Programming

format:

```
#include<stdio.h>
int main()
{
    Printf
     Return 0;
}
```

Meaning:

1) // # is pre-processor directive.
   // stdio.h = standard input output header file.
   // #include<stdio.h> is a pre-processor program.

2) int main()
   //main () is a predefined function and actual program compilation starts from here.

   Printf is a pre-defined output function.

   // return 0; is optional & it is to finish the programme

* Algorithm:

→ Algorithm is defined as "the finite set of steps, which provide a chain of action for solving a problem.
→ It is step by step solution to given problem.
→ Well organized, pre-arrange and defined textual computational module.

Scanned by CamScanner

* Characteristic of good algorithm:

i) Correctness ii) Simplicity iii) Precision
iv) Comprehensibility [easy to read and understand]
v) Abstraction - [Give over-view without referring to low-level program code details.
vi) Efficient - does not waste any space or time.
vii) Easy to implement

* Steps to create Algorithm:

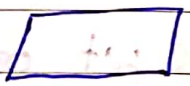
(i) Identify the inputs.
(ii) Identify the outputs.
(iii) Identify the Process.
(iv) Break the solution to steps.

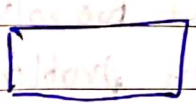
* Flow-chart:

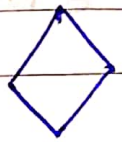
→ Flow chart is pictorial representation of an algorithm.
→ It is easy to understand

 (oval) → Denotes begining or end of algorithm.

⟶ Denotes the direction of logic flow in program.

(Ilgram)  Denotes either an input operation.

(rect)  Denotes a process to be carried out.

Diamond  Denotes a decision to be made.

Scanned by CamScanner

* Pseudocode:

→ Pseudocode statement that appear to have been written in a computer programming language but do not neccessary follow any syntax rule of any specific language.

→ The purpose of using Pseudocode is that it is easier for people to understand then conventional programming language code and that it is an efficient and environment independent description of the key principales of an algorithm.

eg

```
• Start
  Input myNumber
  Set myAnswer = Mynumber * 2
  output myAnswer
• Stop
```

* Variables

→ used to hold the content/value enter by users.

→ Also known as identifier.

* Constant

is

pie = 3.14

'e'

* Datatype:

(tells the Compiler)

→ How much memory space is required.

→ range of value to be stored.

→ types of variables.

Scanned by CamScanner

* Basic Datatype:

→ Integar (15) ⇒ int
→ Float (15.5) ⇒ Float
→ character 'c' ⇒ char
→ string ⇒ "hello

* format specifier:

→ to take any user input

int ⇒ %d
Float ⇒ %f
char ⇒ %c
string ⇒ %s

* variable Declaration:

x=15
//mean 15 is stored in variable x.
int x = 15 to declare the compiler that x is variable which can take only integer type of value.

eg. int x=16, 17, 19. Float y=15.5

int x
int y or int x,y,z
int z

Scanned by CamScanner

* Variable Initialization:

to assign the value to take variable while declaration eg → int X = 20
float Y = 24.5
char Z = 'e'

* token:

→ Every single element in a C-program is token.
→ Smallest unit in a program/statement.
→ It makes the compiler understand what in the program.
→ eg. main, printf, name

* Identifier:

→ So to identify things we have some name given to them.
→ Identifiers are the fundamental building blocks of a program.
→ Used to give names to variables, functions, constant and user defined data.
→ They are user-defined names and consist of an sequence of letters and digits.

* Rules for naming an identifier/variable:

(1) An identifier name is any combination of 1 to 32 alphabet, digits or underscore.
(4) The first character in the identifier name must

Scanned by CamScanner

be an alphabet or underscore.

(iii) No blanks or special symbol other than an underscore can be used in an identifier name.

(iv) keywords are not allowed to be used as identifiers.

