COPYRIGHTED BY

Instagram - @python_world_in

2023/2/5 10:45

Index

2023/2/5 10:45

2023/2/5 10:45

14 C++ Pointers

15 C++ object class concepts

16 C++ constructors

17 C++ destructors

18 C++ Inheritance

19 C++ Polymorphism

20 C++ Overloading

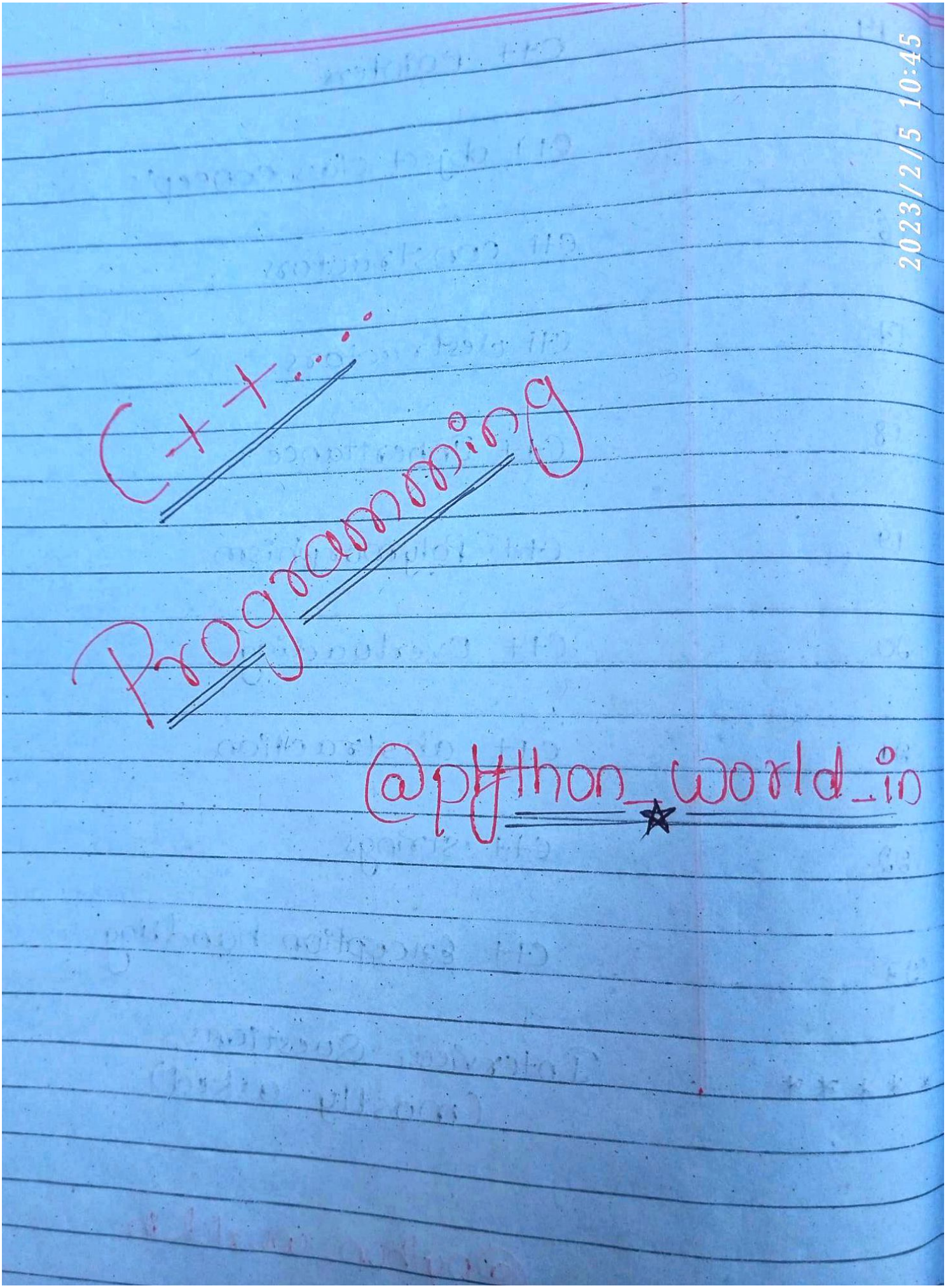
21 C++ abstraction

22 C++ strings

23 C++ Exception handling

***** Interview Questions (mostly asked)

@python_world_in


2023/2/5 10:45
C++...
Programming
@python_world_in

1 Overiew

2023/2/5 10:45

* What is c++ ?

* c++ is a statically typed, compiled, general-purpose free-form programing language that supposes, procedural, object-oriented & generic programing.
* c++ is regarded as middle level language, as it comprises combination of both high-level & low-level language features.
* c++ was developed by Bjarne strousstrup starting in 1979
* c++ is a superset of C & that vertually any C program is a legal c++ program.

Object-Oriented programming:

c++ fully support object-oriented programing including four piller of object-Oriented development.

- Encapsulation
- Data hiding
- Inheritance
- Polymorphism

@python_world_in

* Standard libraries:
* Standard c++ consists of three important parts
* The core language giving all building blocks including variables data types & literals
* The c++ standard library giving a each set of functions manupulating files, strings etc
* The standard template libraries (stc) giving a rich set of methods manupulating data structure

* The ANSI standard:

The ANSI standard is an attempt to ensure that c++ is a portable; that code you write for microsoft's compiler will compile without errors, using compiler on a mac, unix

* learning c++:

* The most important thing while learning c++ is to focus on concepts
* The purpose of learning a programming language is to became a better programmer that is to become more effective at designing & implementing new systems
* c++ is widely used for write devices & other software that rely on direct manupulation of harduare

2023/2/5 10:45

* Usage of C++:

By the help of c++ programming, we can develop different types of secured & robust applications

- window application
- client-server application
- Device drivers
- Embeded Siemware

c++ program:

```
#include<iostream>
using namespace std;

int main() {

    Cout << " Hello C++ programming";

return 0;

}
```

C Vs C++ -

C	C++
C follows procedural style programing	It follows both procedural & object oriented programming
Data is less secured in C	In C++ you can use modifier for class member to make it inaccessible
It follows top-down approach	It follows bottom-up approach
C doesnot support referance variable	C++ supports referance variables
Scanf() & printf() mainly used for o/p input / output	C in & Cout are used to perform input / output operations
C does not supports Inheritance	C++ supports the term inheritance
C dosn't provide feature of name space	It supports feature of namespace

Turbo C++ download & Installation:

There are many compilers available for C++. You need to download anyone, Here we are going to use Turbo C++. It will work for both C & C++

To install the turbo C++ software, you need to follow following steps.

① Download Turbo C++

② create turboc dictionary inside C drive & extract tcp.zip inside C:\turboc

③ Double click on install.exe file

④ click on the tc application file located inside C:\TC\BIN to write C program.

This is how we install C++ in our device ↑

C++ Basic Input / Output

* C++ I/o operation is using the stream concept stream is the sequance of bytes or flow of data. It makes performance fast

* If bytes flow from main memory to device like printers display screen, or network connections, this is called as o/p operation

* If bytes flow from device like printer, displa screen or a network connection to main memory, this is called input operation

I/O library Header files

Header file	function & Description
<iostream>	It is used to define cout, cin & cerr objects, which corresponds to std o/p stream, std i/p stream respectively
<iomanip>	It is used to declare services useful for performing formatted I/O such as setprecision
<Fstream>	It is used to declare services for user-controlled file processing

Standard Output stream (cout)

The cout is predefined object of stream class. It is connected with std o/p device, which is usually a display stream. The cout is used in conjunction with the stream insertion operator (<<) to display output

```
#include<iostream>
  using namespace std;
int main() {
   char ary[] = "welcome to C++ tutorial";
   cout << "value of ary is : " << ary << endl;
}
```

output - Welcome to C++ tutorial

2023/2/5 10:46

Standard input Stream (Cin)

* The Cin is predefined object of istream class. It is connected with the std i/p device, which is usually a Keyword.
* The Cin is used in conjunction with stream exhac -tion operator (>>) to read the i/p from console

```
#include <iostream>
using namespace std;
int main() {
int age;
cout << "enter your age;";
cin >> age;
cout << "your age is " << age << endl
}
```


Output: Enter your age - 22
Your age is - 22

Standard end-line (endl)

The endl is predefined objects of ostream class It is used to insert a new line characters & flushes the stream

```
#include <iostream>
using namespace std;
int main() {
Cout << "C++ Tutorial";
Cout << "Javatpoint" <<endl;
Cout << "End of Line" <<endl;
}
```

Output: C++ Tutorial Javatpoint
End of line

C++ variables:

* A variable is name of memory locat". It is used to store. Its value can be changed.
* It is a way to represent memory locat" through Symbol so that it can be easily identified
* Let's see the syntax to declare a variable

```
int x;
Float y;
char z;
```

* Here X, Y, Z are variables & int, float, char are data types
* A variable can have alphabets, digits & underscore
* No white spaces is allowed.

