

सी++ के अंतर्गत डाटा इनपुट तथा आउटपुट के लिये दो विशिष्ट आदेश प्रयोग किये जाते हैं। ये हैं CIN (इसे सी-इन पडा जाता है) तथा cout (इसे सी आउट पडा जाता है) इन आदेशों का प्रयोग तथा ऑपरेटर के साथ किया जाता है।

CIN आदेश का प्रयोग डाटा इनपुट के लिये होता है। इसका सामान्य प्रारूप है—

CIN>> variable>>>variable2>>.....>>variableN

Penebb variable, variable2 आदि सी++ में प्रयुक्त वैध नाम हैं जिन्हें पहले से घोषित किया जाना आवश्यक है। यह आदेश कम्प्यूटर की अन्य प्रक्रियाओं को बंद कर देगा तथा की.बोर्ड से डाटा प्राप्ति का इंतजार करेगा। इनपुट किये जाने वाले डाटा का प्रारूप निम्न होना चाहिए।

data1 data2..... data n

इनपुट किये जाने वाला डाटा एक दूसरे से एक या अधिक स्पेस के द्वारा अलग किया हुआ होना चाहिए तथा उसके प्रकारों का क्रम वही होना चाहिए जो CIN आदेश के साथ प्रयुक्त वरियेबल के प्रकारों का है।

आदेश का क्रियान्वयन होने पर ऑपरेटर प्रत्येक डाटा का एक-एक कैरेक्टर पढता है तथा उसे सम्बद्ध वेरियेबल दे देता है।

उदाहरण —            `int code;`  
                         `C in >> code;`

डाटा को आउटपुट करने के लिये आदेश का प्रयोग होता है। इसका सामान्य प्रारूप है—

`Cout<<item1<<item2<<....<<item n;`

जहाँ आदिइ विभिन्न वेरियेबल या स्थिरांक है। ये स्पेस द्वारा एक दूसरे से अलग किये होते है।

उदाहरण—            `cout<<"H"=<<no;`  
या                      `cout<<H1<<H2<<H3;`

**put() तथा get() व्यंजक :** इन दो व्यंजको का प्रयोग एक कैरेक्टर को इनपुट करने या प्रिंट करने के लिये होता है।

**get()** व्यंजक का प्रयोग इनपुट ग्रहण करने के लिये तथा **put()** व्यंजक का प्रयोग डाटा प्रदर्शित करने के लिये होता है।

**get()** व्यंजक दो प्रकार से प्रयुक्त होता है। एक तो **get(char\*)** तथा दूसरा **get(void)**। पहले प्रकार का प्रयोग इनपुट ग्रहण कर उसे आर्गुमेंट वेरियेबल को देने के लिये होता है जबकि दूसरा प्रकार इनपुट फंक्शन का मान प्रोग्राम को देता है।

उदाहरण —            `char c;`  
                         `cin.get(c); //get a character from keyboard and`  
                         `//assign it to c.`

यदि इस फंक्शन का उपयोग पूरा टैक्स्ट पढ़ने के लिये करना है तो इसे किसी लूप के अंतर्गत प्रयोग करना होगा। **get(void)** का प्रयोग निम्न प्रकार से हो सकता है

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

प्रोग्राम के क्रियान्वयन के दौरान **get()** फंक्शन का मान C वेरियेबल को दे दिया जाएगा।

**put()** फंक्शन का प्रयोग कैरेक्टर को स्क्रीन पर प्रदर्शित करने के लिये होता है। जैसे —

`cout.put('X');`  
या  
`cout.put(ch);`

पहला उदाहरण 'X' प्रिंट करेगा जबकि दूसरा उदाहरण ch नामक वैरियेबल का मान प्रिंट करेगा। लेकिन इस वैरियेबल का कोई कैरेक्टर मान होना आवश्यक है।

लूप का प्रयोग कर put() फंक्शन से भी एक से अधिक कैरेक्टर प्रिंट किये जा सकते हैं।

**getline() तथा write() फंक्शन :** की बोर्ड से एक पूरी लाइन का इनपुट के रूप में ग्रहण करने सा प्रिंट करने के लिये getline() तथा write() फंक्शन का प्रयोग किया जा सकता है। getline() फंक्शन टेक्स्ट की एक पूरी लाइन को जो कि '\n' पर समाप्त होती है पढ़ना है। उदाहरण के लिये—

```
char name[20];  
cin.getline(name,20);
```

मान लीजिये हमने की-बोर्ड से निम्न इनपुट दिया है।

Madhya Pradesh

तो यह इनपुट सही तरह से पढ़ लिया जाएगा तथा यह मान नामक अरे को दे दिया जाएगा। लेकिन यदि इनपुट निम्न दिया गया है —

All India Society for Electronics and Computer Technology तो इनपुट केवल पहले 19 कैरेक्टर स्वीकार कर रुक जाएगा।

All India Society for E

हालांकि हम CIN आदेश का प्रयोग कर भी इनपुट ग्रहण कर सकते हैं लेकिन CIN केवल वही इनपुट ग्रहण कर सकता है जिनके मध्य स्पेस न हो।

निम्न प्रोग्राम में getline() व्यंजक तथा ऑपरेटर का प्रयोग प्रदर्शित है।

```
#include<iostream.h>  
  
main()  
{  
  
int size = 20  
char city [20]  
  
cout << "Enter city name:\n";  
  
cin>> city;  
  
cout<<"city name is : "city "\n";
```

```

cour<<"Enter city name again:\n";

cin.getline (city, size);

cout << "city name now: "city <<"\n\n";

```

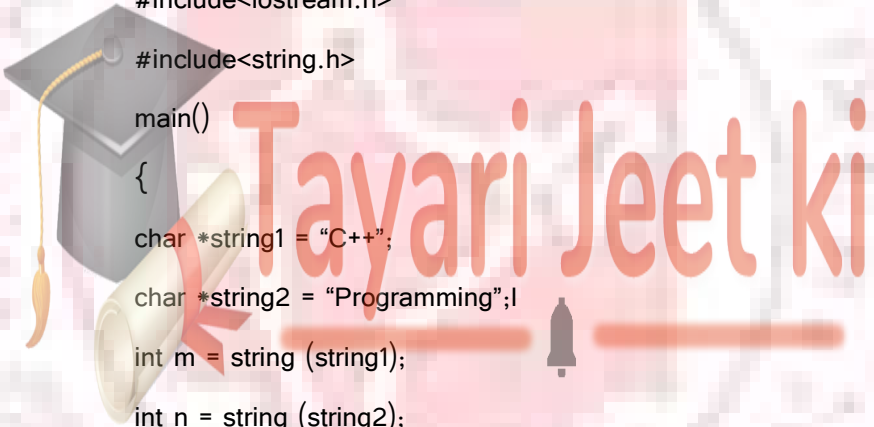
**getline()** फंक्शन के विपरीत **write()** फंक्शन का प्रयोग एक संपूर्ण लाइन को प्रदर्शित करने के लिये होता है। तथा इसका सामान्य प्रारूप है—

```

cout.write (line, size);

```

यहां **line** वह टेक्स्ट या स्ट्रिंग का नाम है जिसे प्रदर्शित करना है तथा **size** उस स्ट्रिंग की अधिकतम कैरेक्टर संख्या है। यहां यह ध्यान रखना आवश्यक है कि कैरेक्टर का प्रदर्शन चही नहीं रुक जहां कोई **null character** उपलब्ध होता है। निम्न प्रोग्राम में **write()** फंक्शन का प्रयोग कर एक उपयोगी प्रोग्राम लिखा गया है।



```

#include<iostream.h>
#include<string.h>
main()
{
char *string1 = "C++";
char *string2 = "Programming";
int m = string (string1);
int n = string (string2);
for (int i=1; i<n; i++)
{
cout.write (string2, i); cout << "\n";
}
for (i=n; i>0; i--)
{
cout.write (string2, i); cout <<"\n";
}
cout <<"\n";
cout.write (string, 10);
}

```

प्रोग्राम की अंतिम लाइन `cout.write (string, 10)` यह प्रदर्शित करता है कि इसके द्वारा `string1` में उपलब्ध कैरेक्टर से भी अधिक कैरेक्टर पढ़े जा सकते हैं।

## फॉर्मेटेड इनपुट/आउटपुट प्रक्रियाएं

सी++ के अंतर्गत आउटपुट को वांछित प्रारूप में प्रदर्शित करने के लिये कई विकल्प उपलब्ध हैं। उनमें से कुछ प्रमुख की यहां चर्चा की गई है।

**width() फंक्शन :** इस फंक्शन का प्रयोग किसी आउटपुट फील्ड के लिये फील्ड का वांछित मान निर्धारित करने के लिये होता है। इसका सामान्य प्रारूप है—

```
cout.width(w);
```

जहां कि `w` फील्ड का वांछित मान है। जैसे

```
cout.width(5);
```

यह ध्यान रखना आवश्यक है कि `width()` फंक्शन केवल एक ही रिकार्ड का फील्ड निर्धारित कर सकता है। अगले रिकार्ड के लिये फिर से `width()` फंक्शन का प्रयोग करना पड़ेगा। यह व्यंजक निम्न प्रोग्राम के माध्यम से समझाया गया है।

```
#include<iostream.h>
main()
{
    int item [4] = (10,8,12,15);
    int cost [4] = (115,100,60,99);
    cout.width [5];
    cout << "items";
    cout.width [8];
    cout << "cost";
    cout.width [15];
    cout << "Total value" <<"\n";
    int sum =0;
```



```

for (int i=0; i=4; i++)
{
    cout.width (5); cout <<item [i];
    cout.width (8); cout <<cost [i];
    int value = items [i] * cost [i];
    cout.width (15); cout <<value "\n";
    sum = sum + value;
}
cout <<"\n" Grand Total = "/n";
cout.width (2); cout <<sum << "\n";

```

**Precision() फंक्शन** : इस फंक्शन का प्रयोग फ्लोटिंग पाइंट अंकों को प्रिंट करते समय दशमलव चिन्ह की स्थिति को निर्धारित करने के लिये होता है। इसका सामान्य प्रारूप निम्न है। –

```
cout.precision(d)
```

जहां d: दशमलव बिन्दु के दायें तरफ अंकों की संख्या है। जैसे

```

cout.precision(3)
cout<<sqrt(2);
cout<<3.14159;
cout<<2.50032;

```

इन आदेशों के आउटपुट निम्न होंगे।

```

1.141
3.142
2.5

```

यह ध्यान रखना चाहिए कि precision फंक्शन द्वारा किये गये निर्धारण तब तक क्रियाशील रहते हैं जब तक कि उन्हें दुबारा निर्धारित न किया जाए।

**Fill() फंक्शन:** यदि हम किसी ऐसे मान को प्रिंट कर रहे हैं जिसकी फील्ड उसके लिये आवश्यक चौड़ाई से गहुत बड़ी है तो हम शेष उपयुक्त जगह को किसी भी वांछित कैरेक्टर से भर सकते हैं। इसके लिये Fill() फंक्शन का प्रयोग किया जाता है।

इसका सामान्य प्रारूप है –

```
cout.fill(ch);
```

यहाँ ch वह कैरेक्टर है जिसका प्रयोग उपयुक्त स्थानों को भरने के लिये होता है जैसे –

```
cout.fill('*');  
cout.width(10);  
cout<<5250<<"\n";
```

इसका आउटपुट होगा –

```
*****5250
```

यह आदेश भी तब स्थापित रहता है जब तक कि इसे परिवर्तित न किया जाए।

**Setf() फंक्शन:** हमने देखा कि जब कोई टेक्स्ट या अंक की प्रिंटिंग width() व्यंजक के साथ की जाती है तो यह हमेशा Right justified रहता है लेकिन वास्तव में टेक्स्ट को हमेशा left justified लिखा जाता है। इस कार्य के लिये तथा फ्लोटिंग पाइंट अंको को घातांक के रूप में व्यक्त करने के लिये setf() (dset flag) फंक्शन का प्रयोग किया जाता है। इसका सामान्य प्रारूप है। –

```
cout.setf(arg1, arg2);
```

यहाँ arg1 कोई flag है जो ios क्लास में पूर्व निर्धारित होता है। तथा arg2 जो कि bit field होता है यह सूचित करता है कि arg1 किस समूह के अंतर्गत है। उदाहरण स्वरूप

```
cout.setf(ios::left, ios::adjusted);  
cout.setf(ios::scientific, ios::floatified);
```

निम्न तालिका में आवश्यक प्रारूप तथा उसके लिये आवश्यक flag तथा बिट फील्ड का मान प्रदर्शित किया गया है। यह ध्यान रखना आवश्यक है कि पहले आर्गुमेंट को दूसरे आर्गुमेंट का ग्रुप-मैम्बर होना आवश्यक है।

आवश्यक प्रारूप	फ्लैग arg1	बिट-फील्ड arg2

लेफ्ट-जस्टिफाइड	ios::left	ios::adjusted
राइट : जस्टिफाइड	ios::Right	ios::adjusted
चिन्ह या बेस के पश्चात् पेडिंग	ios::internal	ios::adjusted
वैज्ञानिक संकेत	ios::scientific	ios::floatified
फिक्स्ड पाइंट संकेत	ios::fixed	ios::floatified
दशमलव आधार	ios::dec	ios::basified
ऑक्टेल् आधार	ios::oct	ios::basified
हेक्साडेसीमल आधार	ios::hex	ios::basified

एक प्रोग्राम के निम्न हिस्से को देखते हैं।

```
Cout.fill ("*");
Cout.setf (ios:: left, ios :: adjusted);
Cout.width (15);
Cout <<"TABLE1"<</n";
```