* List of Data types:

Type	Size (bytes)	minimum range
char	1	-128 to 127
unsigned char	1	0 to 255
int	4	-32768 to 32767
unsigned int	4	0 to 65535
short int	2	-32768 to 32767
unsigned short int	2	0 to 65535
long int	8	-2147483648 to 2147483647
unsigned int	4	0 to 4294967295
[6 digit precision] float	4	3.4e-38 to 3.4e+38
[15 digit precision] double	8	1.7e-308 to 1.7e+308
[20 digit precision] long double	10	3.4e-4932 to 1.1e+4932

* format specifiers:

Specifies the format accordingly to which the value will be printed or entered in C.

Scanned by CamScanner

eg

→ %d → Signed char
→ %ld → long integer
→ %u → unsigned integer
→ %f → float
→ %i → int.

* Operator:

Operator is the symbol which perform some operation on the operands.

* Arthematic operators:

+ → addition of two operands.
- → Subtraction " "
* → & multiplication " "
/ → Division
% → modulo {gives remainder of two operands}

* Uniary Operator:

Increament operator: increase the value of the operand by 1.

i) Pre- Increment: ++count count+1
ii) Post- increment: count++

Scanned by CamScanner

⇒ Pre-Increment: means it will first increase the value of the variable and then evaluate the expression.

Eg. int a=10, b
b = ++a;
a=? b=?

Ans ⇒ a=11, b=11

Increment

a+1
10+1
a=11

Evalute expression

b = a
b = 11

⇒ Post-Increment:
means first evaluate the expression then increment the value of the variable.

Eg. int a=10, b;
b = a++
a=? b=?

Evaluate Expression

b = a
b = 10

Increment

a+1
10+1
a=11

* Decrement operator: decrement the operator by 1.

Scanned by CamScanner

(ii) Pre-decrement operator : --Count
(iii) Post- " " : Count-- = Count-1

⇒ Pre-decrement operator : first decrement the value of the variable and then evaluate the expression.

eg int a=10, b
b = --a

decrement

a-1;
10-1
a=9

b=a
b=9

⇒ Post-decrement operator : first evaluation of expression is done and then the value of variable is decremented.

eg int a=10, b;
b=a--;

Expression

b=9
b=10

a=1
a=9

Scanned by CamScanner

"Object Oriented Programming

Difference between C and C++

	C - language	C++ language
(i)	Program is divided into function.	Program is divided into objects.
(ii)	POP does not have any access specifier.	OOP's has access specific.
(iii)	It is not secure or not protected	It is secure and protected.
(iv)	POP follows top-down approach	OOP follows bottom-down approach.
(v)	In POP overloading is not possible.	In OOP's overloading is possible in the form of function overloading and operator overloading.

```
#include <stdio.h>
main()
{
   int a,b,c;
Input -> scanf
   s = a+b
output -> printf
```

```
#include<iostream>
using namespace std;
main()
{
int a,b,c;
Input -> cin
s = a+b;
output => cout.
```

Scanned by CamScanner

Input →	Input →
scanf ("%d%d", &a, &b)	cin >> a >> b; or cin >> a; cin >> b;
Out	out
printf("Enter a, b") printf ("Sum")	Cout << "Enter a, b"; Cout << "Sum";
printf("Sum = %d", s)	Cout << s;

Eg

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

Cout << " Enter a, b ";
 cin >> a >> b;
    s = a + b;
Cout << "Sum = " << s;
```

Scanned by CamScanner

C++

→ It is developed by : Bjarne Stroustrup

* Header file : #include <iostream>

↓

It include all other libraries { To include C++ input and output function.

* Using name std;

It is designed to overcome this difficulty and it used as additional information to differentiate similar function, classes, variables etc. with the same name available in different libraries. Eg :

Using namespace std; Cout << "A";	std :: Cout << "A";

* Cout → insertion operator
* Cin → extraction operator

Operator

* Unary Operator :

Eg. num = -6+3 = (-3)

↳ unary minus indicates that value is negative.

Scanned by CamScanner

* Relational operators:

It compares two operands upon their relation Expression generates zero (False) or non-zero (true) value.

<	less than
<=	less than equal
>	greater than
>=	greater than or equal to
==	Equality
!=	unequality

* Logic operators:

→ && → Logical and operators.
eg→ (5>3 && 5<10) value is 1 (true)

→ || → Logical or operators.
eg→ (5>3 || 5<2) value is 1 (true)

→ ! → Logical not operators
eg→ !(8==8) value is 0 (false)

Scanned by CamScanner

Variable Scope:

→ **local variable:** Local variables are declared inside the braces of any function and can be accessed only from there.

→ **global variable:** Global variables are declared outside any function and can be access from any where.

#~~using~~

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

int c = 45;  → global variable

int main(){

    *** Build in Data types ****

int a, b, c;

Cout << "enter the value of a & b";
  Cin >> a >> b;
     c = a+b;
  Cout << " the sum is =" << c << endl;
  Cout << "the global c is << ::c;
```

