



सी प्लस प्लस प्रोग्रामिंग

Hindilearn द्वारा प्रकाशित

सी प्लस प्लस

सी प्लस प्लस ट्यूटोरियल के बारे में

सी प्लस प्लस यह सी प्रोग्रामिंग के जैसे ही है लेकिन सी प्लस प्लस में कुछ अतिरिक्त सुविधाएं हैं जिससे हम प्रोग्राम्स को आसानी और सुरक्षा प्रदान कर सकते हैं। हम सी प्लस प्लस में ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग के साथ-साथ और भी अतिरिक्त ट्यूटोरियल पढ़ेंगे जैसे कि exception handling, templates, namespaces और आदि।

ट्यूटोरियल किसे पढ़ना चाहिए

जिसे अपना प्रोग्रामिंग का ज्ञान और भी बेहतर करना है उसे इस ट्यूटोरियल को पढ़ना चाहिए। जिसके पास प्रोग्रामिंग का ज्ञान हो या ना हो वह भी इस ट्यूटोरियल को पढ़ सकता है।

सी प्लस प्लस सिखने के लिए अपेक्षित

जो सी प्लस प्लस सीखना चाहता है तो उसके पास Computer (Desktop, CPU, Keyboard) या laptop और सी प्लस प्लस के कौनसे भी IDE (Integrated Development Environment) की जरूरत होती है। इस PDF File को Run करने के लिए किसी भी Browser या PDF Reader का उपयोग करें।

अपने अधिकार

जो भी सामग्री आप इस PDF के माध्यम से उपयोग में ला रहे हैं वो Hindilearn.in की संपत्ति है जैसे कि छायाचित्र और पाठ। अगर आप इसकी व्यावसायिक रूप में नक़ल करने का प्रयास करेंगे तो आप पर राइट लॉज़ के अनुसार कार्रवाई भी हो सकती है।

संपर्क

अगर आपको कोई गलत जानकारी या अशुद्ध लेखन इस ट्यूटोरियल के माध्यम से प्राप्त होता है तो आप हमें help@hindilearn.in इस पते पर संपर्क कर सकते हैं।

C++ BASICS	11
C++ INTRODUCTION	11
<i>Introduction of C</i>	11
<i>OOPs Concept</i>	12
<i>Reserved Keywords</i>	13
<i>Structure of Program</i>	15
C++ DATA TYPES	18
- Basic Data Types	18
<i>Integer</i>	18
<i>Character</i>	19
<i>Float</i>	21
<i>Double</i>	22
<i>Void</i>	23
<i>Boolean</i>	23
<i>Wide Character</i>	24
- Derived Data Types	24
<i>Array</i>	24
<i>Pointer</i>	24
- User-defined Data Types	24
<i>Structure</i>	24
<i>Pointer</i>	24
C++ CONSTANTS & VARIABLES	25
<i>Constants</i>	25
<i>What is Variable</i>	29
<i>Variable Scopes</i>	33

C++ OPERATORS

35

Arithmetic Operators

35

Relational Operators

36

Logical Operators

37

Bitwise Operators

38

Assignment Operators

40

Increment and Decrement Operators

41

Conditional or Ternary Operators

42

C++ LOOPS

43

While Loop

43

Do While Loop

44

For Loop

45

Nested Loop

46

C++ CONTROL STATEMENTS

47

If Statement

47

If Else Statement

47

Else If Statement

48

Switch Case Statement

50

Break Statement

51

Continue Statement

52

Goto Statement

53

C++ ARRAYS

55

Array Introduction

55

Single or One Dimensional Array

55

Multi Dimensional Array

57

C++ STORAGE CLASSES	59
<i>Automatic</i>	59
<i>External</i>	60
<i>Register</i>	61
<i>Static</i>	62
C++ INPUT AND OUTPUT	63
<i>Input and Output Introduction</i>	63
<i>cout Console Output</i>	63
<i>cin Console Input</i>	64
<i>cerr Console Error</i>	65
C++ FUNCTIONS	66
<i>Function Introduction</i>	66
<i>Predefined or In built Function</i>	66
<i>User defined Function</i>	67
<i>Call By Value And Reference</i>	70
C++ POINTERS	74
<i>Pointer Introduction</i>	74
<i>Referencing Operator</i>	75
<i>Dereferencing Operator</i>	76

C++ STRINGS

77

What is String

77

String Library Functions

79

<i>strcat</i>	80	<i>strncpy</i>	89
<i>strchr</i>	81	<i>strnset</i>	90
<i>strcmp</i>	82	<i>strchr</i>	91
<i>strcmpi</i>	83	<i>strrev</i>	92
<i>strcpy</i>	84	<i>strrstr</i>	93
<i>strdup</i>	85	<i>strset</i>	94
<i>strlen</i>	86	<i>strsr</i>	95
<i>strlwr</i>	87	<i>strupr</i>	96
<i>strncat</i>	88		

C++ STRUCTURE

97

Introduction of Structure

97

Structure using Pointer

102

Array of Structure

103

C++ UNION

106

What is Union

106

Union using Pointer

110

C++ PREPROCESSORS

111

Preprocessors Introduction

111

Macro #define

111

Predefined Macros

112

File Inclusion #include

113

Conditional Compilation

114

Other Directive #undef

120

C++ FILE HANDLING

121

File Introduction

121

File Uses

121

File Modes

122

Opening File

122

Closing File

123

Check File Open or not!

123

Read Data from a File

124

Write Data on a File

125

get()

125

put()

127

write()

128

getline()

130

C++ RECURSION

132

C++ COMMAND LINE ARGUMENT

134

C++ OBJECT-ORIENTED PRORAMMING

C++ CLASSES AND OBJECTS	138
C++ INHERITANCE	141
<i>Introduction of Inheritance</i>	141
<i>Single Inheritance</i>	141
<i>Multilevel Inheritance</i>	144
<i>Multiple Inheritance</i>	146
<i>Hierarchical Inheritance</i>	148
<i>Hybrid Inheritance</i>	150
C++ POLYMORPHISM	154
<i>Introduction of Polymorphism</i>	154
<i>Function Overloading</i>	155
<i>Operator Overloading</i>	157
<i>Virtual Function</i>	162
C++ METHOD OVERRIDING	166
C++ CONSTRUCTOR AND DESTRUCTOR	169
<i>Constructor</i>	169
<i>Destructor</i>	175
C++ ABSTRACTION	176
C++ DATA ENCAPSULATION	177

C++ ADVANCED

C++ NAMESPACES 179

Introduction for Namespaces 179

Nested Namespace 181

Anonymous or Unnamed Namespace 182

C++ DYNAMIC MEMORY ALLOCATION 183

Introduction 183

New Operator 184

Delete Operator 186


C++ TEMPLATES 189

Introduction of Templates 189

Class Templates 190

C++ FRIEND FUNCTIONS 192

C++ EXCEPTION HANDLING 194



विभाग १

C++

Basics

Introduction

Introduction and History of C++

Introduction of C++

C++ Programming Language ये OOP(Object-Oriented Programming) Language है, इससे पहले भी 'Simula' ये भी OOP Language बनाई गयी थी |

C++ में OOps concept के साथ कई features है जैसे कि,

- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

C++ Programming Language ये Middle Level Language है |

C++ Programming को Windows, Linux, Mac OS या आदि Operating Systems पर चलाया जाता है |

History of C++

C++ Programming Language को बनाने की शुरुआत 1979 में हुई |

C Programming में ही कुछ बदल करके 'C with Classes' ये C का concept आया |

'C with Classes' के साथ और भी कई concepts को बनाकर Bjarne Stroustrup ने 1983 में AT & T's Bell Labs में C++ Programming का अविष्कार किया गया |

Uses of C++

C++ Programming Language का इस्तेमाल Computer Software बनाने के लिए किया जाता है |

Computer Software के साथ-साथ Drivers, Computer Hardwares, servers के लिए भी इस्तेमाल किया जाता है |

C++ OOPs Concept

Basic Concepts of OOP(Object-Oriented Programming)

OOP ये Method Classes और Objects पर निर्धारित होता है ।

OOP ये एक ऐसी concept है, जिसमे Object के data और function का data structure होता है ।

C++ ये एक Procedural(C Programming जैसी) और OOP Language भी है ।

नीचे OOP के कुछ concepts short में दिए हुए हैं ।

Class : Class का जो object उससे related उसका behaviour, properties या attributes को define किया जाता है ।for eg. अगर कोई Animal है । तो class में उसका behavior, उसके body के parts और उनकी संख्या इनको define किया जा सकता है ।

Object : Object को class पर create किया जाता है । Object के कई नाम हो सकते हैं, जो उसके data members से related होते हैं ।

Inheritance : Inheritance में एक मुख्य class की attributes वो अपने sub-classes को inherit करता है । Inheritance में मुख्य class को base class या parent class कहते हैं और उसके अन्दर या उसके subclasses को derived class या child class कहते हैं ।

Polymorphism : Polymorphism मतलब एक ही form में अनेक form होते हैं । इसमे same नाम के member function को अलग -अलग parameters होते हैं । example में एक shape के द्वारा square, Circle, Triangle इन सभी shapes को draw किया जा सकता है ।

Abstraction : Abstraction में कुछ जरूरत के हिसाब से important data को दिखाया जाता है और कुछ data या internal processes को hide किया जाता है ।

Encapsulation : Encapsulation data और class को combine करके एक class के अन्दर रखा जाता है । ये 'Data Abstraction' भी कहलाया जाता है ।

C++ Reserved Keywords

Reserved Keywords for C++ Programming			
asm	else	new	this
auto	enum	operator	throw
bool	explicit	private	true
break	export	protected	try
case	extern	public	typedef
catch	false	register	typeid
char	float	reinterpret_cast	typename
class	for	return	union
const	friend	short	unsigned
const_cast	goto	signed	using
continue	if	sizeof	virtual
default	inline	static	void
delete	int	static_cast	volatile
do	long	struct	wchar_t
double	mutable	switch	while
dynamic_cast	namespace	template	

C++ Keywords	Description
asm	Assembly Language लिखने के लिए इस्तेमाल किया जाता है ।
auto	Storage class का एक प्रकार है । जो Local variable के लिए भी इस्तेमाल होता है ।
bool	boolean variable declare करने के लिए इस्तेमाल होता है । for eg. True और False
break	Loops या statements को break करने के लिए इस्तेमाल होता है ।
case	Switch case statement को इस्तेमाल करने के लिए होता है ।
catch	throw से exceptions handle करने के लिए इस्तेमाल होता है ।
char	character variable declare करने के लिए इस्तेमाल होता है ।
class	classes को declare करने के लिए इस्तेमाल होता है ।
const	Const ये एक स्थिर variable के लिए इस्तेमाल होता है ।
const_cast	Constant variable को cast किया जाता है ।
continue	Loop को iterate किया जाता है ।
default	Switch case statement के लिए इस्तेमाल होता है ।
delete	Dynamic memory allocation के लिए इस्तेमाल होता है ।
do	एक loop का प्रकार है । जिसके साथ while loop को इस्तेमाल किया जाता है ।
double	floating-point data-type है ।

dynamic_cast	pointer के साथ इस्तेमाल होता है ।
else	if के साथ statement को इस्तेमाल किया जाता है ।
enum	Enumeration data type का 'Keyword' है ।
explicit	Constructor convert करने के लिए इस्तेमाल होता है ।
export	template की definition को export किया जाता है ।
extern	Storage class का एक प्रकार है । जिसका scope global होता है ।
false	Boolean की एक value है ।
float	Floating-point variable को declare करने के लिए इस्तेमाल होता है ।
for	Loop का एक प्रकार है ।
friend	non-member function; private data में access करने के लिए इस्तेमाल होता है ।
goto	एक statement है , जिसमें label होता है ।
if	एक statement है । जिससे condition सही है या गलत इसका पता चलता है ।
inline	function के लिए इस्तेमाल होता है ।
int	integer variable को declare करने के इस्तेमाल होता है ।
long	long integer variable को declare करने के इस्तेमाल होता है ।
mutable	एक storage class का प्रकार है ।
namespace	same identifiers(variables, functions, classes) अलग-अलग बताने के लिए इस्तेमाल होता है ।
new	Dynamic memory allocation के लिए इस्तेमाल होता है ।
operator	Overloaded operators को declare करने के लिए इस्तेमाल होता है ।
private	Class के private members को declare करने के लिए इस्तेमाल होता है ।
protected	Class के protected members को declare करने के लिए इस्तेमाल होता है ।
public	Class के public members को declare करने के लिए इस्तेमाल होता है ।
register	एक storage class का प्रकार है ।
reinterpret_cast	Variable का cast-type change करने के लिए इस्तेमाल होता है ।
return	function के लिए इस्तेमाल होता है ।
short	short integer variable को declare करने के इस्तेमाल होता है ।
signed	एक variable modifier है ।
sizeof	variable का size return करने के लिए इस्तेमाल होता है ।
static	एक storage class का प्रकार है ।
static_cast	एक type_conversion है ।
struct	structure को define या declare करने के लिए इस्तेमाल होता है ।
switch	एक statement है ।
template	Generic Program को लिखने के लिए इस्तेमाल होता है ।
this	pointer को current object पर लाता है ।
throw	Exception Handling के लिए इस्तेमाल होता है ।
true	Boolean की एक value है ।

try	Exception Handling के लिए इस्तेमाल होता है ।
typedef	data type को alias_name देने के लिए इस्तेमाल होता है ।
typeid	Object का वर्णन किया जाता है ।
typename	Class का alternative है ।
union	ये keyword अपने members को एक ही memory location पर assign किया जाता है ।
unsigned	unsigned integer variable को declare करने के लिए इस्तेमाल किया जाता है ।
using	namespace के साथ इस्तेमाल किया जाता है ।
virtual	Runtime polymorphism के लिए इस्तेमाल होता है ।
void	ये कोई भी value return नहीं करता है ।

C++ Structure of Program

Program का Structure

Preprocessor	#include <iostream.h>
Standard Namespace	using namespace std;
return_type & main function	int main()
Opening curly brace	{
ostream class object	cout<<"Hello World!";
return value to main function	return 0;
Closing curly brace	}

ये Program C++ Programming में सबसे पहला और आसान Program है । ऊपर दिया हुआ Program सात भाग में बटा हुआ है ।

- 1.Preprocessor
- 2.Standard Namespace
- 3.main function
- 4.{
- 5.cout<<"Hello World!";
- 6.return 0;
- 7.}

1.Preprocessor : Program में सबसे पहले preprocessors/header को लिखा जाता है । ये preprocessors अलग-अलग काम के लिए विभाजित किये हुए है, for eg. iostream.h में cin और cout आते है । conio.h में getch function आता है । और भी कुछ preprocessors है , जो Library functions में दिखाएंगे ।

2. **using namespace std;** : ये एक Standard command library है | Program में input/Output के लिए इसका इस्तेमाल होता है | अगर इसका इस्तेमाल नहीं होता, तो cout और cin को कोई scope नहीं रह जाता | Program को बिना using namespace std; से इस्तेमाल किया जाता है | पर cout<< की जगह std::cout<< को लिखना पड़ता है |

3. **int main()** : यहाँ पर main function का return_type integer है | main function का return_type void भी लिखा जाता है पर void कोई भी value return नहीं करता | Program की शुरुआत main() function से होती है |

4. **{** : हर function के codes या statements को curly brace open होने के बाद लिखा जाता है |

5. **cout<<"Hello World!";** : cout को insertion operator(<<) के साथ लिखे हुए statement को output में print किया जाता है | इस statement को दो Double Quotes (" ") के अंदर लिखा जाता है |

6. **return 0;** : return 0 ये Program को बंद करने की अनुमति देता है | ये '0' main function को return करता है |

7. **}** : यहाँ पर } इस curly brace से main function को close किया है |

C++ Hello Program using namespace

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
cout<<"Hello World!";
return 0;
}
```

Output :

```
Hello World!
```

C++ Hello Program without using namespace

Source Code :

```
#include <iostream.h>
int main(){
std::cout<<"Hello World!";
return 0;
}
```

Output :

```
Hello World!
```


Program को Run कैसे करे ?

- Turbo C / C++ Download करे |
- Download किये हुए Turbo C / C++ को अपने Computer में install करे |
- उस Application को Open करके C Hello का Program लिखे |
- Program लिखने के बाद File पर जाकर save या F2 दबाया तो एक dialog box खुल जाएगा |
- Dialog Box खुलने के बाद उसे कोई भी नाम देकर उसे .cpp का extension दे | for eg. hello.cpp
- Save करने के बाद उसे F9 से Compile करे |
- अगर Program बिना error का हो तो CTRL+F9 से करे |

Data Types

- Basic Data Types

C++ Integer

- Integer Data Type में variable को declare करने के 'int' keyword का इस्तेमाल करते हैं।
- Integer Data Type सभी numeric values को store कर सकता है।
- Integer Data Type 2, 4 और 8 bytes के हो सकते हैं।
- अगर Computer का Processor 16-bit हो तो int का size 2 Bytes होता है।
- अगर Computer का Processor 32-bit हो तो int का size 4 Bytes होता है।
- अगर Computer का Processor 64-bit हो तो int का size 8 Bytes होता है।

Date Type	Storage Size	Range
int (16-bit)	2 Bytes	-32,768 to 32,767
int (32-bit)	4 Bytes	-2,147,483,648 to 2,147,483,647
int (64-bit)	8 Bytes	-9,223,372,036,854,775,807 to 9,223,372,036,854,775,807
unsigned int (16-bit)	2 Bytes	0 to 65,535
short int	2 Bytes	-32,768 to 32,767
unsigned short int	2 Bytes	0 to 65,535
signed short int	2 Bytes	-32,768 to 32,767
long int	4 Bytes	-2,147,483,647 to 2,147,483,647
unsigned long int	4 Bytes	0 to 4,294,967,295
signed long int	4 Bytes	-2,147,483,648 to 2,147,483,647

sizeof इस keyword से int Data Types के size का पता करे |

Source Code :

```
#include <iostream.h>
using namespace std;
int main() {
    cout<<"int(16-bit) storage size: "<<sizeof(int)<<endl;
    cout<<"int(32-bit) storage size: "<<sizeof(int)<<endl;
    cout<<"int(64-bit) storage size: "<<sizeof(int)<<endl;
    cout<<"unsigned int storage size: "<<sizeof(unsigned int)<<endl;
    cout<<"short int storage size: "<<sizeof(short int)<<endl;
    cout<<"unsigned short int storage size: "<<sizeof(unsigned short int)<<endl;
    cout<<"signed short int storage size: "<<sizeof(signed short int)<<endl;
    cout<<"long int storage size: "<<sizeof(long int)<<endl;
    cout<<"unsigned long int storage size: "<<sizeof(unsigned long int)<<endl;
    cout<<"signed long int storage size: "<<sizeof(signed long int)<<endl;
    return 0;
}
```

Output :

```
int(16-bit) storage size: 2
int(32-bit) storage size: 4
int(64-bit) storage size: 8
unsigned int storage size: 4
short int storage size: 2
unsigned short int storage size: 2
signed short int storage size: 2
long int storage size: 4
unsigned long int storage size: 4
signed long int storage size: 4
```

C++ Character

- Character Data Type में variable को declare करने के 'char' keyword का इस्तेमाल करते है |
- सिर्फ एक ही character को declare कर सकते है | for eg. 'H'
- अगर multiple character मतलब पूरे एक string को print करना हो तो double quotes("") का इस्तेमाल करते है | for eg. char arr[10] = "Hello";
- Character Data Type 1 byte का होता है |

Source Code :

```
#include <iostream.h>
using namespace std;
int main() {
char str1='H'; // declare variable in single quotes
//char str2[10]='Hello'; // Get Warning Error
char str3[10]="Hello";
    cout<<"Single quoted Print Character : "<<str1<<endl;
    //cout<<"Single quoted Print String : "<<str2<<endl; //Get Warning Error
    cout<<"Double quoted Print Character : "<<str3<<endl;
return 0;
}
```

Note : ऊपरवाला Program Warning Error देगा | User को single quotes और double quotes के बिच का फर्क जानने के लिए ऊपरवाला Program दिया है |

Output :

```
Single quoted Print Character : H
Double quoted Print Character : Hello
```

Date Type	Storage Size	Range
char	1 Byte	-128 to 127
unsigned char (16-bit)	1 Byte	0 to 255
signed char	1 Byte	-128 to 127

sizeof इस keyword से char Data Types के size का पता करे |

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
    cout<<"char storage size : "<<sizeof(char)<<endl;
    cout<<"unsigned char storage size : "<<sizeof(unsigned char)<<endl;
    cout<<"signed char storage size : "<<sizeof(signed char)<<endl;
return 0;
}
```

Output :

```
char storage size : 1
unsigned char storage size : 1
signed char storage size : 1
```

C++ Float

- Floating-point Data Type में variable को declare करने के 'float' keyword का इस्तेमाल करते हैं।
- Floating-point Data Type 4 bytes का होता है।

Source Code :

```
#include <iostream.h>
using namespace std;
int main() {
    float a=5, b=3.145;
    cout<<"Value of Float variable : "<<a<<endl;
    cout<<"Value of b : "<<b<<endl;
    return 0;
}
```

Output :

```
Value of Float variable : 5
Value of b : 3.145000
```

Date Type	Storage Size	Range	Digits of Precision
float	4 Bytes	1.2E-38 to 3.4E+38	6

sizeof इस keyword से float Data Types के size का पता करे | निचे दिया हुआ source code देखे |

Source Code :

```
#include <iostream.h>
using namespace std;
int main() {
    cout<<"Floating-point Storage size : "<<sizeof(float);
    return 0;
}
```

Output :

```
Floating-point Storage size : 4
```

C++ Double

- Double Data Type में variable को declare करने के 'double' keyword का इस्तेमाल करते हैं।
- Double Data Type 8 bytes का होता है।
- Double और Floating-point Data Type में कोई फर्क नहीं है।

Source Code :

```
#include <iostream.h>
using namespace std;
int main() {
double a=5.45;
    cout<<"Value of Double variable : "<<a;
return 0;
}
```

Output :

Value of Double variable : 5.45

Date Type	Storage Size	Range	Digits of Precision
double	8 Bytes	2.3E-308 to 1.7E+308	15
long double	10 Bytes	3.4E-4932 to 1.1E+4932	19

sizeof इस keyword से double Data Types के size का पता करे | निचे दिया हुआ source code देखे |

Source Code :

```
#include <iostream.h>
using namespace std;
int main() {
    cout<<"Double Storage size : "<<sizeof(double);
return 0;
}
```

Output :

Double Storage size : 8

C++ Void

- void मतलब null value |
- void में कोई value नहीं होती |
- ये data type function और function के parameter / argument के लिए इस्तेमाल करते हैं |
- ये कोई value return नहीं करता |

Source Code :

```
#include <iostream.h>
using namespace std;
void hello(void); // function with no return value and parameter
main()
{
    hello();
}
void hello(void)
{
    cout<<"Hello World";
}
```

Output :

```
Hello World
```

C++ Boolean

- Boolean के लिए सिर्फ दो constant values होती हैं |
- अगर true होता है, तो 1 return किया जाता है और false होता है, तो 0 return किया जाता है |

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
    bool value = true;
    cout<<"value : "<<value<<endl;
    value = false;
    cout<<"Changed value is "<<value;
    return 0;
}
```

Output :

```
value : 1
Changed value is 0
```

C++ Wide Character

- `wchar_t` ये wide characters होते हैं।
- ये data type 2 bytes या 4 bytes का होता है।
- `wchar_t` ये Unicode के लिए इस्तेमाल किया जाता है।

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
    wchar_t str[] = L"Hello";
    wcout << "Wide character value : "<<str<<endl; //wcout is used for wide
    stream(Unicode)
    cout << "Size of the wide char : " <<sizeof(wchar_t); //cout is used for standard
    stream(Unicode)
    return 0;
}
```

Output :

```
Wide character value : Hello
Size of the wide char : 2
```

C++ Array - Derived Data Type

Array के लिए यहाँ क्लिक करें।

C++ Pointer - Derived Data Type

Pointer के लिए यहाँ क्लिक करें।

C++ Structure - User-defined Data Type

Structure के लिए यहाँ क्लिक करें।

C++ Union - User-defined Data Type

Union के लिए यहाँ क्लिक करें।

C++ Constants

- Constant की value fixed होती है ।
- Constant किसी भी data type का हो सकता है ।
- Constant को Literals भी कहते है ।
- Constant Pointer भी होता है ।

Syntax

```
const data_type variable_name = value (optional) ;
```

Constant के प्रकार

- Integer Constant
- Decimal Constant
- Octal Constant
- Hexadecimal Constant
- Floating-point / Real Constant
- Character Constant
- String Constant
- Preprocessor