इसका आउटपुट निम्न होगा।

```
TABLE1 *****
```

ऊपर प्रदर्शित तालिका के अलावा कुछ ऐसे फ्लैग भी प्रयुक्त होते हैं। जिनके साथ कोई बिट फील्ड नहीं होता उन्हें निम्न तालिका में प्रदर्शित किया गया है।

फ्लैग	उपयोग
IOS::showbase	आउटपुट में
IOS::showpose	धनात्मक संख्याओं के सामने चिन्ह प्रिंट करना
IOS::showpoint	trailing दशमिक चिन्ह तथा शून्य प्रदर्शित करना।
IOS::uppercase	हेक्स-आउटपुट में अंग्रेजी के बड़े अक्षर प्रयोग करना

अब एक प्रोग्राम देखते हैं जिसमें इनमें से कुछ `setf ()` विकल्पों को प्रयोग किया गया है।

```
#include <iostream.h>

#include <math.h>

Main ()
{
    Cout.fill ('*');
    Cout.setf (IOS::left, ios:: adjusted);
    Cout.width (10);
    Cout <<"value";
    Cout.setf (10&:: right, 10s:: adjusted);
    Cout.width (15);
    Cout <<"sort of value" <<"/n";
    Cout.fill ('*');
    Cout.precision (4);
    Cout.setf (ios:: showpoint);
    Cout.setf (ios:: showpos);
    Cout.setf (ios:: fixed, ios :: floatfied);
    For (intn=1; n10; n++)
    {
        Cout.setf (10s :: internal, 10s::adjusted);
        Cout.width (5);
        Cout <<"/n";
        Cout.setf (IOS:: right, 10& :: adjusted);
        Cout.width (20);
        Cout sqrt (n) <<"/n";
    }
```

```
Cout.setf (10&::scientific, IOS:: floatfied);
Cout << "/n SQRT (100)=" << sqrt (100) << "/n";
```

प्रोग्राम का आउटपुट निम्न होगा।

Value \*\*\*\*\* SQRT OF VALUE

```
+...1...+ 1.0000
+...2...+ 1.4142
+...3...+ 1.7321
+...4...+ 2.0000
+...5...+ 2.2361
+...6...+ 2.4495
+...7...+ 2.6458
+...8...+ 2.8284
+...9...+ 3.0000
+...10...+ 3.1623
SQRT (100)= + 1.0000e+01
```

## आउटपुट प्रबंधन के लिये **Manipulators** का प्रयोग

सी++ में प्रयुक्त हेडर फाइल `iomanip.h` के अंतर्गत फंक्शन का एक समुच्चय प्रयुक्त होता है जिसे **manipulators** कहते हैं। इनका प्रयोग आउटपुट के प्रारूप को नियंत्रित करने के लिये होता है। ये भी फ्लैग के ही समकक्ष हैं।

Manipulator	प्रयोग	समकक्ष आदेश
Setw (intw)	फील्ड की चौड़ाई निर्धारित करना	width ()
Setprecision (intd)	दशमलव चिह्न की स्थिति निर्धारित करना	precision ()
Setfill (int c)	अप्रयुक्त स्थान को भरने के लिये	fill ()

Setiosflags (long f)	फॉर्मेट फ्लेग के निर्धारण के लिये	seft ()
Resetiosflag(long f)	फॉर्मेट फ्लेग के पुर्ननिर्धारण के लिये	unseft ()
Dend 1	नई लाइन प्रविष्ट कर कर तथा stream को “/n” Keeueer (flush) करने के लिये।	

---

उदाहरण- `cout set (10) 12345;`

इस आदेश के आउटपुट का प्रारूप निम्न तरह से परिवर्तित किया जा सकता हैं।

`Cout <<set (10) setiosflags (ios:: left) << 12345;`

अपनी सुविधानुसार हम कुछ अन्य manipulator स्वयं भी परिभाषित कर सकते है। इसका सामान्य प्रारूप है।-

Ostream & manipulator (ostream & output)

```
{
.....
.....
.....
Return output;
}
```

जहाँ manipulator उस manipulator का नाम है जिसका निर्माण करना है।

जैसे-

Ostream & unit (ostream &output)

```
{
Output “inches”;
Return output;
}
```

यह आदेश, यूनिट नामक एक Manipulator का निर्माण करेगा जो “inches” प्रिंट करेगा। यदि आदेश इस तरह लिखा जाए-

Cout <<36<<unit

तो आउटपुट “36inches” होगा।



## 4.1 परिचय

सी++ भाषा में प्रोग्रामिंग के दौरान किसी विशेष परिस्थिति के अनुसार कोई निर्णय लेने के लिये या किसी प्रक्रिया को दोहराने के लिये तीन कंट्रोल संरचनाएं प्रयुक्त होती हैं ये हैं

- (a) (Sequence Structure)
- (b) (Selection Structure)
- (c) (Loop Structure)

उक्त तीनों संरचनाओं को फ्लोचार्ट के रूप में निम्न चित्र में प्रदर्शित किया गया है।

## 4.2 निर्णय प्रक्रिया के लिये if आदेश

निर्णय प्रक्रिया के लिये if आदेश दो प्रकार से प्रयोग होता है।



If आदेश

If...else आदेश

पहले प्रकार का सामान्य प्रारूप कुछ इस प्रकार है।

If (expression)

```
{  
Action 1;  
}  
Action 2;  
Action 3;
```

यदि if के साथ लिखे समीकरण का मान सत्य है तो action 1 अंतर्गत लिखे आदेशों का क्रियान्वयन होता अन्यथा action2, action3, आदेशों का क्रियान्वयन होता है। यहां यह ध्यान रखना आवश्यक है कि action 1 का क्रियान्वयन होने के बाद भी action2, action3, का क्रियान्वयन अवश्य होता है।

दूसरे प्रकार के if .... else संरचना का सामान्य प्रारूप है -

If (expression)

```
{  
Action 1;  
}  
Else  
{  
Action2;  
}  
Action3
```

इस प्रकार यदि if के साथ लिखे समीकरण का मान सत्य है तो action1 क्रियान्वित होता है। उसके पश्चात् action2 को छोड़कर action3 क्रियान्वित होने लगता है जबकि समीकरण को असत्य होने के स्थिति में action1 का क्रियान्वयन नहीं होता है तथा action2 एवं action3 का क्रियान्वयन है।

अब कुछ उदाहरण देखते हैं जिनमें if.... Else संरचनाओं का प्रयोग हुआ है।

```

Include <iostream.h>

Void main ()
{
Int x;

Cout <<"enter number";

Cin >> x;

If (x>100)
{
Cout<<" The number"<< x;
Cou <<"Is greater than 100";
}
}

```

उसी प्रोग्राम को if .... Else संरचना के साथ इस प्रकार लिखा जा सकता है।

```

# include <iostream.h>
Void main ()
{
Int x;
Cout "enter number";
Cin>> X;
It (x >100)
{
Cout << "The number"<<x;
Cout <<is greater than 100";
Else
Cout << "the number" <<x;
Cout << "is not greater than";
}
}

```

### 4.3 एक से अधिक निर्णयों के लिये **switch** आदेश

यदि किसी प्रोग्राम में एक से अधिक निर्णय प्रक्रियाएं हैं तथा वे सभी किसी एक वेरियेबल के मान पर आधारित हैं तो इसके लिये **switch** आदेशका प्रयोग किया जा सकता है। वेरियेबल के विभिन्न मानों के आधार पर कंट्रोल इलग अलग **case** आदेशों पर स्थानांतरित होता रहता है। इसका सामान्य प्रारूप है-

```
Switch (expression)
```

```
{
```

```
Case1:
```

```
{
```

```
Action1;
```

```
}
```

```
Case2:
```

```
{
```

```
Action2;
```

```
}
```

```
Case3:
```

```
{
```

```
Action3;
```

```
}
```

```
Default
```

```
{
```

```
Action4
```

```
}
```

```
}
```

```
Action5
```

यहाँ (expression) के विभिन्न मानों के अनुरूप action1, action2 या action3 क्रियान्वित होते हैं। तीनों में से कोई न होने पर action4 क्रियान्वित होता है तथा सबसे अंत में action5 क्रियान्वित होता है। Expression का मान किसी वेरियेबल का मान या समीकरण से प्राप्त कोई गणना हो सकती है। एक प्रोग्राम देखते हैं।

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

```
Void main ()
```

```
{
```

```
Int n
```

```
Cout <<"enter n";
```

```
Cin >>n;
```

```
Switch (n%2)
```

```
{
```

```
Case 0:
```

```
Cout <<"number is even";
```

```
Break;
```

```
Case1:
```

```
Cout<<"number is odd";
```

```
Break;
```

```
}
```

```
}
```

Switch आदेश के दौरान प्रत्येक case आदेश की जांच कर लेने के पश्चात् यदि case variable मान समीकरण के बराबर है तो उसके अंतर्गत लिखा कार्य किया जाता है तथा इसके बाद break; आदेश का प्रयोग कर उस case से बाहर निकलते हैं।

#### 4.4 लूपिंग संरचनाएं

लूप के द्वारा प्रोग्राम के किसी हिस्से को निश्चित अवधि तक दोहराया जा सकता है। यह दोहराव तक जारी रहता है जब तक कि कोई निश्चित या समीकरण का मान सत्य रहता है। सी कहीं तरह c++ के अंतर्गत भी 3 प्रकार के लूप प्रयुक्त होते हैं, ये हैं-

.for

.while

.do

**For लूप:** यह लूप, आमतौर पर उस समय प्रयुक्त होता है जब हमें यह पता हो कि लूप को कितन बार दोहराना है।

इसका सामान्य प्रारूप है-

```
For (initial value; test; increament)
```

```
{
```

```
Action1;
```

```
{
```

```
Action2;
```

एक उदाहरण देखते हैं।

```
For (i=1;i<=100;i++)
```

```
{
```

```
Cout<<1;
```

```
}
```

For वेरियेबल का निर्माण प्रारंभिक मान (initial value), वह स्थिति जिसकी जांच करनी है। (आमतौर पर यह लूप वेरियेबल का वह अधिकतम मान होता है जब तक लूप को क्रियान्वित करना है) तथा प्रारंभिक मान से अधिकतम मान तक पहुंचने के लिये आवश्यक क्रमवार वृद्धि increament से मिलकर करना होता है। लूप के साथ अन्य विशेषताएं वैसे ही प्रयुक्त होती हैं जैसी की में होती है। इसके अलावा c++ की अन्य विशेषताएं भी लूप में प्रयुक्त हो सकती हैं। एक और उदाहरण देखते हैं।

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

```
Void main ()
```

```
{
```

```
Unsigned int num;
```

```
Unsigned long fact = 1;
```

```
Cout << "enter number ";
```

```
Cin >> Num;
```

```
For (int i = num; i>0; i--)
```

```
Fact *=i;
```

```
Cout << "factorial is "<<fact;
}
```

इस उदाहरण में j वेरियेबल का प्रकार लूप के अंदर निर्धारित किया गया है। चूँकि फेक्बेरियल कोई बहुत बड़ी संख्या भी हो सकता है अतः num वेरियेबल का प्रकार long int निर्धारित किया गया है।

**while लूप:** यह एक entry controlled लूप है जिसमें पहली स्थिति की जाच की जाती है तथा स्थिति के सत्य होने पर ही लूप क्रियान्वत होता है। इसका सामान्य प्रारूप है-

While (condition)

```
{
Action1;
{
Action2;
```

यहां लूप प्रारंभ होने से पहले while के साथ लिखी स्थिति की जाच करता है। यदि स्थिति सत्य है तो action1 क्रियान्वत होता है तथा प्रोग्राम फिर while के साथ लिखी स्थिति की जाच करता है। यदि स्थिति अभी भी सत्य हो तो फिर से action1 क्रियान्वत होता है अन्यथा action1 क्रियान्वत होता है।

एक उदाहरण देखते हैं-

```
#include <iostream.h>

Void main ()
{
Int n=99;
While (n! =0)
Cin >> n;
```

यह प्रोग्राम तब तक क्रियान्वत होता रहेगा जब तक उपयोगकर्ता 0 को टाइप नहीं करते।

लूप के अंतर्गत कोई ऐसा आदेश अवश्य होना चाहिय जिससे लूप वेरियेबल का मान परिवर्तित हो सके। while के साथ लिखी स्थिति असत्य हो सके अन्यथा लूप कभी समाप्त नहीं होगा।

**do लूप:** यह एक **exit controlled** लूप है जिससे कंट्रोल पहले लूप के अंदर प्रविष्ट हो जाता है। लूप के एक बार चलने के पश्चात् स्थिति की जांच की जाती है। यदि स्थिति सत्य है तो लूप फिर से क्रियान्वत होता है। इसका सातान्य प्रारूप है-

```
Do
{
Action1
}
While (condition);
Action2
```

उक्त विवरण से स्पष्ट है कि स्थिति के असत्य होने की स्थिति में भी लूप कम से कम एक बार अवश्य क्रियान्वत होता है। इस प्रकार के लूप का एक उदाहरण देखते हैं-

```
#include <iostream.h>
Void main ()
Int dividend, divisor;
Char ch;
Do
{
Cout <<"Enter dividend"; cin >>dividend;
Cout <<"Enter divisor"; cin >> divisor;
Cout <<"Quotient is"<<dividend/divisor;
Cout <<"Remainder is"<<dividend % divisor;
Cout << "/n want another one (y/n)";
Cin >>ch;
}
While (ch! =n);
}
```