Scanned by CamScanner

=> *** Float, double and long double literals ***

```
Float d=34.4F;
Long double e = 34.4L;

Cout << "the size of 34.4 is " << sizeof(34.4) <<endl;
Cout << "  "  "  "  34.4F "  << sizeof(34.4F) <<endl;
Cout << "  "  "  "  34.4F" << sizeof(34.4F) <<endl;
Cout << "  "  "  "  34.4L" << sizeof(34.4L) <<endl;
Cout << "  "  "  "  34.4L" << sizeof(34.4L) <<endl;

Cout <<"the value of d is " <<d <<endl <<"the value of
        e is " <<e;
```

=> *** Reference variable ***

eg-> Rohan Das ---> Monty ---> Rohu ---> Dangerous Coder
1

Diff. name of same people. same we can assign in the language.

```
Float x=455;
Float &y = x;

Cout << x <<endl; cout << y << endl;
```

=> ** *** Typecasting ****.

In programming we can change the data types of variable in the middle of program. for eg. if we take x in int but if we want x is float at some place. then we can do this.

Scanned by CamScanner

```
int a=45;
Float b=45.46;
Cout<<"the value of a is "<<(Float)a<<endl;
Cout<<"the value of a is "<<Float(a)<<endl;

Cout<<"the value of b is"<<(int)b<<endl;
Cout<<"the value of b is"<<(int(b)<<endl;
int c = int(b);

Cout<<"the expression is "<<a+b<<endl;
Cout<<"the expression is "<<a+(int b)<<endl;
Cout<<"the expression is "<<a+int(b)<<endl;

return 0;

};
```

1.
```
#include<iostream>
#include<iomanip>

using namespace std;

int main(){

int a=34;
Cout<<"the value of a was: "<<a;
a=45;
Cout<<"the value of a is :"<<a;
```

Scanned by CamScanner

#iomanip -> header file to use setw.

* Constants in C++

```
Const int a=3;
Cout << "the value of a was: "<<a << endl;
a=45  //we will get an error bcoz a is constant
Cout << " the value of a is : "<< a << endl;
```

* manipulators in C++

```
int a=31, b=78, c=1233;

Cout<<" the value of a without setw is :"<<a;
Cout<<"   "   "   "   b   "   "   :"<<b;
Cout<<"   "   "   "   c   "   "   :"<<c;
```

Output:

```
the value of a ---- is : 31
"   "   "   b ---- is : 78
"   "   "   c ---- is : 1233
```

```
Cout << " the value of a is : "<< setw(4) <<a;
Cout << "   "   "   "   b "   :" << setw(4) <<b;
Cout << "   "   "   "   c "   :" << setw(4) <<c;
```

Output:

```
the value of a is :   31
"   "   "   b is :   78
"   "   "   c is : 1233
```

→ It will leave 4 bit space

* operators procedure:

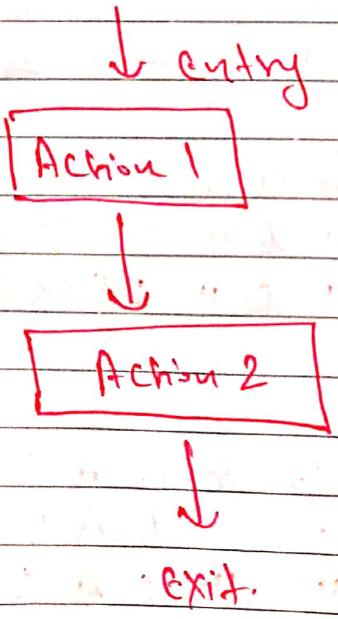
Scanned by CamScanner

```
int a=3; b=4;
// int c = (a*5)+b;
int c = (((((a*5)+b)-45)+87));
Cout << c;
return 0;
}
```

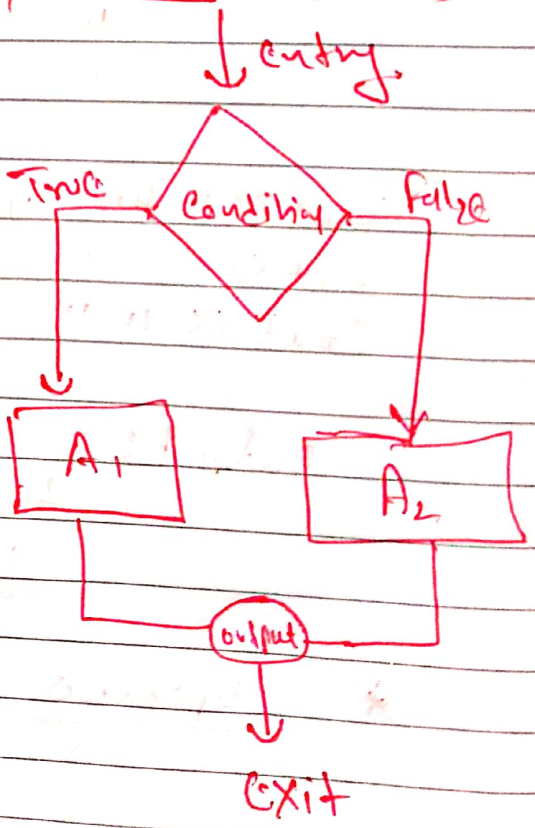
↳ It is the procedure through which compilation will take place.

C++ Control structures

⇒ Sequence structure:

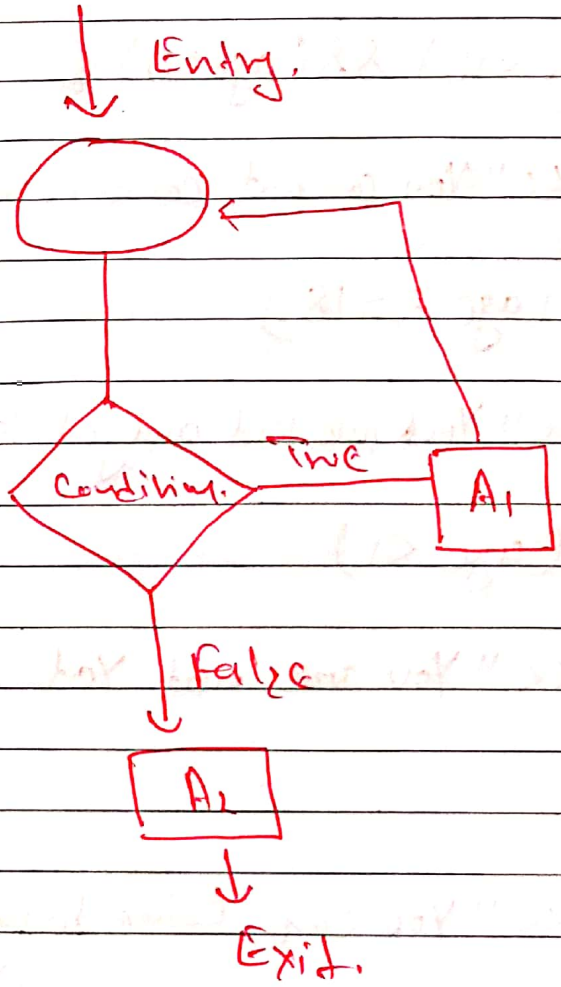


⇒ Selection structure



Scanned by CamScanner

=> Loop structure :