Constant Types	with Examples
Integer	eg. 2, 10, -2
Decimal (Integer)	eg. 2, 10, -2
Octal (Integer)	eg. 02, 010
Hexadecimal (Integer)	eg. 0x12, 0x1f
Floating-point/Real	eg. -2.4, 4.8
Character	eg. 'i', 'j'
String	eg. "Hello", "Hi"
Preprocessor	eg. #define a 5

Integer Constant Types:

- Decimal Integer Constant
- Octal Integer Constant
- Hexadecimal Integer Constant

Integer Constants

- Integer Constant normal Variable की तरह काम करता है ।
- Integer Constant positive (+) या negative (-) हो सकता है ।
- Integer Constant की Range -32768 से 32767 तक होती है ।

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
    const int num = 5; // integer Constant
    cout<<"integer constant value : "<<num;
    return 0;
}
```

Output :

```
integer constant value : 5
```

Character Constants

- Character Constant normal Variable की तरह काम करता है ।
- Character Constant सिर्फ single character लेता है ।
- Escape Sequences के साथ भी character constant इस्तेमाल किया जाता है ।

Escape Sequences	Explanation
\'	Single Quotation Mark
\"	Double Quotation Mark
\\	Backslash
\?	Question Mark
\a	Audible Bell
\b	Backspace
\f	Form Feed
\n	New line

\r	Carriage Return
\h	Horizontal Tab
\v	Vertical Tab

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
const char ch = 'H'; // character constant
const char escape[] = "Hello\tWorld"; // Escape sequence - Horizontal Tab and string constant
    cout<<"Character constant : "<<ch<<endl;
    cout<<"String constant : "<<escape<<endl;
return 0;
}
```

Output :

```
Character constant : H
String constant : Hello    World
```

Floating-point Constant

- Floating-point Constant normal float variable की तरह ही काम करता है ।
- Floating-point Constant में Decimal point (.) होता है ।
- अगर Floating-point Constant की value integer type की हो तो वो Decimal point (.) लेता है ।

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
const float num1 = 5;
const float num2 = 3.525984; // Floating-point constant
    cout<<"Floating-point constant : "<<num1<<endl;
    cout<<"Floating-point constant : "<<num2;
return 0;
}
```

Output :

```
Floating-point constant : 5
Floating-point constant : 3.525984
```

String Constant

- String Constant की value Double Quotation Mark (" ") के अंदर लिखी जाती है ।
- String Constant Single और Multiple characters लेता है ।
- String Constant Escape Sequences के साथ भी इस्तेमाल करते है ।

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
    const char str1 []= "H"; // String Constant with single character
    const char str2 []= "Hello World"; // Normal String constant
    const char str3 [] = "Hello\nWorld"; // String Constant with Escape Sequence
    cout<<"String Constant with single character : "<<str1<<endl;
    cout<<"Normal String constant : "<<str2<<endl;
    cout<<"String Constant with Escape Sequence : "<<str3<<endl;
    return 0;
}
```

Output :

```
String Constant with single character : H
Normal String constant : Hello World
String Constant with Escape Sequence : Hello
World
```

Preprocessor Constant

- Preprocessor के साथ भी Constant value रख सकते है ।

Source Code :

```
#include <iostream.h>
#define a 5 // Constant value with preprocessor
#define b 10 // Constant value with preprocessor
using namespace std;

int main(){
    cout<<"Value of a : "<<a<<endl;
    cout<<"Value of b : "<<b;
    return 0;
}
```

Output :

```
Value of a : 5
Value of b : 10
```

C++ What is Variable

- ये data types की values अपने अंदर store करके रखता है ।
- Variable ये एक memory location का नाम भी होता है ।

Variable के नियम

- Variable ये case-sensitive होता है । for eg int a और int A ये अलग-अलग variables है ।
- Variable की शुरुआत किसी भी alphabet(a-z, A-Z) या underscore(_) से होती है ।
- Variables का नाम alphanumeric हो सकता है । For eg. a1 = 5, var1, var2
- Variable ये space को allow नहीं करता ।
- Variable name कोई भी C++ Keywords नहीं होता ।

हर एक data type का variable होता है ।

Data Type	Example	Description
int	int a = 2;	numeric values को declare करने के लिए int data type का इस्तेमाल होता है ।
char	char char ch = 'H'; char str[]="hello";	Character को declare करने के लिए char data type का इस्तेमाल करते है ।
float	float f = 2.3;	floating-point जैसे variables को declare करने के लिए float data type का इस्तेमाल होता है, लेकिन ये single-precision होते है ।
double	double d = 2.30;	floating-point जैसे variables को declare करने के लिए double data type का इस्तेमाल होता है, लेकिन ये double-precision होते है ।
boolean	bool b = true; bool b = (!false)	bool के लिए दो values होती है । True और False
wchar_t	wchar_t str[] = L"Hello";	ये एक wide character data type है ।

C++ Variable का Declaration और initialization

Variable Declaration

जब Variable declare होता है तब ये variable जिस data type होता है, उसके हिसाब से Memory allocate करता है ।

Variable Declare होने के बाद ये अपने अंदर Garbage Value लेता है ।

Garbage Value : Garbage Value Variable को Compiler द्वारा दी जाती है ।

Syntax for Single Variable Declaration

```
data_type single_variable_name;
```

For Example

```
int a;
```

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
int a;
    cout<<"Value of a : "<<a;
return 0;
}
```

Output :

```
Value of a : 4309822
```

Variable_name	a	
Variable_value	4309822	Garbage Value
Address	0x69fefc	

Syntax for Multiple Variable Declaration

```
data_type multiple_variable_name;
```

For Example

```
data_type multiple_variable_name;
```

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
int a, b, c;
    cout<<"Value of a : "<<a<<endl;
    cout<<"Value of b : "<<b<<endl;
    cout<<"Value of c : "<<c<<endl;
return 0;
}
```

Output :

```
Value of a : 4309822
Value of b : 6946708
Value of c : 4309728
```

Variable_name	a	b	c	
Variable_value	4309822	6946708	4309728	Garbage Value
Address	0x69fefc	0x69fef8	0x69fef4	

Variable Declaration

जब Variable initialize होता है तब ये variable जिस data type होता है, उसके हिसाब से Memory allocate करता है |for eg. int for 2bytes(16-bit) | 4bytes(32-bit) | 8bytes(64-bit), char

Variable initialization में Variable को normal value दी जाती है |

Variable initialization में एक variable सिर्फ एक ही value लेता है for eg. int a = 5, 6 ; int a = 5;

Syntax for Single Variable Initialization

```
data_type single_variable_name = value;
```

For Example

```
int a=5;
```

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
int a = 5;
cout<<"Value of a : "<<a;
return 0;
}
```

Output :

```
Value of a : 5
```

Variable_name	a
Variable_value	5
Address	0x69fefc

Syntax for Multiple Variable Initialization

```
data_type single_variable_name = value, single_variable_name = value;
```

For Example

```
int a=5, b=6;
```

Source Code :

```
#include <iostream.h>
int main(){
int a = 5, b = 6;
cout<<"Value of a : "<<a<<endl;
cout<<"Value of b : "<<b;
return 0;
}
```

Output :

```
Value of a : 5
Value of a : 6
```

Variable_name	a	b
Variable_value	5	6
Address	0x69fefc	0x69fef8

Variable Redeclaration

Variable को redeclare नहीं किया जा सकता | यहाँ पर **b** variable redeclare किया गया है |

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
int a, b;
int b;
    cout << "Value of a : " <<a;
return 0;
}
```

Output :

```
error : redeclaration of 'int b'
```

C++ Variable Scopes

Variable Scope के दो प्रकार है |

- Local Variable
- Global Variable

Local Variable

- Local Variables function के अंदर होते है |
- Local Variables जिस function के अंदर होते है वह पर ही वो visible रहते है |
- Local Variables की default value 'garbage value' होती है |

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
int a = 5, b = 6, c; // Local Variable
cout<<"Value of a : "<<a;
cout<<"Value of b : "<<b;
cout<<"Garbage Value of c : "<<c;
return 0;
}
```

Output :

```
Value of a : 5  
Value of b : 6  
Default Value of c : 4309710 // garbage value
```

Global Variable

- Global Variables function के बाहर होते हैं ।
- Global Variables की visibility पूरे program में होती है ।
- Global Variables की default value '0' होती है ।

Source Code :

```
#include <iostream.h>  
using namespace std;  
int a = 5, b = 6, c; // Global Variable  
int main(){  
    cout<<"Value of a : "<<a;  
    cout<<"Value of b : "<<b;  
    cout<<"Default Value of c : "<<c;  
    return 0;  
}
```

Output :

```
Value of a : 5  
Value of b : 6  
Default Value of c : 0
```

Operators

C++ Arithmetic Operators

Operators	Explanation
+ (Addition)	ये दो Operands को add करता है ।
- (Subtraction)	ये right operand से left operand को निकाल देता है ।
* (Multiplication)	ये दो Operands को multiply करता है ।
/ (Division)	ये right operand द्वारा left operand को divide करता है ।
% (Modulus)	ये right operand द्वारा left operand को divide करके remainder निकालता है ।

Source Code :

```
#include <iostream.h>
using namespace std;
int main() {
    int a,b,c;
    cout<<"Enter two numbers ";
    cin>>a>>b;
    c=a+b;
    cout<<"Addition of a and b is "<<c<<endl;
    c=a-b;
    cout<<"Subtraction of a and b is "<<c<<endl;
    c=a*b;
    cout<<"Multiplication of a and b is "<<c<<endl;
    c=a/b;
    cout<<"Division of a and b is "<<c<<endl;
    c=a%b;
    cout<<"Remainder of a and b is "<<c<<endl;
    return 0;
}
```

Output :

```
Enter two numbers 5
6
Addition of a and b is 11
Subtraction of a and b is -1
Multiplication of a and b is 30
Division of a and b is 0
Remainder of a and b is 5
```

C++ Relational Operators

Operators	Explanation
< (less than)	एक Operand की value दूसरे Operand से कम हो तो ये true return करता है । for eg. num1=5; num2=6; num1 < num2
> (greater than)	एक Operand की value दूसरे Operand से ज्यादा हो तो ये true return करता है । for eg. num1=6; num2=5; num1 > num2
<= (less than or equal to)	एक Operand की value दूसरे Operand से कम हो या बराबर (equal) हो तो ये true return करता है । for eg. num1=5; num2=5; num1 <= num2
>= (greater than or equal to)	एक Operand की value दूसरे Operand से ज्यादा हो या बराबर (equal) हो तो ये true return करता है । for eg. num1=5; num2=5; num1 >= num2
== (equal to)	दो Operands जब बराबर(equal) होते हैं, तब ये true return करता है ।
!= (not equal to)	दो Operands जब एक-दूसरे से अलग होते हैं, तब ये true return करता है ।

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
int a=6, b=5;
if(a > b){
    cout<<" a is greater than b"<<endl;
}else{
    cout<<" a is less than b"<<endl;
}
if(a < b){
    cout<<" a is less than b"<<endl;
}else{
    cout<<" a is greater than b"<<endl;
}
if(a==b){
    cout<<" a is equal to b"<<endl;
}else{
    cout<<" a is not equal to b"<<endl;
}
return 0;
}
```

Output:

```
a is greater than b
is greater than b
a is not equal to b
```

C++ Logical Operators

Operators	Explanation
&& (logical AND)	अगर दोनों conditions true हो तो ये true return करेगा for eg. (5<6) && (6>5)
(logical OR)	अगर दोनों में से एक भी true है , तो ये true return करेगा for eg. (5<6) (6>5)
! (logical NOT)	अगर condition true हो तो ये उसे false कर देता है for eg. !((5<6) && (6>5)) !((5<6) (6>5))

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
if((5<6) && (6>5)){
    cout<<"Condition is true."<<endl;
}else{
    cout<<"Condition is false.[n"<<endl;
}
if((5<6) || (6>5)){
    cout<<"Condition is true."<<endl;
}else{
    cout<<"Condition is false."<<endl;
}
if(!((5<6) && (5>6))){
    cout<<"Condition is true."<<endl;
}else{
    cout<<"Condition is false."<<endl;
}
return 0;
}
```

Output :

```
Condition is true.
Condition is true.
Condition is true.
```

C++ Bitwise Operators

Truth Table for &, |, ^

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

Operation on AND (a&b)

अगर a = 20, b = 12 हो तो,

Decimal Value	Binary Value
20	0 0 0 1 0 1 0 0
12	0 0 0 0 1 1 0 0
4	0 0 0 0 0 1 0 0

Operation on AND (a&b)

अगर a = 20, b = 12 हो तो,

Decimal Value	Binary Value
20	0 0 0 1 0 1 0 0
12	0 0 0 0 1 1 0 0
28	0 0 0 1 1 1 0 0

Operation on XOR(a^b)

अगर a = 20, b = 12 हो तो,

Decimal Value	Binary Value
20	0 0 0 1 0 1 0 0
12	0 0 0 0 1 1 0 0
24	0 0 0 1 1 0 0 0

Binary Left Shift(<<) and Right Shift(>>)

Left Shift(<<) for e.g. a=20; /* 0001 0100 */ a << 2 में numeric value के binary value में हर binary number को 2 binary numbers left side से shift करता है | for e.g.a=20; /* 0001 0100 */ तो इसका 0101 0000 मतलब 80 हो जायेगा |

Right Shift(>>) for e.g. a=20; /* 0001 0100 */ ये Left shift से बिलकुल उलट है | Right Shift a>> 2 में numeric value के binary value में हर binary number को 2 binary numbers right side से shift करता है | for e.g.a=20; /* 0001 0100 */ तो इसका 0000 0101 मतलब 5 हो जायेगा |

Complement Operator (~)

Operation on Complement(~)

Decimal Value	Binary Value
~20	0 0 0 0 1 1 0 0
243	1 1 1 1 0 0 1 1

यहाँ पर Output -13 आने के बजाय 243 आया ऐसा क्यों ?

2's Complement of 243 -(reverse of 243 in binary + 1)

Operation on 2's Complement (~)

Decimal Value	Binary Value	2's Complement
243	1111 0011	-(0000 1100+1) = -(0000 1101) = -13(output)

Source Code :

```
#include <iostream.h>
using namespace std;
int main() {
int a=20; /* 0001 0100 */
int b=12; /* 0000 1100 */
int c;
c=a&b;
cout<<"value of c is "<<c<<endl; /* 4 = 0000 0100 */
c=a|b;
cout<<"value of c is "<<c<<endl; /* 28 = 0001 1100 */
c=a^b;
cout<<"value of c is "<<c<<endl; /* 24 = 0001 1000 */
c=a<<2;
cout<<"value of c is "<<c<<endl; /* 80 = 0101 0000 */
c=a>>2;
cout<<"value of c is "<<c<<endl; /* 5 = 0000 0101 */
cout<<"value of c is "<<~b<<endl; /* -13 = 1111 0011 */
return 0;
}
```

Output :

```
value of c is 4
value of c is 28
value of c is 24
value of c is 80
value of c is 5
value of b is -13
```

C++ Assignment Operators

Assignment Operators ग्यारह प्रकार के होते हैं।

Operators	Example
= (assignment)	c = a + b
+= (add assignment)	c += a same as c = c + a
-= (subtract assignment)	c -= a same as c = c - a
= (multiply assignment)	c *= a same as c = c * a
/= (divide assignment)	c /= a same as c = c / a
%= (modulus assignment)	c %= a same as c = c % a
&= (AND assignment)	c &= a same as c = c & a
= (OR assignment)	c = a same as c = c a
^= (XOR assignment)	c ^= a same as c = c ^ a
<<= (Left Shift assignment)	c <<= a same as c = c << a
>>= (Right Shift assignment)	c >>= a same as c = c >> a

Source Code :

```
#include <iostream.h>
using namespace std;
int main() {
    int a=20, b=12;
    b = a + b;
    cout<<"value of b is "<<b<<endl;
    b += a;
    cout<<"value of b is "<<b<<endl;
    b -= a;
    cout<<"value of b is "<<b<<endl;
    b *= a;
    cout<<"value of b is "<<b<<endl;
    b /= a;
    cout<<"value of b is "<<b<<endl;
    b %= a;
    cout<<"value of b is "<<b<<endl;
    b &= 2;
    cout<<"value of b is "<<b<<endl;
    b |= 2;
    cout<<"value of b is "<<b<<endl;
    b ^= 2;
    cout<<"value of b is "<<b<<endl;
    b <<= 2;
    cout<<"value of b is "<<b<<endl;
    b >>= 2;
    cout<<"value of b is "<<b<<endl;
    return 0;
}
```


value of b is 32
value of b is 52
value of b is 32
value of b is 640
value of b is 32
value of b is 12
value of b is 0
value of b is 2
value of b is 0
value of b is 0
value of b is 0

C++ Increment and Decrement Operators

Increment Operator (++) ये variable की value 1 से बढ़ा देता है ।

Decrement Operator (--) ये variable की value 1 से घटा देता है ।

Operators
++a (Increment Prefix)
--a (Decrement Prefix)
a++ (Increment Postfix)
a-- (Decrement Postfix)

Source Code :

```
#include <iostream.h>
using namespace std;
int main() {
    int a=20;
    cout<<"Print Value with prefix : "<<++a<<endl; // increase value with increment prefix
    cout<<"Value of a : "<<a<<endl;
    cout<<"Print Value with prefix : "<<--a<<endl; // decrease value with decrement
    prefix
    cout<<"Value of a : "<<a<<endl;
    cout<<"Print Value with postfix : "<<a++<<endl; // increase value with increment
    postfix
    cout<<"Value of a : "<<a<<endl;
    cout<<"Print Value with postfix : "<<a--<<endl; // decrease value with decrement
    postfix
    cout<<"Value of a : "<<a<<endl;
    return 0;
}
```

Output :

```
Print Value with prefix : 21
Value of a : 21
Print Value with prefix : 20
Value of a : 20
Print Value with postfix : 20
Value of a : 21
Print Value with postfix : 21
Value of a : 20
```

C++ Conditional Or Ternary Operators

Conditional Operator में तीन Expressions होते हैं ।

Conditional Operator को Ternary Operator भी कहते हैं ।

Conditional Operator में अगर पहला expression true होता है, तो वो दूसरा expression output में print करता है ।

अगर Conditional Operator में पहला expression false होता है, तो वो तीसरा expression output में print करता है ।

Syntax for Conditional / Ternary Operator

```
expression1 ? expression 2 : expression 3
```

Source Code :

```
#include <iostream.h>
using namespace std;
int main()
{
    int a = 100, b ;
    b = ( a == 100 ? 2 : 0 ) ;
    cout<<"Value of a is "<<a<<endl;
    cout<<"Value of b is "<<b;
    return 0;
}
```

Output :

```
Value of a is 100
Value of b is 2
```

Loops

C++ While Loop

- While Loop में variable को initialize करना जरूरी है ।
- अगर While Loop को repeat करना हो तो, increment/decrement operator का इस्तेमाल किया जाता है ।
- जबतक While Loop में Condition true होती तब तक repeat होता रहता है ।

Syntax :

```
variable initialization ;  
while(condition){  
statements;  
variable increment/decrement;  
}
```

For Example :

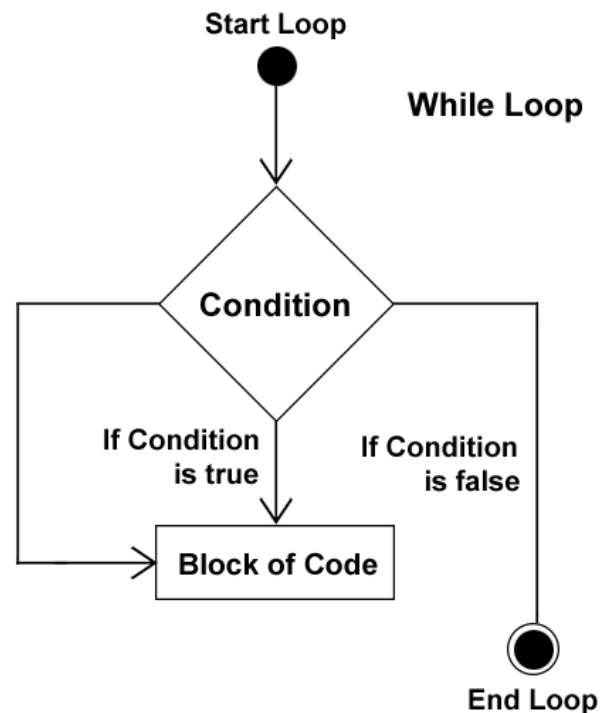
```
while(i<10){  
    cout<<i<<endl; // statement of while loop  
    i++; //increment operator  
}
```

Source Code :

```
#include <iostream.h>  
using namespace std;  
int main(){  
    int i=0; // Variable initialization  
    while(i<10) // condition of while loop  
    {  
        cout<<i<<endl; // statement of while loop  
        i++; //increment operator  
    }  
    return 0;  
}
```

Output :

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```



C++ Do While Loop

- जबतक Do-While Loop में Condition true होती तब तक repeat होता रहता है ।
- Do-While की खासियत ये है कि, अगर condition false भी हो तो ये एक statement को output में print करता है ।

Syntax :

```
do{  
statements;  
}while(condition);
```

For Example :

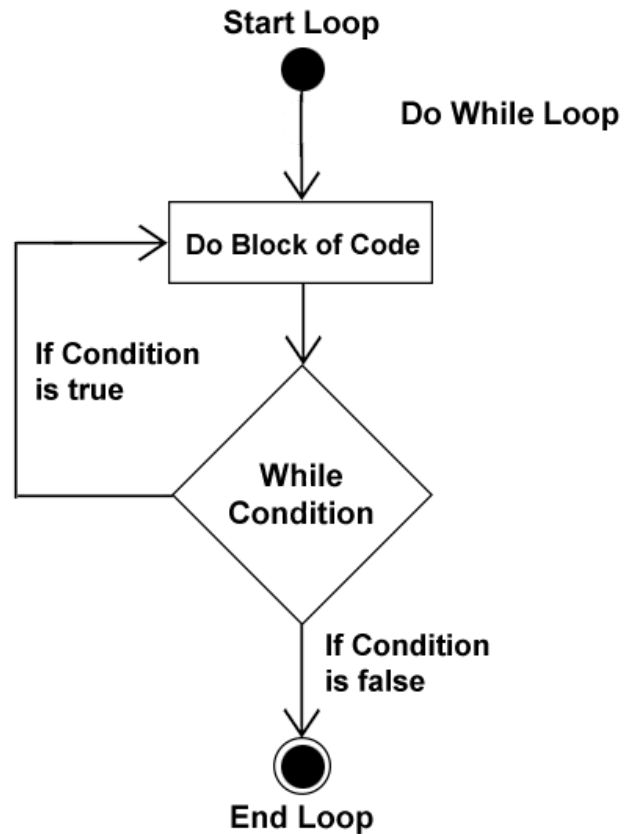
```
do{  
cout<<i<<endl;  
i++;  
}while(i<10);
```

Source Code :

```
#include <iostream.h>  
using namespace std;  
int main()  
{  
int i=0; // Variable initialization  
do{  
cout<<i<<endl;  
i++;  
}while(i<10);  
  
return 0;  
}
```

Output :

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```



C++ For Loop

- For Loop को सिर्फ Variable Declaration कि जरूरत होती है | ये statement छोड़के सभी काम अपने अंदर ही करता है |
- जबतक While Loop में Condition true होती तब तक repeat होता रहता है |

Syntax :

```
for( variable_initialization; condition; increment/decrement){  
statements;  
}
```

For Example :

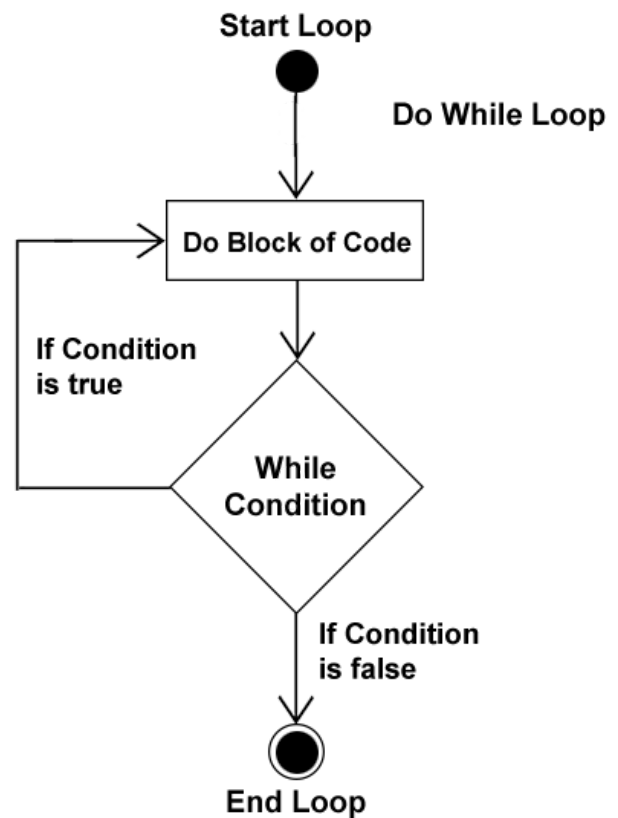
```
for( i=0; i<10; i++){  
    cout<<i<<endl; // statement of while loop  
}
```

Source Code :

