PAGE No.
DATE / /

COPYRIGHTED By

Instagram - @ Codes learning

Youtube - Tech 21

Telegram - Codeslearningnow.

Written By -

JAREKAR ARPITA RAMESH

SOFTWARE ENGINEER

Indexe —

PAGE No.
DATE / /

chaptee numbee	chaptee Name
1	Oveeview
2	C++ Vaeiables
3	C++ Data types
4	C++ keywoeds
5	C++ operatoes
6	C++ Identifiees
7	Control statement
8	C++ loop
9	C++ Aeeays.
10	C++ functions.
11	call by value call by veferance
12	C++ Recuesion.
13	C++ storage classes
14	C++ Pointees.

PAGE NO.
DATE / /

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).

PAGE No.
DATE / /

C++ ....

Programming

@codes learning.

1. OVERVIEW

- What is C++ ?

- C++ is a statically typed, compiled, general-purpose free-form programing language that supports: 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 Stroustrup starting in 1979.
- C++ is a superset of C & that verbally any C program is a legal C++ program.

- Object-Oriented Programming -

C++ fully support object-oriented programing including four pillar of object-oriented development

- Encapsulation
- Data hiding
- Inheritance
- Polymorphism.

Codes learning

- 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 rich set of functions manupulating files, strings etc.
- The standard template libraries (STL) giving a rich set of methods manupulating data structures.

- 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 become 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 hardware.

PAGE No.
DATE / /

- 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 firmware.

- C++ Program -

```
# include <iostream>
Using namespace std;
int main () {
    Cout << " Hello C++ Programming ";
    return 0;
}
```

PAGE NO.
DATE / /

- 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 modifiers for class member to make it inaccessible.
It follows top-down approach.	It follows bottom-up approach.
C doesn't support reference variable.	C++ supports reference variables.
Scanf() & printf() mainly used for o/p input / output	cin & cout are used to perform input / output operations.
C does not supports Inheritance	C++ supports the feature inheritance.
C doesn't provide feature of namespace	It supports feature of namespace.

PAGE No
DATE / /

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 tc3.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 conc
  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 output operation
- If bytes flow from device like printer, displ
  screen or a network connection to main
  memmory, this is called input operation.

I/O Library Header files -

Header file	Function & Discription
<iostream>	It is used to define cout, cin & cerr objects, which corresponds to standard output stream, standard input 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 ostream class. It is connected with standard output device, which is usually a display screen.
The Cout is used in conjuction 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.

Standard input stream (cin)

- The cin is predefined object of istream class. It is connected with the standard input device, which is usually a keyboard.
- The cin is used in conjunction with stream extraction operator (>>) to read the input from a 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 object 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 location. It is used to store data. Its value can be changed.
- It is a way to represent memory location 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.

PAGE No.
DATE / /

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++ -

i) Basic - int, char, float, double
ii) Derived - array, pointer.
iii) Enumeration - enum
iv) User defined - structure.

i) Basic Data Types -

- The basic data types are integer based & floating-point based.
- C++ language supports both signed & unsigned 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.	-

PAGE No.
DATE / /

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.

@ Codes learning :)

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.

Type	operatoe.
Binaey Operatoes: Arithmatic operatoe	+, -, *, /, %
Relational operatoe	<, <=, >, >=, ==
Logical operatoe	&&, ||, !
Bitwise operatoe	&, |, <<, >>, ~, ^
Assignment operatoe	=, +=, -=, /=, %=
Unaey operatoe	++, --
Conditional operatoe	?:

Happy Learning

C++ Identifiers -

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

- Constant
- Variables
- Functions.
- Labels.
- Defined Data Types.

PAGE No.
DATE / /

- Let's look at a simple example to understand concept of identifiers.

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

int main ()

  {
int a;
int A;

Cout << " Enter the values of 'a' & 'A';

  cin >> a;
  cin >> A;

Cout << "\n 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

PAGE No.
DATE / /

- Difference between identifiers & keywords.

	Identifiers	Keywords
i)	Identifiers 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.
ii)	It is used to identify name of variable	It is used to specify type of entity.
iii)	It can consists of - letters, digits and underscore.	It consists only letters.
iv)	It can use both - lowercase and uppercase letters.	It use only lowercase letters.
v)	It can be classified as internal & external identifiers.	It cannot be further classified.
vi)	Example are test, result, sum, power etc.	Examples are 'for', 'if', 'else', 'break' etc.

PAGE No
DATE / /

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

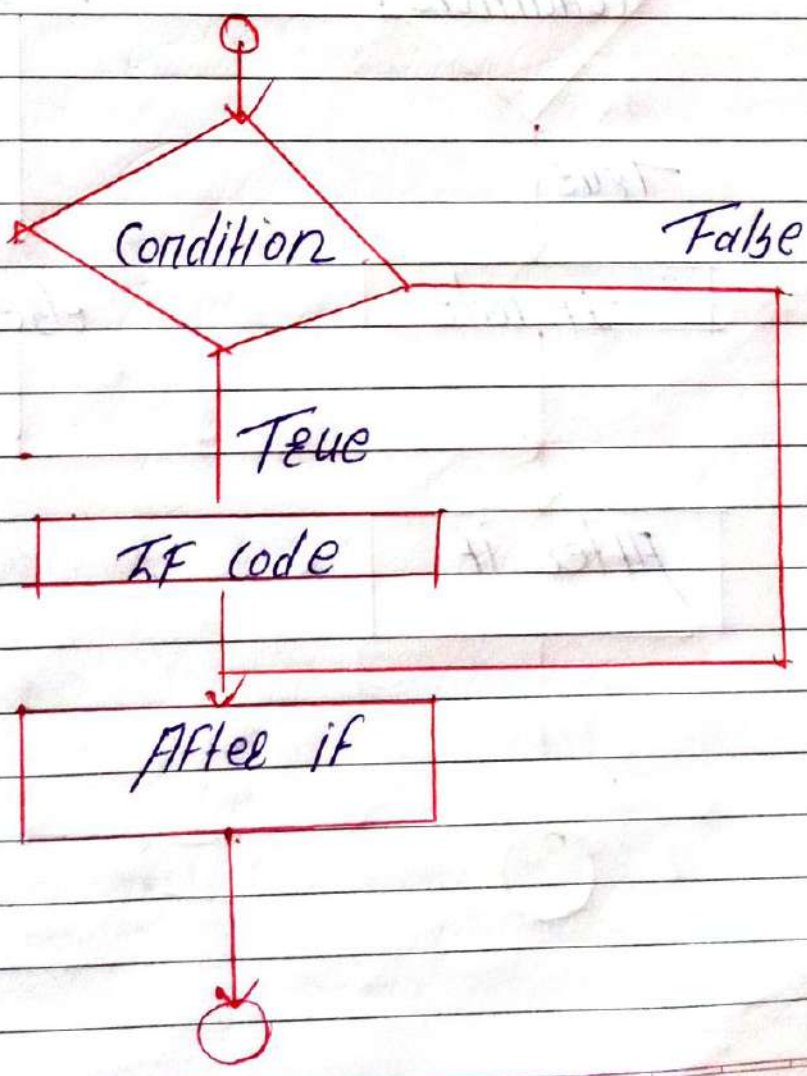
- C++ if statement

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.