2023/2/5 10:46

C++ Data types :

* A data type specifies the type of data that a variable can store, such as integer, floating, character etc.

* Data types in C++ -

1) Basic - int, char, float, double

2) Derived - array, pointer

3) Enumeration - enum

4) User defined - structure

1) Basic Data Types :

- The basic data types are integer based & floating-point based

- C++ language supports both sightened & unsinged literals

- The memory size of basic data types may change according to 32 or 64 bit operating system

Data types	memory size	Range
char	1 byte	-128 to 127
Signed char	1 byte	-128 to 127
Unsigned char	1 byte	0 to 127
Short	2 byte	-32,768 to 32767
Signed short	2 byte	-32768 to 32767
Unsigned short	2 byte	0 to 32,767
int	2 byte	-32767 to 32767
Signed int	2 byte	-32767 to 32767
Unsigned int	2 byte	0 to 32767
short int	2 byte	-32767 to 32767
Signed short int	2 byte	-32767 to 32767
unsigned short int	2 byte	0 to 32767
Float	4 byte	
double	8 byte	

C++ keywords :

* A keyword is a reserved word - you cannot use it as a variable name, constant name
* A list of 32 keywords in c++ language which are also available in c language are given below.

auto	break	case	char	const
double	else	enum	extern	float
int	long	register	return	short
struct	switch	typedef	union	unsigned

C++ Operators :

* An operator is simply a symbol that is used to perform operations
* There can be many types of operations like arithmatic, logical, bitwise
  - arithmatic operator
  - Relational operator
  - logical operator
  - Bitwise operator
  - Assignment operator
  - unary operator
  - conditional operator
  - misc operator

	Types	Operator
Binary Operators	→ Arithmatic operator	+, −, *, /, %
	→ Relational operator	<, <=, >, >=, ==
	→ logical operator	&&, ||, !
	→ Bitwise operator	&, |, <<, >>, ~, ^
	→ Assignment operator	=, +=, -=, *=, /=
	→ Unary operator	++, --
	→ conditional operator	?:

C++ Identifires :

* C++ identifires in a program are used to refer to name of the variable, functions, arrays or other user-defined identifires
* They are basic requirement of any language
* Every language has its own rules for naming the identifires
* In short, we can say that C++ identifires represent essential elements in the program

- constant
- variables
- functions
- labels
- Defined Data types

2023/2/5 10:46

* Let's look at a simple example to understand concept of identifires

```
#include <iostream>
using namespace std;
int main()
{
int a;
int A;
Cout << " Enter the values of 'a' & 'A';
Cin >> a;
Cin >> A;
Cout << " In the values that you have entered':
                "<<a<<", "<<A;
return 0;
}
```

* Output:

Enter the values of 'a' & 'A'
5, 6
The value you have entered are : 5, 6

* Difference bet' identifires & Keywords

Identifires	Keywords
1) Identifires are the name defined by the programmer to the basic elements of a program	Keywords are reserved words whose meaning is known by the Compiler
2) It is used to identify name of variable	It is used to Specify type of entity
3) It can consists of letters, digits and underscore	It consists only letters
4) It can use both lowercase & uppercase letters	It use only lowercase letters
5) It can be classified as internal & external identifires	It cannot be further classified
6) Example are test, result, sum, power etc	Examples are 'for', 'if', 'else', 'break', etc

C++ Expressions

* C++ expression consists of operators, constants, & variables which are arranged according to the rules of language
* It can also contains function call which returns values
* An expression can consists of one or more oprands zero or more operators to compute a value
* An expression can be following types

- constant expression
- Integral expression
- Float expression
- Pointer expression
- Relational expression
- logical expression
- Bitwise expression
- Special assignment expression

C++ Control Statements

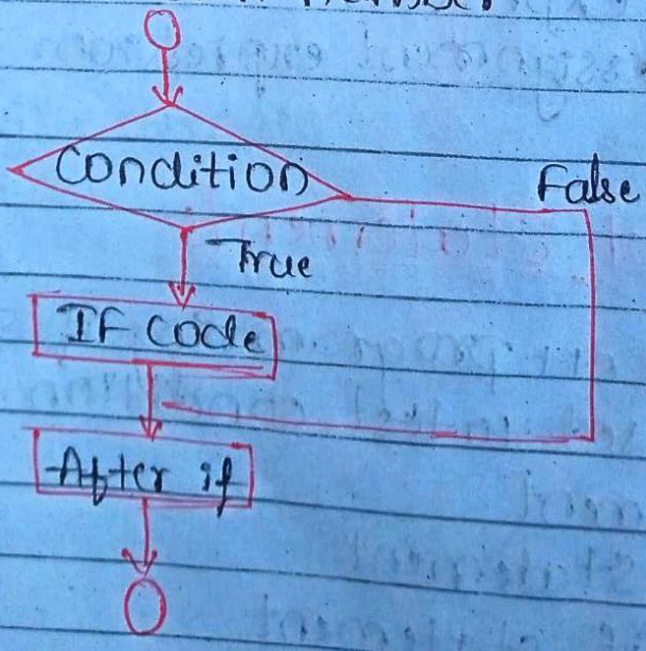
* C++ if-else - In c++ programming if statement is used to test condition
  - If statement
  - if-else Statement
  - nested-if statement
  - if else-if ladder

* C++ if statement

Syntax:
```
if (condition) {
  //code to be executed
}
```

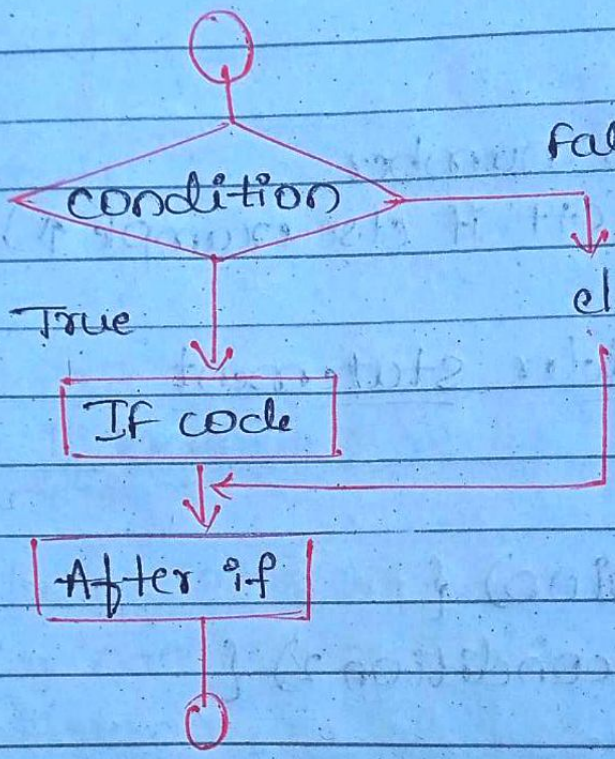
```
#include<iostream>
using namespace std;
int main() {
int num=10;
if (num% 2 ==0)
{
Cout << "It is even number";
}
return 0;
}
```

* Output - It is even number



• C++ if else statement:

Syntax:
```
if (condition) {
// code if condition is true
} else {
} // code if condition is false
```




```
#include <iostream>
using namespace std;
int main() {
int num = 11;
if (num % 2 == 0)
{
Cout << "It is even number";
}
```

```
else
{
cout << "It is odd number";
}
return 0;
}
```

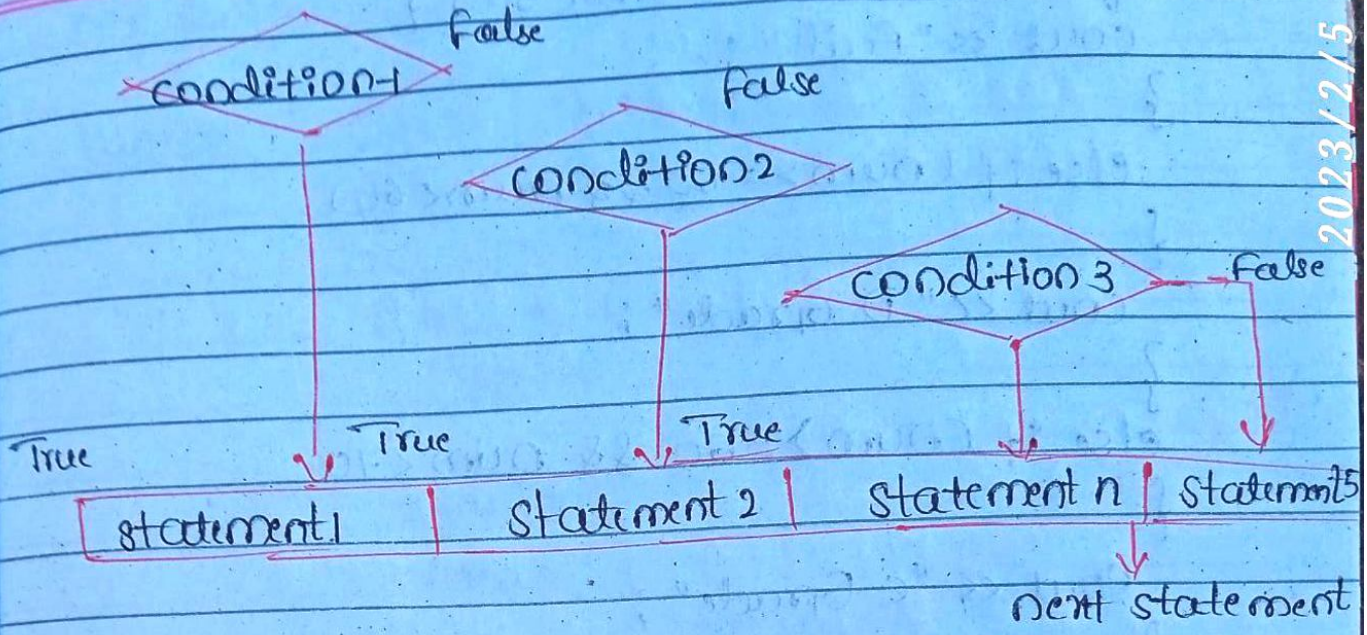
Output - It is odd number

(C++ if else example ↑)

* C++ if-else-if ladder statement

Syntax:

```
if (condition) {
} else if (condition 2) {
}
 else if (condition 3) {
}
....
else {
}
```



* If - else - if example :


```
#include <iostream>
using namespace std;
int main() {
 int num;
Cout << " Enter a number to check grade : ";
Cin >> num;
if (num<0 || num>100)
 {
  cout >> "wrong number";
 }
else if (num >=0 && num < 50) {
 Cout << "Fail";
 }
```

```
else if (num >= 0 && num < 50) {
    cout << "Fail";
  }
   else if (num >= 50 && num < 60)
  {
   cout << "D Grade";
  }
   else if (num >= 60 && num < 70
  {
   cout << "C Grade";
  }
  else if (num >= 70 && num < 80
  {
   cout << "B Grade";
  }
  else if (num >= 80 && num < 90
  {
  cout << "A Grade";
  }
}
```

Output : * Enter a number to check Grade = 66 ↵
C Grade
* Enter a number to check Grade = -2
wrong number

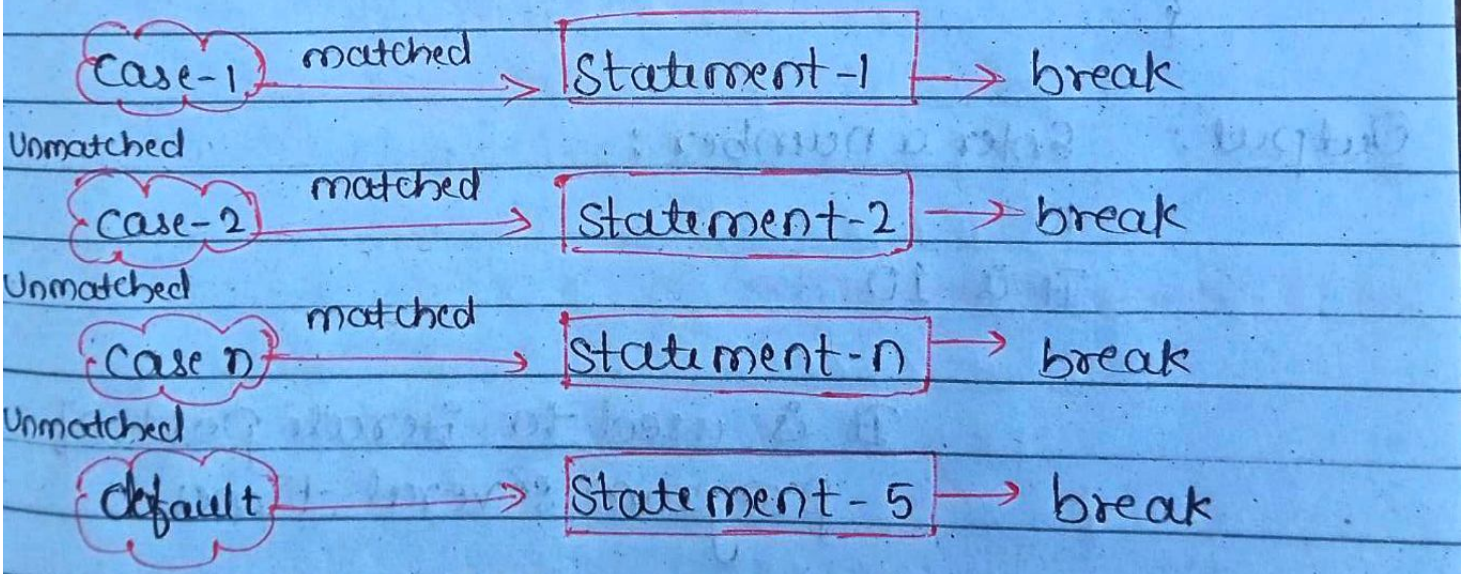
2023/2/5 10:47

* C++ Switch :

Syntax :

```
Switch (expression) {
case value 1;
    break;
case value 2;
    break;
    :
default :
  break;
}
```


expression



* C++ Switch example:

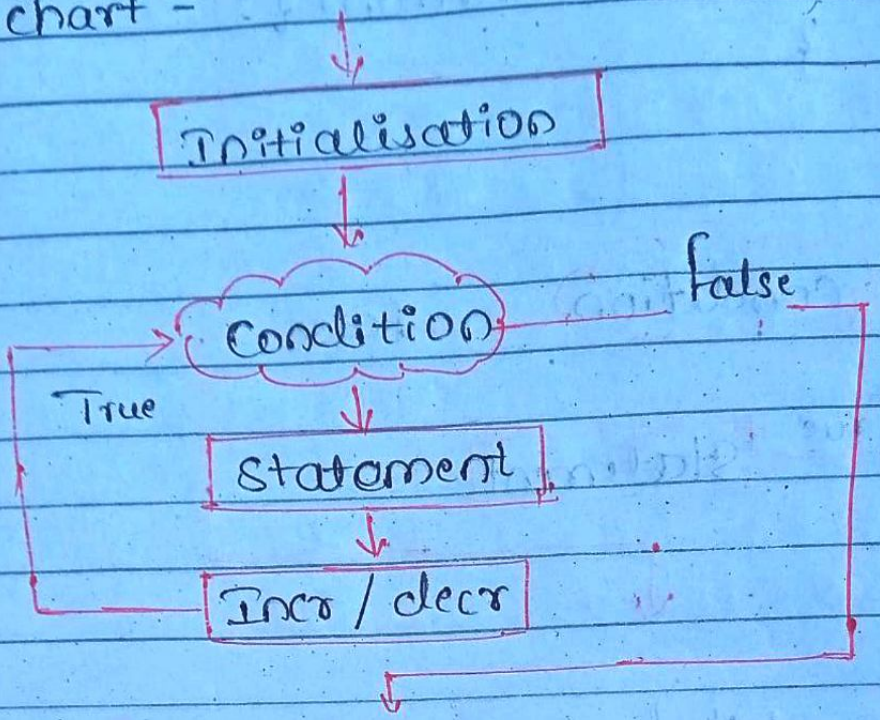
```
#include <iostream>
using namespace std;
int main() {
 int num;
cout << " Enter a number to check Grade:";
cin >> num;
 switch(num)
{
 case 10; Cout << " It is 10" : break;
 case 20; Cout << "It is 20" ; break;
 case 30 ; Cout << "It is 30" : break;
 default; Cout << " Not 10, 20, 30 ", break;
}
}
```

Output: Enter a number:
10
It is 10

It is used to iterate part of program several times

Syntax: for (initialisation ; condition;){
}

Flowchart -



Example -

```
#include <iostream>
using namespace std;
int main() {
 for (int i=1 ; i<=10 ; i++) {
 cout<<i<<"\n";
 }
}
```


Output - 1 2 3 4 5 6 7 8 9 10

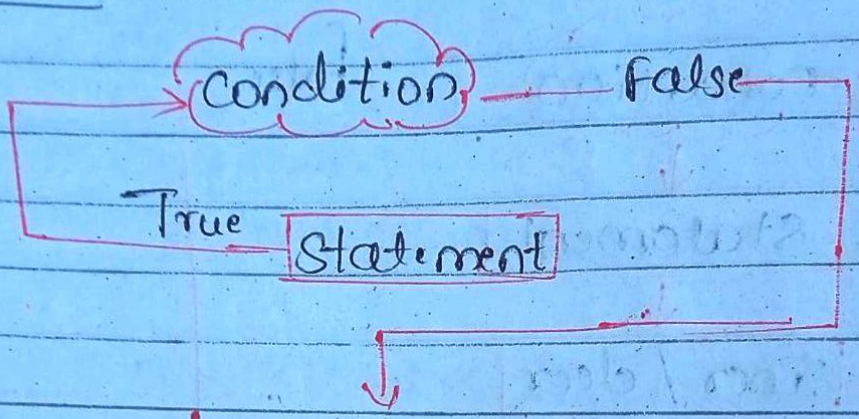
C++ While loop

* In c++ while loop is used to iterate a part of program several times
* If the number of iteration is not fixed it is recommended to use while loop

2023/2/5 10:48

Syntax: while (condition) {
}

Flowchart-



* Example for while loop-

```
#include <iostream>
using namespace std;
int main() {
int i=1;
  while (i<=10)
  {
   Cout<<i<<"\n";
   i++;
  }
}
```

Output -

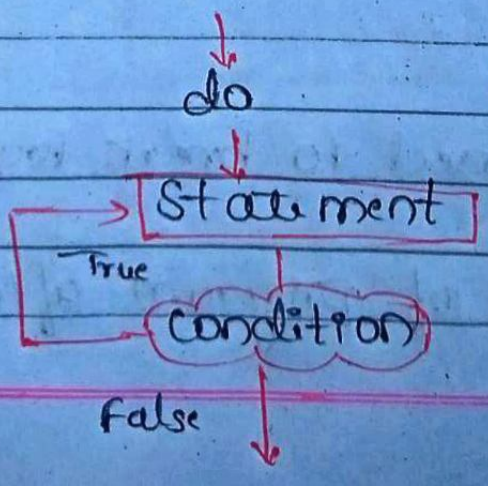
1
2
3
4
5
6
7
8
9
10

C++ Do-While loop

* The do-while loop is used to iterate a part of program several times
* If the number of iterations is not fixed you must have to execute the loop at least once
* The c++ do-while loop is executed at least once because condition is checked after loop body

Syntax -

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



* Do-while loop example -

```
#include <iostream>
using namespace std;
int main() {
int i=1;
 do {
  cout<<i<<"\n";
   i++;
 } while (i<=10);
}
```

Output -

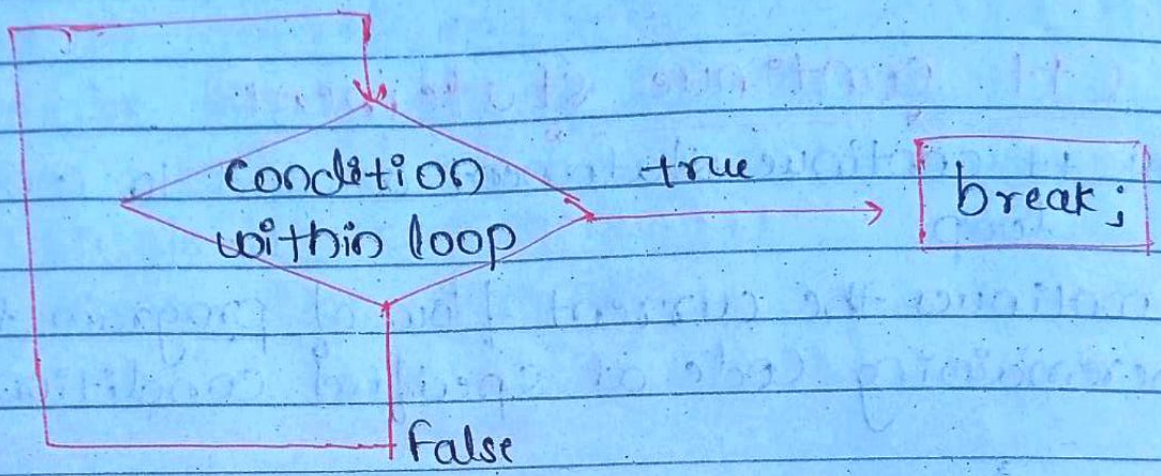
```
1
2
3
4
5
6
7
8
9
10
```


C++ break statement

* The break statement is used to break loop of switch statement
* If break the current flow of program after given condition

jump - statement ;
break ;

Flowchart -



* let us see example -

```
#include <iostream>
 using namespace std;
int main() {
 for (int i=1 ; i<=10 ; i++)
{
  if (i==5)
  {
   break ;
  }
cout << i << "\n";
 }
}
```

Output - 1
2
3
4

C++ continue statement:

* The c++ continue statement is used to continue loop.
* It continues the current flow of program & skip the remaining code at specified condition

```
jump statement;
Continue;
```

Example -

```
#include<iostream>
Using namespace std;
int main() {
for (int i=1; i<10; i++){
if (i==5) {
  continue;
}
cout<<i<<"\n";
}
}
```

Output - 1 2 3 4 5 6 7 8 9 10

C++ Comments

* The c++ comments are statements that are not executed by the compiler
* The comments in c++ programming can be used to provide explanation of the code
* There are two types of comments

i) single-line comment
ii) multi-line comment

i) Single-line Comment:

```
#include <iostream>
using namespace std;
int main() {
int x=11; // x is a variable
Cout <<x<<"\n";
```

Output - 11

2) Multi-line Comment:

```
#include <iostream>
using namespace std;
int main() {
/* declare and
 Print variable in c++ */
int x=35;
Cout <<x<<"\n";
}
```

Output - 35

Note - The C++ multi line comment is used to Comment multiple lines of code. It is surrounded by slash & asterisk.