```
#include <iostream.h>  
using namespace std;  
int main(){  
    int i;  
    for( i=0; i<10; i++){  
        cout<<i<<endl; // statement of while loop  
    }  
    return 0;  
}
```

Output :

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```



C++ Nested Loop

- Nested Loop ये loop का कोई प्रकार नहीं है ।
- Nested Loop में एक loop में दूसरा loop लिया जाता है ।
- Nested Loop While, Do-While, For Loop के होते है ।

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
    int i,j;
    for(i=1;i<4;i++){
        for(j=1;j<4;j++){
            cout<<i<<"\t"<<j<<endl;
        }
    }
    return 0;
}
```

Output :

```
1    1
1    2
1    3
2    1
2    2
2    3
3    1
3    2
3    3
```

Control Statements

C++ If Statement

- if Statement में अगर Condition true होती है तब Statement Execute होता है ।

Syntax :

```
if(condition){  
    statement(s);  
}
```

For Example :

```
int a=10, b=20;  
if( a < b ){  
    cout<<"a is less than b";  
}
```

Source Code :

```
#include <iostream.h>  
using namespace std;  
int main(){  
    int a=10, b=20;  
    if (a < b)  
    {  
        cout<<"a is less than b";  
    }  
    return 0;  
}
```

Output :

```
a is less than b
```

C++ If Else Statement

- if_else Statement में अगर Condition true हो तो वो if का statement Execute करता है ।
- अगर Condition false होती है तो else का Condition करता है ।

Syntax :

```
if(condition){  
statement(s);  
}else{  
statement(s);  
}
```

For Example :

```
int a=10, b=20;  
if( a == b ){  
    cout<<"a is equal to b";  
}else{  
    cout<<"a is not equal to b";  
}
```

Source Code :

```
#include <iostream.h>  
using namespace std;  
int main(){  
int a=10, b=20;  
if( a == b ){  
    cout<<"a is equal to b";  
}else{  
    cout<<"a is not equal to b";  
}  
return 0;  
}
```

Output :

a is not equal to b

C++ Else If Statement

- `else_if` Statement में अगर `if` की Condition true होती है तो `if` का statement execute होता है | अगर `if` का condition false होता है तो वो अगले condition पर जाकर check करता है | अगर वो condition true होता है तो वो उसका statement execute करता है | अगर कोई भी condition true नहीं होती तो वो `else` का statement execute करता है |

Syntax :

```
if(condition){  
statement(s);  
}else if(condition){  
statement(s);  
}else{  
statement(s);  
}
```

For Example :

```
int a=10, b=20;  
if( a < b ){  
    cout<<"a is less than b";  
}else if( a > b ){  
    cout<<"a is greater than b";  
}else{  
    cout<<"a is equal to b";  
}
```

Source Code :

```
#include <iostream.h>  
using namespace std;  
int main(){  
int a=10, b=20;  
if( a < b ){  
    cout<<"a is less than b";  
}else if( a > b ){  
    cout<<"a is greater than b";  
}else{  
    cout<<"a is equal to b";  
}  
return 0;  
}
```

Output :

a is less than b

C++ Switch Case Statement

- Switch case statement में expression होता है और उससे related कुछ cases होते हैं। जो case उस expression या declare किये हुए variable से match होती है तब वो output में print होता है। अगर कोई भी case expression से match नहीं होती तो वो default का statement output में print करेगा। आपको हर statement के बाद break लगाना पड़ता है, इसका मतलब वो उसके पहले का statement ही print करेगा। अगर आप break नहीं लगाते तो वो पहला और दूसरा ये दोनों statement को print करेगा। default case के बाद break नहीं लगाते।

Syntax :

```
switch (expression){  
case value1 :  
statement1 ;  
break;  
case value2 :  
statement2 ;  
break;  
default :  
statement3 ;  
}
```

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
char Day='D';
switch(Day){
case 'A' :
cout<<"Today is Sunday";
break;
case 'B' :
cout<<"Today is Monday";
break;
case 'C' :
cout<<"Today is Tuesday";
break;
case 'D' :
case 'E' :
cout<<"Today is Wednesday";
break;
case 'F' :
cout<<"Today is Thursday";
break;
case 'G' :
cout<<"Today is Friday";
break;
case 'H' :
cout<<"Today is Saturday";
break;
default :
cout<<"Day is Not Found";
}
return 0;
}
```

Output :

Today is Wednesday

C++ Break Statement

- Break Statement Program के loops और switch case के execution का काम किसी condition पर बंद कर देता है ।

Syntax :

```
break;
```

Source Code :

```
#include <iostream.h>
using namespace std;
int main()
{
    int i=0;
    while ( i < 20 ){
        cout<<"value of i is "<<i;
        i++;
        if ( i == 10)
            break;
    }
    return 0;
}
```

Output :

```
value of i is 0
value of i is 1
value of i is 2
value of i is 3
value of i is 4
value of i is 5
value of i is 6
value of i is 7
value of i is 8
value of i is 9
```

C++ Continue Statement

- Continue Statement Program के loops के condition के हिसाब से बीचवाले statements को skip कर देता है और बादवाले statement को execute करता है ।

Syntax :

```
continue;
```

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
    int i;
    for(i=0;i<10;i++){

        if(i==7){
            cout<<"Number %d is skipped."<<i;
            continue;
        }
        cout<<i<<endl;
    }
    return 0;
}
```

Output :

```
0
1
2
3
4
5
6
Number 7 is skipped.
8
9
```

C++ Goto Statement

Go to ये C Programming का statement है | इसमें labels का use किया जाता है |

Goto Statement के दो प्रकार होते हैं |

- Forward
- Backward

जब goto statement कुछ statement को छोड़कर उसके अगले statement को execute करता है , उसे Forward goto statement कहते हैं और किसी पिछले या execute हुए statement को फिरसे execute करने के लिए अपने पिछले label पर जाता है , उसे Backward goto statement कहते हैं |

Syntax for Forward and Backward goto Statement

Syntax For Forward:

```
goto label ;  
statement ;  
-----  
label ;
```

Syntax For Backward:

```
label ;  
statement ;  
-----  
goto label ;
```

Source Code :

```
#include <iostream.h>  
using namespace std;  
int main(){  
    int num1, num2, num3;  
    char ch;  
    yes : cout<<"Enter two values"<<endl;  
          cin>>num1>>num2;  
    num3 = num1 + num2 ;  
    cout<<"Addition of "<<num1<<" and "<<num2<<" is "<<num3;  
    cout<<"\nDo you want to continue y(yes) or n(No)";  
    cin>>ch;  
    if(ch=='y'){  
        goto yes;  
    }  
    else if(ch=='n'){  
        goto no;  
    }  
    else{  
        cout<<"You entered other key";  
    }  
    no : cout<<"Do you want to exit ?, Press Enter";  
    return 0;  
}
```

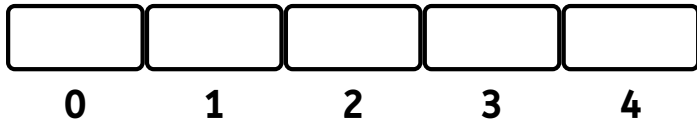
Output :

```
Enter two values  
4  
5  
Addition of 4 and 5 is 9  
Do you want to continue y(yes) or n(No)y  
Enter two values  
6  
4  
Addition of 6 and 4 is 10  
Do you want to continue y(yes) or n(No)n  
Do you want to exit ?, Press Enter
```

Arrays

C++ Introduction of an Array

- Array ये same data type के variables का collection होता है ।
- Array का data type कौनसा भी हो सकता है ।
- Array के variables अपने पासवाले Memory Location पर store होते है
- Array का initialization '0' से शुरू होता है ।



Array का जो element पहला होता है उसका Memory Address सबसे कम होता है ।

Array का इस्तेमाल क्यों किया जाता है ?

अगर किसी students के नाम store करना हो तो उनके अलग-अलग variable बनाने पड़ते है ।
for eg.

```
char stud1 = "Rakesh" , stud2 = "Uday", stud3 = "Raj" ;
```

Array में इन सभी नाम के लिए सिर्फ एक ही array_variable create करना पड़ता है ।
for eg.

```
char arr[] = { "Rakesh", "Uday", "Raj" };
```

C++ Single or One Dimensional Array

Syntax for Single Dimensional Array Declaration

```
data_type array_name[size_of_array];
```

For Example

```
int arr[5];
```

Syntax for Single Dimensional Array Initialization

```
data_type array_name[size_of_array] = {value1, value2,...,value n};
```

For Example

```
int arr[5] = {1, 2, 3, 4, 5};
```

Example for One/Single Dimensional Array

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
int i,num[5]={1, 2, 3, 4, 5};
for(i=0;i<5;i++){
cout<<"Array Number is "<<num[i]<<endl;
}
return 0;
}
```

Output :

```
Array Number is 1
Array Number is 2
Array Number is 3
Array Number is 4
Array Number is 5
```

Array Memory Representation

यहाँ पर Address में 4-4 का फर्क है क्योंकि, System 32-bit होने के कारण integer का size 4bytes है ।
इसीलिए Memory Address में 4-4 का फर्क है ।

अगर array का data type character होता तो Address में 1-1 का फर्क होता ।

index of array	arr[0]	arr[1]	arr[2]	arr[3]	arr[4]
array elements	1	2	3	4	5
Memory Address	6356728	6356732	6356736	6356740	6356744

C++ Multi Dimensional Array

Syntax for Multi Dimentional Array Declaration

```
data_type array_name[ row_size ][ column_size ];
```

For Example

```
int arr[3][4];
```

	column1	column2	column3
row1	<input type="text"/>	<input type="text"/>	<input type="text"/>
	arr[0][0]	arr[0][1]	arr[0][2]
row2	<input type="text"/>	<input type="text"/>	<input type="text"/>
	arr[1][0]	arr[1][1]	arr[1][2]
row3	<input type="text"/>	<input type="text"/>	<input type="text"/>
	arr[2][0]	arr[2][1]	arr[2][2]

Example for Multi Dimensional Array Initialization

```
int arr[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};  
OR  
int arr[3][4] = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```

Example for Multi Dimensional Array

```
#include <iostream.h>  
using namespace std;  
int main() {  
    int arr[3][4] = { {1,2,3,4}, {5,6,7,8}, {9,10,11,12} };  
    int i, j;  
    for ( i = 0; i < 3; i++ ){  
        for ( j = 0; j < 4; j++ ){  
            cout<<"["<<i<<"["<<j<<" = "<<arr[i][j]<<endl;  
        }  
    }  
    return 0;  
}
```

Output :

```
[0][0]=1  
[0][1]=2  
[0][2]=3  
[0][3]=4  
[1][0]=5  
[1][1]=6  
[1][2]=7  
[1][3]=8  
[2][0]=9  
[2][1]=10  
[2][2]=11  
[2][3]=12
```

Storage Classes

- Storage Classes Variables का scope और lifetime तय करता है ।
- Storage Classes Variables को कहाँ पर store करके रखे ये बताता है । for eg. CPU, Register

C++ Automatic

- Automatic Storage Class में 'auto' keyword का इस्तेमाल करते है ।
- ये Normal Variable की तरह ही होता है ।
- ये एक Local Variable है ।
- इनकी visibility या scope function के अंदर होता है ।
- बाहर वो destroyed हो जाते है । इनकी default value 'garbage' होती है ।

Syntax

```
auto data_type variable_name = value(optional);
```

For Example

```
int a ; // and  
auto int a ; // both are same
```

Source Code :

```
#include <iostream.h>  
using namespace std;  
int main(){  
    auto int a;  
    auto int b = 5;  
    cout<<"Value of a : "<<a<<endl;  
    cout<<"Value of b : "<<b;  
    return 0;  
}
```

Output :

```
Value of a : 6946708 // garbage value  
Value of b : 5
```

C++ External

- External Storage Class में 'extern' keyword का इस्तेमाल करते हैं।
- External Storage Class के variables का scope Global होता है।
- Global Variable के कारण इनका इस्तेमाल Program में extern के साथ कहा पर भी और किसी भी function के अंदर होता है। इनकी default value '0' होती है।

Syntax

```
extern data_type variable_name = value(optional);
```

For Example

```
extern int a ;
```

Source Code :

```
#include <iostream.h>
using namespace std;
int num = 5 ;
void func(); // function declaration
int main(){
extern int num ;
    cout<<"Value of num : "<<num<<endl;
func(); // function calling
return 0;
}
void func(){ // function definition
extern int num ;
    cout<<"Value of num : "<<num;
}
```

Output :

```
Value of num : 5
Value of num : 5
```

C++ Register

- Register Storage Class में 'register' keyword का इस्तेमाल करते हैं।
- Register Storage Class के variables का scope Local होता है
- Local Variable के कारण इनका इस्तेमाल Program में जिस function के अंदर इनको declare या initialize किया है उसी function के अंदर visible रहता है।
- Register Storage Variables Computer के Register पर store होते हैं।
- Register Storage Class की Memory 'Limited' होती है। अगर Register की Memory खत्म हो जाए तो वो CPU Memory पर store होते हैं।
- Register Variables का कोई address(&) नहीं होता।
- इनकी default value 'garbage' होती है।

Syntax

```
register data_type variable_name = value(optional);
```

For Example

```
register int a ;
```

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
    register int num = 5 ;
    cout<<"Value of num : "<<num;
    return 0;
}
```

Output :

```
Value of num : 5
```

C++ Static

- Static Storage Class में 'static' keyword का इस्तेमाल करते हैं।
- Static Storage Class के Variables का scope Local और Global ये दोनों होता है।
- इनकी default value '0' होती है।

Code Description

निचे दिए हुए General File में `int num = 1` ; ये value initialize की है और ये variable एक function में मतलब Local Variable लिया है। function1 इस function को for loop से repeat किया है। ये variable सामान्य होने के कारण इसकी value control के बाहर जाने के बाद नष्ट हो जाती है, इसीलिए इस variable की value increase नहीं होती बल्कि ये बार-बार initial होने से ये initial value को ही for loop के द्वारा बार-बार output में print करता है।

निचे दिए हुए Static File में `static int num = 1` ; ये value initialize की है। इस program में General File के मुकाबले सिर्फ static keyword का use किया है। Static variables बिल्कुल सामान्य variables के उलट प्रक्रिया करता है। Static Local variables की value control के बाहर जाने के बाद destroy नहीं होती। Static Local variables की value एक बार ही initialize होती है। Static Variables अपने control के बाहर जाने के बाद initialize हुई value को नष्ट नहीं करता, इसीलिए Static File में बार-बार initial value print नहीं होती।

General File :

Source Code :

```
#include <iostream.h>
using namespace std;
func(){
int num = 1;
cout<<num;
num++;
}
int main()
{
int i;
for(i=0; i<5; i++){
cout<<endl;
func();
}
return 0;
}
```

Output :

```
1
1
1
1
1
```

Static File :

Source Code :

```
#include <iostream.h>
using namespace std;
func(){
static int num = 1;
cout<<num;
num++;
}
int main()
{
int i;
for(i=0; i<5; i++){
cout<<endl;
func();
}
return 0;
}
```

Output :

```
1
2
3
4
5
```

Input and Output

C++ Input and Output Introduction

C++ में Input and Output के लिए streams का इस्तेमाल किया जाता है। सामान्य जीवन में भी दो मुख्य input and output devices हैं। एक Keyboard है और दूसरा Desktop/Monitor। Keyboard से दिया हुआ input Monitor/Desktop की screen पे Output के माध्यम से दिखता है।

C++ में ऐसे ही कुछ header files हैं और इन header files में input/Output होते हैं। for eg. cin और cout

Input/Output के लिए C++ में तीन Header Files हैं।

- `iostream`
- `iomanip`
- `fstream`

iostream : Standard Stream से data को read और write करने के लिए इस header file का इस्तेमाल होता है। C++ के program में input/output के लिए `iostream` इस header file को include करना पड़ता है।

iomanip : `iomanip` ये भी एक input/output library का हिस्सा है। जिसमें arguments के साथ manipulator functions होते हैं। `iomanip` formatted input/Output के लिए इस्तेमाल किया जाता है।

fstream : `fstream` का इस्तेमाल files के लिए होता है। File को read और write करना हो तो इस header file को include करना जरूरी होता है।

C++ cout Console Output

cout(Console Output) : Standard Output Stream

- `cout` output screen पर दिखाता है। `cout` ये `ostream` class का object है।
- `cout` के साथ insertion operator (`<<`) का भी इस्तेमाल किया जाता है।
- insertion operator को एक ही line में variable के साथ अनेक insertion operators (`<<`) का इस्तेमाल किया जाता है।
- insertion operator 'output' के लिए इस्तेमाल किया जाता है।
- insertion operator के साथ `endl` का भी इस्तेमाल किया जाता है। ये line के आखिर में इस्तेमाल होता है।

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
    cout<<"Hello World!"<<endl;
    return 0;
}
```

Output :

Hello World!

C++ cin Console Input

cin(Console Input) : Standard Input Stream

- cin Keyboard से data को input करता है ।
- cin ये istream class का object है ।
- cin के साथ extraction operator(>>) का भी इस्तेमाल किया जाता है ।
- extraction operator को एक ही line में variable के साथ अनेक extraction operators(>>) का इस्तेमाल किया जाता है ।

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
    int a, b;
    cout << "Enter Two numbers"<<endl;
    cin >>a>>b;
    cout <<"Value of a : "<<a<<endl;
    cout <<"Value of b : "<<b<<endl;
    return 0;
}
```

Output :

Enter Two numbers
5
6
Value of a : 5
Value of b : 6

C++ cerr Console Error

cerr(Console Error) : Standard Output Stream for Error

- cout और cerr ये दो Objects एक जैसे हैं ।
- cerr ये object ostream class का है ।
- cerr ये unbuffered standard output stream error है । ये तुरंत ही console पर error message को भेजता है

Source Code :

```
#include <iostream.h>
using namespace std;
int main()
{
    cerr<<"It similar to cout"<<endl;
    return 0;
}
```

Output :

```
It similar to cout
```

Functions

Function Introduction

- C++ Function ये statements का एक समूह होता है | हर एक Program में एक function तो होता ही है | for eg. main() Function को Program में कहा पर भी लिखा जाता है | जहाँ पर Function की जरूरत होती है वहाँ पर Function call किया जाता है |

Function के फायदे :

- Function में लिखा हुआ code बार-बार लिखना नहीं पड़ता |
- Function Programmer का समय और Program की space बचाता है |
- बड़े Program को छोटे-छोटे function में विभाजित किया जा सकता है |
- अगर Program में कहा पर error आ जाए तो उसे आसानी से निकाला जा सकता है |
- जहाँ पर जरूरत हो वहाँ पर function को बार-बार call किया जा सकता है |

Function कैसा होता है ?

Function का एक विशिष्ट नाम होता है | Function की शुरुआत में उसका नाम बाद में दो parenthesis () और function के statements दो curly braces {} होते हैं |

Function के दो प्रकार हैं |

- In-built / Predefined Function
- User-defined Function

Predefined or In built Function

- In-built Functions को predefined या Library Functions भी कहते हैं |
- In-built Functions में हर एक Functions के लिए अलग-अलग header file या preprocessor बनाये गए हैं |
- Function का declaration, definition header files में होता है |
- C++ में बहुत सारे Header files हैं इनमें अलग-अलग functions grouped करके रखे हुए हैं | अगर Programmer चाहे तो अपने खुदके header files भी बना सकता है |

For Example

cout : ये Object iostream.h इस header file के अन्तर्गत आता है | अगर Programmer #include <iostream.h> ये preprocessor Program में include नहीं करता तो cout का इस्तेमाल नहीं कर सकता |

strlen() : ये Function string.h इस header file के अन्तर्गत आता है | अगर Programmer #include ये preprocessor Program में include नहीं करता तो strlen का इस्तेमाल नहीं कर सकता |

User defined Function

User-defined Function के तीन विभाग होते हैं।

1. Function Declaration
2. Function Calling
3. Function Definition

1. In Function Declaration

Function create करते वक्त Compiler को बताना पड़ता है की आप कैसा Function create करना चाहते हो, इस process को Function Declaration कहते हैं।

Syntax for Function Declaration

```
return_type function_name(parameter(s));
```

Function Declaration के चार विभाग हैं।

1. return_type
2. function_name
3. parameter(s)/argument(s)
4. semicolon

1. return_type

हर Function कोई ना कोई value return करता है, वो null (void) हो या numeric(int, float, double) या character(char)। Function का return_type void default होता है। function का return_type program के requirement में होता है।

2. function_name

Function का नाम उसके code के अनुसार होना चाहिए। अगर ना भी हो तो भी compiler को चलता है, लेकिन ये Good Programming नहीं कहलाता। Function का नाम C++ का कोई keyword नहीं होता। C++ के Function के parenthesis () के अंदर parameters भी होते हैं। Function का नाम case-sensitive होता है। for eg. hello() और Hello() ये दोनों अलग-अलग functions हैं।

3. parameter(s)/argument(s)

Function के argument data_types होते हैं या data type के साथ उनके variable के नाम होते हैं। User किस प्रकार की value function calling के जरिये receive करना चाहता है, ये function के argument में declare होता है।

4. semicolon

हर Function के Declaration के बाद semicolon देते है | ये भी function declaration का हिस्सा है |

Example for Function Declaration with two parameters

```
int add(int a,int b); // function declaration
```

Function Declaration without parameter(s)

```
int add();
```

Function Declaration with one parameter

```
int add(int a);
```

2. Function Calling

Syntax for Function Calling

```
function_name(Parameter1 ,Parameter2 ,.Parameter n);
```

Function Calling में सिर्फ function का नाम, function के arguments और semicolon होता है | निचे दिए हुए example function का return type 'integer' है | function का नाम 'add' है और function को दो parameters मतलब a और b पास किये गए है |जब तक function को call नहीं किया जाता तब तक function के declaration और definition को कोई महत्व नहीं रहता |

Example for Function calling with two parameters

```
add(a, b); // funtion calling
```

Function calling without parameter(s)

```
add();
```

Function calling with one parameter

```
add( a );
```

3. Function Definition

Syntax for Function Definition

```
return_type function_name(Parameter(s)){  
  
    function_body;  
  
}
```

Function Calling में सिर्फ function का नाम, function के arguments और semicolon होता है। निचे दिए हुए example function का return type 'integer' है। function का नाम 'add' है और function को दो parameters मतलब a और b पास किये गए हैं। जब तक function को call नहीं किया जाता तब तक function के declaration और definition को कोई महत्व नहीं रहता।

Function Definition के चार हिस्से

1. return_type

हर Function कोई ना कोई value return करता है, वो null (void) हो या numeric(int, float, double) या character(char)। Function का return_type void default होता है। function का return_type program के requirement में होता है।

2. function_name

Function का नाम उसके code के अनुसार होना चाहिए। अगर ना भी हो तो भी compiler को चलता है, लेकिन ये Good Programming नहीं कहलाता। Function का नाम C++ का कोई keyword नहीं होता। C++ के Function के parenthesis () के अंदर parameters भी होते हैं। Function का नाम case-sensitive होता है। for eg. hello() और Hello() ये दोनों अलग-अलग functions हैं।

3. parameter(s)/argument(s)

Function के argument data_types होते हैं या data type के साथ उनके variable के नाम होते हैं। User किस प्रकार की value function calling के जरिये receive करना चाहता है, ये function के argument में declare होता है।

4. function_body

Function body में variables होते हैं, लेकिन function के अंदर होने के कारण इनका scope Local होता है। Body के अंदर कुछ statement(s) होते हैं और value return करता है।

Example for Function Definition

```
int add(int x,int y){ // function definition
int z;
z = x + y ;
return z;
}
```

Full Example for Function

```
#include <iostream.h>
using namespace std;
int add(int a,int b); // function declaration with argument
int main(){
int a,b,c;
cout<<"Enter value of a and b : ";
cin>>a>>b;
c = add(a,b); // function calling
cout<<"Addition of a and b : "<<c;
return 0;
}
int add(int x,int y){ // function definition
int z;
z = x + y ;
return z;
}
```

Call By Value And Reference

Call By Value और Call By Reference सिखने से पहले Parameters को समझे ।

Function में दो प्रकार के Parameters होते हैं ।

- Formal Parameter
- Actual Parameter

Formal Parameter

जो parameter function के declaration में और definition में लिखे जाते हैं, उसे Formal Parameter कहते हैं ।

Full Example