संपूर्ण लूप **do** के अंतर्गत लिखा जाता है। संपूर्ण प्रक्रिया पहले एक बार क्रियान्वयन होती है। उसके बाद **while** के साथ लिखी स्थिति की जांच की जाती है। यदि **ch** वेरियेबल का मान 'n' नहीं है तो लूप फिर से क्रियाशील हो जाता है जबकि **n** होने की स्थिति में लूप समाप्त हो जाता है।





## 5.1 परिचय

सी भाषा की प्रोग्रामिंग के अंतर्गत फंक्शन का विशेष महत्व है। फंक्शन के प्रोग्राम के आकार को अत्यंत छोटा किया जा सकता है जिससे प्रोग्राम का परिचालन अत्यंत तीव्रता से किया जा सकता है। फंक्शन मूलतः आदेशों का एक समूह होता जिन्हें एक इकाई के रूप में इकट्ठा कर कांई नाम दिया जाता है। प्रोग्रामिंग के दौरान किसी भी स्थिति से इस समूह को बुलाया जा सकता है। ये फंक्शन या तो प्रोग्राम के अंदर प्रोग्राम भाग के रूप में सुरक्षित होते हैं या स्वतंत्र रूप से किसी प्रोग्रामर के रूप में सुरक्षित हो सकते हैं जिन्हें अन्य प्रोग्राम अपनी आवश्यकतानुसार प्रयोग कर सकते हैं। पूर्व में पढ़े गये सभी प्रोग्राम में `main ()` नामक फंक्शन का प्रयोग हम देख चुके हैं।

## 5.2 main () फंक्शन

सी प्रोग्रामिंग में `main ()` फंक्शन का विशेष महत्व है। C विपरीत C++ भाषा में `Main` फंक्शन एक पूर्णांक (`integer`) प्रकार का मान ऑपरेटिंग कसस्टम कि पजेंचाता ह इन फंक्शन में जिनकी कोई रिटर्न वेल्थू होती है प्रोग्राम की समाप्ति के लिये `return` आदेशा प्रयोग किया जाता ळ। अतः C++ में फंक्शन निम्न प्रकार से परिभाषित किया जा सकता है।

```
int main()
{
    Cout<<"\n this is main function";
```

```
Return (0);  
}
```

चूँकि फंक्शन की रिटर्न वैल्यू पूर्णांक (integer) ही है। अतः `main()` के साथ शब्द का प्रयोग आवश्यक नहीं है।

### 5.3 फंक्शन प्रोटोटाइन

फंक्शन प्रोटोटाइन वह प्रक्रिया है जिसके अंतर्गत फंक्शन के कंपाइलर के साथ इंटरफेस का विवरण दिया जाता है। इसके लिये फंक्शन की कुछ विशेषताओं जैसे आर्गुमेंट की संख्या एवं प्रकार तथा रिटर्न वैल्यू का प्रकार आदि की मदद ली जाती है। हालांकि सी भाषा में भी फंक्शन प्रोटोटाइप का प्रयोग होता है लेकिन सी भाषा में यह वैकल्पिक है। जबकि सी++ में प्रोटोटाइप अनिवार्य है इसका प्रारूप है।

Type function name (argument-list);

जहाँ `argument-list`, उन आर्गुमेंट का नाम एवं प्रकार है जिन्हें फंक्शन से प्रवाहित होना है जैसे—

```
float volume (intx, floaty, floatz);
```

इस फंक्शन `volume ()` को किसी प्रोग्राम में निम्न प्रकार से किया शील किया जा सकता है।

```
volume (b1,w1,n1);
```

वेरियेबल `b1`, `w1` तथा `n1`, `cub1` के वास्तविक परिमाण है तथा इनका प्रकार प्रोटोटाइप में निकालने के लिये अनुकूल होना चाहिये।

**Cell by reference:** C++ में उपलब्ध संदर्भ वेरियेबल की सुविधा की वजह से हम फंक्शन में विशुद्ध मानों की जगह संदर्भों को भी पेरामीटर के तौर पर प्रविष्ट कर सकते हैं। एक उदाहरण देते हैं।

```
Void swap (int&a, int&b)  
{  
    Int t=a;  
    a=b;  
    b=t;  
}
```

अब यदि `m` तथा `n` दो पूर्णांक हैं तो

swap (m,n)

m तथा n के मान को संदर्भ वेरियेबल a तथा b की मदद से अन्तर्बैदल कर देंगे।

**Return by reference:** कोई फंक्शन किसी मूल मान के स्थान पर कोई संदर्भ भी वापस कर सकता है। जैसे –

```
int&max(int&x, int&y)
{
    if (x>y)
        return x;
    else
        return y;
}
```

चूंकि max का मान int& है अतः फंक्शन x या y का संदर्भ प्रदर्शित करेगा न कि उसका मूल मान।

## 5.4 फंक्शन परिभाषित करना

फंक्शन परिभाषित करने का अर्थ उसके कोड्स को एक जगह इकट्ठा कर लिखने से है।

```
void starline()
{
    for (int i=0; i<35; i++)
        cout<<"*";
    cout<<"\n";
}
```

वह लाइन जहां फंक्शन का नाम परिभाषित किया जाता है, **declarator** कहलाता है। उसके बाद {} के अंदर फंक्शन कोड्स लिखे जाते हैं। जब फंक्शन का प्रयोग किया जाता है तो प्रोग्राम का कंट्रोल फंक्शन की पहली लाइन पर स्थित हो जाता है। फंक्शन की समाप्ति पर कंट्रोल पुनः मूल प्रोग्राम पर वापस आ जाता है।

## 5.5 फंक्शन में आर्गुमेंट प्रविष्ट करना (passing argument to function)

इसे समझने के लिये यह प्रोग्राम देखते हैं।

```

#include<iostream.h>

Int power (int, int);

Void main()
{
    Int num1, num2;

    Cout<< "Enter Number";

    Cin>> num1;

    Cout<< "Enter Power";

    Cin>> num2;

    Power (num1, num2);
}

Int power (intbase, int pow)
{
    Int;
    Int p=1;
    For (i=0; i< pow; i++)
        P = p x base;
    Cout<< "Power is" <<p;
}

```

इस प्रोग्राम में power() नामक फंक्शन का प्रयोग किया गया है। मूल प्रोग्राम की-बोर्ड से दो मान ग्रहण कर उन्हें फंक्शन में प्रवाहित करता है। फंक्शन के अंदर प्रयुक्त वेरियेबल जो इन्हें कहण करते हैं, पैरामीटर कहलाते हैं। जैसे power() में ये base तथा pow है।

## 5.6 फंक्शन से मान प्राप्त करना (Received Values from function)

जब कोई फंक्शन क्रियान्वित होता है तो वह किसी मान की गणना कर उसे मूल प्रोग्राम में वापस भेजता है। आमतौर पर यह मूल समस्या का हल होता है। एक प्रोग्राम देखते हैं।

```

#include<iostream.h>

Float pound-to-kg (float);

```

```

Void main()
{
Float p, kg;

Cout<< "Enter weight in pounds";

Cin>> p;

Kg=pound-to=kg (p);

Cout<< "weight in kilogram is"<<kg;

Float pound-to-kg (float pounds);
{
Float kilogram = 0.45 x pounds;

Return kilogram

```

यह फंक्शन kilogram नामक वेरियेबल का मान मुख्य प्रोग्राम में पहुंचाता है। जब फंक्शन कोई मान वापस करता है। तो उस मान का प्रकार निर्धारित करना आवश्यक है। उक्त प्रोग्राम में pound-to-dg(p) एक ऐसा समीकरण है जो वापस किये जाने वाले मान को ग्रहण करता है।

### Const आर्गुमेंट

C++ के अंतर्गत किसी आर्गुमेंट फंक्शन का आर्गुमेंट const निम्न प्रकार से निर्धारित किया जा सकता है।

```

Int stolen (const char*p);

Int length (const string &&);

```

यह const कंपाइलर को सूचित करता है कि आर्गुमेंट को फंक्शन द्वारा परिवर्तित न किया जा सके। यह प्रकार तब विशेष उपयोगी है जब आर्गुमेंट को संदर्भ (reference) के जरिये प्रविष्ट किया जाता है।

## 5.7 फंक्शन ओवरलोडिंग (Function overloading)

सी++ के अंतर्गत हम एक ही फंक्शन नाम का प्रयोग ऐसे विशिष्ट फंक्शन निर्माण के लिये कर सकते हैं जो अलग-अलग कार्य कर सकें। इसे oops के अंतर्गत फंक्शन ओवरलोडिंग कहते हैं। फंक्शन ओवरलोडिंग की परिकल्पना का प्रयोग करते हुए हम फंक्शनों का एक ऐसा समूह बना सकते हैं जिनमें एक ही फंक्शन नाम अलग-अलग आर्गुमेंट के साथ प्रयोग किया जा सकता है। इन अलग-अलग आर्गुमेंट के आधार पर फंक्शन अलग-अलग प्रक्रियाएं सम्पन्न करेगा। उदाहरण के लिये add() फंक्शन अलग-अलग प्रकार के डाटा का निम्न अलग-अलग प्रकार से प्रयोग करेगा।

```
//Declarations

int add (int a; int b); //prototype1

int add (int a, int b, int c); //prototype 2

double add (double); //prototype 3

double add (int p, double q); //prototype 4

double add (double p, int q); //prototupe 5

//Function calls

cout << add (5,10);           //uses prototype 1

cout << add 15,10,0);         //uses prototype 1

cout << add (12.5,7,5);       //uses prototype 1

cout << add (5,10,15);        //uses prototype 1

cout << add (0,75,5);         //uses prototype 1
```

कोई भी function call पहले अपने समकक्ष संख्या, एक प्रकार के प्रोटोटाइप वाले आर्गुमेंट को ढूँढता है तथा उसके बाद उचित फंक्शन को क्रियान्वि करता है।

अब हम फंक्शन ओवरलोडिंग का प्रयोग करते हुए एक उदाहरण देखते हैं।

```
#include<iostream.h>

int volume (int); //prototype

double volume (double, int); //prototype declration

long volume (long, int, int); //prototype declaration

main()

cout << volume (10) <<"\n";

cout << volume (2.5,8) <<"\n";

cout << volume (100L, 75, 15);

//.... function definition....

int volume (int 5)

{

return (s*s*s);

}

double volume (double r, int h);
```

```

{
    return (3.14159*r*r*h);
}

long volume (long l, int b, int h);
{
    return (l, b, h,)
}

```

इस प्रोग्राम का आउटपुट होगा

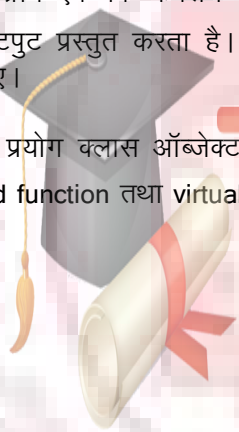
1000

157.2595

112500

यह प्रोग्राम एक कि फंक्शन नाम **volume()** का प्रयोग करता है तथा तीन अलग-अलग आर्गुमेंट के लिये तीन अलग-अलग आउटपुट प्रस्तुत करता है। फंक्शन ओवरलोडिंग का प्रयोग सिर्फ एक जैसे आपस में संबंधित प्रक्रियाओं के लिये ही करना चाहिए।

इनका प्रयोग क्लास ऑब्जेक्ट को संभालने के लिये अधिकतर किया गया है। इसी प्रकार दो और विशिष्ट प्रकार के फंक्शनों **friend function** तथा **virtual function** का विवरण भी आगे दिया गया है।



Tayari Jeet ki



## 6.1 परिचय

सी++ की सभ्यतः सबसे प्रमुख विशेषता 'क्लास' है। यह भी में पढ़े गये संरचना (structures) हो उन्नत रूप है जिसके जरिये उपयोगकर्ता द्वारा परिभाषित प्रकार के डाटा टाइप बनाये एवं प्रयोग किये जा सकते हैं। हमने सी भाषा के अध्ययन के समय यह पढ़ा है कि संरचना के प्रयोग से हम विभिन्न प्रकार के डाटा को इकट्ठा कर एक सम्मिलित प्रकार का डाटा प्रकार निर्मित कर सकते हैं। जैसे उदाहरण देखिये।

```
Struct complex
```

```
{
```

```
Float s;
```

```
Float y;
```

```
Int t;
```

```
};
```

```
Struct complex c1, c2, c3
```

उक्त उदाहरण में दो विभिन्न प्रकार के डाटा को इकट्ठा कर एक विशिष्ट डाटा प्रकार **complex** निर्मित किया गया है जिसके वेरियेबल **c1**, **c2** तथा **c3** हैं। इन वेरियेबल को डॉट-ऑपरेटर की मदद से कोई मान दिया जा सकता है। लेकिन सी भाषा में प्रयुक्त इस **strict** प्रकार की प्रमुख कमी यह है कि इसके वेरियेबल को **built-in** प्रकार के वेरियेबल की तरह प्रयोग नहीं कर सकते हैं।



जैसे  $c3=c1+c2$  आदेश वैध नहीं है।

इसके अलावा एक अन्य प्रमुख कती यह है कि इसमें डाटा को छुपाने (Data hiding) की कोई सुविधा उपलब्ध नहीं है। किसी एक फंक्शन में परिभाषित तथा प्रयुक्त डाटा अन्य फंक्शन द्वारा भी आसानी से प्रयोग किया जा सकता है।