C++ Arrays

* Arrays in C++ is a group of similar types of elements that have continuos memory location
* In C++ std::array is a container that encapsulates fixed size arrays
* Arrays index start from 0

Data	10	20	30	40
Index	0	1	2	3

* **Advantages of C++ Array:**
  - code optimization
  - Random access
  - Easy to traverse data
  - Easy to manupulate data
  - Easy to sort data

* **Disadvantages of array**
  - Fixed size

* C++ Array types -
  i) Single diamensional array
  ii) multi diamensional array

i) Single diamensional array -

```
#include <iostream>
using namespace std;
int main() {
int arr[5] = {10, 0, 20, 0, 30}
for (int i=0 ; i<5 ; i++)
{
cout << arr[i] << "\n";
}
}
```

Output - 10 0 20 0 30

C++ Array to function

* In c++ to reuse the array logic, we can create function in c++, we need to provide only array name

```
#include <iostream>
using namespace std;
void printarray (int arr[5]);
int main()
{
int array1[5] = {10, 20, 30, 40, 50};
```

```
int array 2[5] = { 5, 15, 25, 35, 45 };
 Print-Array (arr1);
 Print-Array (arr2);
}
Void printArray(int arr[5])
{
 Cout << " Printing array elements : " << endl;
      For (int i=0 ; i<5 ; i++
    {
     Cout << arr[i] << "\n";
    }
  }
```

Output -

```
Printing array elements :
10
20
30
40
50
Printing array elements :
5
15
25
35
45
```

ii) C++ multidiamentional array:

* The multidiamentional array is also known as rectangular arrays. It can be two dimen-sional or three diamentional. The data is stored in tabular form.

```
#include <iostream>
using namespace std;
int main()
{
  int test[3][3] = 5;
      test[0][0] = 10;
      test[0][1] = 15;
      test[1][1] = 20;
      test[1][2] = 30;
      test[2][0] = 30;
      test[2][2] = 10;
for(int i=0; j<3; ++j)
{
 For(int j=0; j<3; ++j)
 {
 cout << test[i][j] << " ";
 }
 cout << "\n";
}
 return 0;
}
```

Output -

```
5  10  0
0  15  20
30  0  10
```

C++ Functions

* The functions in c++ language is also known as Procedure or subroutine in other programming language
* To perform any task, we can create function. A function can be called many times

Advantages of functions
- i) Code Reusability
- ii) Code optimization

1) Code Reusability:
- By creating functions in c++ you can call many times, so we don't need to write the same code again

2) Code Optimisation:
- It makes the code optimized, we don't need to write much code

You need to write logic only once & you can use it several times

Types of functions

Library funct°
- The functions which are declare in the c++ header files such as - ceil(x), cos(x), exp(x).

User-defined funct°
- The funct's which are created by the c++ programer so that he can use many times. It reduces complexity of big problem.

Syntax for creating function

```
return_type function_name
{
  // code to be executed
}
```

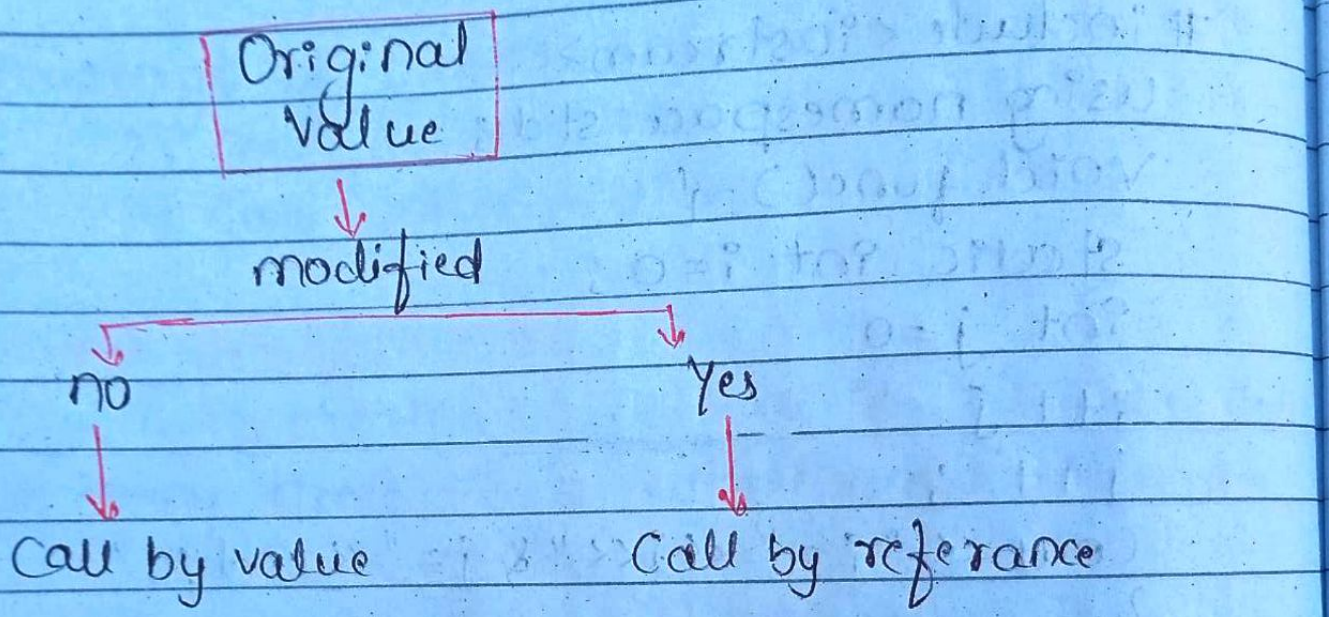
Examples

```
#include <iostream>
using namespace std;
void func() {
static int i=0;
int j=0
++i;
j++;
cout << "i=" << i << " & j=" << endl;"
}
int main()
{
  func();
  func();
  func();
}
```


Output - i=1 & j=1
i=2 & j=1
i=3 & j=1

Call by value & call by referance

There are two way to pass value or data to function in c language. Original value is not modified in call by value but it is modified in call by referance



i) Call by value:

In call by value in C++ -
Original value is not modified

```
#include <iostream>
using namespace std;
void change (int data);
int main()
{
int data = 3;
change (data);
cout<<" value of data is:" << data << endl;"
```

```
return 0;
}
void change (int data)
{
  data = 5;
}
```

Output - Value of data is 3

ii) call by Reference

Original value is modified cos we pass reference

```
#include<iostream>
using namespace std;
void swap (int*x, int* y)
{
int swap;
Swap = *x;
*x = *y;
*y = swap;
}
int main()
{
 int x=500, y=100
 Swap(&x, &y);
 Cout << "Value of x is :"<< x << endl;
 Cout <<"Value of y is :"<< y << endl;
 return 0;
}
```

Output - Value of X is : 100
Value of Y is : 500

Call by Value	Call by referance
A copy of value is pass to the function	An address of value is pass to function
changes made inside the funct^n isn't reflected on the other functions	changes made inside the function is reflected on the other functions
Actual & formal argum-ents will be created in diff memory locat^n	Actual & formal arguments will be created in same memory location
Original value isn't modified	Original value is modified cos we pass the referance

C++ Recusion

```
#include <iostream>
using namespace std;
int main()
{
int factorial (int);
int fact, value;
Cout << "Enter any number: "}
Cin >> value;
```

```
Fact = Factorial (value);
Cout << "Factorial of a number is: "<< Fact <<endl;
return 0;
}
int Factorial (int n)
{
if (n<0)
return(1);
else
   {
return (n*factorial (n-1));
  }
}
```


Output - Enter any number : 5
factorial of number is : 120

* We can understand above program of recursive method call by figure -

return 5 * factorial (4) = 120
return 4 * factorial (3) = 24
return 3 * factorial (2) = 6
return 2 * factorial (1) = 2
return 1 * factorial (0) = 1

∴ 1 * 2 * 3 * 4 * 5 = 120

fig - Recursion

* When a function is called within same function it is known as recursion
* The function which calls the same function is known as recursive function

C++ Storage Classes

* Storage classes is used to define the lifetime & visibility of a variable and/or function within C++ program.