```
void swap(int x, int y); // Formal Parameters
int main(){
int a=2; b=10
swap (a, b)
}
void swap (int x, int y){ //Formal Parameters
-----
-----
}
```

जो parameter function call में लिखे जाते हैं, उसे Actual Parameter कहते हैं।

Full Example

```
void swap(int x, int y);
int main(){
int a=2; b=10
swap (a, b) //Actual Parameter
}
void swap (int x, int y){
-----
-----
}
```

Function Calling के दो प्रकार

1. Call By Value
2. Call By Reference

1. Call By Value

Call By Value में Variable के value को parameter के रूप से function को pass किया जाता है।
Call By Value में Actual Parameter की value Formal Parameter पर copy की जाती है।

यहाँ पर Program में variable 'a' और 'b' function 'add' को pass किये गए हैं।
यहाँ पर 'a' और 'b' की values 'x' और 'y' variable पर copy की जाती है।

For Example

```
#include <iostream.h>
using namespace std;
void swap(int x, int y);
int main(){
    int a = 2, b = 10;
    cout<<"Before Swapping a = "<<a<<" and b "<<b<<endl;
    swap(a, b);
}
void swap(int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
    cout<<"After Swapping a = "<<x<<" and b "<<y;
}
```

Output :

```
Before Swapping a = 2 and b 10
After Swapping a = 10 and b 2
```

2. Call By Reference

Call By Reference में Variable के address को parameter के रूप से function को pass किया जाता है ।

Call By Value में Actual Parameter की value Formal Parameter पर copy नहीं की जाती है ।

यहाँ पर Program में variable 'a' और 'b' address को function 'add' को pass किये गए है ।

यहाँ पर 'a' और 'b' की values 'x' और 'y' variable पर copy नहीं की जाती है ।

ये सिर्फ variables का address hold करके रखता है ।

For Example

```
#include <iostream.h>
using namespace std;
void swap(int *x, int *y);
int main(){
int a = 2, b = 10;
cout<<"Before Swapping  a = "<<a<<" and b "<<b<<endl;
swap(&a, &b);
}
void swap(int *x, int *y)
{
int temp;
temp = *x;
*x = *y;
*y = temp;
cout<<"After Swapping  a = "<<*x<<" and b "<<*y;
}
```

Output :

```
Before Swapping  a = 2 and b 10
After Swapping  a = 10 and b 2
```

Pointers

Pointer Introduction

- हर एक Variable का एक Memory Address होता है और Memory Address देखने के लिए Programmer को variable से पहले '&' (address operator) लगाना पड़ता है।

For Example

```
int a ;  
cout<<"Address of a : "<<&a;
```

Output :

```
Address of a : 0x69fefc  // 0x69fefc is address of variable
```

इसी memory address की मदद से Variable की value को access किया जाता है, इसी को 'Pointers' कहते हैं। Pointer किसी दूसरे variable का address hold करके रखता है। हर एक variable का एक address होता है। जब variable declare होता है तभी उसका एक memory location पर वो store होता है।

For Example

```
int a = 10;
```

Variable_or_Location_name	a	
Variable_value	10	Location
Memory Address	0x69fefc	

Syntax for Pointer Declaration

```
data_type *pointer_name;
```

Example for Pointer Declaration

```
int *ptr1; // integer pointer variable  
char *ptr2; // character pointer variable  
float *ptr3; // float pointer variable  
double *ptr4; // double pointer variable
```

किसी integer data type variable का address hold करना है तो pointer भी उसी data type का होगा जिस data type का variable हो, मतलब variable int है तो pointer भी int ही होगा, अगर variable char हो तो pointer variable भी char ही होगा। Pointer किसी की value hold करके नहीं रखता, बल्कि सिर्फ उसका address hold करके रखता है और उस address से ही pointer के साथ variable के value को print किया जाता है।

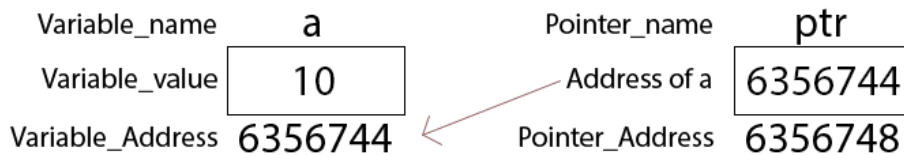
Full Example for Pointer

```
#include <iostream.h>
using namespace std;
int main(){
int a = 10;
int *ptr; // Integer Pointer Variable
ptr = &a; //address of a stored in ptr
    cout<<"Value of a is: "<<a<<endl;
    cout<<"Address of a is: "<<ptr;
return 0;
}
```

Output :

```
Value of a is: 10
Address of a is: 0x69fef4
```

Note : Pointer का address साधारणतः Hexadecimal Numbers होते हैं, पर कुछ compiler अपना address integer में print करते हैं | Variable का address print कराने के लिए '%x' या '%u' इन Format Specifiers का इस्तेमाल करते हैं |



Referencing Operator

Referencing मतलब किसी दूसरे variable का address hold करके रखना होता है |

हर variable का address hold करने के लिए जिसका address hold करना है और जिस Pointer variable में hold करना है, तो उन दोनों का data type same होना चाहिए |

For Example

```
int a = 10;
int *ptr;
ptr = &a; //address of a stored in ptr
```

Dereferencing Operator

Dereferencing में asterisk (*) का इस्तेमाल करते पर pointer में इसे Dereferencing Operator भी कहते हैं।
Dereferencing में pointer में store किये हुए value को access किया जाता है।

For Example

```
int a = 10;  
int *ptr;  
ptr = &a;  
int b = *ptr; // ptr means dereferencing  
cout<<"Value of a : "<<b;
```

Output :

```
Value of a : 10
```

Strings

C++ What is String

- Strings characters का समूह होता है ।
- Strings ये One-dimensional array होता है, जिसमे सिर्फ characters होते है ।
- String का आखिरी character 'NULL'(\0) होता है ।
- अगर पूरा string लिखना हो तो उसे double quotes (" ") में लिखा जाता है । अगर एक-एक character को लिखना हो तो उसे single quotes (' ') में लिखा जाता है ।
- String का data type character (char) होता है ।

Example for Single Character String

```
char str1[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

Example for Multiple Character String

```
char str2[6] = "Hello";
```

Example for Multiple Character String without using Array_size

```
char str3[] = "Hello";
```

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
char str1[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
char str2[6] = "Hello";
char str3[] = "Hello";
    cout<<"Value of str1 : "<<str1<<endl;
    cout<<"Value of str2 : "<<str2<<endl;
    cout<<"Value of str3 : "<<str3<<endl;
return 0;
}
```

Output :

```
Value of str1 : Hello
Value of str2 : Hello
Value of str3 : Hello
```

String Representation

index of array	str[0]	str[1]	str[2]	str[3]	str[4]	str[5]
Array Elements	H	e	l	l	o	\0
Memory Address	0x69fef0	0x69fef1	0x69fef2	0x69fef3	0x69fef4	0x69fef5

String Working with size (sizeof)

Program में हर एक String के initialization में अलग-अलग size है | दिए हुए array के size की memory allocate की जाती है | अगर Array का size नहीं दिया जाता तो जितनी size string की है उतनी size array allocate करता है |

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
char str1[20] = {'H', 'e', 'l', 'l', 'o', '\0'};
char str2[10] = "Hello"; //using array_size
char str3[] = "Hello"; //without using array_size
    cout<<"Size of of str1 : "<<sizeof(str1)<<endl;
    cout<<"Size of str2 : "<<sizeof(str2)<<endl;
    cout<<"Size of str3 : "<<sizeof(str3)<<endl;
return 0;
}
```

Output :

```
Size of of str1 : 20
Size of str2 : 10
Size of str3 : 6
```

String using *Pointer

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
char *ptr = "Hello";
    cout<<ptr;
return 0;
}
```

Output :

```
Hello
```

C++ String Library Functions

इन Functions को Program में इस्तेमाल करना हो तो string.h या strings.h इन header file को include करना पड़ता है ।

String Functions	Description
strcat	एक String से दूसरे String को जोड़ा जाता है ।
strchr	दिए हुए string से एक character का पहला occurrence के आगे का string pointer को return करता है ।
strcmp	दो String को Compare किया जाता है । ये case-sensitive है ।
strcmpi	दो String को Compare किया जाता है । ये case-sensitive नहीं है ।
strcpy	एक string को दूसरे string में copy करता है ।
strdup	String का duplicate बनाता है ।
strlen	String की Length निकाली जाती है ।
strlwr	Uppercase के Characters को Lowercase में convert किया जाता है ।
strncat	दिए हुए number के जितने character है उनसे String को जोड़ा जाता है ।
strncpy	दिए हुए number के जितने character एक string से दूसरे string में copy किया जाता है ।
strnset	दिए हुए number और दिए हुए character के हिसाब से string को replace करता है ।
strrchr	दिए हुए string से एक character का आखिरी occurrence के आगे का string pointer को return करता है ।
strrev	String को उलटी दिशा से print करता है ।
strrstr	दिए हुए String का आखिरी string occurrence के आगे का string pointer को return करता है ।
strset	दिए हुए character से पूरे string को replace करता है ।
strstr	दिए हुए String का पहला string occurrence के आगे का string pointer को return करता है ।
strupr	Lowercase के Characters को Uppercase में convert किया जाता है ।

strcat() - String Function

strcat ये String का एक Function है | एक String से दूसरे String को जोड़ा जाता है |

Syntax

```
strcat(destination_string, source_string);
```

- **Destination_string** - ये वो parameter है जिसके साथ source का string जोड़ा जाता है | String के आखिर में null character (\0) होता है | Source string destination string के साथ जुड़ते समय उसको remove कर देता है |
- **Source_string** - ये वो parameter है जिसका string destination string के साथ आखिर में जोड़ा जाता है | किसी integer value को एक variable को दूसरे variable से जोड़ना हो तो arithmetic operator (+) का use नहीं कर सकते |

for e.g.

```
int num1 = 5 ;i
```

```
nt num2 = 5 ;
```

```
int num3 = num1 + num2 ;
```

इनका output 55 मिलना चाहिए लेकिन इनका output 10 मिलेगा |

किसी char को भी एक दूसरे से arithmetic operators के साथ नहीं जोड़ा जा सकता |

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
int main (){
char str1[20] = "Welcome";
char str2[20] = " Friend";
strcat(str1, str2);
    cout<<"Concatenation String : "<<str1;
return 0;
}
```

Output :

```
Concatenation String : Welcome Friend
```


strchr() - String Function

दिए हुए string से एक character का पहला occurrence के आगे का string pointer को return करता है ।

Syntax

```
strchr(string, int character);
```

- **string** - ये एक normal string है ।
- **int character** - ये दिए हुए string में से दिए हुए character का पहला occurrence के आगे का string pointer को return करता है ।

अगर दिया हुआ character string को नहीं मिलता तो वो NULL character return करता है ।

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
int main (){
char str[] = "Hello Friends";
char *ptr;
ptr = strchr(str, 'F');
cout<<ptr;
return(0);
}
```

Output :

```
Friends
```

strcmp() - String Function

दो String को Compare किया जाता है | ये case-sensitive है |

Syntax

```
strcmp(string1, string2);
```

- **string1** - ये वो String है जिसके साथ String2 को Compare किया जाता है |
- **string2** - ये वो String है जिसके साथ String1 को Compare किया जाता है |

अगर दोनों string एक जैसे होते हैं तो ये '0' return करता है | अगर दोनों string अलग-अलग होते हैं तो '1' या '-1' return करता है |

Note : ये strcmp() function case-sensitive है | इसमें 'h' और 'H' ये दोनों अलग-अलग हैं |

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
int main(){
char str1[] = "Hello" ;
char str2[] = "World" ;
int a = strcmp(str1, str2) ;
    cout<<a<<endl;
int b = strcmp(str1, "Hello") ;
    cout<<b<<endl;
int c = strcmp(str1, "hello") ; // strcmp is case-sensitive
    cout<<c<<endl;
return 0;
}
```

Output :

```
-1
0
-1
```

strcmpi() - String Function

दो String को Compare किया जाता है | ये case-sensitive नहीं है |

Syntax

```
strcmpi(string1, string2);
```

- **string1** - ये वो String है जिसके साथ String2 को Compare किया जाता है |
- **string2** - ये वो String है जिसके साथ String1 को Compare किया जाता है |

अगर दोनों string एक जैसे होते हैं तो ये '0' return करता है | अगर दोनों string अलग-अलग होते हैं तो '1' या '-1' return करता है |

Note : ये strcmpi() function case-sensitive है | इसमें 'h' और 'H' ये एक ही अलग-अलग हैं |

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
int main(){
char str1[] = "Hello";
char str2[] = "World";
int a = strcmpi(str1, str2) ;
    cout<<a<<endl;
int b = strcmpi(str1, "Hello") ;
    cout<<b<<endl;
int c = strcmpi(str1, "hello") ; // strcmpi is not case-sensitive
    cout<<c<<endl;
return 0;
}
```

Output :

```
-1
0
0
```

strcpy() - String Function

दो String को Compare किया जाता है | ये case-sensitive नहीं है |

Syntax

```
strcpy(destination_string, source_string);
```

- **destination_string** - ये वो parameter है जिसपर source के string की value copy की जाती है | अगर destination string पर कोई value भी हो तो वो overwrite हो जाती है |
- **source_string** - ये वो parameter है जिसकी value destination पर copy की जाती है |

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
int main(){
char str1[20] = "Welcome";
char str2[20] = "Friend";
    cout<<"Value of str2 = "<<str2<<endl;
strcpy(str2, str1);
    cout<<"Copy str1 to str2 = "<<str2<<endl;
return 0;
}
```

Output :

```
Value of str2 = Friend
Copy str1 to str2 = Welcome
```

strdup() - String Function

String का duplicate बनाता है।

Syntax

```
strdup(string);
```

- **string** - ये String है जिसका duplicate बनाया जाता है।

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
int main(){
char str[] = "Hello World";
char *ptr;
ptr = strdup(str);
    cout<<"Duplicate string : "<<ptr;
return 0;
}
```

Output :

```
Duplicate string : Hello World
```

strlen() - String Function

String की Length निकाली जाती है ।

Syntax

```
strlen(string);
```

- **string** - ये एक normal string है, जिसकी length निकली जाती है ।

strlen से निकला हुआ output integer value ही होती है ,ये किसी दूसरे integer variable में भी store करके रख सकते हैं ।

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
int main(){
char str[30] = "Hello World";
int length ;
    cout<<"String : "<<str<<endl;
length = strlen(str);
    cout<<"Length of str is "<<length;
return 0;
}
```

Output :

```
String : Hello World
Length of str is 11
```

strlwr() - String Function

Uppercase के Characters को Lowercase में convert किया जाता है ।

Syntax

```
strlwr(string);
```

- **string** - ये वो string है जिसको lowercase में convert किया जाता है ।

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
int main(){
char str[50] = "HELLO WORLD";
int lwr ;
    cout<<"String : "<<str<<endl;
    cout<<"Lowercase of str = "<<strlwr(str);
return 0;
}
```

Output :

```
String : HELLO WORLD
Lowercase of str = hello world
```

strncat() - String Function

दिए हुए number के जितने character है उनसे String को जोड़ा जाता है ।

Syntax

```
strncat(destination_string, source_string, size_t num);
```

- **destination_string** - ये वो string जिसके साथ source string को जोड़ा जाता है ।
- **source_string** - ये वो string जिसके साथ destination string को बाद में जोड़ा जाता है ।
- **size_t num** - यहाँ पर जो integer value दी जाती है उतने character वो source string से लेता है ।

Program में strncat(str1, str2, 2); ऐसा लिखा है , इसका मतलब ये है कि, ये 2 characters str2 से लेगा ।

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
int main(){
char str1[20] = "Hello";
char str2[20] = " World";
strncat(str1, str2, 2);
    cout<<"String : "<<str1;
return 0;
}
```

Output :

```
String : Hello W
```


strncpy() - String Function

दिए हुए number के जितने character एक string से दूसरे string में copy किया जाता है ।

Syntax

```
strncpy(destination_string, source_string, size_t num);
```

- **destination_string** - ये वो parameter है जिसपर source के string की value copy की जाती है । अगर destination string पर कोई value भी हो तो वो overwrite हो जाती है ।
- **source_string** - ये वो parameter है जिसकी value destination पर copy की जाती है ।
- **size_t num** - यहाँ पर जो integer value दी जाती है उतने character वो destination string से लेकर source string पर copy कर देता है ।

Program में strncpy(str1, str2, 2); ऐसा लिखा है , इसका मतलब ये है कि, ये 2 characters str2 से लेगा ।

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
int main(){
char str1[20] = "Welcome";
char str2[20] = "Friend";
    cout<<"Value of str2 = "<<str2<<endl;
    strncpy(str2, str1, 2);
    cout<<"Copy str1 to str2 = "<<str2;
    return 0;
}
```

Output :

```
Value of str2 = Friend
Copy str1 to str2 = Weiend
```

strnset() - String Function

दिए हुए number और दिए हुए character के हिसाब से string को replace करता है।

Syntax

```
strnset(string, char ch, int c);
```

- **destination_string** - ये एक normal string है।
- **char ch** - ये वो character है जिससे string के हर character को replace किया जाता है।
- **int c** - यहाँ पर जितना number है उतने character string से replace किया जाते हैं।

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
int main(){
char str[20] = "Hello World";
    cout<<"Value of str = "<<str<<endl;
    strnset(str, '.', 2);
    cout<<str;
    return 0;
}
```

Output :

```
Value of str = Hello World
..llo World
```

strrchr() - String Function

दिए हुए string से एक character का आखिरी occurrence के आगे का string pointer को return करता है।

Syntax

```
strrchr(string, int character);
```

- **string** - ये एक normal string है।
- **int character** - ये वो character है जिसका आखिरी occurrence के आगे का string pointer को return किया जाता है।

अगर strrchr() function को कोई character नहीं मिलता तो वो NULL character return करता है।

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
int main(){
char str[20] = "Hello World";
char *ptr;
ptr = strrchr(str, 'o');
cout<<ptr;
return 0;
}
```

Output :

```
orld
```

strrev() - String Function

String को उलटी दिशा से print करता है।

Syntax

```
strrev(string);
```

- **string** - ये एक normal string है।

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
int main(){
char str[20] = "Hello World";
strrev(str);
    cout<<str;
return 0;
}
```

Output :

```
dlroW olleH
```

strstr() - String Function

दिए हुए String का आखिरी string occurrence के आगे का string pointer को return करता है ।

Syntax

```
strstr(string1, string2);
```

- **string** - ये एक normal string है ।
- **string2** - string1 में से ये string find करके उसका आखिरी occurrence pointer को return करता है ।

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
int main(){
char str[100] = "Hi How are you? Hi I am Fine";
char *ptr;
ptr = strstr(str, "Hi");
    cout<<ptr;
return 0;
}
```

Output :

```
Hi I am Fine
```

strset() - String Function

दिए हुए String का आखिरी string occurrence के आगे का string pointer को return करता है ।

Syntax

```
strset(string, int character);
```

- **string** - ये एक normal string है ।
- **int character** - ये एक-एक character करके सभी characters को replace कर देता है ।

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
int main(){
char str[50] = "Hi How are you? Hi I am Fine";
strset(str, '$');
    cout<<str;
return 0;
}
```

Output :

```
$$$$$$$$$$$$$$$$$$$$$$$$$
```

strstr() - String Function

दिए हुए String का आखिरी string occurrence के आगे का string pointer को return करता है ।

Syntax

```
strstr(string1, string2);
```

- **string** - ये एक normal string है ।
- **int character** - string1 में से ये string find करके उसका पहला occurrence pointer को return करता है ।

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
int main(){
char str[100] = "Hi How are you? Hi I am Fine";
char *ptr;
ptr = strstr(str, "Hi");
    cout<<ptr;
return 0;
}
```

Output :

```
Hi How are you? Hi I am Fine
```

strupr() - String Function

Lowercase के Characters को Uppercase में convert किया जाता है ।

Syntax

```
strupr(string);
```

- **string** - ये वो string है जिसको Uppercase में convert किया जाता है ।

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
int main(){
char str[50] = "hello world";
int lwr ;
    cout<<"String : "<<str<<endl;
    cout<<"Uppercase of str = "<<strupr(str);
return 0;
}
```

Output :

```
String : hello world
Uppercase of str = HELLO WORLD
```


Introduction Of Structure

Structure ये एक अलग-अलग data types का collection होता है ।

अगर structure का इस्तेमाल करना हो तो 'struct' keyword का इस्तेमाल करते है ।

Structure ये array के जैसा ही होता है ।

Array similar data types का collection होता है और Structure different data type का collection होता है ।

Structure में किये हुए हर variable के decaration को 'member' कहते है ।

Structure हर एक member के लिए अलग-अलग memory allocate करता है ।

Syntax for Structure Definition

```
struct structure_name{  
    data_type member 1;  
    data_type member 2;  
    -----  
    -----  
    data_type memeber n;  
};
```

Example for Structure Definition :

```
struct Employee{  
    int emp_id;  
    char emp_name[30];  
    float salary;  
};
```

यहाँ पर example में structure का नाम 'Employee' लिया है और structure के तीन Members है जिनके अलग-अलग तीन data types है । एक 'emp_id' के लिए integer है, एक 'emp_name' के लिए character है, एक float है जो 'salary' के लिए है ।

Structure Variable Declaration

Structure का variable declare करने के दो प्रकार है ।

जब Structure की definition लिखी जाती तब वहा पर भी Structure variable को लिखा जाता है ।

Structure के variable को main() function के अंदर भी किया जाता है ।

Syntax for Structure Variable outside of Structure Definition

```
struct structure_name{
    data_type member 1;
    data_type member 2;
    -----
    -----
    data_type memeber n;
}structure_variable(s);
```

Example for Structure Variable in main() Function

```
struct Employee{
    int emp_id;
    char emp_name[30];
    float salary;
}info;
```

Syntax for Structure Variable in main() Function

```
struct structure_name{
    data_type member 1;
    data_type member 2;
    -----
    -----
    data_type memeber n;
}structure_variable(s);
int main(){
    struct structure_name structure_variable_name;
}
```

Example for Structure Variable in main() Function

```
struct Employee{
    int emp_id;
    char emp_name[30];
    float salary;
};
int main(){
    struct Employee info;
}
```

Structure के Members या Elements को access कैसे किया जाता है ?