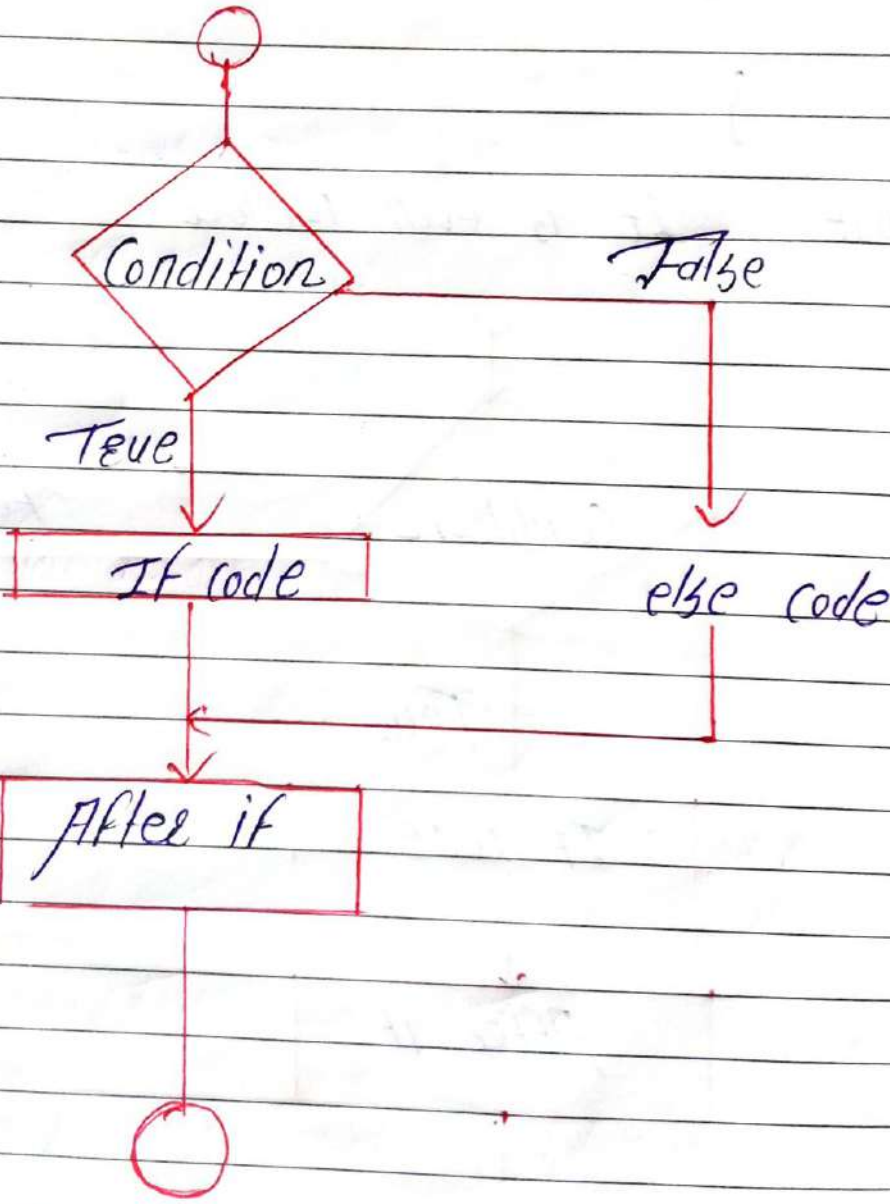


PAGE NO.
DATE / /

- C++ if else statement

Syntaxe -

```
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)

            {

                Coub << "It is even numbee";

            }

        else

            {

                Coub << "It is odd numbee";

            }

    Eeturn 0;

    }
```

Output -

It is odd numbee.

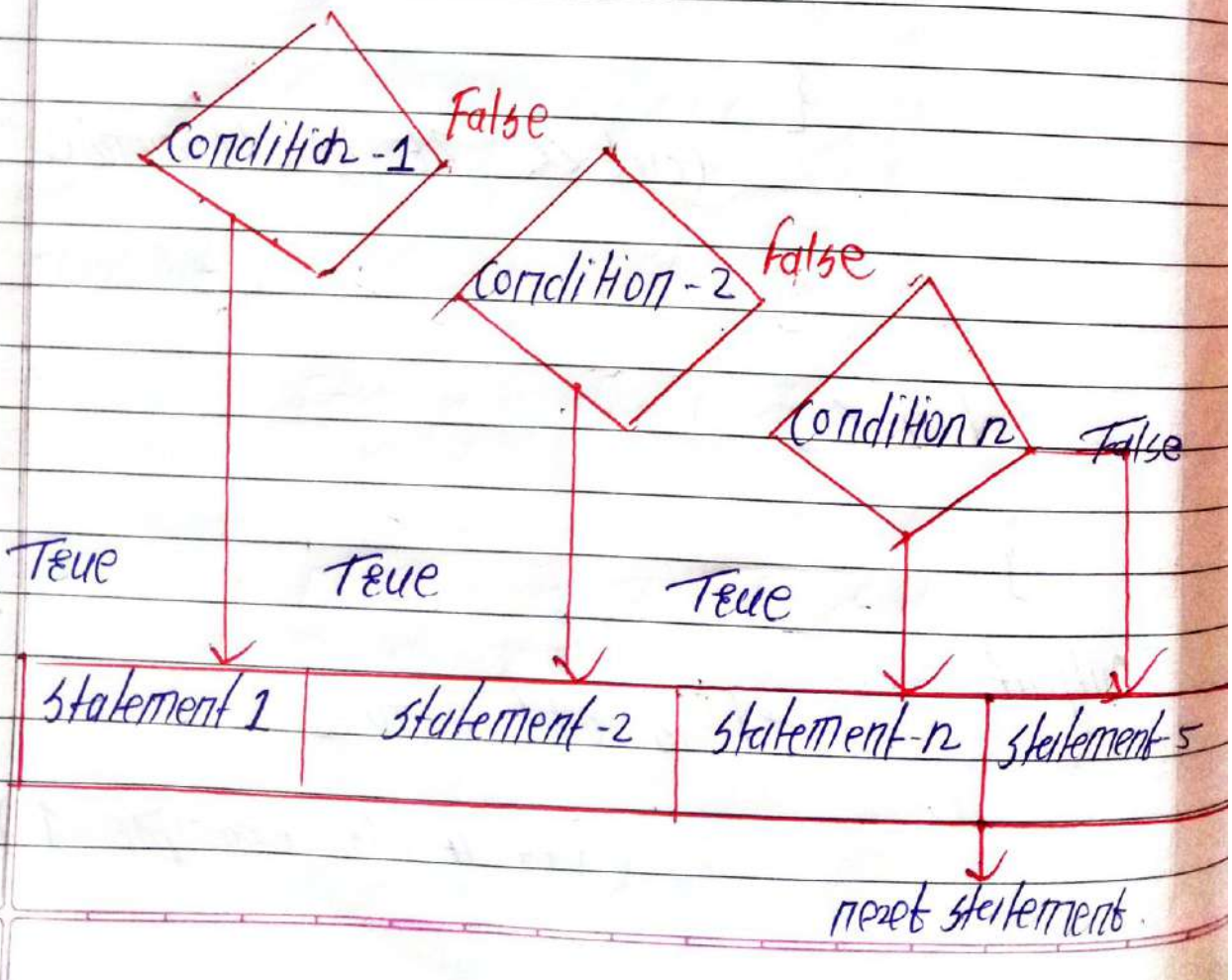
(C++ if else example ↑)