There are five types of storage classes -

storage class	Keyword	lifetime	visibility	Initial value
Automatic	auto	function block	local	Cearbage
Register	register	function Block	local	Cearbage
mutable	mutable	class	local	Cearbage
External	external	whole Program	global	zero
Static	Static	whole program	local	zero

C++ Pointers

The pointer in c++ language is a variable. it is also known as locator or indicator that points to an address of variable.

address ⟶ value

Pointer ⟶ variable

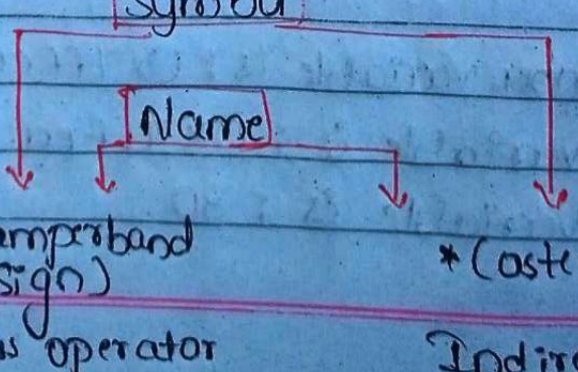
Advantage of pointer:

1) Pointer reduces the code & improves the performance. it is used to retrieving strings, trees etc
2) We can return multiple values from the function using pointer
3) It makes you able to access any memory location in the computer's memory

Usage of pointer -

1) Dynamic memory allocation
2) Array, Function & Structure

Symbol used in pointer



Description

It determines the address of variable

Access the value of an address

Declaring a pointer -

```
int *a; // pointer to int
char *c; // pointer to char
```

Ex:

```
#include <iostream>
using namespace std;
int main() {
int number = 30;
int *P;
P = &number;
Cout << "Address of number variable is :"
<< &number << endl;
Cout << "Address of p variable is : " << P << endl;
Cout << " value of P variable is : " << *P << endl;
return 0;
}
```

Output -

Address of number variable is : 0x7ffccc8724c4
Address of P variable is : 0x7ffccc8724c4
Value of P variable is : 30

C++ Array of pointers

```
#include <iostream>
using namespace std;
int main()
{
 int* ptr;
 int marks[10];
 Std:: Cout << "Enter elements of array"
            << std:: endl;
 for (int i=0; i<10; i++)
 {
  Cin >> marks[i];
 }
 Ptr = marks;
 Std:: Cout << Value of * ptr is:" << * Ptr << Std::endl;
 Std:: Cout << Value of * marks is:" << marks << std::endl;
 }
```

Output - Enter the elements of an array:

1
2
3
4
5
6
7
8
9
10

The value of *Ptr is : 1
The value of *marks is : 1

Array of pointer:

Array & pointers are closely relates to each other, the name of an array is considered as pointer; the name of array contains the address of an element.

* In the above code, we declare integer, pointer & an array of integer type, we assign address of marks to ptr by using the statement ptr = marks; it means both point to same element
* Hence, it is proved that array name stores the address of first element of array.

C++ void pointers

```
#include <iostream>
using namespace std;
int main()
{
void* Ptr;
int a=9;
Ptr = &a;
std:: Cout << &a << std:: endl;
std:: Cout << Ptr << std:: endl;
return 0;
}
```

Output - 0x7ffcf1da5e04
0x7ffcf1da5604
.... Program finished with exit code 0
Press enter to exit console

- Syntax - void * ptr ;

* A void pointer is general-purpose pointer that can hold address of any data type but it is not associated with any data type
* In C++, we cannot assign the address of a variable to the variable of diff data types.

Syntax of void pointer

```
int * ptr ;
float a = 10.2 ;
ptr = &a ;
```


* In the above ex, we declare declare of type integer, After declaration, we try to store address of 'a' variable in 'ptr', but this is not possible in C++ as the variable cannot hold the address of different data types

Pointers

1) A pointer is a variable that holds memory address of another variable

2) A pointer can be reassigned

3) A pointer has its own memory address & size on the stack

4) Pointers can be assigned null directly

5) int a = 10
int * P = &a;
OR
int * P;
P = &a;

References

1) A reference variable is an alias that is another name of an already existing variable

2) A reference cannot be reassigned

3) Reference shares the same memory address but also takes up some space on the stack

4) Reference cannot assigned null directly

5) int a = 10;
int &P = a; // it is correct
but
int &P;
P = a; // it is incorrect

Function pointer in C++

* As we know that pointers are used to point some variables; similarly, the function pointer is a pointer used to point functions.
* They are mainly useful for event-driven applications, callbacks

What is the address of functions

<<Program-Cpp>>	<<program-exe>>	Application
int main()	10010010	Heap
{	11001100	Stack
Cout << "Hello world";	111 000 11	Global
return 0;	1010 1010	Code
}		
→ compiler →	→	
Source code	machine code	

Syntax of declaration

```
int (*Funcptr)(int, int);
```

Example:

```
#include <iostream>
using namespace std;
int add (int a, int b)
{
```

```
    return a+b;
}
int main()
{
        int (*funcptr)
(int, int);
Funcptr = add;
int sum = funcptr(5,5);

Std :: Cout << "value of sum is:"
<< sum << std :: endl;
return 0;
}
```

@python_world_in

Output :-

Value of sum is : 10

④ C++ Object class-Concepts

* The major purpose of C++ programming is to introduce the concept of object orientation to C programing language

Object Oriented programing is a paradigm that provides many concept such as inheritance, data binding, polymorphism

Object means a real word entity such as pen, chair etc

Object Oriented programing is a methodology or

Paradigm to design a program using classes & objects

i) object
ii) class
iii) inheritance
iv) Polymorphism
v) Abstraction
vi) Abstraction
vii) Encapsulation

Advantages of OOP Over procedure Oriented Programming language

1) oops makes development & maintanance easier where as in procedure-Oriented programming language it is not easy to manage it code grows as a project size grows

2) oops provide data hiding where as in procedure Oriented programing language a global data can be accessed from anywhere

3) Oops provide ability to simulate real world event much more effectively, we can provide the sol^n of real world problem.

C++ Object

- Object is the real world entity for ex- chair, Car, Pen

- Object is a runtime entity that has state & behaviour
- Object is instance of class

C++ class

- class is a group of similar objects it is a template from which object are created

① C++ Object and class Examples

```
#include <iostream>
using namespace std;
class student {
 public:
  int id;
 String name;
};
 int main() {
     Student S1;
     S1.id = 201;
     S1.name = "SONOO Jaiswal";
Cout << S1.id << endl;
Cout << S1.name << endl;
return 0;
}
```

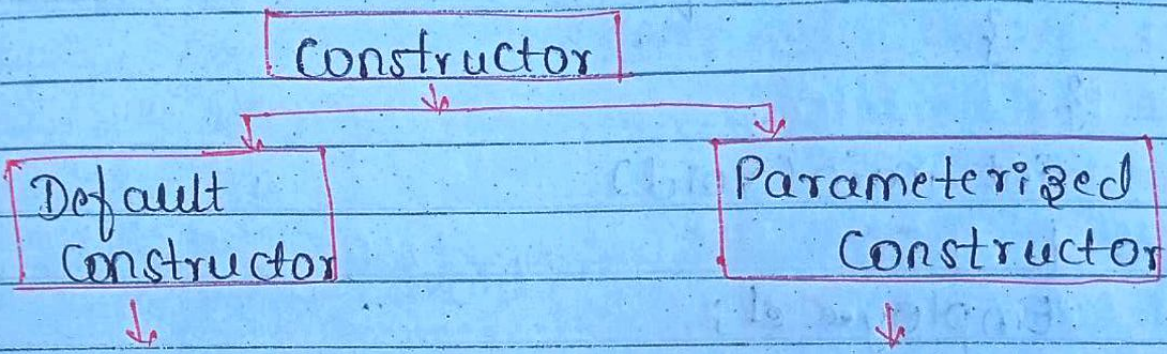

Output -

```
201
SONOO Jaiswal
```

C++ Constructor

* In a C++ constructor is a special method which is invoked automatically at the time of object creation.
* It is used to initialize the data member of new object generally

-There a two types of constructor



A constructor which has no argument is known as default constructor	A constructor which has parameter is called parameterized constructor
Has no parameter	Has one or more Parameter
If the programer has not written a constructor the default constructor is automatically called	Programer should write his own constructor writing parameterised constructor

Example for default Constructor

```
#include <iostream>
using namespace std;
class Employee
  {
   Public
     Employee ()
    {
     cout << " default constructor Involved
    }
}
 int main(void)
 {
 Employee e1;
 Employee e2;
return 0;
}
```

Output- Default constructor invoked
Default constructor invoked

Example for parameterized Constructor

```
#include <iostream>
using namespace std;
class Employee {
      Public:
    int id;
    String name;
```

```
Float salary;
Employee (int i, string n, float s)
{
  id = i;
  name = n;
  Salary = s;
}
void display()
{
Cout <<id<<" "<< name <<" "salary<<endl;
  }
};
int main(void)
  {
  Employee e1 = Employee (101, "sonoo890000);
Employee e2 = Employee (102, "Nakul", 59005);
  e1. display();
  e2. display();
  return 0;
  }
```


Output -

```
101   sonoo   890000

102   Nakul   59000
```

C++ Copy constructor

A copy constructor is an overloaded constructor used to declare & initialize an object from another object.

copy constructor

- Default copy constructor
- User-defined construction

The compiler defines the default copy constructor If the user defines no copy constru-ctor, compiler supplies its constructor.