Syntax for Accesing Structure members

```
structure_variable_name . member_of_structure = value(optional);
```

Example for Accesing Structure members

```
info.emp_id = 10;
```

Structure Initilization

Structure Initialization के दो प्रकार है |

Type 1 :

```
struct Employee{
    int emp_id;
    char emp_name[30];
    float salary;
};
int main(){
    struct Employee info;
    info.emp_id = 34;
    strcpy( info.emp_name, "Raj Biradar");
    info.salary = 20000.00;
}
```

Type 2 :

```
struct Employee{
    int emp_id;
    char emp_name[30];
    float salary;
};
int main(){
    struct Employee info = {34, "Raj Biradar", 20000.00};
}
```

Full Example for Structure

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
struct Employee {
    int emp_id;
    char emp_name[30];
    float salary;
}info;
int main(){
    info.emp_id = 34;
    strcpy( info.emp_name, "Raj Biradar");
    info.salary = 20000.50;
    cout<<"Employee id is : "<<info.emp_id<<endl;
    cout<<"Employee name is : "<<info.emp_name<<endl;
    cout<<"Employee salary is : "<<info.salary<<endl;
    return 0;
}
```

Output :

```
Employee id is : 34
Employee name is Raj Biradar
Employee salary is : 20000.5
```

Structure working with size(sizeof)

Structure के member अलग-अलग memory allocate करता है और पूरा Structure का size उनके members के जितनी होती है ।

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
struct Employee {
    int emp_id;
    char emp_name[20];
    float salary;
};
int main(){
    struct Employee info;
    cout<<"Size of Employee id is : "<<sizeof(info.emp_id)<<" bytes."<<endl; // size of
emp_id
    cout<<"Size of Employee name : "<<sizeof(info.emp_name)<<" bytes."<<endl; // size of
emp_name
    cout<<"Size of Employee salary is : "<<sizeof(info.salary)<<" bytes."<<endl; // size of
salary
    cout<<"Size of Employee structure : "<<sizeof(info)<<" bytes."<<endl; // size of
Employee
    return 0;
}
```

Output :

```
Size of Employee id is : 4 bytes
Size of Employee name : 20 bytes
Size of Employee salary is : 4 bytes
Size of Employee structure : 28 bytes
```

Structure Memory Representation

Memory Representation में देखे तो System 32-bit होने के कारण integer का size '4 Bytes' होता है | character का size bracket में दिए हुए size जितना ही होता है और float का size 4 Bytes ही होता है | Structure का size दिए हुए सभी Members के size जितना होता है | for eg. 4 (int) + 20 (char) + 4 (float) = 28 Bytes होता है |

In 32-Bit					
Member Name	int emp_id;		char emp_name[20];		float salary;
Member Value	34		Raj Biradar		20000.00
Member Address	6356724	4 gap	6356728	20 gap	6356748
Memory Allocation	4 Bytes		20 Bytes		4 Bytes

Structure Using Pointer

Structure के Members को दो प्रकार से access किया जाता है ।

- . (dot Operator)
- -> (pointer Operator)

Accessing Members using pointer Operator

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
struct Employee {
    int emp_id;
    char emp_name[30];
    float salary;
};
int main(){
    struct Employee info;
    struct Employee *ptr;
    ptr = &info;
    ptr->emp_id = 34;
    strcpy( ptr->emp_name, "Raj Biradar");
    ptr->salary = 20000.00;
    cout<<"Employee id is : "<<ptr->emp_id<<endl;
    cout<<"Employee name is : "<<ptr->emp_name<<endl;
    cout<<"Employee salary is : "<<ptr->salary<<endl;
    return 0;
}
```

Output :

```
Employee id is : 34
Employee name is Raj Biradar
Employee salary is : 20000.000000
```

Array of Structure

Syntax for Array Structure Declaration

```
struct structure_name structure_variable_name[array_size];
```

Example for Array Structure Declaration

```
struct Employee info[3];
```

अगर एक से ज्यादा Employee की information store करनी हो तो array का इस्तेमाल करता है।

Example for Structure Without using Array

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
struct Employee{
    int emp_id;
    char emp_name[20];
    float salary;
};
int main(){
    struct Employee info1, info2, info3;
    info1.emp_id=34;
    strcpy(info1.emp_name, "Raj");
    info1.salary = 20000.00;
    info2.emp_id=35;
    strcpy(info2.emp_name, "Maruti");
    info2.salary = 30000.00;
    info3.emp_id=36;
    strcpy(info3.emp_name, "Suraj");
    info3.salary = 40000.00;
    cout<<"Student 1"<<endl;
    cout<<"Employee id is : "<<info1.emp_id<<endl;
    cout<<"Employee name is : "<<info1.emp_name<<endl;
    cout<<"Employee salary is : "<<info1.salary<<endl<<endl;
    cout<<"Student 2"<<endl;
    cout<<"Employee id is : "<<info2.emp_id<<endl;
    cout<<"Employee name is : "<<info2.emp_name<<endl;
    cout<<"Employee salary is : "<<info2.salary<<endl<<endl;
    cout<<"Student 3"<<endl;
    cout<<"Employee id is : "<<info3.emp_id<<endl;
    cout<<"Employee name is : "<<info3.emp_name<<endl;
    cout<<"Employee salary is : "<<info3.salary<<endl;
    return 0;
}
```

Ouput :

```
Student 1
Employee id is : 34
Employee name is Raj
Employee salary is : 20000.000000
Student 2
Employee id is : 35
Employee name is Maruti
Employee salary is : 30000.000000
Student 3
Employee id is : 36
Employee name is Suraj
Employee salary is : 40000.000000
```

Same Example for Structure using Array

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
struct Employee{
    int emp_id;
    char emp_name[20];
    float salary;
};
int main(){
    struct Employee info[3];
    int i;
    info[0].emp_id=34;
    strcpy(info[0].emp_name, "Raj");
    info[0].salary = 20000.00;
    info[1].emp_id=35;
    strcpy(info[1].emp_name, "Maruti");
    info[1].salary = 30000.00;
    info[2].emp_id=36;
    strcpy(info[2].emp_name, "Suraj");
    info[2].salary = 40000.00;
    for(i=0; i<3; i++){
        cout<<"Student "<<i+1<<endl;
        cout<<"Employee id is : "<<info[i].emp_id<<endl;
        cout<<"Employee name is : "<<info[i].emp_name<<endl;
        cout<<"Employee salary is : "<<info[i].salary<<endl<<endl;
    }
    return 0;
}
```


Ouput :

Student 1

Employee id is : 34

Employee name is Raj

Employee salary is : 20000.000000

Student 2

Employee id is : 35

Employee name is Maruti

Employee salary is : 30000.000000

Student 3

Employee id is : 36

Employee name is Suraj

Employee salary is : 40000.000000

Union

What is Union

Union ये एक अलग-अलग data types का collection होता है ।
अगर union का इस्तेमाल करना हो तो 'union' keyword का इस्तेमाल करते है ।
Union ये structure के जैसा ही होता है ।
Union में किये हुए हर variable के decaration को 'member' कहते है ।
Union हर एक member के लिए अलग-अलग memory allocate नहीं करता है ।
Union के members एक ही memory location को share करते है ।
Union में जो member अपने size में बड़ा होता है, तो वो पूरे Union की size होती है ।

Syntax Difference between Structure and Union Defintion

Syntax for Structure Definition

```
struct structure_name{  
    data_type member 1;  
    data_type member 2;  
    -----  
    -----  
    data_type memeber n;  
};
```

Syntax for Union Definition

```
struct union_name{  
    data_type member 1;  
    data_type member 2;  
    -----  
    -----  
    data_type memeber n;  
};
```

Syntax में सिर्फ struct और union keyword में फर्क है । Structure के members अलग-अलग memory allocate करते है । Union के members एक ही member की memory पर store होते है ।

Example for Structure Definition

```
struct Employee{  
    int emp_id;  
    char emp_name[30];  
};
```

Example for Union Definition

```
union Student{  
    int stud_id;  
    char stud_name[30];  
};
```

Memory Representation

	Structure		Union		
Name	Employee		Student		
Member_name	emp_id	emp_name[30]	stud_id and stud_name[30]		Location
Memory Address	6356728	6356732	6356762		
Member Size	4bytes	30bytes	4bytes	30bytes	
Total Size	34bytes		30bytes		

Union Variable Declaration

Structure का variable declare करने के दो प्रकार है ।

जब Union की definition लिखी जाती तब वहा पर भी Structure variable को लिखा जाता है ।

Union के variable को main() function के अंदर भी किया जाता है ।

Syntax for Union Variable outside of Union Definition

```
struct Union_name{
    data_type member 1;
    data_type member 2;
    -----
    -----
    data_type memeber n;
}Union_variable(s);
```

Example for Union Variable in main() Function

```
struct Employee{
    int emp_id;
    char emp_name[30];
    float salary;
}info;
```

Syntax for Union Variable in main() Function

```
union Union_name{
    data_type member 1;
    data_type member 2;
    -----
    -----
    data_type memeber n;
}Union_variable(s);
int main(){
    union Union_name Union_variable_name;
}
```

Example for Union Variable in main() Function

```
union Employee{
    int emp_id;
    char emp_name[30];
    float salary;
};
int main(){
    union Employee info;
}
```

Full Example for Union

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
union Employee {
    int emp_id;
    char emp_name[30];
    float salary;
}info;
int main(){
    info.emp_id = 34;
    strcpy( info.emp_name, "Raj Biradar");
    info.salary = 20000.00;
    cout<<"Employee id is : "<<info.emp_id<<endl;
    cout<<"Employee name is : "<<info.emp_name<<endl;
    cout<<"Employee salary is : "<<info.salary<<endl;
    return 0;
}
```

Output :

```
Employee id is : 1184645120
Employee name is
Employee salary is : 20000
```

Union working with size(sizeof)

Union के member एक ही memory location में store होता है और पूरे Union का size जो सबसे बड़ा member होता है, वो size Union की होती है ।

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
union Employee {
    int emp_id;
    char emp_name[20];
    float salary;
};
int main(){
    union Employee info;
    cout<<"Size of Employee id is : "<<sizeof(info.emp_id)<<" bytes."<<endl; // size of
emp_id
    cout<<"Size of Employee name : "<<sizeof(info.emp_name)<<" bytes."<<endl; // size of
emp_name
    cout<<"Size of Employee salary is : "<<sizeof(info.salary)<<" bytes."<<endl; // size of
salary
    cout<<"Size of Employee union : "<<sizeof(info)<<" bytes."<<endl; // size of Employee
return 0;
}
```

Output :

```
Size of Employee id is : 4 bytes
Size of Employee name : 20 bytes
Size of Employee salary is : 4 bytes
Size of Employee union : 20 bytes
```

Union using Pointer

Union के Members को दो प्रकार से access किया जाता है ।

- . (dot Operator)
- -> (pointer Operator)

Accessing Members using pointer Operator

Source Code :

```
#include <iostream.h>
#include <string.h>
using namespace std;
union Employee {
    int emp_id;
    char emp_name[30];
    float salary;
};
int main(){
    union Employee info;
    union Employee *ptr;
    ptr = &info;
    ptr->emp_id = 34;
    strcpy( ptr->emp_name, "Raj Biradar");
    ptr->salary = 20000.00;
    cout<<"Employee id is : "<<ptr->emp_id<<endl;
    cout<<"Employee name is : "<<ptr->emp_name<<endl;
    cout<<"Employee salary is : "<<ptr->salary<<endl;
    return 0;
}
```

Output :

```
Employee id is : 1184645120
Employee name is :
Employee salary is : 20000
```

Preprocessors

Preprocessors Introduction

Preprocessor क्या होता है ?

Program के हर header area में `#include <stdio.h>` इस प्रकार के files को include करते हैं। इनकी शुरुआत '#' से होती है। इन्हें 'Preprocessor' कहते हैं। Preprocessor को statement की तरह semicolon (;) नहीं दिया जाता।

Preprocessor ये एक Program है, जो Source Code compile होने से पहले ही execute होता है।

Normal Example for Preprocessor

Source Code :

```
#include <iostream.h>
int main(){
    cout<<"Hello";
    return 0;
}
```

यहाँ पर Preprocessors को अलग-अलग विभाग में विभाजित किया है।

Directive Types	Preprocessors
Macro	<code>#define</code>
File Inclusion	<code>#include</code>
Conditional Compilation	<code>#if</code> , <code>#else</code> , <code>#ifdef</code> , <code>#endif</code> , <code>#ifndef</code> , <code>#elif</code>
Other Directives	<code>#pragma</code> , <code>#undef</code>

Macro #define

macros identifiers होते हैं, Program में जहाँ पर identifier होता है, तो वो replace होकर वहाँ पर दी हुई value आ जाती है।

Source Code :

```
#include <iostream.h>
#define PI 3.145
using namespace std;
int main(){
    cout<<"Value of PI : "<<PI;
    return 0;
}
```

Output :

```
Value of PI : 3.145000
Overwrite value of PI : 3.140000
```

Example for #define function macro

Source Code :

```
#include <iostream.h>
#define PI 3.145
#define Area(rad) PI*rad*rad
using namespace std;
int main()
{
    int radius = 5;
    float area;
    area = Area(radius);
    cout<<"Area of circle is "<<area;
    return 0;
}
```

Output :

```
Area of circle is 78.625000
```

Predefined Macros

Macro	Description
__DATE__ :	ये current date के string को return करता है ।
__FILE__ :	ये current file name को उसके path के साथ return करेगा ।
__LINE__ :	ये macro जिस line पर है उस line का number return करेगा ।
__STDC__ :	अगर compiler ANSI Standard C++ का पालन करता है तो वो '1' return करेगा ।
__TIME__ :	ये current time के string को return करता है ।

Example for predefined macro

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
    cout<<"Date : "<<__DATE__<<endl;
    cout<<"File : "<<__FILE__<<endl;
    cout<<"Line : "<<__LINE__<<endl;
    cout<<"STDC : "<<__STDC__<<endl;
    cout<<"Time : "<<__TIME__<<endl;
    return 0;
}
```


Output :

```
Date : Dec 19 2016
File : C:\UD\Documents\predefined_macro.c
Line : 7
STDC : 1
Time :16:38:22
```

File Inclusion #include

Header files को program में include करने के लिए #include का इस्तेमाल करते हैं।
हर एक program में #include का सन्दर्भ होता है।
Preprocessor दो प्रकार में include जाते हैं।

- #include
- #include "file_name"

कुछ header files predefined होते हैं और कुछ header files user-defined भी होती हैं।

- **Predefined header files** : iostream.h, conio.h, string.h, math.h
- **User-defined header file** : myheader.h

header files में functions होते हैं, जिस program में इन functions का इस्तेमाल करना हो तो header files को include करना पड़ता है।

- **stdio.h header files** : strlen(), strcat()
- **myheader.h header file** : myfunction()

Source Code :

```
#include <iostream.h>
#include
using namespace std;
int main(){
    cout<<"Hello World!";
    getch();
}
```

Output :

```
Hello World!
```

Conditional Compilation

Conditional Compilation - #if

#if ये एक conditional preprocessor है , जिसके आगे expression या condition को लिखा जाता है ।
अगर दी हुई condition true हो तो वो नीचेवाला statement / code execute करता है ।

Note :अगर Programmer CodeBlock के compiler में program बना रहा हो तो, उसे Program लिखते वक्त ही पता चलता है कि, कौनसा statement 'true' है और कौनसा 'false' ।

Syntax for #if

```
#if condition
    statement(s);
#endif
```

Example

```
#include <iostream.h>
#define num 1
using namespace std;
int main(){
    #if (num==1)
    {
        cout<<"Number is equal to 1";
    }
    #endif
    return 0;
}
```

Output :

```
Number is equal to 1
```

Conditional Compilation - #else

अगर #if की condition 'false' होती है तो #else का statement 'true' होता है।

Syntax for #else

```
#if condition  
statement(s);  
#else  
statement(s);  
#endif
```

Example

```
#include <iostream.h>  
#define num 1  
using namespace std;  
int main() {  
    #if (num==0)  
    {  
        cout<<"Number is equal to 1";  
    }  
    #else  
    {  
        cout<<"Number is equal to 1";  
    }  
    #endif  
    return 0;  
}
```

Output :

```
Number is equal to 1
```

Conditional Compilation - #elif

#elif ये else if का combination होता है | अगर #if की condition false हो तो, वो #elif के condition पर जाता है , अगर condition true होती तो उसका statement print होता है |

Syntax for #elif

```
#if condition
statement(s);
#elif condition
statement(s);
#else
statement(s);
#endif
```

Example

```
#include <iostream.h>
#define num 15
using namespace std;
int main(){
    #if (num<10)
    {
        cout<<"Number is less than 10";
    }
    #elif (num>10)
    {
        cout<<"Number is greater than 10";
    }
    #else #if (num==10)
    {
        cout<<"Number is equal to 10";
    }
    #endif
    return 0;
}
```

Output :

```
Number is greater than 10
```

Conditional Compilation - #ifdef

`#ifdef`(if defined) पहले program में `#define` ये preprocessor लिखा है या नहीं ये check करता है | अगर `#ifdef` का macro और `#define` का macro same होता है तो, `#ifdef` का statement 'true' होता है |

Syntax for #ifdef

```
#ifdef macro  
statement(s);  
#endif
```

Example

```
#include <iostream.h>  
#define POSITIVE 5  
using namespace std;  
int main() {  
#ifdef POSITIVE  
    cout<<"Defined Positive value";  
#else  
    cout<<"Defined Negative Value";  
#endif  
return 0;  
}
```

Output :

```
Defined Positive value
```

Conditional Compilation - #ifndef

#ifndef (if not defined) ये पहले #define के macro को check करता है, अगर #define का macro और #ifndef का macro अलग-अलग है तो, वो true return करता है।

Syntax for #ifndef

```
#ifndef macro  
statement(s);  
#endif
```

Example

```
#include <iostream.h>  
#define POSITIVE 5  
using namespace std;  
int main() {  
#ifndef NEGATIVE  
    cout<<"Defined Negative value";  
#endif  
return 0;  
}
```

Output :

```
Defined Negative value
```

Conditional Compilation - #endif

जहाँ पर #if का इस्तेमाल होता है वहाँ पर #endif का इस्तेमाल होता है।

Syntax for #ifndef

```
statement(s) of #if or #else;  
#endif
```

Example

```
#include <iostream.h>  
#define num 1  
using namespace std;  
int main() {  
    #if (num==1)  
    {  
        cout<<"Number is equal to 1";  
    }  
    #endif  
    return 0;  
}
```

Output :

```
Number is equal to 1
```

Other Directives #undef

Conditional Compilation - #endif

#undef ये preprocessor #define किये हुए macro को undefine कर देता है ।

Syntax for ##undef

```
#undef macro
```

Example 1

```
#include <iostream.h>
#define num 1
#undef num
using namespace std;
int main(){
    cout<<"num is "<<num;
    return 0;
}
```

Output :

```
error: 'num' undeclared
```

Example 2

```
#include <iostream.h>
#define num 1
using namespace std;
int main(){
    cout<<"num is "<<num;
    #undef num
    #define num 2
    cout<<"num is "<<num;
    return 0;
}
```

Output :

```
num is 1
num is 2
```


File Handling

C++ File Introduction

File Handling में data को secondary storage device(Hard disk) में permanently store किया जाता है।

File Handling को open, close, read, write के लिए इस्तेमाल किया जाता है। इससे पहले cin और cout इन दोनों object के लिए iostream.h का इस्तेमाल करते आये हैं। जब iostream इस header file का इस्तेमाल नहीं करते तो, istream class का cin और ostream class का cout को कोई किम्मत नहीं रहती। वैसे ही File Handling के लिए भी है।

C++ में File Handling एक ऐसा topic है जिसको अलग header file को दिया है, उसका नाम है fstream।

C++ File Uses

Program में cin और cout ये थोड़े समय के लिए ही memory को store करके रखता है यानी कि, Program जब Programmer बंद करता है, तब program का सारा data destroy हो जाता है। Program में data store करने के Programmer कुछ variables, arrays, structures, unions का इस्तेमाल करता है, पर ये data permanently store नहीं होता। इसे permanently store करने के लिए ही File Handling का इस्तेमाल होता है। File Handling के दरम्यान create हुई files चाहे वो अलग-अलग types(.txt, .doc etc.) की हो वो portable होती है। दूसरे Computer में भी वो इस्तेमाल होती है।

अन्ततः File Handling के लिए fstream इस header file का इस्तेमाल करना पड़ता है।

fstream इस header file में तीन classes आते हैं।

ifstream : ifstream का इस्तेमाल file को read करने के लिए किया जाता है।

ofstream : ofstream का इस्तेमाल file को write करने के लिए किया जाता है।

fstream : fstream का इस्तेमाल file को read और write करने के लिए किया जाता है।

fstream class में दो ifstream और ofstream ये दो classes होते हैं।

अगर file को read करना हो तो, ifstream की जरूरत पड़ेगी और file पर data को write करने हो तो, ofstream की जरूरत पड़ेगी।

लेकिन file को किस mode पे Open करना है ये समझ लेना चाहिए।

C++ File Modes

File के सात modes बनाए गए हैं।

Modes	Type of Stream	Description
ios::in	ifstream	read mode पर file को Open किया जाता है।
ios::out	ofstream	write mode पर file को Open किया जाता है।
ios::binary	ifstream ofstream	Binary file को open किया जाता है। अगर file को कोई extension ना दिया जाए तो default .txt mode पे file Open होती है।
ios::ate	ifstream ofstream	end-of-file पर data को जोड़ा जाता है और file के पूरे data को existing data के साथ control किया जाता है।
ios::app	ofstream	end-of-file पर data को append किया जाता है। मतलब data को जोड़ने के लिए इसका इस्तेमाल होता है।
ios::trunc	ofstream	File में से data को मिटाया जाता है।
ios::nocreate	ofstream	File पहले से ही exist ना हो तो open function fail हो जाता है और file create भी नहीं होती।
ios::noreplace	ofstream	File पहले से ही exist हो तो open function fail हो जाता है और new file create भी नहीं होती।

अगर stream class 'ifstream' इस्तेमाल किया जाए तो file default ios::in(read) mode पर open होती है।

अगर stream class 'ofstream' इस्तेमाल किया जाए तो file default ios::out(write) mode पर open होती है।

C++ Opening File

Syntax For Opening File

```
stream-class stream-object;  
stream-object.open("file_name");
```

For Example

```
ofstream fout;    // create stream  
fout.open("sample.txt"); // default mode is ios::out
```

File को open करने के लिए Bitwise OR Operator(|) के साथ एक से ज्यादा modes भी इस्तेमाल किये जाते हैं।

For Example

```
fstream fout;  
fout.open("sample.txt", ios::in|ios::out);
```

C++ Closing File

File जब open होती है, तब उसे close भी करना पड़ता है। जब file close की जाती है, तब file जो memory allocate करता उसे free कर दिया जाता है।

Syntax For Closing File

```
stream-object.close();
```

For Example

```
fout.close();
```

C++ Check File Open or not!

जब file open होती है तो, उसे जाँचना पड़ता है कि वो file open हुई है या नहीं। इसके लिए is_open() या fail() इन member functions का इस्तेमाल किया जाता है।

is_open() और fail ये दोनों member function boolean values return करते हैं।

Source Code :

```
#include <iostream.h>  
#include  
using namespace std;  
int main(){  
    ifstream fout;  
    fout.open("sample.txt");  
    if(fout.is_open()){    // or if(!fout.fail()){  
        cout<<"File is Opened."<<endl;  
    }else{  
        cout<<"Unable to open file."<<endl;  
    }  
    fout.close();  
    return 0;  
}
```

Output :

```
File is Opened.
```

C++ Read Data from a File

निचे दिया हुआ program File को read करने के लिए लिया है ।

वहा पर character data type का एक variable लिया है, जिसमे file के data को store करने के लिए लिया है ।

ifstream के stream-object का नाम fout लिया है और बाद में file को open किया है । पर open member function में mode के लिए कोई argument नहीं लिया , इसका कारण ये है कि, अगर ifstream class का इस्तेमाल किया जाए तो ios::in वहा पर पहले से ही स्थित(default) होता है ।

बाद में file open हुई है या नहीं हुई है ये condition के द्वारा check किया गया है । अगर file open हुई है, तो Program continue होता है और अगर file open नहीं होती तो, Program exit()(stdlib.h) इस function के द्वारा terminate किया जाएगा ।

बाद में while loop के द्वारा file end-of-file तक पहुंची है या नहीं पहुंची ये check करने के लिए इस्तेमाल किया गया है ।

sample.txt

Hello World!

Source Code :

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
using namespace std;
int main(){
char ch[30];
ifstream fout;
fout.open("sample.txt") ;
if(!fout){
    cout<<"Unable to opened file!";
    exit(1);    //Program terminate if file is not opened.
}
    cout<<"File Contents : ";
while(!fout.eof()){
    fout>>ch;
    cout<<ch<<" ";
}
fout.close();
return 0;
}
```

Output :

File Contents : Hello World!

C++ Write data on a File

Source Code :

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
using namespace std;
int main(){
    ofstream fout;
    fout.open("sample.txt") ;
    if(!fout){
        cout<<"Unable to opened file!";
        exit(1);    //Program terminate if file is not opened.
    }
    fout<<"Hello Friends!";
    fout.close();
    return 0;
}
```

Output :

sample.txt

Hello Friends!

C++ get() Member Function

Syntax for get() for File Handling

```
stream-object_of_ifstream_class.get(char ch); or
char ch = stream-object_of_ifstream_class.get();
```

Syntax for get() for Console

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

get() memeber function से single character को input किया जाता है |g

et() function istream class में define होता है | इसका मतलब ये function इस्तेमाल करने के लिए iostream इस header file को include करना पड़ता है |

get() unformatted stream IO functions है |

get() function data को read करने के लिए इस्तेमाल किया जाता है |

cin>>ch और cin.get(ch); में फर्क क्या है ?