सी++ के अंतर्गत सी के संरचना की सभी खूबियों को प्रयोग करते हुए एक उन्नत रूप उपलब्ध कराया गया है जो ऑब्जेक्ट ओरिएन्टेड प्रोग्राम की विचारधारा के अनुकूल है। इसे क्लास (Class) कहते हैं। इसके अन्तर्गत उपयोगकर्ता द्वारा निर्मित डाटा प्रकार को **built-in** डाटा प्रकार के अनुकूल रखने का अधिकाधिक प्रयास किया गया है। साथ ही **data-hiding** की भी सुविधा उपलब्ध कराई गई है।

## 6.2 क्लास परिभाषित करना

क्लास ऐसा प्रकार है जिसके जरिये डाटा तथा उससे संबंधित फंक्शन को इकट्ठा किया जा सकता है। साथ ही डाटा का बाहरी प्रयोग से बचाया जा सकता है। एक क्लास के निर्धारण के दो भाग हैं।

- क्लास निर्धारण
- क्लास के फंक्शन निर्धारण

क्लास के निर्धारण का सामान्य प्रारूप निम्न है।

```
Class<class_name>
```

```
{
```

```
Private;
```

```
Variable declaration;
```

```
Function declaration;
```

```
Public;
```

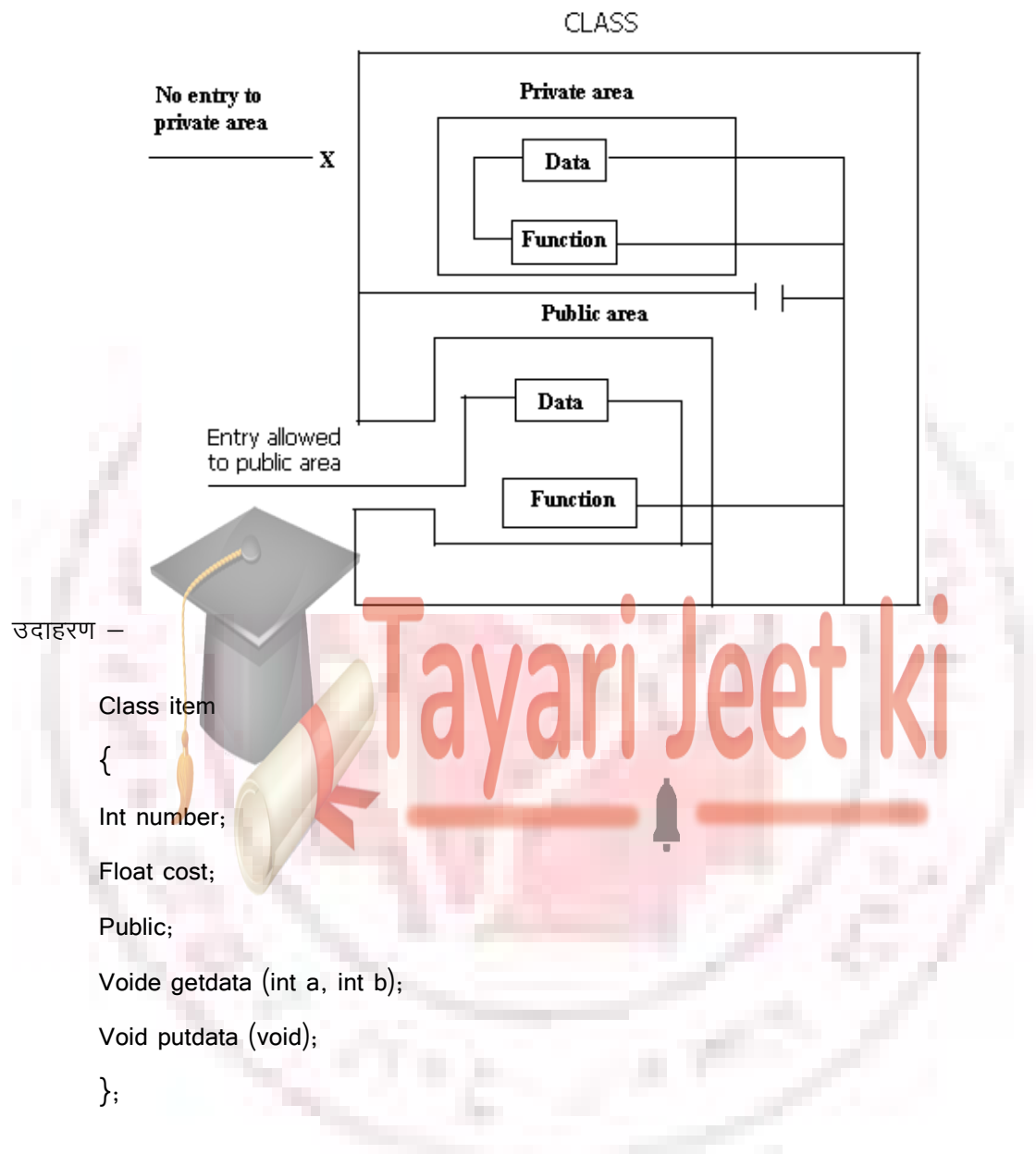
```
Variable declaration;
```

```
Function declaration;
```

```
};
```

क्लास निर्धारण में **class** शब्द के साथ उस क्लास का नाम तथा उसके अंतर्गत वेरियेबल तथा फंक्शन निर्धारण किये जाते हैं। इन्हें उस क्लास का मेम्बर वेरियेबल तथा मेम्बर फंक्शन कहते हैं। वे सदस्य जो **private** के साथ परिभाषित किये जाते हैं केवल क्लास के अंतर्गत ही प्रयोग किये जा सकते हैं जबकि **public** के साथ परिभाषित सदस्य को क्लास के बाहर से भी प्रयोग किया जा सकता है। यदि कोई प्रकार निर्धारित नहीं किया है तो स्वतः ही (**bydefault**)

private मान लिया जाता है। डाटा फंक्शन को एक क्लास के अंतर्गत इकट्ठा करने की प्रक्रिया को encapsulation कहते हैं।



उक्त उदाहरण के बाद item नामक नया डाटा प्रकार उपलब्ध हो जाएगा जिसके अंतर्गत दो डाटा मेम्बर हैं जो private हैं तथा दो फंक्शन मेम्बर हैं जो public प्रकार हैं। फंक्शन getdata() का प्रयोग डाटा मेम्बर number तथा cost को कोई मान प्रदान करने के लिये होता है जबकि putdata() का प्रयोग उनका मान प्रदर्शित करने के लिये होता है। यहां यह ध्यान रखना आवश्यक है कि फंक्शन को सिर्फ निर्धारित (declare) किया गया है, परिभाषित (defined) नहीं फंक्शन को प्रोग्राम में आगे परिभाषित किया जाता है।

### 6.3 ऑब्जेक्ट का निर्माण

क्लास का निर्धारण करने के पश्चात् उस प्रकार का ऑब्जेक्ट निर्मित किया जा सकता है। इसके लिये क्लास के नाम के साथ उस ऑब्जेक्ट का नाम लिखा जाता है जैसे –

Item x;

यह आदेश **x** नामक एक वैरियेबल निर्मित करेगा जिसका प्रकार **item** है। एक ही आदेश में एक से अधिक वैरियेबल का निर्माण कर सकते हैं।

Item x, y, z;

## 6.4 मेम्बर फंक्शन परिभाषित करना

मेम्बर फंक्शन दो अलग-अलग स्थानों पर परिभाषित किये जा सकते हैं।

- क्लास परिभाषा के बाहर
- क्लास परिभाषा के अंदर

इसका सामान्य प्रारूप है।—

```
Return type class name :: function name (argument declaration)
{
    Function body
}
```

मेम्बरशिप लेबर **class-name::** यह सूचित करता है कि लिखे गये फंक्शन का नाम (**function-name**) उस **class name** से संबंधित है।

### क्लास के साथ एक C++ प्रोग्राम

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

```
Class item
```

```
{
```

```
    Int number;
```

```
    Float cost;
```

```

Public;

Void getdata (int a, int b);

Void putdata (void);
{
Cout <<"number: " <<number <<"/n";

Cout <<"cost:" <<cost<<"/n";

}

};

Void item :: getdata (int a, int b )
{
Number =a;
Cost =b;
}

Main ()
{
Item x; //create object x
Cout <<"/n object x" <<"/n";
x.getdata (100, 299. 50);
x.putdata ();

Item y; //create object y
Cout <<"/n object y" <<"/n";
y.getdata (200, 150.65);
y.putdata()

```

यह प्रोग्राम दो ऑब्जेक्ट X तथा Y का निर्माण दो अलग-अलग आदेशों में करता है। इन्हें एक साथ इस प्रकार लिखा जा सकता है।

```
Item x,y;
```

इस प्रोग्राम का आउटपुट निम्न होगा।

```
Object x
```

Number: 100  
Cost: 229. 50  
Number: 200  
Cost: 229.50  
Number: 200  
Cost: 150. 65

**नेस्टेड मेम्बर फंक्शन:** किसी एक मेम्बर फंक्शन के नाम को किसी दूसरे मेम्बर फंक्शन का नाम प्रयोग करते हुए प्रयोग कर सकते हैं। इसे मेम्बर फंक्शन की नेस्टिंग कहते हैं।

```
#include <iostream.h>

Class set
{
Public:
Voide input (void);
Void display(void);
Int largest (void);
};

Int set :: largest (void)
{
If (m>=n)
Return (m);
Else
Return (n);
}

Void set :: input (void)
{
Cout << "input value of mandn" <<"/n";
Cin >>m>>n;
}

Void set:: display (void)
{
Cout <<"largest valued is"
<<largest () <<"/n";
```

```

    }
    Main ()
    {
    Set A:
    A.    input ();
    A.display ();
    }

```

उक्त प्रोग्राम का आउटपुट होगा—

Input value of m and n

30,50

Largest value= 50

## 6.5 क्लास के अंदर अरे का प्रयोग

क्लास के अंदर अरे को मेम्बर वेरियेबल की तरह प्रयोग किया जा सकता है। जैसे

```
Const int size = 10//value for array size
```

Call array

```

{
Int a [size];
Public:
Void setval (void);
Void display (void);
};

```

अरे वेरियेबल `a[ ]` को अरे क्लास के `private` मेम्बर की तरह निर्धारित किया गया है।

## 6.6 स्टैटिक डाटा मेम्बर

स्टैटिक वेरियेबल का प्रयोग उन मानों के लिये होता है जो पूरी क्लास के लिये उभयनिष्ठ (common) यह आदेश देखते हैं।

```
Int item :: count;//static data member
```

अब स्टैटिक डाटा मेम्बर का प्रयोग करते हुए एक प्रोग्राम देखते हैं।

```
#include <iostream.h>

Class item
{
    Static int cout;//cou is static
    Int member;
    Public:
    Void getdata (inta)
    {
        Number =a;
        Count ++;
    }
    Void getcout (void)
    {
        Cout <<"cout";
        Cout << cout "/n"
    };
};
```

## 6.7 कंस्ट्रक्टर तथा डिस्ट्रक्टर (constructor and Destructor)

कंस्ट्रक्टर एक विशेष प्रकार का मेम्बर फंक्शन है जिसका प्रयोग क्लास के ऑब्जेक्ट को प्रारंभिक मान देने के लिये होता है। यह विशेष इसलिये है क्योंकि इसका नाम वही होता है जा क्लास का नाम है। इसे **constructor** इसलिये कहते हैं क्योंकि यह क्लास के डाटा मेम्बर का निर्माण (constructor) करता है कंस्ट्रक्टर को निम्न तरह से परिभाषित किया जाता है।

```
Class integer
{
```

```

Int m, n;

Public;

Integer (void); //constructor declared

::

};

Integer :: integer (void) //constructor defined
{
M=n; n=0;
}

```

जब किसी क्लास के साथ कंस्ट्रक्टर का प्रयोग होता है। तो उसके ऑब्जेक्ट का मान स्वतः ही अपनी प्रारंभिक स्थिति में आ जाता है।

**डिस्ट्रक्टर (destructor)**, जैसा कि नाम से जाहिर है, उन ऑब्जेक्ट को नष्ट कर देता है जो कंस्ट्रक्टर के साथ निर्मित किये जाते हैं। यह भी एक मेम्बर फंक्शन होता है जिसका नाम वही होता है जो क्लास का नाम है। जिसके पहले एक tilde (~) चिन्ह प्रयुक्त होता है। उदाहरण के लिये integer नामक class के लिये desrtuctoc परिभाषित करने लिये आदेश होगा—

```

~ Integer () {}

```

डिस्ट्रक्टर के साथ न तो कोई आर्गुमेंट प्रयुक्त होता है, न ही कोई मान मूल प्रोग्राम को वापस करता है।

```

Int item :: cout
Main ()
{
Item a,b,c;
A.get cout ();
b.getcout ();
c.getcout ();
a.getdata (100);
b.getdata(200);
c.getdata (300);
cout <<"After reading data" <<"\n";
}

```



```
a.getcout();  
b.getcout();  
c.getcout();
```

प्रोग्राम में स्टैटिक वेरियेबल `cout` का प्रारंभिक मान शून्य कर लिया गया है। जब भी ऑब्जेक्ट के अंदर एक डाटा पढ़ा जाता है तो `cout` का मान बढ़ जाता है। चूंकि डाटा तीन बार ऑब्जेक्ट में पढ़ा जाता है अतः `cout` वेरियेबल का मान तीन बार बढ़ता है।