PAGE No.
DATE / /

- C++ if-else-if ladder statement

Syntax -

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



PAGE No.
DATE / /

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 >= 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";
    }
}
```

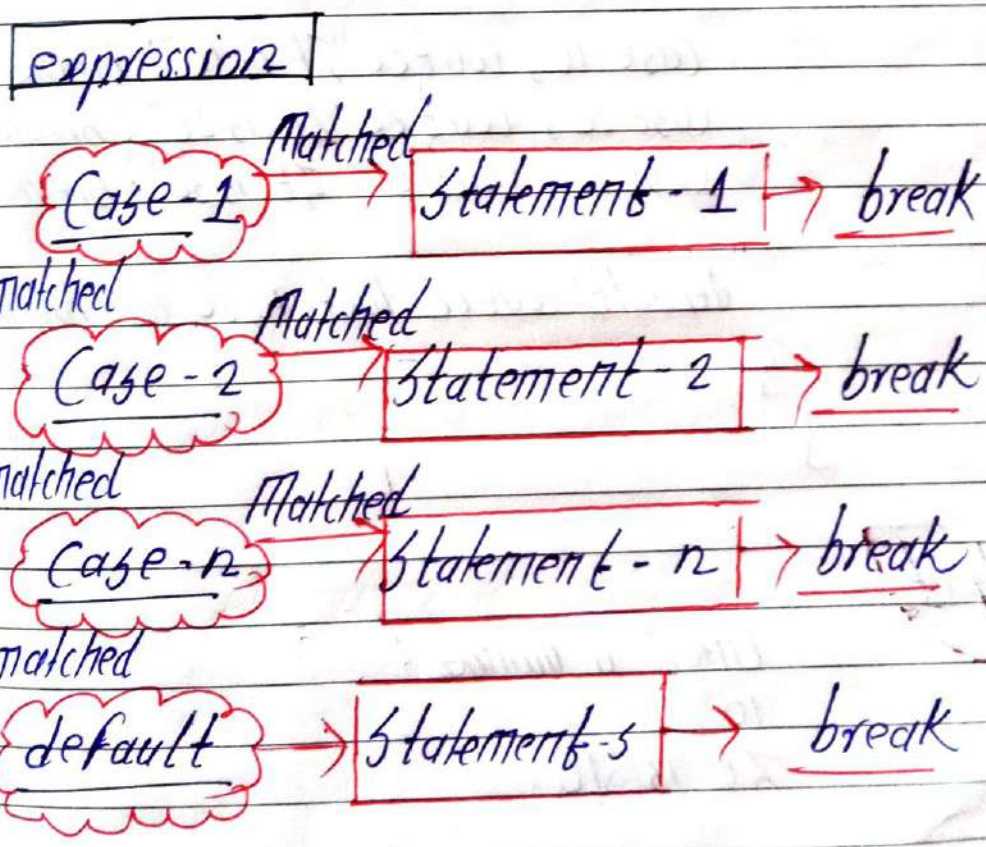
Output -

- Enter a number to check Grade = 66
  C Grade
- Enter a number to check Grade = -2
  Wrong Number.

C++ switch -

Syntaxe -

```
switch (expression) {

Case value 1;
   break;

Case value 2;
   break;
.
.
default:
   break;
}
```



PAGE NO
DATE / /

- 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.
```

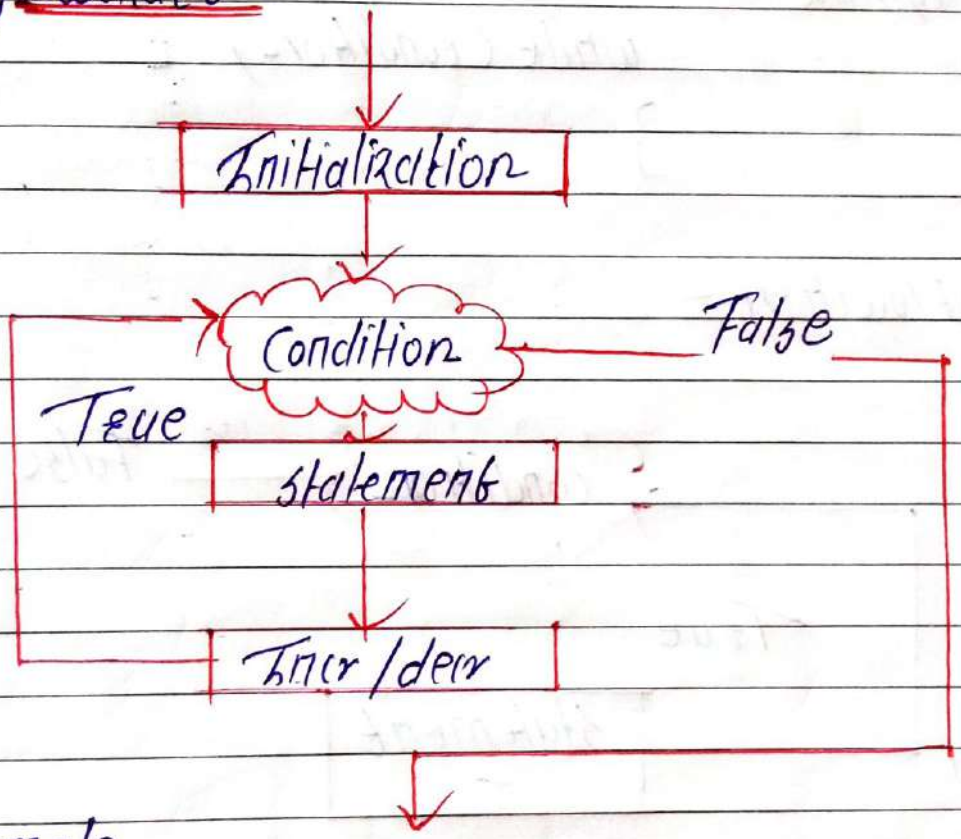
PAGE No.
DATE / /

- C++ For loop - It is used to iterate part of program several times.