~~#using #include<iostream>~~

#include<iostream>

Using namespace std;

int main()
{
int age;
cout << "tell me Your age" << endl;
cin >> age;

Scanned by CamScanner

1. Selection Control structure: If else- If else ladder

```
if ((age<18) && (age>0)){
  {
  Cout << "You can not come to my party" <<endl;
   }
else if (age == 18)
  {
  Cout << "You are kid and get kid pass for party" <<endl;
   }
else if (age <1)
  {
  Cout << "You are not yet born" <<endl;
   }
   else
  {
  Cout << "You can come to my party" <<endl;
  }
```

2. Selection Control structure : Switch Case statement

```
switch (age)
  {
  Case 18:
    Cout << "You are 18" <<endl;
    break;
  Case 22;
    Cout<< " You are 22" <<endl;
    break;
  default:
  Cout<< "No special cases" <<endl;
        break;
   }
```

Scanned by CamScanner

* Rules of using switch Case:

- → Case label must be unique.
- → Case level must end with colon.
- → Case level must have constant expression.
- → Case label must be of integer, character type like Case 2, Case 1+1, Case 'a'.
- → Case level should not be floating point
- → Default can be placed anywhere in switch
- → Multiple Case cannot use same expression.
- → Relational operators are not allowed in switch.
- → Nesting of switch is allowed.
- → variable are not allowed in switch Case level.

```
#include <iostream>

Using namespace std;
int main()
{
```

Loops in C++:

There are three types of loops in c++:

1. for loop
2. while loop
3. do-while loop.

Scanned by CamScanner

* for loop in c++ :

Syntax :

```
for (initialization; condition; updation)
{
    loop body (c++ code);
}
```

eg

```
for (int i=1; i<=40; i++)
{
    cout <<i<< endl;
}
```

eg of Infinite loop :

```
for (int i=1; 34<=40; i++)
{
    cout <<i<<endl;
}
```

* while loop in c++ :

syntax:

```
while (condition)
{
    c++ statement;
}
```

Scanned by CamScanner

eg-> Printing 1 to 40 using while loop:

```
int i=1;
while (i<=40){
    cout <<i<<endl;
    i++;
}
```

eg of infinite loop:

```
int i=1;
while (true)
{
    cout<<i<<endl;
    i++;
}
```

* do while loop in c++:

Syntax:

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

eg.

```
int i=1;
do{
    Cout<<i<<endl;
    i++;
} while (false);
return 0;
}
```

Scanned by CamScanner

* Jump Statement:

i) Break:

→ It is keyword
→ It allow the program to terminate the loop.
→ A break statement causes control to transfer to the first statement after the loop or block.

ii) Continue:

→ It is exactly opposite to break.
→ Continue statement is used to Continuing the next iteration of the loop statement.

iii) goto:

→ Unconditionally transfer control.
→ goto may be used for transversing control from one place to another.

Syntax:

```
goto identifier.
```

Eg

```
int x;
Cout << " Enter the no";
Cin >> x;
if (x % 2 == 0)
        goto even;
else
        goto odd;
```

Scanned by CamScanner

```
even:
    cout<< x << " is even no";
odd:
    cout<< x << " is odd no";
}.
```

* Return Statement:

→ Exit the function.
→ return exits immediatly from the currently executing function to the calling routine, optionally returning a value. The syntax is:

```
return [expression];
```

for eg.:

```
int sqr (int x)
{
    return (x * x);
}
```

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

int main() {

    for (int i=0; i<40; i++)
    {
        if (i==2) {
            break;
```

Scanned by CamScanner

```
    }
    cout << i << endl;
    }.

for (int i=0; i<40; i++)
    {
        if(i==2)
        {
            continue;
        }
        cout << i << endl;
        }.
```

Function and Recursion

Sometimes our program gets bigger in size and its not possible for a programmer to track which piece of code is doing what.

function is a way to break over code into chunks so that it is not possible for a programmer to reuse them.

* What is function?

A function is a block of code which performs a particular task.

A function can be reused by the programmer in a program any no. of times.

Scanned by CamScanner

Date
Page

eg and syntax of function:

```
#include <iostream>

void display();  => function prototype.

int main(){
    int a;
    display();  => function call
    return 0;
}

void display(){  -> function definition
    cout<<"Hi, I am display";
}
```

* function prototype:

function prototype is a way to tell the compiler about the function we are going to define in the program.
Here void indicates that the function returns nothing.

* function Call:

function Call is a way to tell the compiler to execute the function body at the time the call is made.
Note that the program execution starts from the main function in the sequence the instruction are written.

Scanned by CamScanner

* Function defination:

This part contains the exact set of instruction which are executed during the function call when a function called from main(), the suspended. During this time the control goes to the function being called. When the function body is done executing main() resume.

* Important Points:

→ Execution of a C++ program starts from main().
→ A C++ program can have more than one function.
→ Every function gets called directly or indirectly from main().
→ There are two types of function in C++. Let talk about them.

* Types of function:

1. Library function: Commonly required grouped together in a libary file on disk.
2. User defined function: These are the function declared and defined by the user.

* Why use function?

→ To avoid rewritting the same logic again and ag
→ To keep track of what are doing in a prog
→ To test and check logic independently.

Scanned by CamScanner

* Passing values to functions:

We can pass values to a function and can get a value in return of a function.

```
int sum(int a, int b)
```

The above prototype means that sum is a function which takes values a (of type int) and b (of type int) and return a value of type int.

function defination of sum can be:

```
int sum(int a, int b){
    int c;
    c = a+b;  => a and b are parameters
    return c;
}
```

Now we can call sum (2,3); from main to get 5 in return.
↳ 2 & 3 are arguments

```
int d = sum (2,3);  => d becomes 5
```

Note:

(1) Parameters are the values or variable placeholder in the function defination. eg a & b.

Scanned by CamScanner

(II) Arguments are the actual values passed to the function to make a call. eg -> 2, 33.

(III) A function can return only one value at a time.

(IV) If the passed variable is changed inside the function, the function call does not change the value in the calling function.

```
int change(int a) {
    a=77                    => Mismatch.
    return 0;
}
```

change is a function which changes a to 77. No if we call it from main like this

```
int b=22
change(b);   => The value of b remains 22
cout<<"b is "<<b;  prints b is 22.
```

This happens because a copy of b is passed to the change function.

```
#include<iostream>

using namespace std;
```

Scanned by CamScanner

function Prototype

type function-name (arguments);
int sum(int a, int b); ---> Acceptable
int sum(int a, b); ---> Not-Acceptable.
int sum(int, int); ---> Acceptable.
void g(void); ---> Acceptable
void g(); ---> Acceptable

```
int main(){
int num1, num2;
   cout<<"Enter the 1st num: ";
   cin>>num1;
cout<<"Enter the 2nd num: ";
   cin>>num2;

// num1 & num2 are actual parameters.

cout<<"the sum is "<<sum(num1, num2);
 g();
   return 0;
}

int sum(int a, int b){
// formal parameters a and b will be taking
values from actual parameters num1 and num2
   int c=a+b;
   return c;
}
void g(){
cout<<"\n Hellow, Good morning";
   }
```

Scanned by CamScanner

* Call by value in C++ :

Call by value is a method in C++ to pass the values to the function arguments. In case of call by value the copies of actual parameters are sent to the formal parameters which means that if we change the value inside the function that will not affect the actual values.

* Call by Reference :

Call by reference is a method in C++ to pass the values to the function arguments. In the case of call by reference, the reference of actual parameters is sent to the formal parameters, which means that if we change the value inside the function that will affect the actual values.

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

int sum(int a, int b) {
    int c = a + b;
    return c;
}
```

Scanned by CamScanner

This will not swap a and b.

```
void swap(int a, int b){          // temp  a  b
    int temp = a;                 //  4    4  5
    a = b;                        //  4    5  5
    b = temp;                     //  4    5  4
}
```

// Call by reference using pointers.

```
void swapPointer(int* a, int* b){   // temp  a  b
    int temp = *a;                  //  4    4  5
    *a = *b;                        //  4    5  5
    *b = temp;                      //  4    5  4
}
```

---

* Recursion:

A function defined in C++ can call itself. This is called recursion.
A function calling itself is also called recursive function.