**स्टैटिक मेम्बर फंक्शन :** स्टैटिक मेम्बर वेरियेबल की तरह स्टैटिक मेम्बर फंक्शन का भी प्रयोग किया जा सकता है। स्टैटिक मेम्बर फंक्शन को उसकी क्लास का नाम प्रयोग करते हुए निम्न तरह से लिखा जा सकता है।

```
class-name :: function-name;
```

एक स्टैटिक फंक्शन केवल उसी क्लास के दूसरे स्टैटिक मेम्बर को ही पढ़ सकता है।

## 6.8 ऑब्जेक्ट की अरे (Array of objects)

हम अरे के अंदर ऐसे वेरियेबल प्रयुक्त कर सकते हैं जिनका प्रकार क्लास है। ऐसे वेरियेबल को ऑब्जेक्ट की अरे कहते हैं। यह उदाहरण देखिये।

```
Class employee
```

```
{
```

```
Char name [30];
```

```
Float age;
```

```
Public
```

```
Void getdata(void);
```

```
Void putdata (void);
```

```
};
```

यहां `employee` एक उपयोगकर्ता द्वारा परिभाषित डाटा प्रकार तथा इसका प्रयोग अलग अलग ऑब्जेक्ट निर्माण करने के लिये हो सकता है। जैसे

```
Employee manger [3];
```

```
Employee officer [20];
```

यहां **manager** नामक अरे तीन ऑब्जेक्ट का निर्माण करेगा जबकि **officer** नामक अरे बीस ऑब्जेक्ट का निर्माण करेगा।

```
Manager (i).putdata ();
```

यह आदेश **manager** नामक इरे के **ith** प्रविष्टि को प्रदर्शित करेगा। ऑब्जेक्ट की यह इरे एक बहुआयामी अरे की तरह मेमोरी में सुरक्षित होती है।

फंक्शन आर्गुमेंट की तरह ऑब्जेक्ट का प्रयोग: किसी भी अन्य डाटा की तरह ऑब्जेक्ट को भी फंक्शन के आर्गुमेंट की तरह प्रयोग किया जा सकता है। यह दो प्रकार से किया जाता है।

- पूरे ऑब्जेक्ट को फंक्शन से प्रवाहित करना जिसे **pass-by-value** कहते हैं।
- ऑब्जेक्ट के सिर्फ एड्रेस को फंक्शन में स्थानांतरित करना जिसे **pass-by-reference** कहते हैं।

## 6.9 फ्रेंडली फंक्शन (Friendly function)

एक ऐसी स्थिति देखते हैं जिसमें दो क्लास **manager** तथा **scientist** परिभाषित किये गये हैं। अब यदि हम एक फंक्शन **incomtax ()** प्रयोग करना चाहते हैं जो दोनों क्लास के ऑब्जेक्ट पर कार्य करता है। ऐसी स्थिति में **c++** इस उभयनिष्ठा फंक्शन को **friendly** तौर पर प्रयोग कर सकते हैं जिससे फंक्शन इन दोनों क्लास के **private** डाटा को भी प्रयोग कर सकता है। इसके लिये इस फंक्शन को **friend** शब्द के साथ निधारित करना आवश्यक है। जैसे-

```
Class ABC
{
.....
.....
Public:
.....
.....
Friend void xyz(void):
};
```

इस प्रकार के फंक्शन को दोनों में से किसी भी क्लास का मेम्बर होना आवश्यक नहीं है। निम्न प्रोग्राम में **Friend** फंक्शन का प्रयोग प्रदर्शित है।

```

#include <iostream.h>

Class sample
{
    Int a;

    Int b;

    Public:

    Void setvalue () {a=25; b=40;}

    Friend float mean (sample s);
};

Float mean (samle &
{
    Return float (s.a+s.b)/2.0;
}

Main ()
{
    Sample x;
    x.setvalue();
    Cout <<"Mean value -"<<mean (x) "/n";
}

```

उक्त प्रोग्राम का आउटपुट Mean value:32.5

## 6.10 मेम्बर के पॉइन्टर (Pointer of member)

किसी क्लास के मेम्बर का ऐड्रेस प्राप्त कर उसे एक पॉइन्टर पर निर्धारित किया जा सकता है। मेम्बर का ऐड्रेस किसी मेम्बर के नाम के साथ '&' ऑपरेटर का प्रयोग कर ज्ञात किया जा सकता है। क्लास मेम्बर का पॉइन्टर क्लास के नाम के साथ '::\*' का प्रयोग कर निर्धारित किया जा सकता है।

अब मेम्बर m के लिये पॉइन्टर निम्न तरह से निर्धारित किया जा सकता है।

```
Int A::&ip=&a::m;
```

यहां lp एक पॉइन्टर है। यहां A::\* का अर्थ है “क्लास A के मेम्बर के लिये पॉइन्टर” ।



## 7

### इनहेरिटेंस

---

#### 7.1 परिचय

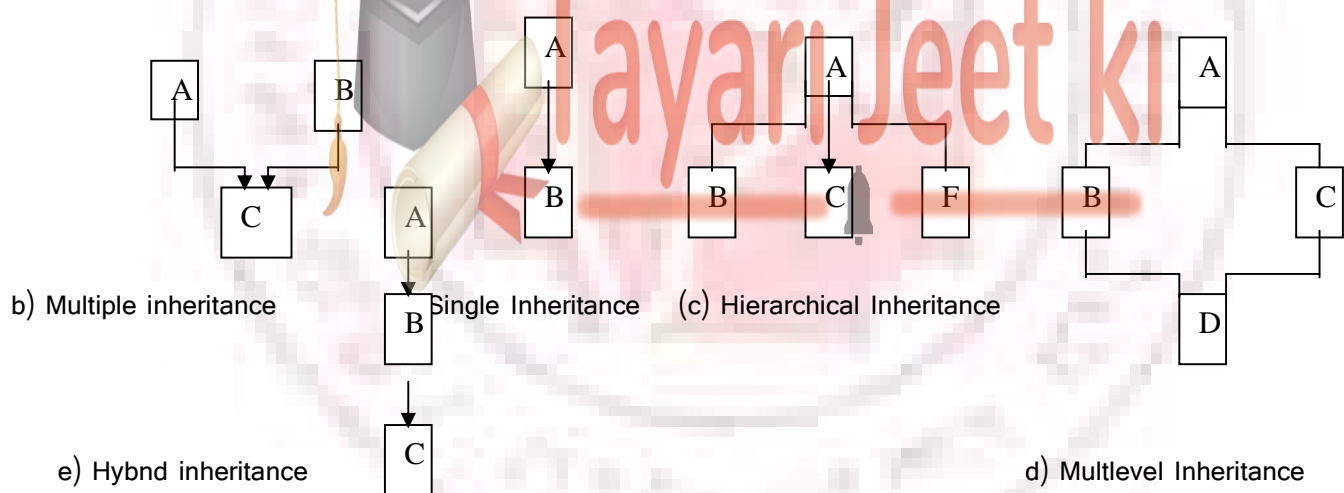
सी++ की एक प्रमुख विशेषता यह है कि इसके क्लास को एक से अधिक बार प्रयोग किया जा सकता है। ऐसा मुख्यतः नयी क्लास का निर्माण कर किया जाता है जिसमें पुरानी क्लास की विशेषताएं inherit हो जाती हैं। अतः

**inheritance** को ऐसी प्रक्रिया के रूप में परिभाषित किया जा सकता है जिसमें पुरानी क्लास से नयी क्लास का निमर्सन होता है। पुरानी क्लास को **base** क्लास तथा नयी क्लास को **derived** क्लास कहते हैं।

नयी क्लास में पुरानी क्लास के सभी या कुछ गुण समावेशित हो सकते हैं। साथ ही नयी क्लास में इसके अलावा गुण जोड़े जा सकते हैं तथा इसके अलावा कोई क्लास एक से अधिक पुरानी क्लास से भी गुण प्राप्त कर सकती है। यदि किसी नयी क्लास में सिर्फ एक ही पुरानी **base class** है तो ऐसी नयी क्लास को **single inheritance** कहते हैं जबकि यदि एक से अधिक **base** क्लास हो तो इसे **multiple inheritance** कहते हैं। किसी नयी क्लास से दूसरी नयी क्लास प्राप्त करने की प्रक्रिया को **multiple inheritance** कहते हैं। इसी प्रकार एक क्लास एक से अधिक क्लास को अपनी विशेषताएं प्रदान कर सकती है। इसे **hierarchical inheritance** कहते हैं।

निम्न चित्रों में **inheritance** के विभिन्न प्रकार प्रदर्शित हैं।

ऑब्जेक्ट ओरियन्टेड प्रोग्राम में क्लास के बाद **inheritance** संभवतः सबसे अधिक उपयोगी विशेषता है। इसकी मुख्य विशेषता यह है कि कोड्स को बिना दुबारा लिखे अन्य प्रोग्राम में प्रयोग करने की सुविधा प्रदान करता है। (**reusability**) एक बार एक **base** क्लास लिखे जाने के बाद उसे विभिन्न स्थितियों के लिये प्रयोग किया जा सकता है। इससे प्रोग्राम को दुबारा लिखने में समय तो बचता ही है, प्रोग्राम की विश्वसनीयता भी बढ़ जाती है।



## 7.2 नयी क्लास परिभाषित करना

किसी नयी क्लास को **base** क्लास के साथ उसके संबंध तथा उसके अपने विवरणों के साथ मिलाकर परिभाषित किया जाता है। इसका सामान्य प्रारूप है—

```
class derived-class-name: visibility mode base-class-name
{
```

.....

.....

Members of derived class

};

यहां **derived-class-name** नयी क्लास का नाम तथा **base-class-name** उस पुरानी क्लास का नाम है जिससे नयी क्लास का निर्माण हुआ है। **visibility-mode** का प्रयोग वैकल्पिक है तथा इसका मान **private** **public** होता है। इसका प्रारंभिक मान (default) **private** हैं। जैसे-

```
ClassABC: private xyz
```

```
{
```

```
Members of ABC
```

```
};
```

```
Class ABC: public xyz
```

```
{
```

```
Memberof ABC
```

```
};
```



# Tayari Jeet ki



## 7.3 एकल इनहेरिटेंस (single inheritance)

एकल inheritance को समझने के लिये एक प्रोग्राम देखते हैं निम्न प्रोग्राम में एक **base** क्लास B तथा एक **drived class** D हैं। B क्लास में एक प्रायवेट डाटा मेम्बर एक पब्लिक डाटा मेम्बर तथा 3 पब्लिक मेम्बर फंक्शन है। जबकि D क्लास में एक प्रायवेट डाटा मेम्बर तथा 2 पब्लिक फंक्शन है।

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

```
Class B
```

```
{
```

```
Int a;
```

```
Public:
```

```

Int b;

Void get ab ();

Int get a (void);

};

Class D: public B
{
Int c;

Public:

Void nul (void);

};

//function definition
Void :: get-ab (void)
{a=5; b=10;}

Int B :: get-a ()
{return a;}

Void B ::show- a ()
{cout <<"a=" << a<< "/n";}

Void D :: nul ()
{c=b x get-a();}

Void :: display ()
{
Cout << "a=" << get_a() << "/n";
Cout << "b=" << b<< "/n";
}

//MAIN PROGRAM

Main()

D d;

d.get- ab();

d.nul();

d.show-a();

```

```

d.display();

d.b=20;

d.nul();

d.display();

}

```

इस प्रोग्राम का आउटपुट निम्न होगा।

```

a=5

a=5

b=10

c=10

a=5

b=20

c=100

```

यह ध्यान रखना आवश्यक है कि मूल क्लास के सिर्फ उन ही मेम्बर को नयी क्लास में समावेशित किया जाता है जिनका प्रकार **public** निर्धारित किया गया है।

**Private मेम्बर को समावेशित करना :** सी++ भाषा में **private** तथा पब्लिक के इलावा एक और प्रकार है- **protected**, यदि किसी मेम्बर को प्रोटेक्ट परिभाषित किया गया है तो वह उस क्लास तथा उससे ठीक अगली **derive** क्लास द्वारा प्रयोग किया जा सकता है।

```

Class alpha
{
Private:

.....

.....

Protected:

.....

.....

Public:

.....

```



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

जब किसी प्रोटेक्ट प्रकार के मेम्बर के मेम्बर को **public** मोड में **inherit** किया जाता है तो यह नयी क्लास में भी **protected** ही रहता है। जबकि यदि **protected** प्रकार के मेम्बर को **private** मोड में **inherit** किया जाए तो यह नयी क्लास में भी **private** ही रहता है। अतः यह नयी **derived** क्लास में तो उपलब्ध रहते हैं लेकिन इसके आगे की स्थिति के लिये इसे **inherit** नहीं किया जा सकता।

किसी क्लास को परिभाषित करने में **private**, **public** तथा **protected** शब्द किसी भी क्रम से तथा कितनी भी बार प्रयोग किये जा सकते हैं।

## 7.4 बहुस्तरीय इनहेरिटेंस (Multilevel inheritance)

कोई एक नयी क्लास किसी दूसरी नयी क्लास से प्राप्त की जा सकती है जैसे कि इस चित्र में प्रदर्शित हैं।

base class

intermediate base class

Derived class

किसी multilevel inheritance को निम्न तरह से परिभाषित किया जा सकता हैं।

```
Class A{...}; //Base Class
```

```
Class B: publicA {...}; //B derieved from A
```

```
Class C: publicB {...}; //C derived from B
```

## 7.5 एकाधिक इनहेरिटेंस (Multiple inheritances)

B-1

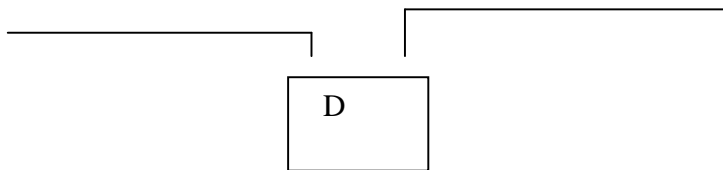
B-2

B-3

|

|

|



कोई एक क्लास एक से अधिक क्लास के गुण धारण कर सकता है जैसा कि निम्न चित्र से प्रदर्शित है।

इसे **multiple inheritance** कहते हैं।

किसी नयी क्लास को जिसमें एक से अधिक (**base**) क्लास है, निम्न प्रकार से परिभाषित किया जा सकता है।

Class D: visibility B-1, visibility B-2....