- Syntax -

```
For (initialization; 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

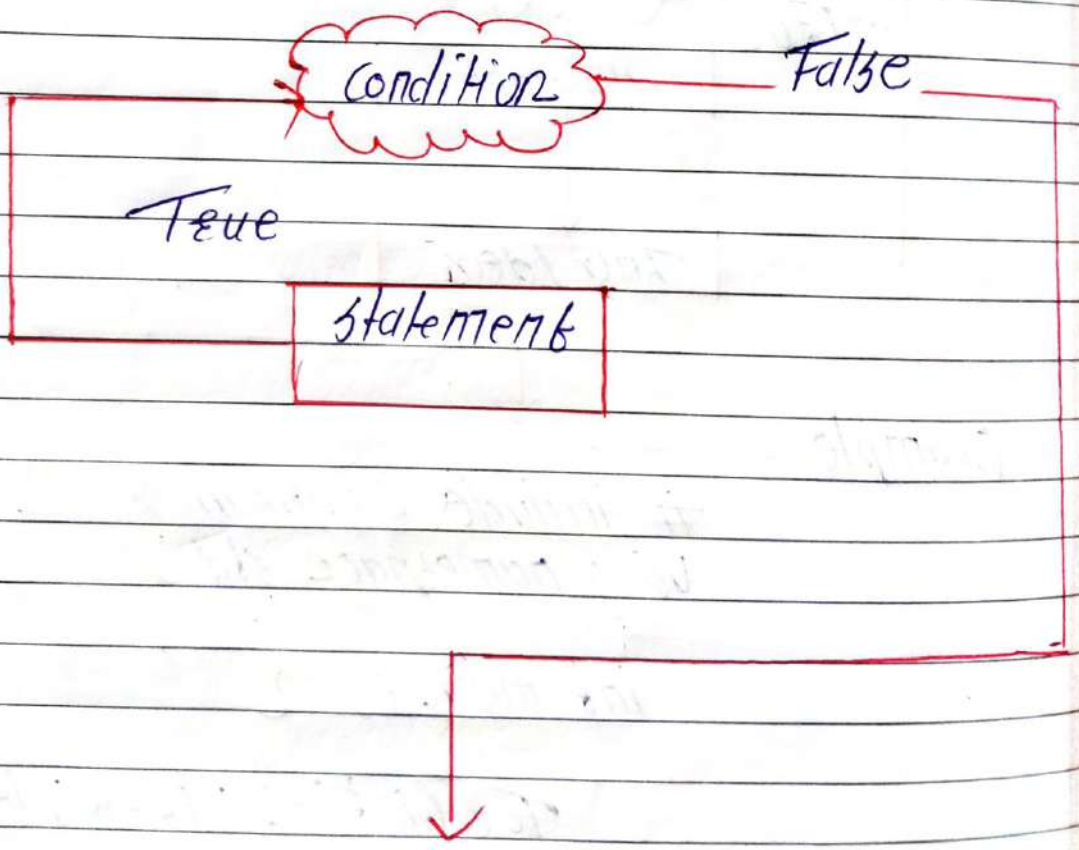
C++ While loop

- 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

— Syntax —

```
while (condition) {
}
```

— Flowchart —



PAGE No.
DATE / /

- Example for while loop -

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

int main() {
int i = 1;

    while (i<=10)
    {
        coub << i<< "\n";

        itt;
    }

}
```

Output -

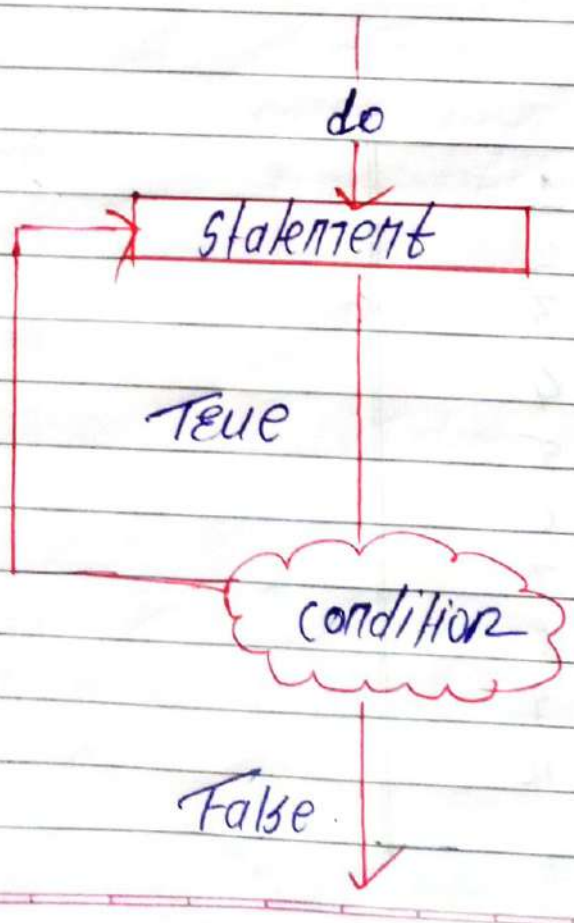
1
2
3
4
5
6
7
8
9
10

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);



PAGE No.
DATE / /

- 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
```

@codes learning

C++ Break Statement

- The break statement is used to break loop or switch statement.
- It break the current flow of program at given condition.