The programmer defines the user define constructor

Syntax

class_name(const class_name & old_object)

```
// Program of copy constructor
#include<iostream>
using namespace std;
Class A
{
Public :
int x;
```

```
A(int a)
{
 x=a;
}
A(A &i)
{
  X=i.x;
}
};
 int main()
 {
A a1(20);
A a2(a1);
Cout << a2.X;
return 0;
 }
```


Output - 20

Two types of copies are produced by constructor

- shallow copy
- Deep copy

Shallow copy:

i) It is defined as the process of creating the copy of an object by copying data

of all the member variables as it is

ii) The default copy constructor can only produce the shallow copy

* Deep copy

i) Deep copy dynamically allocated the memory for the copy & then copies the actual value, both the source & copy have distinct memory locations

ii) Deep copy requires us to write the user-defined constructors

// Program of shallow copy

```
#include <iostream>
using namespace std;
class Demo
{
 int a;
 int b;
 int *p;
   Public
     Demo()
 {
   P= new int;
 }
  void setdata(int x, int y, int z)
 {
```

```
    std::cout<<"value of a is: "<<a<<std::endl;
    std::cout<<"value of b is: "<<b<<std::endl;
    std::cout<<"value of *p is: "<<*p<<std::endl;
    }
};
int main()
{
Demo d1;
d1.setdata(4, 5, 7);
Demo d2=d1;
d2.showdata();
return 0;
}
```


Output -

```
value of a is : 4
value of b is : 5
value of *p is : 7
```

// Program of deep copy

```
#include<iostream>
    using namespace std;
    class Demo
     {
      Public:
      int a;
      int b;
```

```
    int *P;
Demo()
{
 P=new int;
}
Demo(Demo &d)
{
 a=d.a;
 b=d.b;
 P=new int;
 *p=*(d.P)
}

Void showdata()
std::Cout<<"value of a is:"<<a<<std::endl
}
};

int main()
{
 Demo d1.
 d1.setdata(4,5,7);
 Demo d2=d1;
 D2.showdata();
 return 0;
}
```

Output - value of a is : 4
value of b is : 5
value of *p is : 7

C++ destructor

* A destructor works opposite to constructor. It destructs the object of class
* It can be defined only once in the class like constructors, it is invoked automatically
* A destructor is define like constructors It must have same name as class
* C++ destructor cannot have parameter modifiers can't be applied on destructor

// let us see Example of Constructor & destructor

```
#include <iostream>
  using namespace std;
   class Employee
    {
     Public :
      Employee ()
       {
  Cout << "Constructor invoked" << endl;
  }
   ~ Employee ()
  {
   Cout << "Destructor invoked " <<endl;
```

```
    }
};
int main(void)
  {
    Employee e1;
    Employee e2;
    return 0;
  }
```

Output -

Constructor Invoked
Constructor Invoked
destructor Invoked
destructor Invoked.

C++ this pointer

* It can be used to pass current object as a parameter to another method.

* It can be used to refer current class instance variable
* It can be used to declare indexes

```
// let us see Example
#include <iostream>
  using namespace std;
    class Employee {
      Public:
```

```
int id;
string name;
Float salary;
Employee (int id, string name, float salary)
{
 this -> id = id;
 this -> name = name;
 this -> salary = salary;
}
 void display()
 {
 Cout << id << "  " << name << "  " << salary << endl;
 }
};
int main(void){
Employee e1 = Employee(101, "Sonoo", 8900);
Employee e2 = Employee(102, "Nakul", 5900);
e1. display()
e2. display()
 return 0;
}
```

Output - 101 Sonoo 8900
102 Nakul 5900

C++ Friend function

i) If a function is defined as friend function in C++, then protected & private data of class can be accessed using function.

ii) By using keyword friend function compiler knows the given function is a friend function

Declaration of friend function

```
class class_name
    {
    Friend data_type function_name
                        (argument/s);
    };
```

// Let us see the Example

```
#include <iostream>
  using namespace std;
  class Box:
  {
  Private:
    int length;
  Public:
    Box(): length(0) { }
    friend int printlength(Box);
  };
```

```
int printlength (Box b)
{
 b length t = 10;
 return b length;
}
 int main ()
 {
  BOX b;
  Cout << "length of box:" << Printlength (b) << endl;
  return 0;
 }
```


Output - length of BOX : 10

C++ Inheritance

i) Inheritance is a process in which one object acquires all the properties & behavioral of its parent object automatically.

ii) In such a way you can reuse, extend & modify the attributes & behaviour

iii) The class which inherits the members of another class is called derived class & the class whose members are inherited is called base class

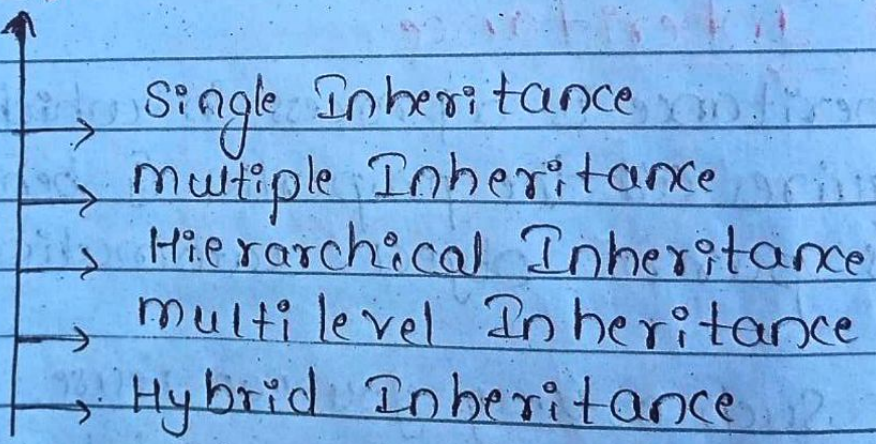
iv) Advantage - code Reusability) -

Now you can reuse the members of your parent class, so there is no need to define the member again.

Note :

- In C++, the default mode of visibility is private.
- The private member of base class are never Inherited.