{

....

....

Body of D

....

};



Tayari Jeet ki



जहाँ **visibility** का मान **private** हो। मूल क्लास को कॉमा (,) के जरिये अलग किया जाता है।

Class p: public m, public n

{

Void display (void);

};

नयी क्लास **p** में **m** तथा **n** दोनों के मेम्बर होंगे। साथ ही इसके अपने मेम्बर भी इसमें हो सकते हैं।

निम्न प्रोग्राम में यह प्रदर्शित किया गया है कि किस तरह तीन क्लास को **multiple inheritance** मोड में प्रकट किया जा सकता है।

```
////////////////[MULTIPLE INHERITANCE]////////////////
```

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

Class M

{

Protected:

Int m;

Public:

Void get\_m (int);

};

Class N

{

Protected:

Int n;

Public:

Void get\_n (int);

};

Class P: public M, public N

{

Public:

Void display (void);

};

Void M :: get\_m (int x)

{m= x ;}

Void N :: get\_n (int y)

{n= y;}

Void P :: display (void)

{

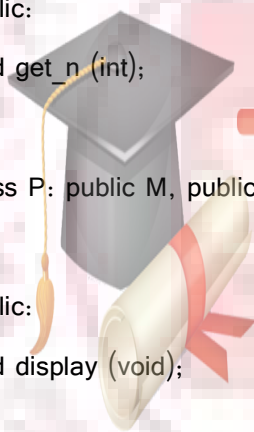
Cout <<"m=" <<m <<" /n";

Cout <<"n=" <<" /n";

Cout <<"m\*n=" <<m\*m <<" /n"

}

Main ()



Tayari Jeet ki

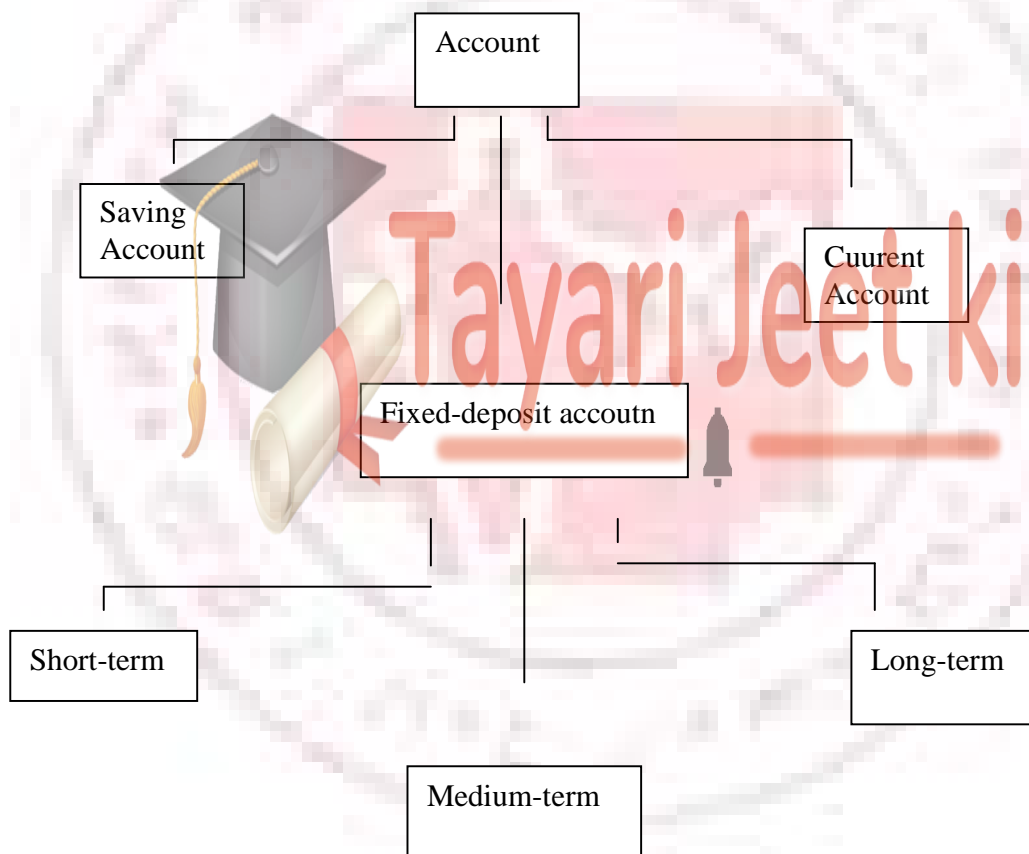


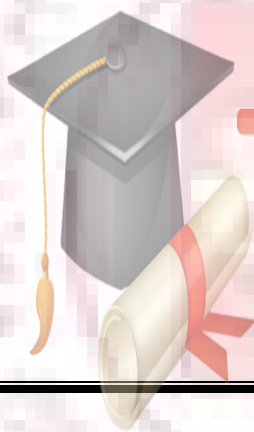
```

| {
  P p;
  p.get_m (10);
  p.get_n (20);
  p.display ()

```

**Hierarchical inheritance:** यह inheritance का ऐसा प्रकार है जिसमें किसी एक लेबर की कुछ विशेषताएं उसके नीचे के कई लेबल द्वारा एक साथ प्रयोग की जाती हैं। निम्न दो चित्रों में ऐसी दो संरचनाएं प्रदर्शित हैं।





# Tayari Jeet ki

8

ऑपरेटर ओवरलोडिंग

## 8.1 परिचय

सी++ की विभिन्न विशेषताओं में ऑपरेटर ओवरलोडिंग प्रमुख है जिसने सी++ के विस्तार करने की क्षमता में वृद्धि की है। ऑपरेटर ओवरलोडिंग वह प्रक्रिया है जिसके जरिये किसी ऑपरेटर को उसके मूल अर्थ के साथ कोई अन्य अर्थ भी प्रदान किया जा सकता है। जैसे कि + ऑपरेटर का प्रयोग आंकिक वेरियेबल को जोड़ने के लिये है लेकिन क्लास के प्रयोग से उनका प्रयोग **structure variable** को भी जोड़ने के लिये हो सकता है।

ऑपरेटर ओवरलोडिंग के जरिये सी++ में प्रयुक्त निम्न ऑपरेटरों को छोड़कर सभी के अर्थ परिवर्तित किये जा सकते हैं।

Class member access operator (...\*)

Scope resolution operator (::)

Size operator (size of)

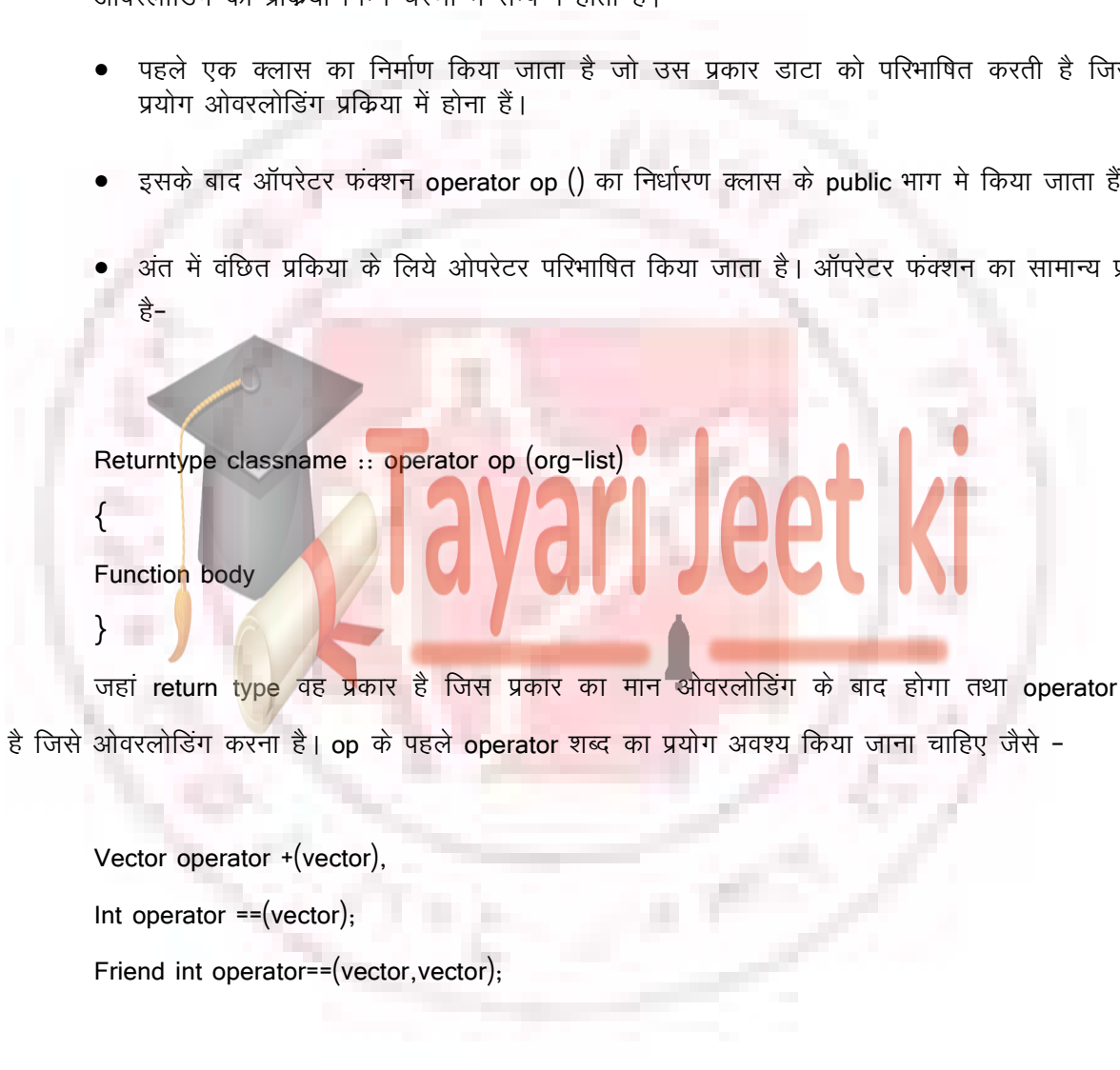
Conditional operator (? :)

यहां पर ध्यान रखना आवश्यक है कि ऑपरेटरो का अर्थ परिवर्तित होने पर भी उनका मूल अर्थ नष्ट नहीं होता। साथ ही यह भी ध्यान रखना आवश्यक है सिर्फ ऑपरेटर का अर्थ ही परिवर्तित होता है उसका **syntax** तथा उसे प्रयोग करने के अन्य नियम परिवर्तित नहीं होते हैं।

## 8.2 ऑपरेटर ओवरलोडिंग को परिभाषित करना

ओवरलोडिंग की प्रक्रिया निम्न चरणों में सम्पन्न होती हैं।

- पहले एक क्लास का निर्माण किया जाता है जो उस प्रकार डाटा को परिभाषित करती है जिसका प्रयोग ओवरलोडिंग प्रक्रिया में होना है।
- इसके बाद ऑपरेटर फंक्शन `operator op ()` का निर्धारण क्लास के **public** भाग में किया जाता है।
- अंत में वंछित प्रक्रिया के लिये ओपरेटर परिभाषित किया जाता है। ऑपरेटर फंक्शन का सामान्य प्ररूप है-



```
Returntype classname :: operator op (arg-list)
{
    Function body
}
```

जहां **return type** वह प्रकार है जिस प्रकार का मान ओवरलोडिंग के बाद होगा तथा **operator** वह ऑपरेटर है जिसे ओवरलोडिंग करना है। **op** के पहले **operator** शब्द का प्रयोग अवश्य किया जाना चाहिए जैसे -

```
Vector operator +(vector),
Int operator ==(vector);
Friend int operator==(vector,vector);
```

## 8.3 Unary ऑपरेटर की ओवरलोडिंग

Unary ऑपरेटर में एक ही **operand** होता है जैसा ऑपरेटर **++** या **....** Unary निम्न प्रोग्राम में एक **unary minus** ऑपरेटर की ओवरलोडिंग प्रदर्शित है।

```
#include iostream.h
```

```
Class space
```

```
{
```

```
Int x;
```

```
Int y;
```

```
Int z;
```

```
Public:
```

```
Void getdata (int a, int b, int c);
```

```
Void display (void);
```

```
Void operator - (); // operator unary minus};
```

```
};
```

```
Void space :: getdata (int a, int b, int c)
```

```
{
```

```
X=a;
```

```
Y=b;
```

```
Z=c;
```

```
}
```

```
Void space :: display (void)
```

```
{
```

```
Cout <<x<<"";
```

```
Cout <<y<<"";
```

```
Cout <<z<<"";
```

```
}
```

```
Void space :: operator -() //defining operator -()
```

```
{
```

```
X= - x;
```

```
Y= - y;
```

```
Z = -z;
```

```
}
```

```
Main
```