- cin>>ch; ये newline और whitespace लेता नहीं है |
- cin.get; ये newline और whitespaces को लेता है |

Source Code :

```
#include <iostream.h>
#include <fstream.h>
using namespace std;
int main(){
char ch;
    cout<<"Enter one character : ";
    cin.get(ch);
    cout<<"Entered character : ";
    cout<<ch;
return 0;
}
```

Output :

```
Enter one character : o
Entered character : o
```

get() with file

sample.txt

```
Hello Friends!
```

Source Code :

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
using namespace std;
int main(){
char ch;
ifstream fin;
fin.open("sample.txt") ;
if(!fin){
    cout<<"Unable to opened file!";
    exit(1);    //Program terminate if file is not opened.
}
cout<<"File Contents : ";
while(!fin.eof()){
    fin.get(ch);
    cout<<ch;
}
fin.close();
return 0;
}
```

Output :

```
File Contents : Hello Friends!
```

C++ put() Member Function

Syntax for put() for File Handling

```
stream-object_of_ofstream_class.put(char ch);
```

Syntax for put() for Console

```
cout.put(char ch);
```

put() member function से single character को Output किया जाता है ।

put() function ostream class में define होता है । इसका मतलब ये function इस्तेमाल करने के लिए iostream इस header file को include करना पड़ता है ।

put() unformatted stream IO functions है ।

put() function data को write करने के लिए इस्तेमाल किया जाता है ।

Source Code :

```
#include <iostream.h>
#include <fstream.h>
using namespace std;
int main(){
char ch='H', c = 65;
    cout<<"Value of ch : ";
    cout.put(ch)<<endl;
    cout<<"Value of c : ";
    cout.put(c);    // ASCII value of a is 65
return 0;
}
```

Output :

```
Value of ch : H
Value of c : A
```

put() with file

Source Code :

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
using namespace std;
int main(){
char ch='H';
ofstream fout;
fout.open("sample.txt") ;
if(!fout){
    cout<<"Unable to opened file!";
    exit(1);    //Program terminate if file is not opened.
}
    fout.put(ch);
fout.close();
return 0;
}
```

Output :

sample.txt

H

C++ write() Member Function

Syntax for write() for File Handling

```
stream-object_of_ofstream_class.write(line_or_string, stream_size);
```

Syntax for write() for Console

```
cout.write(line_or_string, stream_size);
```

Concatenate write() function with dot operator

```
cout.write(line_or_string, stream_size).write(line_or_string, stream_size);
```


write() function ये output function है, जो line को display करता है |

write() function ये ostream class का member function है |

write() function के लिए दो arguments होते है |

line_or_string : ये character type का array होता है |

stream_size : Line से कितने characters लेना है ये यहाँ पर लिखा जाता है | अगर stream_size ज्यादा हुई तो whitespaces भी लिए जाते है |

Source Code :

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
using namespace std;
int main(){
char ch[12] = "Hello World!";
    cout.write(ch, 12);
return 0;
}
```

Output :

sample.txt

Hello World!

write() with file

Source Code :

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
using namespace std;
int main(){
char ch[12] = "Hello World!";
ofstream fout;
fout.open("sample.txt") ;
if(!fout){
    cout<<"Unable to opened file!";
    exit(1);    //Program terminate if file is not opened.
}
    fout.write(ch, 12);
fout.close();
return 0;
}
```

Output :

sample.txt

```
Hello World!
```

C++ getline() Member Function

Syntax for getline() for File Handling

```
stream-object_of_ifstream_class.getline(line_or_string, stream_size);
```

Syntax for getline() for Console

```
cin.getline(line_or_string, stream_size);
```

getline() function ये input function है, जो line को keyboard से input करता है ।

getline() function ये istream class का member function है ।

getline() function के string के आखिर में NULL character(\0) होता है ।

sample.txt

```
Hello World!
```

Source Code :

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
using namespace std;
int main(){
char ch[12];
    cout<<"Enter String : ";
    cin.getline(ch, 12);
    cout<<"Entered String : "<<ch;
return 0;
}
```

Output

```
Enter String : Hello World!
Entered String : Hello World
```

getline() with file

sample.txt

Hello World!

Source Code :

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
using namespace std;
int main(){
char ch[12];
ifstream fin;
fin.open("sample.txt") ;
if(!fin){
    cout<<"Unable to opened file!";
    exit(1);    //Program terminate if file is not opened.
}
    fin.getline(ch, 12);
    cout<<ch;
fin.close();
return 0;
}
```

Output

Hello World

Recursion

जिस function में वही function को call किया जाता है, उसे recursion कहते हैं।

Recursion एक ऐसा process है, जो loop के तरह काम करता है।

Recursion को एक satisfied condition लगती है, जिससे recursive function काम करना बंद कर दे।

Recursive Function तबतक call होता रहता है, जबतक उसका satisfaction नहीं होता।

अगर Recursive function का satisfaction नहीं हो तो 'infinite looping' की संभावना होती है।

Recursive function कैसा होता है ?

```
void recursive_function(){
//some code;
    recursive_function();
//some code;
}
int main(){
//some code;
    recursive_function();
//some code;
}
```

Source Code :

```
#include <iostream.h>
using namespace std;
int factorial (int n){
if (n == 0){
    return 1;
}else{
    return (n * factorial(n-1));
}
}
int main(){
int a,fact=0;
    cout<<"Enter Number : ";
    cin>>a;
fact=factorial(a);
    cout<<"Factorial of "<a<<" is "<fact;
return 0;
}
```

Output

```
Enter Number : 4
Factorial of 4 is 24
```

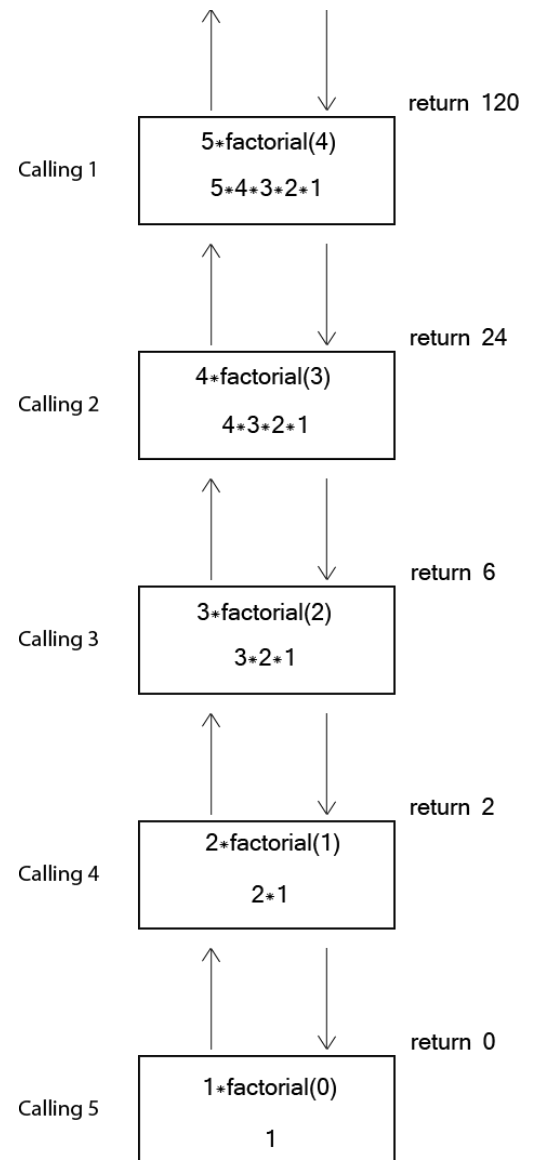
Example for Addition of Two Numbers using Recursion

Source Code :

```
#include <iostream.h>
using namespace std;
int add(int, int);
int main()
{
    int num1,num2,sum;
    cout<<"Enter Two Values"<<endl;
    cin>>num1>>num2;
    sum = add(num1, num2);
    cout<<"sum of "<<num1<<" and "<<num2<<" : "<<sum;
    return 0;
}
add(int a, int b){
    if(b==0){
        return a;
    }else{
        return (1+add(a,b-1));
    }
}
```

Output

```
Enter Two Values
2
4
sum of 2 and 4 : 6
```



Command Line Argument

C++ Program को run करने के लिए Programmer अलग-अलग Compilers का इस्तेमाल करते हैं जैसेकि, Turbo C/C++, CodeBlock और Dev C/C++ इत्यादी .

Compiler से program run कैसे होता है ?

Programmer पहले Source Code बनाता है, जिसका नाम Programmer कुछ भी रख सकता है, लेकिन .c extension/type लगाना अनिवार्य होता है। बाद में Preprocessor pre-process होता है। Pre-process होने के बाद Compiler के द्वारा Program को compile किया जाता है, Compile होने से जो IDE Programmer इस्तेमाल कर रहा है उसके द्वारा जिस file का नाम .cpp file type के साथ रखा है, उस file के नाम के साथ .obj extension की Object File तैयार होती है। बाद में Linker के साथ .lib library file तैयार होती है जो object file के साथ merge होती है। उसके बाद .exe executable file बनती है। जो हर Computer User अपने Computer में इस्तेमाल करता है।

Syntax for Command Line Arguments

```
int main( int argc, char *argv[])
```

Command Line के लिए main function में दो arguments होते हैं।

1. argc
2. argv[]

1. argc(Argument Counter)

ये integer type का variable या argument है।
इसमें arguments के total numbers main function को pass किये जाते हैं।

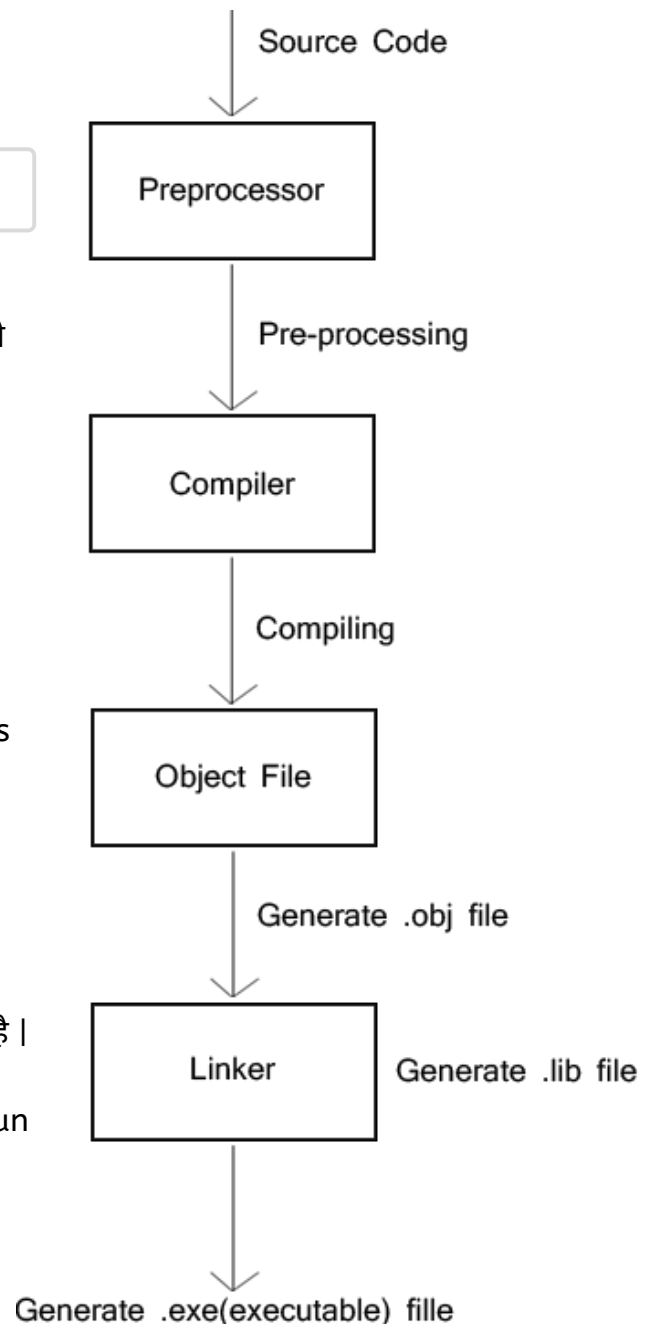
Command line में file के नाम को भी वो argument लेता है

2. argv[(Argument Vector)

ये character pointer type का argument होता है।
ये actual arguments को main function को pass किये जाते हैं।
ये सभी argument array बनाकर लेता है।

Command Line Arguments किसी भी IDE पर Compile या Run नहीं होता।

Command Line Arguments सिर्फ Command Prompt पर Run होते हैं।



Example for Command Line Argument

command.cpp

Source Code :

```
#include <iostream.h>
using namespace std;
int main(int argc, char *argv[]){
    int i;
    cout<<"Number of Argument : "<<argc<<endl;
    for(i=1; i<argc; i++){
        cout<<"Argument "<<i<<" = "<<argv[i]<<endl;
    }
}
```

Command Line से Program Run कैसे किया जाता है ?

पहले Command Prompt को open करे |

Command Prompt को Open करने के लिए Window+R Press करे |

बाद में एक Run का Dialog Box खुल जायेगा उसमे 'cmd' नाम का Program डाले |Command Prompt खुल जाएगा |

Drive को change करने के लिए जिस Drive पर जाना है उस Drive का नाम और उसके साथ 'colon' दे | for eg.E:, C:, D:

```
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.
C:\Users\UD>E:
E:\>
```

Directory को change करने के लिए 'cd' type करे और जिस directory पर जाना है उसका path दे |

```
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.
C:\Users\UD>E:
E:\>cd E:\UD\Documents
E:\UD\Documents>
```

command.cpp

Source Code :

```
#include <iostream.h>
using namespace std;
int main(int argc, char *argv[]){
    int i;
    cout<<"Number of Argument : "<<argc<<endl;
    for(i=1; i<argc; i++){
        cout<<"Argument "<<i<<" = "<<argv[i]<<endl;
    }
}
```

Program Command line से run करने से पहले IDE से run करे, अगर बिना error से program चले तो फिर Command Line से चलाये |

```
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.
C:\Users\UD>E:
E:\>cd E:\UD\Documents
E:\UD\Documents>command Hi! Hello Friends
Number of Argument 4
Argument 1 : Hi!
Argument 2 : Hello
Argument 3 : Friends
E:\UD\Documents>
```

Example for Addition two Numbers using Command Line Arguments


commandline.cpp

Source Code :

```
#include <iostream.h>
#include <stdlib.h>
using namespace std;
int main(int argc, char * argv[]) {
    int i, sum = 0;
    if (argc != 3) {
        cout<<"Enter only Two Arguments.";
        exit (1);
    }
    cout<<"Addition = "<<sum;
    for (i = 1; i < argc; i++)
        sum = atoi(argv[1]) + atoi(argv[2]); // atoi is used to convert string to integer
    cout<<sum;
}
```

Output :

```
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.
C:\Users\UD>E:
E:\>cd E:\UD\Documents
E:\UD\Documents>commandline 4 6
Addition = 010
E:\UD\Documents>
```

विभाग २

C++

Object-Oriented Programming

Classes & Objects

OOP के लिए Class और Object बहुत ही महत्वपूर्ण हिस्सा है ।

OOP का concept Objects और Classes पर निर्भर है ।

Class के अन्दर बहुत सारे member function और data members हो सकते है । जिसको object के जरिये access किया जाता है ।

Class के अन्दर जो variables होते है उन्हें data members कहते है और जो functions होते है उन्हें member functions कहते है ।

उदाहरण में,

अगर कोई प्राणी है तो उस प्राणी का behavior और properties जैसे कि, उसका भौकना, चलना, देखना , उसके शरीर की रचना को data members और member functions के जरिये लिखा जाता है ।

हर एक से अधिक प्राणी को अलग-अलग नाम से उनके objects भी बनाये जाते है ।

Class क्या होता है ?

- Class एक structure के जैसा होता है । जिसमे variable(data members) और function(member functions) को एक ही जगह पर इकट्ठा किये जाते है ।
- Data और functions class के members होते है ।
- Class data को hold करने का काम करता है ।
- Class ये object का layout होता है ।

Defining Class

Class को define करने के लिए class keyword का इस्तेमाल किया जाता है । इसके साथ 'Classname' जाता है ।

Class के नाम का पहला character Uppercase या Lowercase दोनों भी हो सकता है । लेकिन Class के नाम का पहला अक्षर Uppercase में होना एक 'Good Programming' कहलाता है ।

Class के लिए और भी कुछ महत्वपूर्ण हिस्से है । जैसे कि, Access Specifier

Access Specifier / Modifier का use Access Control करने के लिए किया जाता है ।

Access Specifier तीन प्रकार के होते है ।

- 1.private
- 2.public
- 3.protected

1. **private** : private में class के members मतलब variables लिखे जाते हैं। private के बिना private के members लिखे जाते हैं वो default private member होता है। private member सिर्फ उसके class के लिए ही काम करते हैं। ये class के बाहर access नहीं होते।

2. **protected** : protected का काम private की तरह ही होता है। ये inherited होते हैं। इनका इस्तेमाल Inheritance में किया जाता है।

3. **public** : public के members class के अंदर और बाहर दोनों ही जगह पर access किये जाते हैं। इसके अन्दर data members को भी लिखा जाता है।

Syntax for Class

```
class Classname{  
    data member(s);  
    member function(s);  
};
```

For Example,

निचे दिए हुए example में 'class' keyword के साथ class का नाम 'Number' और curly brace को open({) किया है। इसके साथ data members और member function define किया गया है। इसके बाद curly brace close(}) करके आखिर में semicolon (;) दिया है।

```
class Classname{  
    data member(s);  
    member function(s);  
};
```

```
class Number{  
    private:  
    int a;    //data member  
    public:  
    getdata(); //member function  
};
```

Object क्या होता है ?

- Objects variables जैसे होते हैं।
- Object के लिए class को as a data type लिया जाता है और उसे एक या एक से अधिक variables(objects) दिए जाते हैं।
- Class के Object को main() function के अन्दर लिखा जाता है।
- Object और access operator के साथ class के member functions को access किया जाता है।

Syntax for Object Declaration

जैसे variables को declare किया जाता है, वैसे ही object को classname को as a data type को लेकर declare किया जाता है।

```
classname object(s);
```

Output :

```
Number n1, n2; // n1 and n2 is a objects of class Number
```

Accessing Class Members

Class के data को access करने के लिए object और member function का उपयोग किया जाता है। इसके साथ access operator(.) की जरूरत पड़ती है।

Syntax for Accessing Class Members

```
object_name.member_function();
```

For Example

```
n1.getdata();
```

Source Code :

```
#include <iostream.h>
using namespace std;
class Number{
private:
    int num;
public:
    getdata(){
        cout <<"Enter value of num : ";
        cin>>num;
    }
    putdata(){
        cout<<"Value of num : "<<num;
    }
};
int main(){
    Number n1;
    n1.getdata();
    n1.putdata();
    return 0;
}
```

Output :

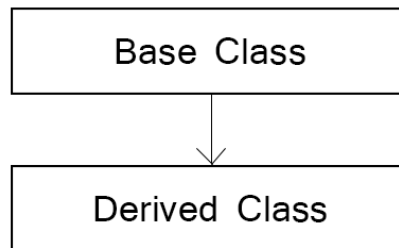
```
Enter value of num : 5
Value of num : 5
```

Inheritance

C++ Introduction of Inheritance

OOP में Inheritance ये बहुत ही महत्वपूर्ण हिस्सा है | Inheritance में दो प्रकार के के classes होते है |

- **Base Class** : Base class को 'parent या super class' भी कहा जाता है | Base class में data members और member functions होते है |
- **Derived Class** : Derived class को 'child या sub class' भी कहा जाता है | Derived class; Base class की properties को inherit करता है |



Inheritance के पांच प्रकार है |

- Single Inheritance
- Multilevel Inheritance
- Multiple Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

Syntax for Inheritance

```
class Derived_Class : Access_Specifier Base_Class
```

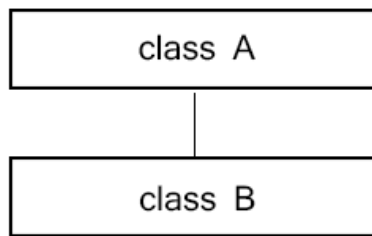
C++ Single Inheritance

Single Inheritance ये Inheritance का सबसे पहला और आसान प्रकार है |

Single Inheritance में एक Base Class की properties एक derived class को inherit की जाती है |

Syntax

```
class Base_class_Name
{
    // body_of_Base_class
};
class Derived_class_Name : Access_Specifier Base_class_Name
{
    //body_of_Derived_class
};
```



जब member functions class के अन्दर declare किये जाते हैं, तब Syntax

```
return_type Member_function(){ some_code; }
```

जब member functions class के बाहर declare किये जाते हैं, तब Syntax

```
return type class_name :: member_function(){ some_code; }
```

निचे दिए हुए program में member functions को class के अन्दर declare किया गया है ।

Example

```
#include <iostream.h>
using namespace std;
class Employee{
int emp_id;
char emp_name[20];
int emp_salary;
public:
void getdata(){
cout<<"Enter Employee id : ";
cin>>emp_id;
cout<<"Enter Employee name : ";
cin>>emp_name;
cout<<"Enter Employee salary : ";
cin>>emp_salary;
}
void putdata(){
cout<<"Employee id : "<<emp_id<<endl;
cout<<"Employee name : "<<emp_name<<endl;
cout<<"Employee salary : "<<emp_salary<<endl;
}
};
```

```

class fitness : public Employee
{
float height,weight;
public:
void accept(){
cout<<"Enter Height in feet: ";
cin>>height;
cout<<"Enter Weight in kg: ";
cin>>weight;
}
void display(){
cout<<"Employee Height is "<<height<<" feet"<<endl;
cout<<"Employee Weight is "<<weight<<" kg"<<endl;
}
};
int main(){
fitness F;
cout<<" Enter Employee details"<<endl;
F.getdata();
F.accept();
cout<<"Employee details are"<<endl;
F.putdata();
F.display();
return 0;
}

```

Output :

```

Enter Employee details
Enter Employee id : 5
Enter Employee name : Rakesh
Enter Employee salary : 20000
Enter Height in feet: 5.9
Enter Weight in kg: 65
Employee details are
Employee id : 5
Employee name : Rakesh
Employee salary : 20000
Employee Height is 5.9 feet
Employee Weight is 65 kg

```

C++ Multilevel Inheritance

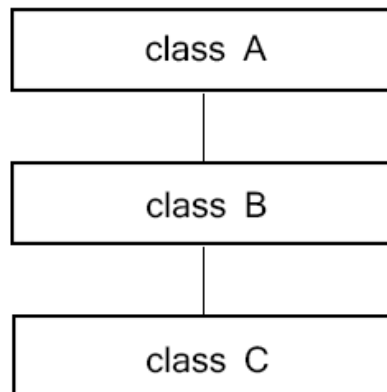
Multilevel Inheritance; Inheritance का दूसरा प्रकार है ।

Multilevel Inheritance में एक base class और दो derived class होते हैं ।

Multilevel Inheritance में derived class का base class होता है और derived class का भी derived class होता है ।

Syntax

```
class Base_class_Name
{
    // body_of_Base_class
};
class Derived_class_Name1 : Access_Specifier Base_class_Name
{
    //body_of_Derived_class1
};
class Derived_class_Name2 : Access_Specifier Derived_class_Name1
{
    //body_of_Derived_class2
};
```



Source Code :

```
#include <iostream.h>
using namespace std;
class base_class{
public:
    void show(){
        cout << "I am a Base Class."<<endl;
    }
};
class derived_class1 : public base_class{
public:
    void get(){
        cout << "I am a Derived Class1."<<endl;
    }
};
class derived_class2 : public derived_class1{
public:
    void display(){
        cout << "I am a Derived Class2."<<endl;
    }
};
int main(){
    derived_class2 d;    // Access all data from derived_class2
    d.show();
    d.get();
    d.display();
    return 0;
}
```

Output :

```
I am a Base Class.
I am a Derived Class1.
I am a Derived Class2.
```

C++ Multiple Inheritance

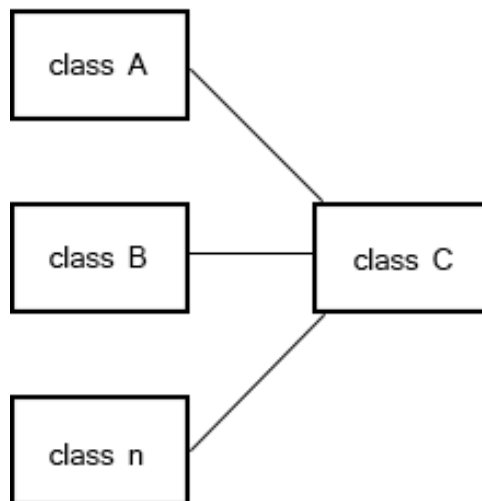
Multiple Inheritance ये Inheritance का तीसरा प्रकार है ।

Multiple Inheritance में एक से अधिक base classes हो सकते हैं और एक derived class होता है ।

Derived class को दो या दो से ज्यादा base classes को inherit किया जाता है, उसे Multiple Inheritance कहते हैं ।

Syntax

```
class Base_class_Name1
{
    // body_of_Base_class
};
class Base_class_Name2
{
    //body_of_Base_class2
};
class Derived_class_Name : Access_Specifier Base_class_Name1, Access_Specifier
Base_cass_Name2,.....
{
    //body_of_Derived_class2
};
```



Source Code :

```
#include <iostream.h>
using namespace std;
class A{
public:
    display1(){
        cout<<"Class A"<<endl;
    }
};
class B{
public:
display2(){
    cout <<"Class B"<<endl;
}
};
class C: public B, public A{
public:
    display3(){
        cout <<"Class C"<<endl;
    }
};
int main(){
    C obj;
    obj.display1();
    obj.display2();
    obj.display3();
    return 0;
}
```