```
jump-statement;

break;
```

- Flowchart

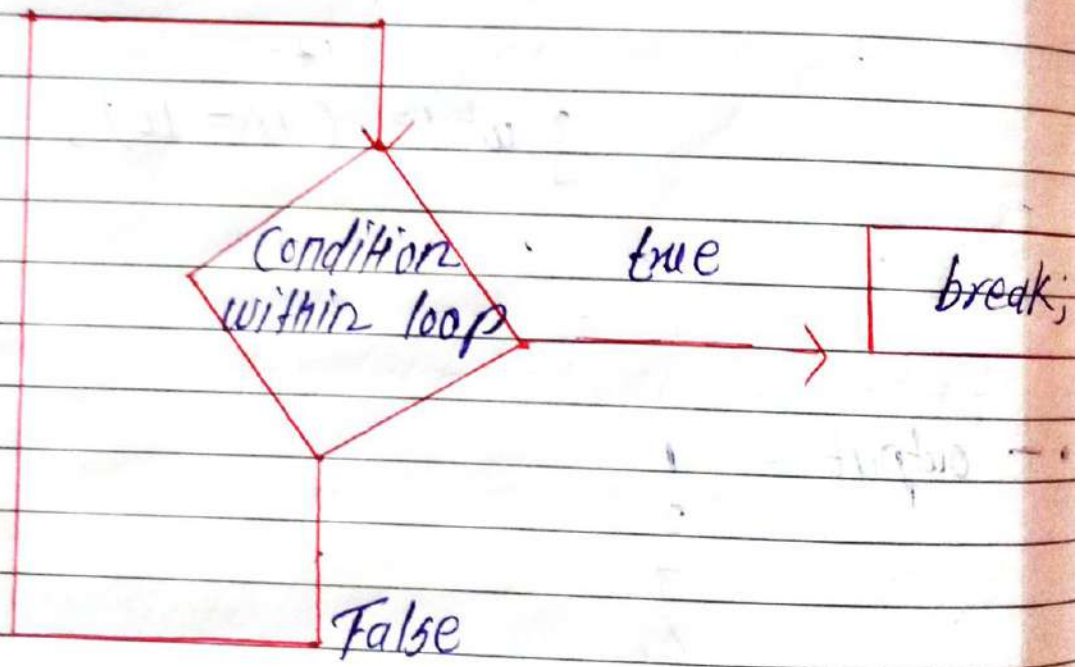


Fig- Flowchart of break statement

- let us see example -

```
# include <iostream>
Using namespace std;
int main () {
    for(int i = 1; i<=10; i+1)
        {
            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

PAGE No.
DATE / /

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

ii) 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 contigious 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.

PAGE NO
DATE / /

- 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,
    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 array2[5] = {5, 15, 25, 35, 45};

    printArry(arr1);
    printArry(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 diamentional or three diamentional. The data is stored in tubular form.

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

int main ()
{

    int test [3] [3];
```

```
test[0][0] = 5;
test[0][1] = 10;
test[1][1] = 15;
test[1][2] = 20;
test[2][0] = 30;
test[2][2] = 10;
for(int i=0; i<3; ++i)
{
    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 function in c++ language is also known as procedure or subroutine in other programing 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.

i) Code Reusability -

– By creating functions in c++ you can call it many times. so we don't need to write the same code again.

ii) Code optimization -

– 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.

PAGE No.
DATE / /

Types of Functions

Library Functions

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

User-defined function

The functions which are created by the c++ programmer, so that he can use many times. It reduces complexity of big problem.

- **Syntax for creating functions**

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

- **Example For function -**

```
# 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 & and j=1

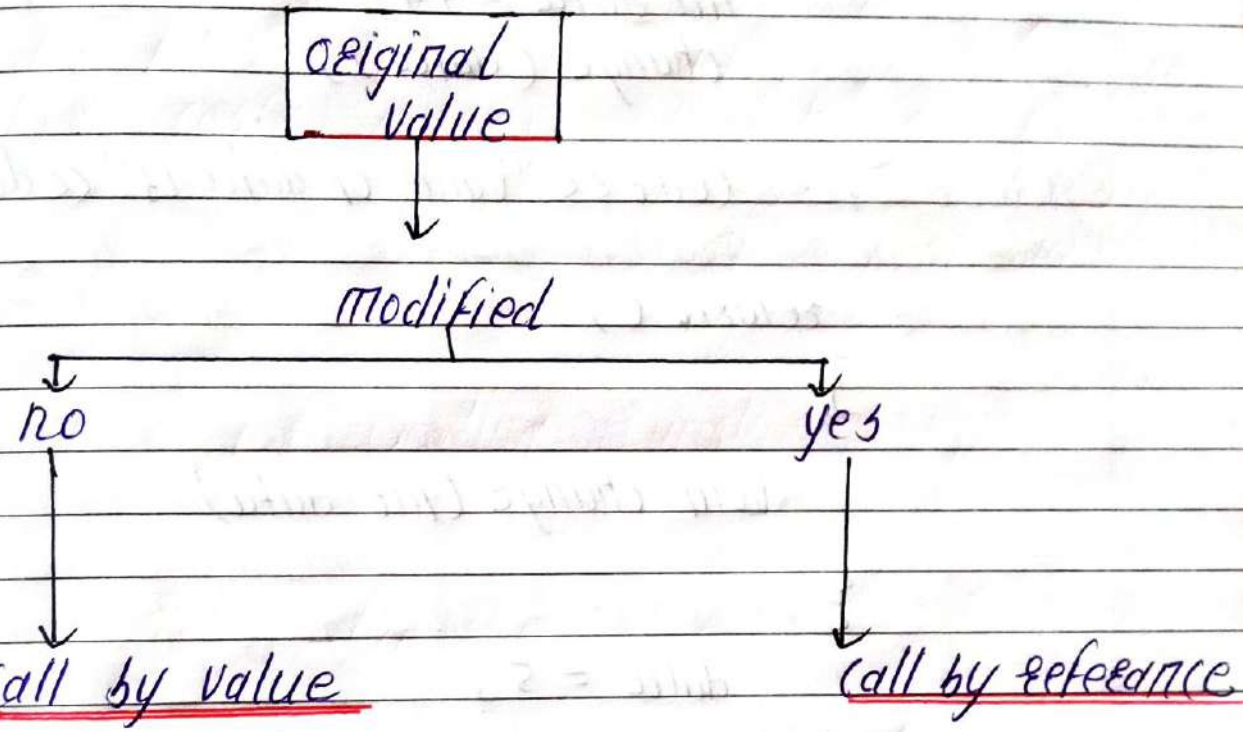
i=2 & j=1

i=3 & j=1

PAGE No
DATE / /

• Call by value & call by reference -

There are two ways 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 reference.



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<<en

   return 0;
  }

     Void change (int data)
  {
     data = 5;
  }
```

- Output -

Value of data is 3.

ii) Call by Reference -

- original value is modified as 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 Reference
A copy of value is pass-to the function	An address of value is pass to function.
changes made inside the function is not reflect--ed on the othee - functions.	changes made inside the function is reflected on the othee functions.
Actual & formal arguem--ents will be created in different memory location	Acutal & formal argu-ments will be created in same memory location.
Original value is - not modified.	Original value is modified, cos we pass the referance.