Tayari Jeet ki



```
{
Spaces s;
.sgetdata (10, -20, 30);
Cout <<"s:";
s.display ();
```

यह प्रोग्राम निम्न आउटपुट प्रस्तुत करेगा।

S:10-20 30

S: - 10, 20 - 30

इस प्रोग्राम में ऑपरेटर फंक्शन का कार्य यह है कि यह ऑब्जेक्ट के डाटा मेम्बर का मूल चिन्ह परिवर्तित कर देता है।

## 8.4 बायनरी ऑपरेटर की ओवरलोडिंग

बायनरी ऑपरेटर को भी आसानी से ओवरलोडिंग किया जा सकता है इसे समझने के लिये पहले एक प्रोग्राम देखते हैं।

```
[// foverloading + operator]
Include <iostream.h>
Class complex
Float x;
Float y;
Public:
Complex () {} //constructor 1
Complex {float real, float image} //constructor2
{X=real; y=imag;}
Complex operator +(complex);
Void display (void );
};
Complex complex :: operator (complex c)
{
Complex temp;
```



```

Temp. x= x+c.x; //float addition
Temp.y =y+c.y; //float addition
Return (temp);
}

Void complex :: display (void)
{
Cout <<x <<"+"<<y<<"\n";
}

Main ()
Complex c1,c2,c3; //invoke constructor 1
C1=complex (2.5, 3.5); //invoke constructor
C2= complex (1.6,2.7); //invoke constructor2
C3= c1+c2; //invoke +() operator
Cout <<"c1 ="; c1.display ();
Cout <<"c2="; c2, display ();
Cout <<"c3="; c3 . display ();
}

```

उक्त प्रोग्राम का आउटपुट निम्न होगा।

```

c1 =25+j3.5
c2 =1.6+2.7
c3 =4.1+j6.2

```

उक्त प्रोग्राम में फंक्शन **operator + ()** को विशेष ध्यान से देखने पर हम समझ सकते हैं की ऑपरेटर ओवरलोडिंग प्रक्रिया किस प्रकार कार्यान्वित की गई है।

```

complex complex :: operator + (complex ()
{
Complex temp;
Temp.x=x+c.x;

```

```
Temp.y=y+cy;
Return (temp);
}
```

प्रोग्राम कमे केवल **complex** प्रकार का आर्गुमेंट ग्रहण होता है।

बयनरी ऑपरेटर की ओवरलोडिंग में बांयी तरफ के **operand** का प्रयोग **operator** फंक्शन को प्रारंभ करने के लिये होता है जबकि दायीं तरफ के **operand** का प्रयोग फंक्शन में आर्गुमेंट स्थिर करने के लिये होता है।

## 8.5 ऑपरेटर का प्रयोग करते हुए स्ट्रिंग प्रबंधन

ANSIC के अंतर्गत स्ट्रिंग के प्रबंधन के लिये कोई ऑपरेटर उपलब्ध नहीं है लेकिन C++ के अंतर्गत हम ऑपरेटर को इस प्रकार परिभाषित कर सकते हैं जिससे उन्हें स्ट्रिंग प्रबंधन के लिये प्रयोग किया जा सके। इससे निम्न तरह के आदेश प्रयोग किये जा सकते हैं जो ANSIC में संभव नहीं है।

```
String 3= string 1+string2;
If (string1> = string2)
```

स्ट्रिंग को क्लास ऑब्जेक्ट की तरह परिभाषित किया जा सकता है। तथा फिर उन्हें **built-in** प्रकार से प्रयोग किया जा सकता है। इसे एक उदाहरण से देखते हैं।

```
////[Mathmetical operations on string]
Include <string.h>
Include <iostream.h>
Class string
{
Char *p;
Int ten;
Public:
String() {len=0; p=0;}
String (const char* &);
String (const string &s);
String () {delete p;}
```

```

Friend string operator +(const string &s, const string &t);

Friend int operator <=(const string &s, const string &t);

Friend void show (const string &);

};

String ::string (const char* &)
{
    Len=strlen (&);
    P=new char [len+1];
    strcpy (p,&);
}

String ::string (const string &s)
{
    String :: string (const string &s)
    {
        Len s.len;
        P= new char [len +1];
        strcpy (p,s.p);
    }

    // ....overloading + operator....

    String operator + (const string&s, const string & t)
    {
        String temp;

        Temp len = s.len +t.len;p

        Temp .p =new char [temp.len+1];

        strcpy (temp.p, s.p);

        Strcat (temp.p, t.p);

        Return (temp);

    Main ()

    {

        String S1= "New ";

```

```
String s2 ="York";
String s3 = "Delhi";
String t1, t2, t3, t4;
T1 =s1;
T2 =s2;
T3 =s1+s2;
T4= s1 +s3;
Cout <<"int1="; show (t1);
Cout <<"int2="; show (t2);
Cout <<"/n";
Cout <<"int3="; show (t3);
Cout <<"int4= "; show (t4);
Cout <<"/n/n";
If (t3=t4);
{
Show (t3);
Cout <<"smaller than";
Show (t4);
Cout <<"/n";
```

इस प्रोग्राम का आउटपुट निम्न होगा

```
T1=New
T2=york
T3=Newyourk
T3=New Delhi
Newyourk smaller than New Delhi
```

## 8.6 ऑपरेटर ओवरलोडिंग के नियम

ऑपरेटर को ओवरलोड करने के निम्न नियम हैं।

- सिर्फ उपलब्ध (existing) ऑपरेटर को ही ओवरलोडिंग किया जा सकता है। नये ऑपरेटर का निर्माण नहीं किया जा सकता।
- ओवरलोड किये जाने वाले ऑपरेटर में कम से एक operand उपयोगकर्ता परिभाषित (user defined) होना चाहिए।
- इससे ऑपरेटर का मूल गुण परिवर्तित नहीं किया जा सकता। जैसे+ ऑपरेटर का प्रयोग दो संख्याओं को घटाने के लिये नहीं किया जा सकता।
- Friend फंक्शन का प्रयोग कुछ विशेष ऑपरेटर की ओवरलोडिंग के लिये नहीं किया जा सकता।



## 9.1 परिचय

पाइंटर सी++ की एक विशिष्ट उपयोगिता है जिसके जरिये बिना वेरियेबल को पढ़े उसका मान पढ़ा जा सकता है। ऐसा वेरियेबल का एड्रेस पढ़ कर किया जाता है। हम जानते हैं कि कंप्यूटर में नयी समझना किसी न किसी मेम में सुरक्षित की जाती है तथा प्रत्येक सेल का एक विशेष एड्रेस होता है। यह एड्रेस ही वेरियेबल तथा उस प्रोग्राम, जिसके द्वारा इस वेरियेबल को पढ़ा जाता है, के बीच संबंध स्थापित करता है। यदि एक आदेश निम्न तरह से लिखा है—

```
intQty = 100;
```

तो यह आदेश एक आंकित वेरियेबल परिभाषित करेगा जिसका नाम qty है तथा जिसका मान 100 है। यह वेरियेबल किसी विशेष एड्रेस पर सुरक्षित होगा। मान लो यह एड्रेस 2000 है, तो इसे निम्न तरह से प्रदर्शित किया जाएगा।



यह मेमोरी एड्रेस 2000 भी कंप्यूटर के किसी सेल पर सुरक्षित रहेगी यदि यह मान 5000 है तथा इस एड्रेस का नाम y है तो यह y उक्त वेरियेबल qty का पाइन्टर है। वेरियेबल x तथा y के मध्य सहसंबंध इस चित्र में प्रदर्शित है।

Variable	value	address
x	100	2000
y	2000	5000

## 9.2 वेरियेबल का एड्रेस ज्ञात करना

किसी वेरियेबल का डिस्क पर एड्रेस ज्ञात करने के लिये & ऑपरेटर का प्रयोग किया जाता है। जैसे -

```
p=&x;
```

यह आदेश **x** नाम के वेरियेबल के एड्रेस को **P** नामक वेरियेबल के अंतर्गत सुरक्षित करेगा। इसका प्रयोग किसी वेरियेबल या अरे के मान के लिये ही किया जा सकता है। किसी स्थिरांकया संपूर्ण अरे के नाम के लिये इसका प्रयोग नहीं किया जा सकता।

अब एक प्रोग्राम देखते हैं जिसमें **&** ऑपरेटर का प्रयोग किया गया है।

```
include <iostream.h>

void main()
{
    Int var1=11;
    Int var2=22;
    cout << var1;
    cout << var2;
}
```

यह प्रोग्राम दो वेरियेबल **var1** तथा **var2** परिभाषित करेगा तथा उनके प्रारंभिक मान 11 तथा 22 निर्धारित कर देगा। इसका आउटपुट निम्न हो सकता है।

Ox8fuffff4 address of var1  
Ox8f4ffff2 address of var2

हालांकि यह मान भिन्न-भिन्न कम्प्यूटर पर अलग-अलग होगा। किसी कम्प्यूटर में वेरियेबल का एड्रेस ऑपरेटिंग सिस्टम के आकार तथा मेमोरी में अनय प्रोग्राम की उपस्थिति पर निर्भर करता है।

### 9.3 पॉइन्टर वेरियेबल

किसी भी अन्य वेरियेबल की रिह पॉइन्टर वेरियेबल को भी प्रयोग से पहले परिभाषित करना आवश्यक है। वेरियेबल जो किसी सेल का एड्रेस सुरक्षित रखते हैं, पदइन्टर वेरियेबल या पॉइन्टर कहलाते हैं। पॉइन्टर वेरियेबल परिभाषित करने का सामान्य प्रारूप है।

data- type\*pointer variable name

जैसे -                      dint\*balance;

यहां \* यह प्रदर्शित करता है कि **balance** नामक वेरियेबल एक पॉइन्टर वेरियेबल है।

यहां यह ध्यान रखना आवश्यक है कि वेरियेबल **balance** के साथ प्रयुक्त पदज इस वेरियेबल का प्रकार नहीं है बल्कि उस वेरियेबल का प्रकार है जिसका **balance** एक पॉइंटर वेरियेबल है।

```
char*cptr;
```

यह सूचित करेगा कि **cptr** एक पाइन्टर वेरियेबल है जो किसी कैरेक्टर मान का पाइन्टर है।

एक उदाहरण देखते हैं-

```
#include <iostream.h>
Void main()
{
    Intvar1 =11;
    Intvar2 = 22;
    Int*ptr;
    Ptr = &var1;
    Cout <<endl<<*ptr;
    Cout <<endl<<*ptr;
```

यह प्रोग्राम **ptr** नामक एड्रेस में सुरक्षित वेरियेबल का नाम प्रिंट करेगा इसका आउटपुट होगा.

11

22

जब किसी वेरियेबल के नाम के साथ **\*** का प्रयोग किया जाता है तो इसे **indirection** ऑपरेटर कहते हैं।

## 9.4 पॉइन्टर के द्वारा वेरियेबल का मान ज्ञात करना

निम्न उदाहरण देखते हैं।

```
int rollno, *roll, n;
rollno=20010;
roll=&20010;
```



```
roll=&rollno;  
n=*roll;
```

यहां पहली लाइन दो आंकिक वेरियेबल rollno तथा n तथा एक पाइन्टर वेरियेबल \*roll परिभाषित करेगी। दूसरी लाइन आंकिक वेरियेबल rollno को एक मान प्रदान करेगी। तीसरी लाइन rollno नामक वेरियेबल के ऐड्रेस का roll नामक वेरियेबल के अंतर्गत सुरक्षित करेगी। अंत में indirections ऑपरेटर का प्रयोग किया गया है। जब यह प्रयोग किसी पॉइन्टर वेरियेबल के पहले किया जाता है तो यह उस वेरियेबल का मान प्रदर्शित करना है जिसका यह पॉइन्टर है।

## 9.5 पॉइन्टर तथा अरे

पॉइन्टर तथा अरे के मध्य अत्यंत निकट संबंध है। अरे के अवयवों को अरे के संकेतक या पाइन्टर संकेतक का प्रयोग कर ज्ञात किया जा सकता है। निम्न प्रोग्राम देखिये।

```
#include<iostream.h>  
Void main()  
{  
    Int group[5]={9,12,40,21,5};  
    For {int i=0; i<<5,i++}  
    Cout <<endl<<*(int array+i);  
}
```

उक्त प्रोग्राम में समीकरण  $*(int\ array+i)$  का अभिप्राय  $int\ array[i]$  के समकक्ष है। यदि i का मान 3 है तो समीकरण का मान  $*(int\ array+3)$  होगा। यह आदेश अरे के चौथे अवयव का मान अर्थात 21 प्रदर्शित करेगा।

## 9.6 ऑब्जेक्ट के लिये पॉइन्टर

पॉइन्टर का प्रयोग किसी क्लास द्वारा निर्मित ऑब्जेक्ट को इंगित करने के लिये हो सकता है।

निम्न प्रोग्राम में ऑब्जेक्ट के पॉइन्टर का प्रयोग प्रदर्शित हैं।

```
#include<iostream.h>  
  
class item  
{  
  
    int code;
```

```

float price;

public :

void getdata (inta, floats)
{
code =a; prive =b;
}

void show (void)
{
coout<<"Code: "<<Code <<"\n";
cout<< "price:" <<price<<"\n";
}
};

const int size =2;
main()
{
item *p = new item [size];
item *p = new item [size];
item *d = p;

int x, i;

float y;

for (i=0, i<size; i++)
{
cout <<"input code and price for item" i+1;

cin>>x>>y;

p -> getdata (x,y);

p++;
}

for (i=0; i<size; i++)
{
cout<<"item:" <<i+1 <<"\n";