=> Eg of Recursion:

A very good exam of recursion is factorial.

factorial(n) = 1x2x3 --- xn.
factorial(n) = 1x2x3 --- n-1xn
factorial = factorial(n-1)xn

Scanned by CamScanner

Since we can write factorial of a number in terms of itself, we can program it using recursion.

```
int factorial(int x){
    int f;
    if(x==0 || x==1)
        return 1;
    else
        f = x * factorial(x-1)

    return f;
    }
```

⇒ A program to Calculate factorial using recursion.

How does it work:

factorial(5)

5 x factorial(4)

5 x 4 x factorial(3)

5 x 4 x 3 x factorial(2)

5 x 4 x 3 x 2 x factorial(1)

5 x 4 x 3 x 2 x 1.

* Notes

→ Recursion is sometimes the most direct way to code an algorithm.

→ The condition which doesnot call the function further in a recursive function is called as the base condition.

→ Sometimes, due to a mistake made by the programmer a recursive function can keep running without resulting in a memory error.

Scanned by CamScanner

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

int fib(int n){

  if(n<2){
        return 1;
    }
   return fib(n-2) + fib(n-1);
}
```

* fib(5)
  fib(4) + fib(3)
  fib(2) + fib(3) + fib(2) + fib(3)

```
int factorial(int n){
   if (n<=1){
       return 1;
    }
   return n* factorial(n-1);
    }

int main(){

   int a;
    Cout <<"Enter a number"<< endl;
     Cin >>a;
Cout <<" the fachonal of "<<a<<" is "<<factorial(a);
cout <<" the term in fabonacci seq. at position "<<a<<
          "is" << fib(a) <<endl;
       return 0; }
```

Scanned by CamScanner

Pointers

→ A pointer is variable which stores the address of another variable.

Syntax: data_type *ptr_name;

eg→ int var = 10
int *p;
p = &var → address of var.

→ P is the pointer that stores the address of variable var. The data types of pointer p and variable var should match because an integer pointer can only hold the address of integer variable.

int x = 10;
float y = 2.0;
int *ptr = &y; → Invalid {diff, both have diff datatypes}
↓ ↓
int float
int *ptr = &x; → valid

→ Any number of pointers can point to the same address.

eg. int x = 12
int *p1 = &x, *p2 = &x, *p3 = &x;
↓
All the three pointers are pointing towards x.

Scanned by CamScanner

→ Memory taken by any kind of pointer (i.e int, float, char float :-) is always equivalent to the memory taken by unsigned integers, as pointer will always store address of a variable (which is always unsigned integer), so the type of pointer will not make any difference.

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

int main(){
  int a=3;
  int *b;
  b = &a;
      ↳ (Address of) operator.

  cout<<"the address of a is"<<&a<<endl;
  cout<<"  "  "  "  "  "<<b<<endl;

  * ---> (value at) Reference operator.
  cout<<"the value at address b is"<<*b<<endl;

  * ---> pointer to pointer.
  int **c = &b;
    cout<<"the add of b is"<<&b<<endl;
    cout<<"  "  "  "  "  "<<c<<endl;
    cout<<"the value at address of c is"<<*c<<endl;
cout<<"the value at address value_at(value_at(c)) is"<<
                                        **c<<endl;

  return 0;
  }
```

Scanned by CamScanner

Types of Pointer

(I) Null pointer :

A null pointer is a pointer that does not point to any memory location. It is used to initialize a pointer variable when the pointer does not point to a valid memory address.