Difference Between - Call by Value / Call by Reference

C++ Recursion -

```
#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 16 : 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.

PAGE No.
DATE / /

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	Garbage
Register	register	function Block	local	Garbage
Mutable	mutable	class	local	Garbage
External	extern	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.

- **Advantage of pointer.**

1) pointer reduces the code & improves the performance, it is used to retrieving strings, trees etc.

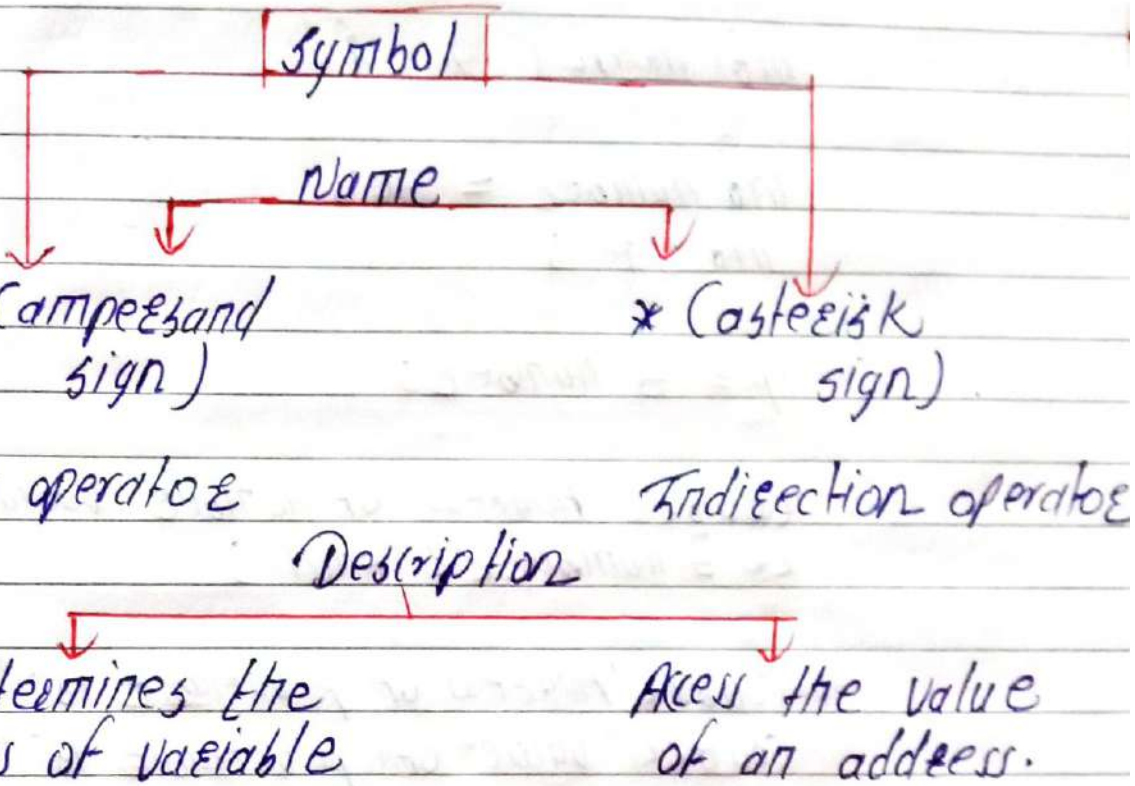
2) We can return multiple values from the function using pointer

3) It makes you able to aces any memory location in the computer's memory.

- **Usage of pointer -**

1) Dynamic memory allocation
2) Array, function & structure.

- Symbol used in pointer -



- Declaring a pointer -

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

- Lets see pointer Example -

```
#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

PAGE No.
DATE / /

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 related 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 Pointer

```
# 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
0x7ffcf1da5e04

.... program finished with exit code 0
press enter to exit console.
```

— Syntaxe - `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 different data-types.
- Syntax of void pointer -

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

- In the above example, we declare pointer 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.

• Pointer Vs References in C++

	Pointers	References
Def^{n}	i) A pointer is a variable that holds memory address of another variable	i) A Reference variable is an alias, that is another name of an already existing variable.
Reassignment	ii) A pointer can be reassigned.	ii) A Reference cannot be reassigned
Memory address	iii) A pointer has its own memory address & size on the stack.	iii) Reference shares the same memory address but also takes up some space on the stack.
Null-value	iv) Pointer can be assigned null directly	iv) Reference cannot assigned null directly.

Initialization.	int a = 10
	int *p = &a;
	OR
	int *p;
	p = &a;
		v) 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 memory
int main()                           10010010                 Heap
{                                    11001100                 Stack
cout << "Hello word";                11100011                 Global
return 0;                            10101010                 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;
}
```

Output -

Value of sum is: 10.

C++ Object class - Concepts.

- The major purpose of C++ programming is to introduce the concept of object oriented to c programming language
- object oriented programming 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 programming is a methodology or paradigm to design a program using classes & objects.

i) Object

ii) class

iii) Inheritance

iv) Polymorphism

v) Abstraction

vi) Encapsulation

— Advantages of OOP over Procedure - Oriented programming language →

1) OOPs makes development and maintainance easier where as in procedure - oriented programming language it is not easy to manage if code grows as a project size grows.

2) OOPs provide data hiding whereas in procedure oriented programming language a global data can be accessed from anywhere

3) OOPs provide ability to simulate real world events much more effectively, we can provide the solution of real world problem

— C++ Object —

- object is the real world entity for example - chair, car, pen.
- object is a runtime entity that has state and behaviour.
- object is instance of class

— C++ Class -

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

C++ Object and class Example -

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

class student {
    public :

        int id;
        steing name;
};
    int main () {

        student s1;
        s1.id = 201;
        s1.name = "sonoo Jaiswal";

    cout<< s1.id << endl ;

    cout << s1.name << endl;

    return 0;
}
```

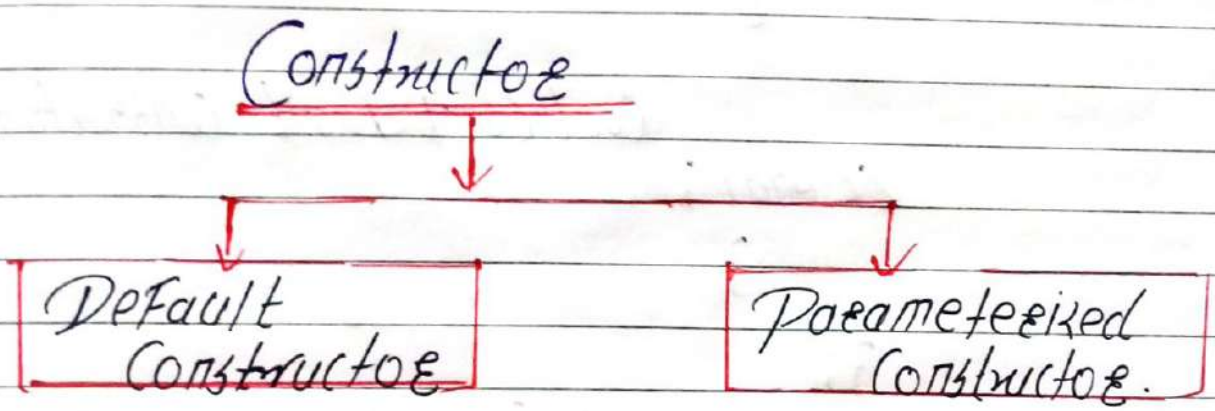
Output -

```
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 programe has not written a constructor the default constructor is automatically called.	programe should write his own constructor writing parameterized Constructor.

- Example for default Constructor -

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

class Employee
  {
    public
      Employee ()
      {
        cout << "Default Constructor Invo
<< endl;
      }
};
  int main (void)
  {
  Employee e1;
  Employee e2;
  return 0;
}
```

- Output -

Default Constructor invoked.
Default Constructor invoked.