Output :

```
Class A
Class B
Class C
```

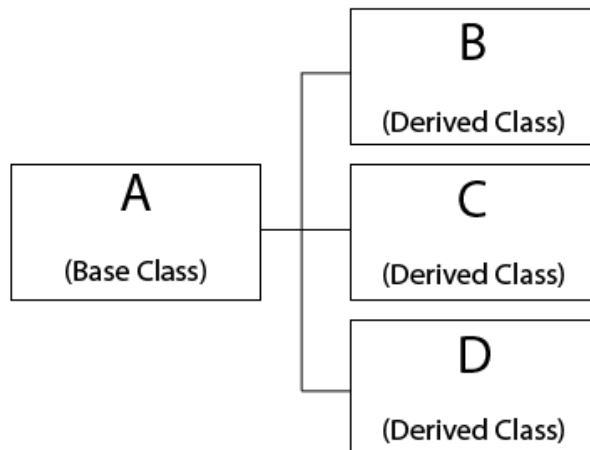
C++ Hierarchical Inheritance

Hierarchical Inheritance ये Inheritance का चौथा प्रकार है।

Hierarchical Inheritance में एक base class और एक से अधिक derived classes होते हैं।

Syntax

```
class Base_classname
{
    //body_of_Base_class
};
class derived_class_Name1 : Access_Specifier Base_class_Name
{
    //body_of_derived_class1
};
class derived_class_Name2 : Access_Specifier Base_class_Name
{
    //body_of_derived_class2
};
```



Source Code :

```
#include <iostream.h>
using namespace std;
class A{
public:
    display1(){
        cout<<"Class A"<<endl;
    }
};
class B : public A{
public:
    display2(){
        cout <<"Class B"<<endl;
    }
};
class C : public A{
public:
    display3(){
        cout <<"Class C"<<endl;
    }
};
int main(){
    B b;
    C c;
    c.display1();
    b.display2();
    c.display3();
    return 0;
}
```

Output :

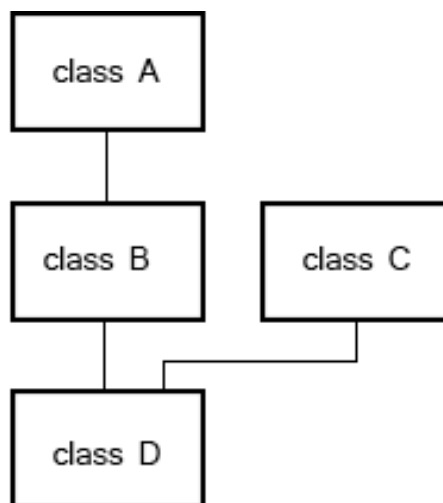
```
Class A
Class B
Class C
```

C++ Hybrid Inheritance

Hybrid Inheritance ये Inheritance का पांचवा और आखिरी प्रकार है ।
इसे 'Virtual Inheritance' भी कहा जाता है ।Hybrid Inheritance; एक से ज्यादा inheritance का combination है ।

Syntax

```
class base_class_Name
{
    //body_of_Base_class
};
class derived_class_Name1 : Access_Specifier base_class_Name
{
    //body_of_derived_class1
};
class derived_class_Name2
{
    //body_of_derived_class2
};
class derived_class_Name3 : class derived_class_Name1, class derived_class_Name2
{
    //body_of_derived_class3
};
```



Source Code :

```
#include <iostream.h>
using namespace std;
class A{
public:
    display1(){
        cout<<"Class A"<<endl;
    }
};
class B : public A{
public:
    display2(){
        cout <<"Class B"<<endl;
    }
};
class C{
public:
    display3(){
        cout <<"Class C"<<endl;
    }
};
class D : public C, public B{
public:
    display4(){
        cout <<"Class D"<<endl;
    }
};
int main(){
    D obj;
    obj.display1();
    obj.display2();
    obj.display3();
    obj.display4();
    return 0;
}
```

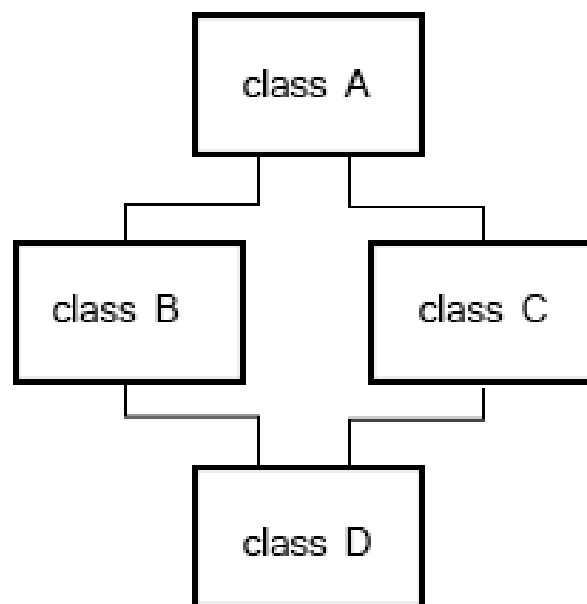
Output :

```
Class A
Class B
Class C
Class D
```

Hybrid Inheritance using Virtual Function

Source Code :

```
class base_class_Name
{
//body_of_Base_class
};
class derived_class_Name1 : virtual Access_Specifier base_class_Name //or
Access_Specifier virtual base_class_Name
{
//body_of_derived_class1
};
class derived_class_Name2 : virtual Access_Specifier base_class_Name //or
Access_Specifier virtual base_class_Name
{
//body_of_derived_class2
};
class derived_class_Name3 : class derived_class_Name1, class derived_class_Name2
{
//body_of_derived_class3
};
```



अगर निचे दिए हुए program को देखे तो virtual function दिखेगा ।

दिया हुआ program अगर बिना virtual function के compile करे तो 'member ambiguous' का error आ जाता है ।

इसका कारण ये है कि, class A; class B और class C इन दोनों पर derived किया गया है और class D पर class B और class C की वजह से class A की दो copies class D पर आयी है । और आखिर में class D का object ही सब inheritance का data access कर रहा है ।

Virtual Function लेने का कारण ये है कि, class A की सिर्फ एक copy ही class D को मिल जाए ।

Source Code :

```
#include <iostream.h>
using namespace std;
class A{
public:
    display1(){
        cout<<"Class A"<<endl;
    }
};
class B : virtual public A{    //or public virtual A
public:
    display2(){
        cout <<"Class B"<<endl;
    }
};
class C : virtual public A{    //or public virtual A
public:
    display3(){
        cout <<"Class C"<<endl;
    }
};
class D : public C, public B{
public:
    display4(){
        cout <<"Class D"<<endl;
    }
};
int main(){
    D obj;
    obj.display1();
    obj.display2();
    obj.display3();
    obj.display4();
    return 0;
}
```

Output :

```
Class A
Class B
Class C
Class D
```

Polymorphism

C++ Introduction of Polymorphism

Polymorphism ये एक 'poly' और 'morph' इन दोनों ग्रीक शब्दों से बना है | इसका अर्थ होता है 'अनेक रूप होना' |

एक ही रूप में अनेक रूप होना polymorphism होता है |

Polymorphism में inheritance का भी उपयोग किया जाता है |

Polymorphism के inheritance में base class और derived class में एक ही नाम का function होता है, लेकिन उनकी definition अलग-अलग होती है |

Polymorphism दो प्रकार के होते हैं |

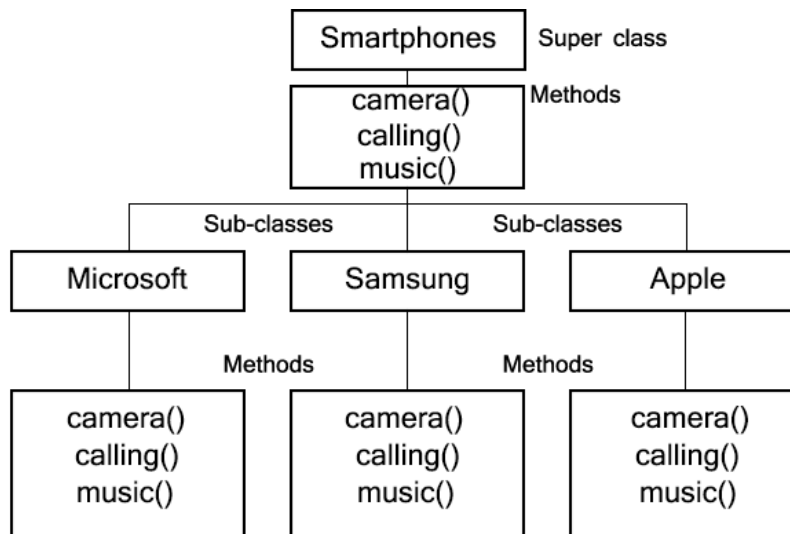
- Compile-time Polymorphism
- Run-time Polymorphism

Compile-time Polymorphism के दो प्रकार हैं |

- Function Overloading
- Operator Overloading

Run-time Polymorphism का एक ही प्रकार है |

- Virtual Function



Function Overloading : Compile-time/Static Binding

C++ में एक से ज्यादा एक जैसे नाम के function को इस्तेमाल किया जाता है | पर C में इसका इस्तेमाल नहीं हो सकता |

एक program में एक से ज्यादा एक ही नाम के function इस्तेमाल करना 'Function Overloading' कहते हैं | एक function दूसरी बार पहले जैसा इस्तेमाल नहीं किया जा सकता |

For Example,

जब ऐसा होता है तो, Compiler Function cannot overloaded का error देता है |

```
void func(int a, int b){ }  
void func(int a, int b){ }
```

Function Overloading का इस्तेमाल करना हो तो, हर same name के function का parameter का data_type, parameters की संख्या अलग-अलग इस्तेमाल किया जाता है |

For Example,

```
void func(){ }  
void func(int a){ }  
void func(float a){ }  
void func(int a, int b){ }  
void func(int a, int b, int c){ }
```

Source Code :

```
#include <iostream.h>
using namespace std;
void func();
void func(int);
void func(float);
void func(int, float);
void func(int, float, int);
int main() {
int x = 10;
float y = 3.14;
int z = 20;
    func();
    func(x);
    func(y);
    func(x, y);
    func(x, y, z);
return 0;
}
void func(){
    cout<<"Hello World!"<<endl;
}
void func(int a){
    cout<<"Value of a : "<<a<<endl;
}
void func(float b){
    cout<<"Value of b : "<<b<<endl;
}
void func(int a, float b){
    cout<<"Value of a : "<<a<<endl;
    cout<<"Value of b : "<<b<<endl;
}
void func(int a, float b, int c){
    cout<<"Value of a : "<<a<<endl;
    cout<<"Value of b : "<<b<<endl;
    cout<<"Value of c : "<<c<<endl;
}
```

Output :

```
Hello World!
Value of a : 10
Value of b : 3.14
Value of a : 10
Value of b : 3.14
Value of a : 10
Value of b : 3.14
Value of c : 20
```

C++ Operator Overloading

Hybrid Operator Overloading : Compile-time/Static Binding/Early Binding using Virtual Function

C++ Operator में Arithmetic, Logical, Conditional इत्यादी Operators के प्रकार होते हैं।
C++ के हर Operator का अलग-अलग काम होता है।

For Example,

+ : दो numbers का addition करना।

***** : दो numbers का multiplication करना।

Operator Overloading में इन Operators को किसी दूसरे तरह से भी define किया जा सकता है। इन Operators का इस्तेमाल 'operator' keyword के साथ किसी function_name की तरह किया जाता है। अगर user + Operator को overload करता है तो, वो अपनी पहली definition को भी same program में इस्तेमाल कर सकता है।

कौनसे Operators 'Operator Overloading' के लिए सही हैं ?

C++ के सभी Operators का overloading में इस्तेमाल किया जा सकता है। पर कुछ Operators ऐसे हैं की जीना इस्तेमाल Operator Overloading में नहीं हो सकता।

निचे दिए हुए Operators का इस्तेमाल Overloading में नहीं किया जा सकता।

Operator	Name of Operator
.	Accessing Members
.*	Pointer to accessing members
?:	Conditional
::	Scope Resolution

निचे दिए हुए Operators का इस्तेमाल Overloading में किया जाता है।

Operator	Name of Operator
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
=	Assignment

&	Bitwise AND and Address
+=	Addition assignment
-=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
&=	Bitwise AND assignment
%=	Modulus assignment
>=	Greater than or equal to
==	Equal
++	Increment
--	Decrement
<	Less than
>	Greater than
<<	Left shift
>>	Right shift
<<=	Left shift assignment
>>=	Right shift assignment
	Bitwise inclusive OR
=	Bitwise inclusive OR assignment
^	Exclusive OR
^=	Exclusive assignment
~	One's Complement
,	Comma
!	Logical NOT
!=	NOT or equal
&&	Logical AND
	Logical OR
*	Pointer
->	Member selection
->*	Pointer-to-member

*	Array Subscript
new	New
delete	Delete

Syntax for Operator Overloading

```
return_type operator operator_to_overload(parameters_list){
    //statement(s);
}
```

Operator Overloading का इस्तेमाल क्यों किया जाता है ?

C++ में हमें चाहे वैसा program 'Operator Overloading' के बिना भी कर सकते हैं। Operator Overloading का इस्तेमाल करना और program समझ आने में आसानी होती है। पहली बार जब user; Operator Overloading पढ़ता है, तो उसे कुछ समझ में नहीं आता और जब समझ आये तो उसे हर Operator की Overloading समझ आती है।

Normal Program में जैसे दो variables को add किया जाता है। C++ में दो object को add करना आसान नहीं होता, लेकिन Operator Overloading में ये मुमकिन है।

For Example.

```
int main(){
    Number n1, n2, n3;
    n3 = n1 + n2;    //+ is overloaded operator
}
```

Example for '+' Operator Overloading

```
#include <iostream.h>
using namespace std;
class Complex{
private:
    int a, b;
public:
    void put(int x=0, int y=0){
        a = x;
        b = y;
    }
    Complex operator+(Complex c){
        Complex com;
        com.a = a + c.a;
        com.b = b + c.b;
        return com;
    }
    void get(){
        cout<<"Real Number : "<<a<<endl;
        cout<<"Imaginary Number : "<<b<<endl;
        cout<<"Complex Number : "<<a<< " + "<<b<<"i";
    }
};

int main()
{
    Complex c1, c2, c3;
    c1.put(5, 10);
    c2.put(7, 9);
    c3 = c1 + c2; //Use also Complex c3 = c1.operator+(c2);
    c3.get();
    return 0;
}
```

Output :

```
Real Number : 12
Imaginary Number : 19
Complex Number : 12 + 19i
```

उपरवाला Program Binary Operator का है ।

Operator Overloading के लिए Operators के लिए दो प्रकार हैं ।

1. **Binary Operators** : Binary Operators के लिए दो operands दिए जाते हैं | for eg. +, -
2. **Unary Operators** : Unary Operators के लिए एक operand दिया जाता है | for eg. ++, --

'-' Unary Operator Overloading

Example for '+' Operator Overloading

```
#include <iostream.h>
using namespace std;
class Number{
private:
int a;
int b;
public:
Number(){ // Constructor is required
    a=5;
    b=-7;
}
void operator -(){
    a = -a;
    b = -b;
}
void show(){
    cout<<"Value of a : "<<a<<endl;
    cout<<"Value of b : "<<b<<endl;
}
};
int main(){
    Number x;
    cout<<"Before Overloading"<<endl;
    x.show();
    -x; // Unary Operator overloading
    cout<<"After Overloading"<<endl;
    x.show();
    return 0;
}
```

Output :

```
Before Overloading
Value of a : 5
Value of b : -7
After Overloading
Value of a : -5
Value of b : 7
```

C++ Virtual Function

Virtual Function : Run-time/Dynamic Binding/Late Binding

Introduction for Virtual Function

निचे दिये हुए program में एक base class और उससे नया derived class लिया है और Base Class का pointer और derived class का object बनाया है ।

अगर Base class को Base class pointer से point किया जाए तो base class के member functions को access जा सकता है ।

यहाँ पर base class के pointer से derived class के members को point करके access नहीं किया जा सकता ।

लेकिन Virtual Function से Base class के pointer से derived class के members को point करके उनके members को access किया जा सकता है ।

Without Virtual Function

Source Code :

```
#include <iostream>
using namespace std;
class Base{
public:
void display(){
    cout << "Base Class";
}
};
class Derived : public Base{
public:
void display(){
    cout << "Derived Class";
}
};
int main(){
    Base *ptr;    //Base class pointer
    Derived d;
    ptr = &d;
    ptr->display(); //Early Binding
    return 0;
}
```

Output :

Base Class

Virtual Function से यहाँ पर Late Binding होता है ।

With Virtual Function

Source Code :

```
#include <iostream>
using namespace std;
class Base{
public:
virtual void display(){ // or void virtual display(){
    cout << "Base Class";
}
};
class Derived : public Base{
public:
void display(){
    cout << "Derived Class";
}
};
int main(){
Base *ptr; //Base class pointer
Derived d;
ptr = &d;
ptr->display(); //Late Binding
return 0;
}
```

Output :

Derived Class

Abstract Class and Pure Virtual Function

जिस class में एक या एक से अधिक pure virtual function होता है, तो उसे Abstract Class कहते हैं ।

Example for Abstract Class

```
class A{ // Abstract Class
virtual void show() = 0; // pure virtual function
};
```

जिस virtual function की कोई definition नहीं होती, तो उसे 'Pure virtual Function' कहते हैं ।

```
virtual void show() = 0;
```

Base Class में Virtual function की कोई definition नहीं होती, अगर derived class; base class को inherit कर रहा हो तो वहा पर virtual function की definition दी जाती है। जब Pure virtual function के Abstract class को कोई derived class; inherit कर रहा है तो, उसे भी Abstract class कहते हैं, चाहे उसमें Pure virtual function हो या ना हो।

Abstract Class की विशेषताएं

Abstract Class का कोई object नहीं बनाया जाता। लेकिन उसका pointer बनाया जा सकता है।

अगर abstract class को कोई derived class; inherit कर रहा हो और inherit हुए pure virtual function की definition नहीं लिखता तो, वो भी Abstract Class बन जाती है और उसका Object भी नहीं बन पाता।

Example for Abstract and Pure Virtual Function

Source Code :

```
#include <iostream>
using namespace std;
class A{
public:
    void virtual show() = 0;
};
class B : public A{
public :
    void show(){
        cout<<"Hello World!";
    }
};
int main(){
    A *ptr;
    B b;
    ptr = &b;
    ptr->show();
    return 0;
}
```

Output :

```
Hello World!
```

Example for Pure Virtual Function and Run-time Polymorphism

Source Code :

```
#include <iostream.h>
using namespace std;
class A{
public:
    void virtual show() = 0;
};
class B : public A{
public :
    void show(){
        cout<<"Hello World!"<<endl;
    }
};
class C : public A{
public :
    void show(){
        cout<<"Hello Friend!"<<endl;
    }
};
int main(){
    A *ptr;
    B b;
    C c;
    ptr = &b;
    ptr->show();
    ptr = &c;
    ptr->show();
    return 0;
}
```

Output :

```
Hello World!
Hello Friend!
```

Method Overriding

Introduction for Method Overriding

Method Overriding ये Polymorphism का ही एक प्रकार है लेकिन Function Overloading से थोड़ासा अलग-अलग होने की वजह से उसे अलग से बनाया है ।

जैसे Function Overloading में एक की नाम के member functions को अलग-अलग तरीके से बनाया जाता था, वैसे ही Function or Method Overriding में अलग -अलग classes में एक ही नाम के member functions को बनाया जाता है ।

For Example,

```
class A{  
void show(){  
-----  
}  
};  
class B : public A{  
void show(){  
-----  
}  
};
```

Method Overriding में उन एक जैसे member functions की definition अलग-अलग होती है ।

Method Overriding में Inheritance के लिए एक Base class और एक Derived class की जरूरत होती है ।

Method Overriding के लिए एक जैसे member functions को एक ही class में नहीं लिखा जाता ।

Method Overriding Early Binding का काम करता है ।

Example for Method Overriding

Source Code :

```
#include <iostream.h>
using namespace std;
class A{
public:
void show(){
    cout << "class A"<<endl;
}
};
class B : public A{
public:
void show(){
    cout << "class B"<<endl;
}
};
int main(){
A a;
B b;
a.show();    //Early Binding
b.show();
return 0;
}
```

Output :

```
class A
class B
```

Different type of Method Overriding

निचे दिया हुआ program कुछ अलग तरीके का है ।

Program में हर एक class में दो-दो नाम के same member functions लिए है ।

लेकिन display() ये member function दोनों class पर है । लेकिन एक में parameter नहीं है और एक में parameter है ।

यहाँ पर Inheritance के नजरिये से class A का derived class B लिया है । इसकी वजह से Class A के जो members है, वो B class के object से भी access किये जा सकते है ।

Program में B class में void display(int x) इस प्रकार से लिया है और A class में void display() ऐसा लिया है ।

और आखिर में B class का object लेने की वजह से वो ज्यादा ध्यान अपने members की तरह देगा ।

class A और class B में एक ही नाम का member function हो तो, जिस class के object के साथ दिया है उस data को access किया जाता है ।

Program के आखिर में error आया है वो ये है कि, B class में display नाम का member function है, लेकिन वो parameter के साथ है । लेकिन उसे int main() में उसे parameter के साथ नहीं दिया ।

Source Code :

```
#include <iostream.h>
using namespace std;
class A{
public:
void show(){
    cout << "class A"<<endl;
}
void display(){
    cout << "Base Class"<<endl;
}
};
class B : public A{
public:
void show(){
    cout << "class B"<<endl;
}
void display(int x){
    int a = x;
    cout << "Value of a : "<<a<<endl;
}
};
int main(){
A a;
B b;
a.show();    //Early Binding
b.show();
//b.display(); //No matching function for B class
b.display(5);
return 0;
}
```

Output :

```
class A
class B
Value of a : 5
```


Constructor & Destructor

C++ Constructor

Constructor ये एक special type का member function होता है, जो अपने class के नाम के जैसा होता है। जिस class का constructor बना हो, अगर उसी class का जब object जब बनता है तब वो automatically call होता है।

Constructor का कोई return type नहीं होता। void भी return नहीं करता।

अगर कोई value को initialize करना हो तो अलग से उसका member function बनाकर उसे object के साथ access करना पड़ता है। Constructor ये काम सिर्फ object बनाते ही कर देता है।

Constructor के साथ virtual keyword का इस्तेमाल नहीं किया जाता।

Example for Constructor inside of class

```
class A{
private:
-----
public :
A(){      // Constructor
//constructor body;
}
};
```

Constructor को class के बाहर भी define किया जा सकता है।

Example for Constructor outside of class

```
class A{
private:
-----
public :
A(); //Constructor declaration
};
A::A() //Constructor Definition
{
-----//Constructor body;
}
```

Example for Constructor

Source Code :

```
#include <iostream.h>
using namespace std;
class Example
{
public :
    Example(){ //Constructor
        cout<<"Hello World!";
    }
};
int main(){
    Example e;
    return 0;
}
```

Output :

```
Hello World!
```

Constructor के चार प्रकार होते है |

- 1.Default Constructor
- 2.Parameterized Constructor
- 3.Default Copy Constructor
- 4.Constructor Overloading

1. Default Constructor

Default Constructor कोई parameter या argument नहीं लेता | ऊपर दिया हुआ program Default Constructor का है |

Program में 'A' नाम का class है और उसका constructor बनाया गया है | जब A class का object बनेगा तब Constructor automatically call होगा |जितनी बार A class का object बनेगा उतनी बार constructor call होता है |

Source Code :

```
#include <iostream.h>
using namespace std;
class A
{
public :
    A(){ //Constructor
        cout<<"Default Constructor."<<endl;
    }
};
int main(){
    A a1, a2, a3;
    return 0;
}
```

Output :

```
Default Constructor.
Default Constructor.
Default Constructor.
```

2. Parameterized Constructor

Parameterized Constructor में Constructor को parameters pass किये जाते हैं।

Parameterized Constructor में constructor को अलग-अलग arguments दिए जाते हैं। इसमें arguments की कोई मर्यादा नहीं होती।

Parameterized Constructor में class के object में parameters की values देनी पड़ती है।

निचे दिए हुए program Addition के लिए दो values initialize करके उनका addition किया गया है।

Source Code :

```
#include <iostream.h>
using namespace std;
class A{
private:
    int a, b, c;
public :
    A(int x, int y){    //Constructor
        a = x;
        b = y;
        c = a + b;
    }
    void display(){
        cout<<"Addition of "<<a<<" and "<<b<<" is "<<c;
    }
};
int main(){
    A a(5, 6);
    a.display();
    return 0;
}
```