```

```
d-> show();
```

```
d++;
```

```
}
```

```
}
```

इस प्रोग्राम का आउटपुट निम्न होगा।

```
input code and price for item1 40 500
```

```
input code and price for item2 50 600
```

```
item : 1
```

```
code : 40
```

```
price : 500
```

```
item : 2
```

```
code : 50
```

```
price : 60
```

## 9.7 This पॉइन्टर

यह एक विशिष्ट प्रकार का पॉइन्टर है जो उस ऑब्जेक्ट को इंगित करता है जिसके लिये जीपे फंक्शन का प्रयोग किया गया है। उदाहरण के लिये फंक्शन कॉल में जीपे पॉइन्टर को। ऑब्जेक्ट के एड्रेस पर निर्धारित कर दिया जाएगा।



**10**

अरे (Array)

---

## 10.1 परिभाषा

प्रोग्रामिंग की कई अन्य भाषाओं की तरह C++ में भी सुविधा प्रदान की गई है कि एक जैसे प्रकृति वाले वेरियेबल को अलग-अलग परिभाषित करने के स्थान पर उन्हें एक समूह के रूप में परिभाषित किया जा सकता है। यह कार्य अरे की मदद से किया जा सकता है। अरे एक ऐसी डाटा संरचना है जिसमें मूल रूप से केवल एक ही वेरियेबल परिभाषित किया जाता है तथा विभिन्न वेरियेबल को सब-स्क्रिप्ट के रूप में परिभाषित किया जाता है। जैसे यदि किसी छात्र के इस विषयों के अंक परिभाषित करना है तो दसों विषयों के रूप में अलग वेरियेबल परिभाषित करने के बजाय हम सिर्फ एक वेरियेबल **Marks(i)** परिभाषित कर सकते हैं जहां **i** के विभिन्न मानों के लिये दस अलग-अलग वेरियेबल **Marks(1), Marks(2)....** आदि परिभाषित कर सकते हैं।

अरे तथा स्ट्रक्चर वैसे तो एक जैसे ही हैं जिनमें एक ही नाम के अंतर्गत कई वेरियेबल सुरक्षित किये जाते हैं लेकिन दोनों में मूल फर्क यह है कि अरे में सुरक्षित सभी वेरियेबल एक ही प्रकार के होते हैं जबकि स्ट्रक्चर के अंतर्गत सुरक्षित वेरियेबल अलग-अलग प्रकार के ही सकते हैं। स्ट्रक्चरके वेरियेबल को उनके नाम से ही सूचित किया जाता है जबकि अरे में वेरियेबल को उनके सबस्क्रिप्ट क्रमांक से सूचित किया जाता है।

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

```
void main()
```

```
{
```

```
int age[4];
```

```
cout <<endl;
```

```
for (int j=0; j<<4; j++)
```

```
{
```

```
cout <<"Enter an age:";
```

```
cin >>age [i]
```

```
}
```

```
for(int j=0; j<<4; j++)
```

```
{
```

```
cout <<"\n You entered" <<age[i];
```

```
}
```

```
}
```

उक्त प्रोग्राम में पहले एक अरे परिभाषित की गई है जिसमें चार अवयव हैं। इसके बाद पहले FOR लूप में वेरियेबल के मान पढ़े गए हैं तथा दूसरे for लूप से उन्हें प्रिंट किया गया है।

**अरे को परिभाषित करना :** जिस तरह किसी भी वेरियेबल को प्रयोग करने से पहले उसे परिभाषित करना आवश्यक है उसी प्रकार अरे को भी उपयोग से पहले परिभाषित करना आवश्यक है। इसका सामान्य प्रारूप है।

वेरियेबल प्रकार वेरियेबल (अवयव संख्या)

जैसे `int age [4];`

यहां यह ध्यान रखना आवश्यक है कि अवयव की संख्या कोई स्थिरांक होना चाहिए।

**अरे को प्रारंभिक मान देना:** अरे के वेरियेबल को कोई पूर्व निर्धारित प्रारंभिक मान भी दिया जा सकता है।

उदाहरण के लिये

```
int V1[]={1,2,3,4};  
char V2[]={ 'a', 'b', 'c', 0};
```

जब किसी अरे को बिना किसी निश्चित आकार के परिभाषित किया जाता है तो अरे का आकार प्रारंभिक मानों की तालिका में दिये अवयवों की संख्या के आधार पर ज्ञात किया जाता है। जैसे उक्त दोनों उदाहरणों में पहला `int[4]` तथा दूसरा `char[4]` आकार का है। यदि अरे का आकार पहले से निर्धारित है तो उससे अधिक पूर्व निर्धारित मान देने पर अशुद्धि प्रदर्शित होगी। जैसे—

```
char V3[2]={ 'a', 'b', 0};
```

गलत है क्योंकि अरे का आकार दो ही है जबकि 3 मान प्रारंभिक तौर पर दिये गये हैं।

यदि अरे के आकार से कम मान दिये गये हैं तो शेष अवयवों के प्रारंभिक मान की शून्य मान लिया जाता है।

जैसे —

```
int V5[8]={1,2,3,4};
```

आदेश `int V5[]={1,2,3,4,0,0,0,0};` के समकक्ष है।

**कैरेक्टर अरे:** C++ में सभी स्ट्रिंग वेरियेबल के मानों को अरे के जरिये ही पढ़ा जा सकता है। किसी स्ट्रिंग के कैरेक्टर की कुल वास्तविक संख्या उसमें प्रदर्शित होने वाले कैरेक्टर से एक ज्यादा होती है क्योंकि प्रत्येक स्ट्रिंग के अंत में एक `nul` character होता है जो स्ट्रिंग का अंत प्रदर्शित करता है। जैसे यदि स्ट्रिंग का मान 'Bohr' है तो इसके कैरेक्टर की वास्तविक संख्या पांच होगी।

**अरे को पढ़ना:** पहले लिखे प्रोग्राम में अरे के प्रत्येक मान को पढ़ने के लिये समीकरण है:—

```
age [j];
```

इसके अंतर्गत अरे का नाम, फिर ब्रैकेट का प्रारंभ, फिर एक वेरियेबल जो अरे का कोई एक अवयव इंगित करता है, सम्मिलित रहता है। जैसे यदि अरे का समीकरण Age [J] है तो J के पहले मान के लिये समीकरण Age [1] दूसरे के लिये Age [2] आदि होंगे।

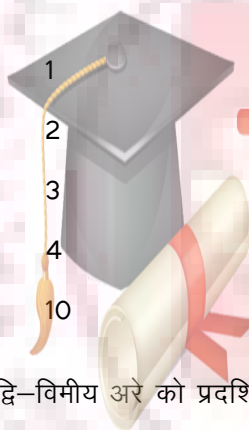
ब्रैकेट के अंदर लिखे स्थिरांक को अरे का इन्डेक्स कहते हैं।

## 10.2 बहुविमीय अरे

अभी तक हमने एक विमीय अरे के विषय में पढ़ा जिसमें केवल एक कॉलम होता है। एक ही वेरियेबल के जरिये पूरी तालिका को प्रदर्शित किया जा सकता है। लेकिन अरे का प्रयोग बहुविमीय प्रारूप में भी हो सकता है। उदाहरण के लिये, यदि हमें 10 विद्यार्थियों के 5 अलग-अलग विषयों में अंक लिखने हैं तो उनका प्रारूप कुछ इस प्रकार हो सकता है—

विद्यार्थी इस प्रकार हो सकता है—

विद्यार्थी विषय 1 विषय 2 विषय 3



# Tayari Jeet ki



इस उदाहरण में एक द्वि-विमीय अरे को प्रदर्शित किया गया है।

```
#include<iostream.h>
#include<iomanip.h>
const int DISTRICTS =4;
const int MONTHS =3;
void main()
{
    int d, m;
    float sales [DISTRICTS] [MONTHS];
    cout <<endl;
    for (d=0; d<<DISTRICTS, d++)
        //get array values
```

```

for (m=0; m<<MONTHS; m++)
{
    cout <<"Enter sales for district" <<d+1;
    cout <<"month" <<m+1<<":";
    cin>>sales[d][m];
    //put number in array
}
cout << "\n\n";
cout <<"Month\n";
cout <<"123";
for (d=0; d<<DISTRICTS;d++)
{
    cout <<"\nDistrict"<<d+1;
    for (m=0; m<<MONTHS;m++) //display array values
        cout <<,setiosflags (ios:: fixed) //not exponential
        <<setiosflags (ios::showpoint) //always use point
        <<setprecision(2) //digits to right
        <<setw (10) //field width
        <<sales[d][m]; //get number for array
    }
}

```

**द्विविमीय अरे को परिभाषित करना :** एक द्विविमीय अरे को परिभाषित करने का सामान्य प्रारूप निम्न हो सकता है—

```
Name [m][n];
```

जहां m रो की संख्या तथा n कॉलम की संख्या है **Name** उस अरे का नाम है तथा **type** अरे के अवयवों के प्रकार है। जैसे `int sales [2] [3];`

एक अरे जोड़ी जिसका नाम `sales`, रो की संख्या 2 तथा कॉलम की संख्या 3 होगी।

यहां यह ध्यान रखना आवश्यक है कि अरे के प्रत्येक इंडेक्स का स्वयं का एक ब्रैकेट होता है।

`int sales [2,3]` आदेश गलत होगा।



**द्वि-विमीय अरे को पढ़ना :** जिस तरह द्विविमीय अरे को परिभाषित करने के लिये दो इंडेक्स की जरूरत पड़ती है। उसी प्रकार उसके किसी एक अवयव को पढ़ने के लिये भी दो इंडेक्स की आवश्यकता पड़ेगी। पूर्व के उदाहरण में यदि हमें छात्रों की तालिका में से दूसरे छात्र के चौथे विषय के अंक को इंगित करना है तो `marks [2], [4];` आदेश देना होगा।

### 10.3 अरे के अंतर्गत **Structure** का प्रयोग

सधारण डाटा प्रकार के अलावा **Structure** को भी अरे के अवयवों की तरह प्रयोग किया जा सकता है। निम्न प्रोग्राम में हम **structure** की एक अरे देखते हैं।

```
#include<<iostream.h>>
const int SIZE = 4
struct part
{
    Int modelumber;
    Int partnumber;
    Float cost;
}
void main()
{
    int n;
    part apart [SIZE];
    for (n=0; n<<SIZE; n++)
    {
        cout <<endl;
        cout <<"Enter model number:";
        cin >> apart[n].modelumber;
        cout <<"Enter part number:";
        cin >> apart [n].partnumber;
        cout << "Enter cost:";
        cin >> apart [n].cost;
    }
```


```

for (n=0; n<<SIZE; n++)
{
cout <<"\nModel" <<aprt[n].modelnumber;
cout <<"part" <<apart[n].partnumber;
cout <<"cost" << apart [n].cost;
}
}

```

उपयोगकर्ता द्वारा मॉडल क्रमांक पार्ट, क्रमांक तथा प्रत्येक पार्ट की कीमत प्रविष्ट की जाती है। प्रोग्राम के जरिये इस डाटा को Structure के रूप में सुरक्षित कर लिया जाता है। प्रोग्राम चार विभिन्न पार्ट के लिये डाटा ग्रहण करता है व वांछित सूचना प्रिंट करता है।

प्रोग्राम का आउटपुट कुछ इस प्रकार होगा।



```

Enter model number      : 44
Enter pat number        : 4954
Enter cost               : 133.45
Enter model number      : 44
Enter part number       : 8431
Enter cost              : 97.59
Enter model number      : 77
Enter part number       : 9343
Enter cost              : 109.99
Enter model number      : 77
Ente part number       : 4297
Enter cost              : 3456.55

Model 44 part 4954 cost 133.45
Model 77 part 8431 cost 97.59
Model 77 part 9343 cost 109.99
Model 77 part 4297 cost 3456.55

```

अरे के अंतर्गत **structure** की साधारण वेरियेबल की ही तरह परिभाषित किया जाता है केवल फर्क यह है कि अरे का प्रकार **part** ही यह सूचित करता है कि यह एक ज्यादा क्लिष्ट प्रकार है।

ऐसे डाटा को जो किसी **structure** के सदस्य हों, जो स्वयं ही किसी अरे का अवयव हो, पढ़ने के लिये निम्न प्रारूप है—

**Apart[n]—modelnumber**

यह आदेश **structure** के **modelnumber** सदस्य को इंगित करता है। साथ ही **aststructure** स्वयं **apart** नामक अरे का एक अवयव है।

**कैरेक्टर अरे :** C++ के अंतर्गत किसी भी कैरेक्टर स्ट्रिंग को अरे के रूप में ही प्रदर्शित किया जा सकता है। किसी स्ट्रिंग के अंतर्गत कैरेक्टर की संख्या, प्रकट रूप में प्रदर्शित होने वाले कैरेक्टर की तुलना में एक अधिक होती है क्योंकि इसमें एक अतिरिक्त **null character** होता है जो अरे की समाप्ति सूचित करता है।

```
char name [20];
```

**पॉइन्टर तथा अरे :** C++ के अंतर्गत पॉइन्टर तथा अरे में अत्यंत निकट संबंध है। अरे के नाम को पॉइन्टर की तरह प्रयोग किया जा सकता है। यह पॉइन्टर अरे के पहले अवयव को सूचित करता है।

```
int v[] = {1,2,3,4};
```

```
int*p1 = v;
```

```
int*p2 = &v[0];
```

```
int*p2 = &v[4];
```