PAGE No.
DATE / /

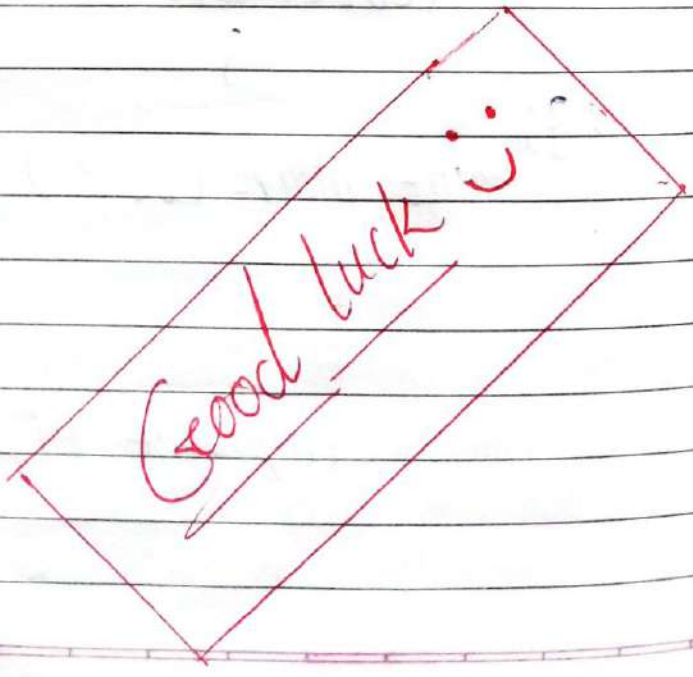
-Example for parameterized constructor-

```
#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, "Sonoo
                                890000);
```

```
Employee e2 = Employee (102, "Nakul", 5900
e1.display();
e2.display();
return 0;
}
```

⊙ Output -

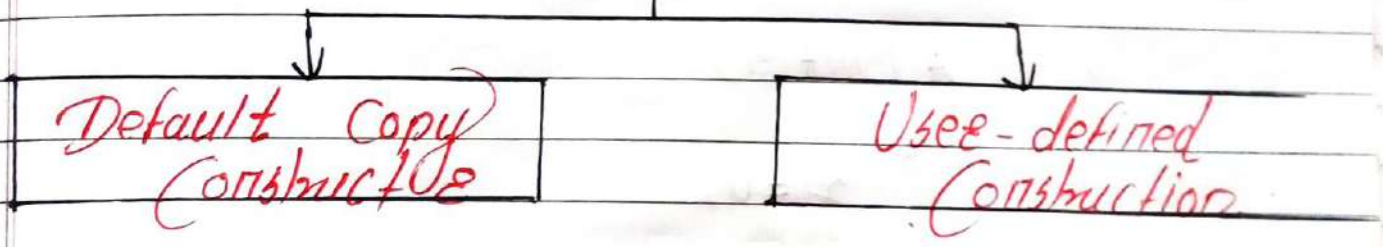
```
101 50000 890000
102 Nakul 59000
```



C++ Copy Constructor -

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

Copy Constructor.



The Compiler defines the default copy Constructor If the user defines no copy Constructor, 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 allocates 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;