Types of Inheritance



// let us see Example

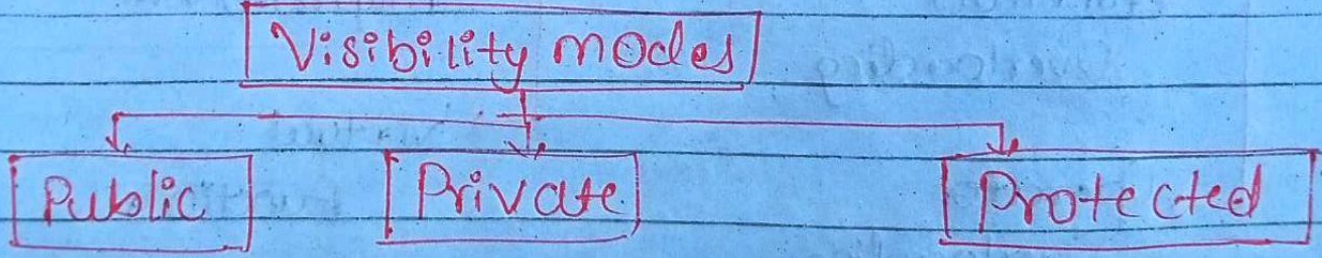
```
#include <iostream>
using namespace std;
class Account {
Public:
  Float Salary = 60000;
};
```

```
class programmer : Public account {
Public :
float bonus = 5000;
};
int main (void) {
programmer P1 ;
Cout << "salary:" << P1.salary << endl ;
Cout << " Bonus:" << P1. bonus << endl;
return 0;
}
```

Output -

Salary : 60000
Bonus : 5000

- visibility modes can be classified into three categories :



When the member is declare as a Public, it is accessible to all the functions of program	when the member is declare as a Private, it is accessible within the class only	when the member is declare as Protected, it is accessible within its own class

- Visibility of Inherited members

Base class	Derived class visibility		
Private	Not Inherited	Not Inherited	Not Inherited
Protected	Protected	Private	Protected
Public	Public	Private	Protected

C++ Polymorphism

The term "Polymorphism" is the combination of poly + morphs, which means many forms

Polymorphism Types

Compile time polymorphism
- Function Overloading
- Operator Overloading

Runtime polymorphism
- Polymorphism
- virtual Functions

// let us see Example

```
#include<iostream>
using namespace std;
class Animal {
```

```
Public :
  String color = "Block";
};
class Dog : Public Animal
{
 Public :
   String color = "Grang";
};
 int main (void) {
 animal d = Dog ( );
  Cout << d. color;
}
```

Output -

Black

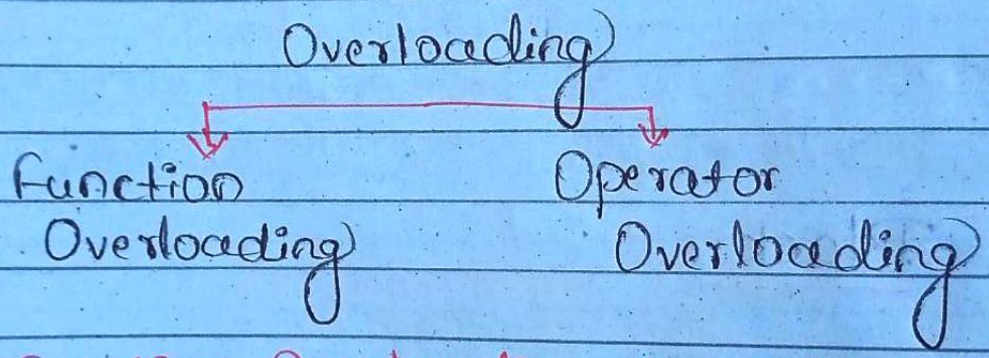
#learn more
Grow more

C++ Overloading

If we create two or more members having the same name but different in number or a type of parameter, is known as C++ Overloading.

In C++, we can Overload -

i) methods
ii) Constructors
iii) Indexed properties



// Function Overloading Example

```
#include <iostream>
using namespace std;
class cal {
  Public
static int add (int a, int b) {
    return a+b;
  }
  Static int add (int a, int b, int c)
  {
  return a+b+c;
  }
};
```

```
int main(void){
cal c;
cout << c.add (10, 10) <<endl;
cout << c add (12, 20, 23);
return 0;
}
```

Output - 30
55

C++ Operator Overloading

// operator that cannot be Overloaded are follows

i) Scope operator (::)

ii) Size of

iii) member selector(.)

iv) member pointer selector (*)

v) ternary operator (?:)

// Syntax of Operator Overloading

```
return_type class_name :: Operator op
{
  // body of Function.
}
```

```
#include <iostream>
using namespace std;
class Test
{
  Private :
    int num;
  Public:
    Test() : num(8) { }
    void operator++() {
    num = num + 2 ;
  }
  Void print() {
    Cout << The Count is: " << num;
  }
};
int main()
{
  Test tt;
  tt++;
  tt.print();
  return 0;
}
```

Output -

The count is : 10

C++ Function Overloading

If derived class defines same function as defined in its base class, it is known as function Overriding.

// C++ Function Overloading Example

```
#include<iostream>
using namespace std;
 class Animal {
  Public:
  Void eat() {
 Cout<<" eating...";
  }
 };
 class Dog: Public Animal
   {
   Public
   Void eat()
   {
 Cout<<" Eating bread...";
  }
};
 int main(void) {
 Dog d = Dog();
  d. eat();
  return 0;
  }
```

Output - Eating bread.

Abstraction

Interfaces in c++ (Abstract classes)

Abstract classes are the way to achieve abstraction in c++. Abstraction in c++ is the process to hide the internal details & showing functionality only

Abstraction can be achieved by -

1. Abstract class
2. Interface

// let us see Example

```
# include <iostream>
  using namespace std;
  class shape
  {
  Public:
   virtual void draw()=0;
  };
  class Rectangle : shape
  {
    Public
    void draw()
  {
    Cout << " drawing rectangle :: " <<endl;
  }
};
```

```
class circle : shape
 {
  Public
  void draw()
  {
  Cout <<"drawing circle "<<endl;
  }
};
  int main() {
   Rectangle rec;
   Circle cir;
   rec-draw();
   cir-draw();
   return 0;
  }
```


Output - drawing rectangle
drawing circle

C++ Strings

In c++ string is an object of std :: string class that represents sequence of charactors we can perform many functions on string such as concatination, comparison

Example

```
#include<iostream>
  using namespace std;
```

```
int main() {
  String S1 = "Hello";
   char ch[] = {'C', '+', '+'};
String S2 = String(ch);
Cout << S1 << endl;
Cout << S2 << endl;
}
```

Output - Hello
C++

C++ Exception Handling

Exception Handling is the process to handle runtime errors.

We perform exception handling so the normal flow of application can be maintained even after runtime errors.

Advantage - It maintains the normal flow of application... rest of code is executed even after exception

C++ Exception classes

Std : exception

- Std : bad_alloc
- Std : domain_error
- Std : bad_cast
- Std : invalid_argument

std : bad_typeid

std : length_error

std : bad_exception

std : out_of_range

std : bad_logic failure

std : runtime_error

std : runtime_error →
- std : overflow_error
- std : range_error
- std : underflow_error

Exeeption	Description
std :: exception	It is an exception & parent class of all std c++ exceptions
std :: logic_failure	It is an except that can be detected by reading a code
std :: runtime_error	It is an exception that cannot be detected by reading a code
std :: bad_exception	It is used to handle the unexpected exceptions
std :: bad_typeid	This exception is generally be thrown by typeid
std :: bad_alloc	This exception is generally thrown by new

Exception handling Keywords

(a) try
(b) catch
(c) throw

In c++ programming, exception handling is performed using try/catch statements.

i) The c++ try block is used to place the code that may occur exception

ii) The catch block is used to handle the exception

Catch: A program catches an exception with an exception handler at the place in the program where you want to handle problem

try: It identifies block of code for which particular exception will occures

throw: A program throws an exception when a problem shows up

C++ Interview Questions

1. What are the different data-types present in C++?
2. What are the class & object in C++?
3. What is Operator Overloading?
4. Explain Constructor in C++?
5. Tell me about vertual function
6. What do you know about friend class & friend function
7. In destructor Overloading possible? if yes then explain & if no then why?
8. What is the difference between vertual functions & pure vertual function?
9. What are the void pointers?
10. How do you allocate & deallocate memory in C++?
11. What is the size of void in C++?
12. What is a referance in C++?

C++ Interview Questions

1. What are the different data-types present in C++?
2. What are the class & object in C++?
3. What is Operator Overloading?
4. Explain constructor in C++?
5. Tell me about vertual function
6. What do you know about friend class & friend function
7. In destructor Overloading possible? if yes then explain & if no then why?
8. What is the difference between vertual functions & pure vertual function?
9. What are the void pointers?
10. How do you allocate & deallocate memory in C++?
11. What is the size of void in C++?
12. What is a referance in C++?