Output :

```
Addition of 5 and 6 is 11
```

3. Default Copy Constructor

Copy Constructor से Object को बनाया जाता है | Copy Constructor में पहले Constructor Object को किसी दूसरे Object पर Copy किया जाता है |

जो data पहले object में होता है वही data दुसरे Object पर copy होता है |

Copy Constructor में दो तरीके से Object को copy किया जाता है |

```
A a2(a1); //or  
A a2 = a1;
```

Source Code :

```
#include <iostream.h>  
using namespace std;  
class A{  
private:  
    int a, b, c;  
public :  
    A(int x, int y){    //Constructor  
        a = x;  
        b = y;  
        c = a + b;  
    }  
    void display(){  
        cout<<"Addition of "<<a<<" and "<<b<<" is : "<<c<<endl;  
    }  
};  
int main(){  
    A a1(5, 6);  
    A a2(a1);    // Copy Constructor a1 to a2  
    A a3 = a2;    // Copy Constructor a2 to a3  
    a1.display();  
    a2.display();  
    a3.display();  
    return 0;  
}
```

Output :

```
Addition of 5 and 6 is : 11  
Addition of 5 and 6 is : 11  
Addition of 5 and 6 is : 11
```

4. Constructor Overloading

Constructor Overloading में class में multiple Constructor overloading की जा सकती है , सिर्फ उनकी parameters की संख्या और उनके type अलग-अलग होते हैं ।

Constructor Overloading; Function Overloading के तरह ही होता है ।

Source Code :

```
#include <iostream.h>
using namespace std;
class A
{
private:
    int a, b;
public:
    A(){
        a = 2;
        b = 3;
    }
    A(int x, int y){
        a = x;
        b = y;
    }
    void show(){
        cout<<"a : "<<a<<endl;
        cout<<"b : "<<b<<endl;
    }
};
int main(){
    A a1, a2(5, 7);
    cout<<"Default Constructor"<<endl;
    a1.show();
    cout<<"Parameterized Constructor"<<endl;
    a2.show();
    return 0;
}
```

Output :

```
Default Constructor
a : 2
b : 3
Parameterized Constructor
a : 5
b : 7
```

C++ Destructor

Destructor ये एक special type member function है, जो object को destroy कर देता है।
जब object out of scope जाता है, तब Destructor automatically call होता है।
Destructor; Constructor के तरह ही होता है, लेकिन Destructor में parameters नहीं होते।
Destructor prefix पर ~(tilde) sign के साथ इस्तेमाल होता है।

Syntax

```
~class_name(){  
    //statement(s);  
}
```

Source Code :

```
#include <iostream.h>  
using namespace std;  
class A{  
    int a;  
public:  
    A(int x){    //Constructor  
        a = x;  
        cout<<"Constructor is created."<<endl;  
    }  
    ~A(){        //Destructor  
        cout<<"Constructor is deleted."<<endl;  
    }  
    void show(){  
        cout<<"Value of a : "<<a<<endl;  
    }  
};  
int main(){  
    A a(5);  
    a.show();  
    return 0;  
}
```

Output

```
Constructor is cretaed.  
Value of a : 5  
Constructor is deleted.
```

Abstraction

Data Abstraction ये OOP में बहुत ही महत्वपूर्ण है | Data Abstraction ये data को hide करने का काम करता है | Data Abstraction में data type काम कैसे करता ये नजरंदाज होता है |

अपने दैनंदिन जीवन का विचार करे , जो हम क्रिया करते वो अपने जरूरत के हिसाब से करते है जैसे कि, खाना हो , पीना हो , सोना हो ये हर क्रिया होने का एक कारण होता है और वो प्रक्रिया शरीर के अन्दर से होती है | कोई खा रहा है और कोई पि रहा है ये देखा जा सकता है | लेकिन उस वक्त शरीर के अन्दर क्या चल रहा है ये नहीं जान सकता , इसे ही दैनंदिन जीवन 'Data Abstraction' कहा जाता है |

Data Abstraction में inside data है जो data members होते है, वो सुरक्षित रखे जाते है |

Source Code :

```
#include <iostream.h>
using namespace std;
class A{
private :
    int a; // private data hidden from outside world
public:
    A(int x){
        a = x;
        cout<<"Value of a : "<<a;
    }
};
int main(){
    A a(5);
    return 0;
}
```

Output

```
Value of a : 5
```


Data Encapsulation

Data Encapsulation का इस्तेमाल हर program में किया जा सकता है | OOP में ये काफी महत्वपूर्ण feature है | Data Encapsulation में Object के methods और data को एक ही जगह पर मतलब एक ही class पर store करके रखा जाता है |


Data Encapsulation में class के data members और member function को एक ही class में define किये जाते हैं | Data Encapsulation ये Data Abstraction का भी एक अच्छा उदाहरण है | ये object की सारी information एक ही जगह पर bind करके रखी जाती है |

Source Code :

```
#include <iostream.h>
using namespace std;
class A{
    int a, b;
public:
    A(){
        cout<<"Enter two Numbers"<<endl;
        cin>>a>>b;
    }
    void display(){
        cout<<"Addition of "<<a<<" and "<<b<<" is "<<a+b<<endl;
        cout<<"Subtraction of "<<a<<" and "<<b<<" is "<<a-b<<endl;
        cout<<"Multiplication of "<<a<<" and "<<b<<" is "<<a*b<<endl;
        cout<<"Division "<<a<<" and "<<b<<" is "<<a/b<<endl;
    }
};
int main(){
    A obj;
    obj.display();
    return 0;
}
```

Output

```
Enter two Numbers
12
6
Addition of 12 and 6 is 18
Subtraction of 12 and 6 is 6
Multiplication of 12 and 6 is 72
Division 12 and 6 is 2
```



विभाग ३

C++

Advanced

Namespaces

C++ Introduction for Namespaces

Namespace ये variable, function और classes का एक समूह होता है।

जैसे class को class keyword के साथ नाम दिया जाता है उसी तरह namespace keyword के साथ namespace का नाम दिया जाता है।

अगर एक ही program में एक ही variable को दो बार declare किया जाए तो re-declaration का error आ जाता है। लेकिन इन एक जैसे variable को एक ही program में अलग-अलग namespaces में declare किया जा सकता है।

Namespace पर आखिर में class की तरह semicolon(;) नहीं दिया जाता।

Namespace को function के अन्दर define नहीं किया जाता इसका मतलब namespace का scope global होता है।

Namespace की विशेषताएँ

Namespace के आखिर में class की तरह semicolon(;) नहीं दिया जाता।

Namespace का object नहीं बनाया जाता।

Syntax

```
namespace namespace_name{  
    // body of namespace;  
}
```

अगर namespaces के data को access करना हो तो ,

Syntax for Accessing Namespace members

```
namespace_name :: member(); or member(arguments_list);
```

Example for Namespace

```
#include <iostream.h>
using namespace std;
namespace ns1{           //Namespace 1
    void show(int x){
        int a = x;
        cout<<"Namespace 1"<<endl;
        cout<<"Value of a : "<<a<<endl;
    }
}
namespace ns2{           //Namespace 2
    void show(){
        cout<<"Namespace 2"<<endl;
    }
}
int main () {
    ns1::show(6); //Accessing Methods of Namespace
    ns2::show();
    return 0;
}
```

Output :

```
Namespace 1
Value of a : 6
Namespace 2
```

अगर एक ही namespace को access करना हो तो function को main() function में call करना काफी होता है ।

For Example

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

Namespaces को alias name भी दिया जाता है ।

For Example

```
namespace m = ns1; // alias name of ns1 is m
namespace n = ns2; // alias name of ns2 is n
int main () {
    m::show(6);
    n::show();
    return 0;
}
```

C++ Nested Namespace

Namespace nested type का भी होता है।

Nested Namespace के members को access करना हो तो scope resolution का इस्तेमाल किया जाता है।

अगर ns1 के member को access करना हो तो,

```
using namespace ns1; // ns1 namespace member accessing
int main () {
    show();
    return 0;
}
```

अगर ns2 के member को access करना हो तो,

```
using namespace ns1 :: ns2; // ns2 namespace member accessing
int main () {
    show();
    return 0;
}
```

Example for Nested Namespace

```
#include <iostream.h>
using namespace std;
namespace ns1{
    void show(){
        cout<<"Namespace 1"<<endl;
    }
namespace ns2{
    void show(){
        cout<<"Namespace 2"<<endl;
    }
}
using namespace ns1 :: ns2; //ns2 namespace member accessing
int main () {
    show();
    return 0;
}
```

Output :

```
Namespace 2
```

C++ Anonymous or Unnamed Namespace

Namespaces बिना नाम के भी define किये जा सकते हैं।

लेकिन multiple namespaces के variable, function और class अलग-अलग होते हैं।

Unnamed Namespace में ऐसा नहीं की उसे कोई नाम नहीं होता, Compiler की तरफ से उसे एक unique नाम दिया जाता है।

Syntax for Unnamed Namespace

```
namespace{  
    //some_code;  
}
```

Source Code :

```
#include <iostream.h>  
using namespace std;  
namespace{  
    void show(){  
        cout<<"Namespace 1"<<endl;  
    }  
}  
namespace{  
    void get(){  
        cout<<"Namespace 2"<<endl;  
    }  
}  
int main () {  
    show();  
    get();  
    return 0;  
}
```

Output :

```
Namespace 1  
Namespace 2
```

Dynamic Memory Allocation

C++ Introduction of Dynamic Memory Allocation

Dynamic Memory Allocation ये c++ के लिए बहुत ही अच्छा feature है ।

जब Variable declare होता है, तब variable; data type के हिसाब से Memory Allocate की जाती है ।

अगर User ने integer data type का variable बनाया तो 16-bit पर 2 bytes या 32-bit पर 4 bytes की memory allocate की जाती है ।

Memory Allocation के लिए दो प्रकार है ।

1. Compile-time/Static Memory Allocation

2. Run-time/Dynamic Memory Allocation

1. Compile-time/Static Memory Allocation

जब variable को बनाया जाता है, तब Compiler के जरिये compile-time पर memory allocate की जाती है ।

जब variable बनता है , तब वो किस data type से बनता है, तो compiler इसके हिसाब से खुद ही वो memory allocate करता है ।

Variable का declaration Compile-time पर होता है ।

For Example,

32-bit Integer के लिए 4 bytes

Character के लिए 1 byte

float के लिए 4 bytes

जब array को Compile-time पर memory allocate करे तो ये memory का space ये constant होता है ।

For Example,

```
int arr[10];
```

जब ऐसा array बनता है, तो इसमें 10 ही integer की values declare की जा सकती है और हर variable 4 bytes की memory allocate करेगा ।

2. Run-time/Dynamic Memory Allocation

Dynamic Memory Allocation में Run-time पर Memory allocate की जाती है ।

Dynamic Memory Allocation में जब program run होता है तब Memory allocate की जाती है ।

यहाँ पर जरूरत के हिसाब से variables को बनाया जाता है ।

Dynamic Memory Allocation में variables का declaration नहीं किया जाता । सिर्फ compile-time पे variable को declaration किया जाता है ।

C Programming में जैसे Dynamic Allocation के लिए malloc, calloc, realloc और free का इस्तेमाल किया जाता है , वैसे ही C++ में दो ऐसे Operators है , जिनका इस्तेमाल Dynamic Memory Allocation के लिए किया जाता है |

- new operator
- delete operator

C++ New Operator

new operator का इस्तेमाल Dynamic memory को allocate करने के लिए किया जाता है |
ये memory allocation; Run-time पर होता है |

Syntax for new

यहाँ पर new operator के साथ कौनसे भी data type की और array के साथ dynamic memory allocate की जा सकती है | यहाँ पर memory allocation के class को भी लिया जा सकता है |

```
new data_type  
new data_type[array_size];
```

For Example

```
new int;  
new int[10];
```

जब dynamic memory allocation करना हो तो 'pointer' का इस्तेमाल किया जाता है | जब new operator एक dynamic allocation है तब उसका address; new operator से दिए हुए pointer में store किया जाता है |

Syntax for new operator with pointer

```
pointer = new data_type;  
pointer = new data_type[array_size]; //for Array
```

Dynamic Memory Allocation के Example में देखा जाए, तो वहाँ new के साथ int का address; store करने के लिए int *ptr लिया है और ptr में new से int का address return होकर ptr के अन्दर store होगा |
Pointer में जैसे integer data type variable का address hold करना है तो pointer भी उसी data type का होता है वैसे ही new int address store करना हो तो pointer भी integer का ही होना चाहिए |

For Example

```
int *ptr;  
ptr = new int;  
ptr = new int[50]; //for array
```

or


```
int *ptr = new int;  
int *ptr = new int[50]; //for array
```

किसी User से भी array का size define किया जाता है ।

For Example

```
int a;  
cin>>a;  
int *ptr;  
ptr = new int[a]; or int *ptr = new int[a];
```

class with Constructor, Destructor and array

Source Code :

```
#include <iostream.h>  
using namespace std;  
class A{  
public:  
    A(){  
        cout << "Constructor"<<endl;  
    }  
    ~A(){  
        cout << "Destructor"<<endl;  
    }  
};  
int main()  
{  
    A *a = new A[3];  
    delete[] a;  
    return 0;  
}
```

Output :

```
Constructor  
Constructor  
Constructor  
Destructor  
Destructor  
Destructor
```

C++ Delete Operator

delete operator ये allocated dynamic memory को de-allocate करता है ।

जब new operator; memory allocate करता है, तब delete operator से allocate की हुई memory को de-allocate या free कर देता है ।

जब program में new operator इस्तेमाल किया जाता है, तब delete operator का इस्तेमाल किया जाता है ।

Syntax for delete

```
delete pointer_name;  
delete []pointer_name; //for array
```

Example for delete

```
delete ptr;  
delete []ptr; // for array
```

Example for new and delete operator

Source Code :

```
#include <iostream.h>  
using namespace std;  
int main(){  
int *ptr;  
ptr = new int;  
cout<<"Enter value of ptr : ";  
cin>>*ptr;  
cout << "Value is ptr : " <<*ptr<< endl;  
delete ptr;  
return 0;  
}
```

Output :

```
Enter value of ptr : 2  
Value is ptr : 2
```

Example for new and delete operator with array

Source Code :

```
#include <iostream.h>
using namespace std;
int main(){
int a, i;
int *ptr;
    cout<<"Enter number of elements : ";
    cin>>a;
ptr = new int[a];
    cout<<"Enter "<<a<<" elements "<<endl;
for(i=1; i<=a; i++){
    cout<<"Element "<<i<<" : ";
    cin>>ptr[i];
}
cout<<"Entered Elements : "<<endl;
for(i=1; i<=a; i++){
    cout<<ptr[i]<<endl;
}
delete []ptr;
return 0;
}
```

Output :

```
Enter number of elements : 4
Enter 4 elements
Element 1 : 6
Element 2 : 4
Element 3 : 8
Element 4 : 9
Entered Elements :
6
4
8
9
```

class with Constructor, Destructor and array

Source Code :

```
#include <iostream.h>
using namespace std;
class A{
public:
    A(){
        cout << "Constructor"<<endl;
    }
    ~A(){
        cout << "Destructor"<<endl;
    }
};
int main()
{
    A *a = new A[3];
    delete[] a;
    return 0;
}
```

Output :

```
Constructor
Constructor
Constructor
Destructor
Destructor
Destructor
```

C++ Introduction of Templates

C++ का Templates ये feature बहुत ही उपयुक्त है। ये generic programming के लिए इस्तेमाल किया जाता है।

C++ Templates को दो प्रकार से इस्तेमाल किया जाता है।

- Function Templates
- Class Templates

1. Function Templates

C++ Templates में 'template' keyword के साथ 'typename' या 'class' keyword का इस्तेमाल किया जाता है।

Syntax for Function Templates

ये Function Template का syntax है। इसमें जो type है वो placeholder है, जो parameter की values दी जाती है, उसके हिसाब से Compile-time पर basic data type लेता है।

For Example,

अगर parameter में 4 और 5 ये values ली तो वो दोनों के data type integer लेगा।

```
template //can replace 'class' keyword by 'typename' keyword
return_type function_name(parameters_list){

    // Function body;
}
```

Program में template keyword के साथ 'class' ये keyword लिया है, चाहे तो 'typename' keyword का भी इस्तेमाल किया जा सकता है।

class 'X' को define किया गया है। इसका मतलब जो भी User parameter(s) की values लिखेगा वहां पर उसका data type; compile-time पर compiler दे देता है।

Source Code :

```
#include <iostream.h>
using namespace std;
template //can replace 'class' keyword by 'typename' keyword
X func(X a, X b){
    return a;
}
int main()
{
    cout<<func(9, 5)<<endl; // func(int, int);
    cout<<func('a', 'b')<<endl; //func(char, char);
    cout<<func(3.7, 5.6)<<endl; //func(double, double);
    return 0;
}
```

Output :

```
9
a
3.7
```

C++ Class Templates

Class Templates; Function Templates के जैसे ही होता है ।

Class Templates को generic Templates भी कहा जाता है ।

Class का इस्तेमाल जैसे सामान्य तरीके से c++ program में किया जाता है वैसे ही Class Template में किया जाता है ।

Syntax for Class Template

```
template //can replace 'class' keyword by 'typename' keyword
class class_name{

    // Class body;
}
```

Example for Class Template

```
#include <iostream.h>
using namespace std;
template
class A{
private:
    C a, b;
public:
    A(C x, C y){
        a = x;
        b = y;
    }
    void show(){
        cout<<"Addition of "<<a<<" and "<<b<<" is "<<add()<<endl;
    }
    C add(){
        C c = a + b;
        return c;
    }
};

int main(){
    Aaddint(4, 5);
    Aaddfloat(4.6, 8.9);
    Aadddouble(3.145, 5.268);
    addint.show();
    cout<<endl;
    addfloat.show();
    cout<<endl;
    adddouble.show();
    return 0;
}
```

Output :

```
Addition of 4 and 5 is 9
Addition of 4.6 and 8.9 is 13.5
Addition of 3.145 and 5.268 is 8.413
```

Friend Function

Introduction for Friend Function

Friend function का इस्तेमाल 'friend' keyword के साथ किया जाता है | Friend function ये c++ के लिए एक उपयोगी function है | Friend Function; जिस class में उसे declare किया गया है , उस class का member नहीं बल्कि 'friend' होता है | Friend function class के अन्दर declare किया जाता है | Friend function की definition; class के बाहर होती है | ये functions class के member function नहीं होते हैं | ये non-member functions होते हैं |

Friend Function की विशेषताएँ

Friend Function से class के private members को access किया जाता है |

जब class के बाहर member function define किया जाता था, तब scope resolution(::) का इस्तेमाल किया जाता था | friend function के साथ class के बाहर scope resolution(::) का इस्तेमाल नहीं किया जाता |

Member functions को access करने के लिए object की जरूरत पड़ती थी, लेकिन friend function ये non-member function होने के कारण उसे access करने के लिए कोई object की जरूरत नहीं पड़ती |

जब friend definition बनायी जाती है, तब उसके parameters; class के object(s) होते हैं |

Syntax for Friend Function Declaration

Friend Function का declaration class के अन्दर किया जाता है |

```
class class_name
{
    friend return_type function_name(parameter(s));
}
```

Syntax for Friend Function Definition

Friend Function का definition; class के बाहर की जाती है |

Class के private और protected इन access specifier से data को friend function को access किया जाता है |

```
return_type function_name(parameter(s))
{
    //body of function_name;
}
```


Example for Friend Function

Program में show नाम का friend function लिया है और A नाम का class लिया है | ये friend function directly A class के 'x' इस private member को access कर रहा है |
main function में देखा जाए तो, A class के object का नाम 'obj' लिया है | यहाँ पर constructor से assign हुई value 'x' 0 call हो जायेगी और बाद में show function; call होकर '5' ये value assign हो जायेगी |

Source Code :

```
#include <iostream.h>
using namespace std;
class A{
private:
    int x;
public:
    A(){
        x = 0;
    }
    friend int show( A ); // declaring friend function
};
int show(A s){ // friend function definition
    s.x = 5;
    return s.x;
}
int main(){
    A obj;
    cout<<"Value of a : "<< show(obj);
    return 0;
}
```

Output :

```
Value of a : 5
```

Exception Handling

Exceptions ये Error का एक प्रकार है जो Run time पर या execution के time पर आ जाता है | जब program के किसी specific code को handle नहीं किया जा सकता या असामान्य स्थिति होने पर exception handling का इस्तेमाल किया जाता है |

इन Exceptions में divided by zero, out of bound array, out of memory या अन्य कारणों की वजह भी हो सकती है |

Exceptions दो प्रकार के होते हैं |

- Compile-time Error
- Run-time Exceptions
- **Compile-time Errors** : compile time के वक्त error आ जाने पर Compile-time Errors कहते हैं | For Example, Logical Errors, Syntax Errors.
- **Run-time Errors(Exceptions)** : Run-time के वक्त error आ जाने पर Exceptions कहते हैं | For Example, divided by zero, out of memory, array out of bound.

जब Program successfully execute होता है तब भी Exception या run time error आ सकता है | आप इस्तेमाल कर रहे IDE(Application) crash होने पर भी ये आ सकता है |

Simple Example for Exception:

Example पर 'Divided by zero' exception occur हुआ है | इसे try-catch block द्वारा handle किया जायेगा |

```
#include <iostream>
using namespace std;
int main(){
int a = 5;
int b = 0;
int c;
c = a/b;
return 0;
}
```

Output :

```
[Warning] division by zero [-Wdiv-by-zero]
```

Try to Handle Exception/Run-time Error

Exceptions हो handle करने के लिए 'try catch throw' statement का इस्तेमाल किया जाता है ।

try : जिस source code से exception आ जाने की संभावना होती है वो source code यहाँ पर दिया जाता है । इस try block से exception को throw किया जाता है ।

throw : throw keyword का इस्तेमाल exception को throw करने के लिए किया जाता है । ये error के बारे में information को provide करता है । throw keyword के साथ दिए हुए parameter को handler पर pass किया जाता है ।

catch : throw statement द्वारा throw किये हुए exception को catch करने के लिए catch block का इस्तेमाल किया जाता है । catch block पर exception को handle किया जाता है ।

Syntax

```
try{
    some_statements;
    throw exception_parameter;
}
catch (data_type e){
    some_statements;
}
```

Syntax for multiple catch Block

```
try{
    some_statements;
    throw exception_parameter;
}
catch (data_type e1){
    some_statements;
}
catch (data_type e2){
    some_statements;
}
-----
catch (data_type eN){
    some_statements;
}
```

Handle Divided By Zero Exception(Single catch Block)

```
#include <iostream>
using namespace std;
int main(){
    int a = 5;
    int b = 0;
    int c;
    try{
        if(b == 0){
            throw b;
        }
        c = a/b;
        cout<<"Value of c : " <<c;
    }
    catch(int ex){
        cout<<"You cannot declare "<<ex<<" as denominator.";
    }
    return 0;
}
```

Output :

```
You cannot declare 0 as denominator.
```

Example for Multiple catch Block

```
#include <iostream>
using namespace std;
int main(){
    int a = 5;
    int b = 10;
    int c;
    try{
        if(b == 0){
            throw b;
        }else if(b > a){
            throw "Not allowed - denominator is greater than numerator.";
        }
        c = a/b;
        cout<<"Value of c : " <<c;
    }catch(int ex1){
        cout<<"You cannot declare "<<ex1<<" as denominator.";
    }catch(char const* ex2){
        cout<<ex2;
    }
    return 0;
}
```

Output :

Not allowed - denominator is greater than numerator.

Exceptions	Description
bad_alloc	Memory allocate करने में असफल होने पर exception throw किया जाता है ।
bad_cast	dynamic cast करने में असफल होने पर exception throw किया जाता है ।
bad_exception	unexpected handler द्वारा exception throw किया जाता है ।
bad_function_call	bad call पर exception throw किया जाता है ।
bad_typeid	typeid द्वारा exception throw किया जाता है ।
bad_weak_ptr	Bad weak pointer होने पर exception throw किया जाता है ।
ios_base::failure	ये stream exception का base class होता है ।
logic_error	source code को read करने में असफल होने पर exception throw किया जाता है ।
runtime_error	Runtime होने पर exception throw किया जाता है ।
domain_error	invalid domain होने पर exception throw किया जाता है ।
future_error	future error होने पर exception throw किया जाता है ।
invalid_argument	invalid argument होने पर exception throw किया जाता है ।
length_error	length error होने पर exception throw किया जाता है ।
out_of_range	out of range होने पर exception throw किया जाता है ।
overflow_error	Arithmetic overflow error होने पर exception throw किया जाता है ।
range_error	internal computation में range error होने पर exception throw किया जाता है ।
system_error	System error होने पर exception throw किया जाता है ।
underflow_error	mathematical underflow error होने पर exception throw किया जाता है ।

हमारी ई-बुक खरीदने के लिए धन्यवाद !