  }
```

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 << "Cons
```

```
    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 indexers.

// 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 and 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 += 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 & behaviours of its parent object automatically.

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

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

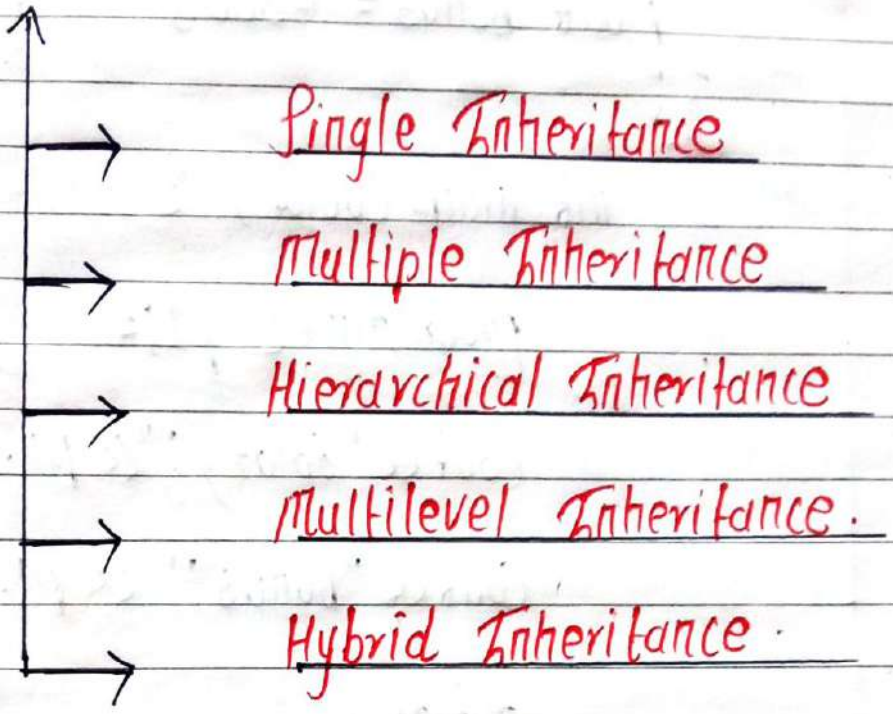
iv) Advantage - Code Reusability -

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) {
programmee 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 :

Visibility Modes

Public — when the member is declare as a public, it is accessible to all the functions of the program.

Private — when the member is declare as a private, it is accessible within the class only.

Protected — when the member is declare as protected, it is accessible within its own class.

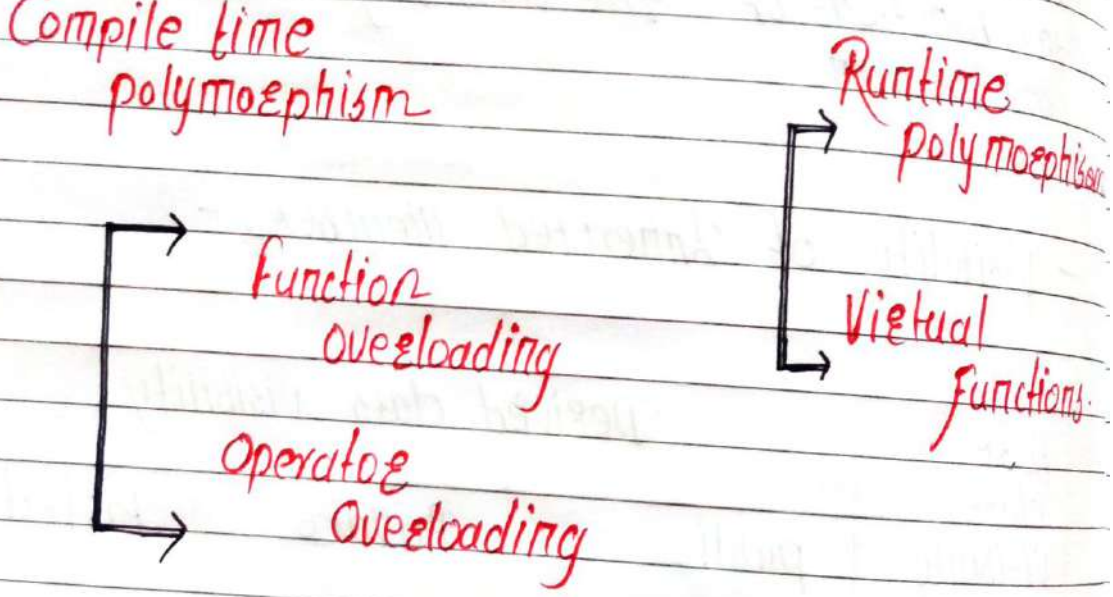
– Visibility of Inherited Members –

Base class visibility	Derived class visibility		
	public	Private	Protected.
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



// let us see Example -

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

class Animal {
```

```
public:
  string color = "Black";
};
  class Dog: public Animal
  {
      public:
        string color = "Gray";
  };
      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.

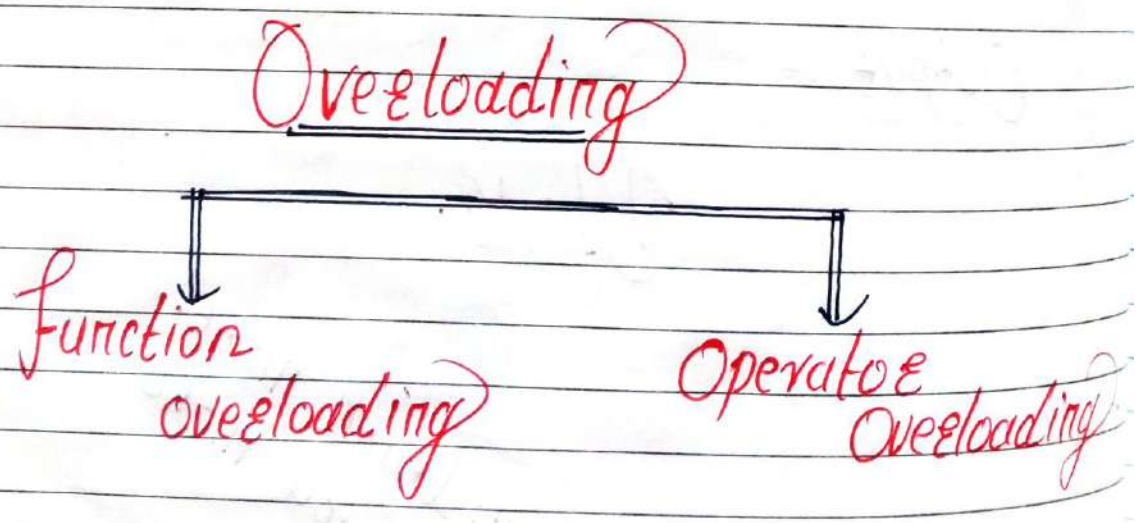
In C++, we can overload -

i) methods,

ii) Constructors

iii) Indexed properties

- Types of Overloading -



// 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, 20) << 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) teenary 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 Overeeiding -

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

// C++ function overeeiding 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...

Happy Coading

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 characters. We can perform many functions on string such as concatination, comparison.

String 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.

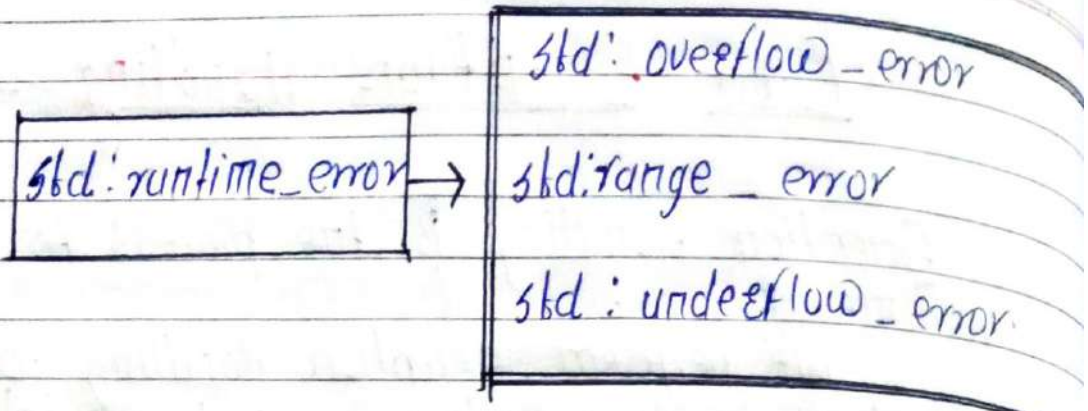
Advantage -

It maintains the normal flow of applicition... 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	



Exception	Description
std:: exception	It is an exception & parent cla of all standard c++ exceptions.
std:: logic_ failure	It is an exception 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 _ castypeid	This exception is generally be thrown by typeid.
std:: bad _ alloc.	This exception is generally thrown by new.

Exception Handling Keywords

ⓐ try
ⓑ catch
ⓒ 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 exceptions.

try - It identifies block of code for which particular exception will occurs.

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

throw - A program throws an exception when a problem shown 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. Is destructor overloading possible? if yes then explain & if no then why ?
8. What is the difference between virtual functions & pure virtual 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 reference in C++?