Note : It is invalid to dereference a null pointer.

eg

```
#include<iostream>
using namespace std;
int main()
{
    int *ptr = NULL
    int a = 10;
    cout << ptr; // 0 will be displayed.
    cout << *ptr; // Invalid (dereferencing) as ptr is
                  null pointer.
    ptr = &a;
    cout << *ptr; // Now it is allowed, as Null pointer
                  has starting pointing somewhere.
    return 0;
}
```

(II) Wild pointer :

→ Pointer which are not initialized during its declaration holding some junk value (or garbage addr) are wild pointer.

→ Every pointer when it is not initialized is declared as wild pointer.

Scanned by CamScanner

→ A pointer get initialized, start pointer to some variable is defined as pointer, not a wild one.

eg

```
#include<iostream>
using namespace std;
int main()
{
    int *ptr; // wild pointer.
    int a=10;
  Cout<<ptr; // gives garbage address value
  Cout<<*ptr; // gives garbage value stored in the garbage address.
   ptr=&a; //.
        └──> Now pointer is not wild pointer.
  Cout<<*ptr;
    return 0;
}
```

(iii) Void pointer:

→ It a pointer that can hold the address of different data types at different time also called generic pointer.
→ Syntax → void *pointer-name;
→ Here, the void is keyword which can point to any value of any data type.
→ But before accessing the value through generic pointer by dereferencing it, it must be properly typecasted.

eg

Scanned by CamScanner

```
#include<iostream>
using namespace std;
    int main()
{
    int x=10;
    char ch = 'A';
    void *gp;
      gp = &x;
  Cout<< *(int*)gp);
     gp = &ch;
  Cout << *(char*)gp);
    return 0;
      }
```

(IV) A Constant Pointer :

→ A Constant pointer, ptr is a pointer that is initialized with an address and cannot point to anything else. But we can use ptr to change the content of variables pointing to

eg;

```
int value = 22;
int *const ptr = &value;
```

(V) Dangling Pointer :

→ It is a type of pointer which point towards such as memory location which is already delocated / or deallocated

→ It is a problem associated with pointer, when a pointer is unnecessarily pointing towards delocated memory location.

Scanned by CamScanner

→ It can be resolved through assigning Null address once, the memory has been deallocated.

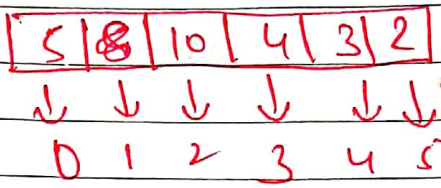
```
#include<iostream>
int main() using namespace std;
 int main() {

    int *ptr;
    {
        int val=23;
          ptr =&val;
        Cout<< *ptr;    // 23 is printed.
        Cout<<ptr; // address of val is printed.
    }

Print Cout<<ptr; //same address is printed, even val is destroyed, hence ptr is dangling pointer.
        ptr = Null; //solution.
    Cout<< ptr; // Now ptr is not a dangelling pointer so address value is printed.

    return 0;
        }.
```

Scanned by CamScanner

Array

→ It is collection of related data items of same data type.

→ The order of list of Array is :



N-Size → Index(n-1).

→ Syntax : Array_name[] = { — }.

X[5] = {1, 2, 3, 4, 5, 6};

⇒ 1-D Array :

↳ It is organized in linear sequence.
↳ Array is called group of consecutive memory location.
↳ First element is always at position zero.

* Way of initializing 1-D array :

(I) int a[5] = {1, 2, 3, 4, 5};
(II) int a[] = {1, 2, 3, 4, 5}; // Here compiler will automatically depict the size : 5
(III) int a[5] = {1, 2, 3}; // Partial initialization (Remaining element will be initialized to default value for integers, i.e 0].
(IV) int a[5] = {}; // All element will be initialized to 0
(V) int a[4] = {1, 2, 3, 4, 5}; // Compiler will give error As 5 element is initialized.

Scanned by CamScanner

Rank → It is used to know about the dimension of passed array.

classmate Date Page

Eg:

```
#include <iostream>
using namespace std;
  int main()
{
   int a[100], n, i;
   Cout << "Enter the element : ";
   Cin >> n;
   Cout << "Entered element are";
    for (i=0; i<n; i++)
      {
          Cin >> a[i];
      }
    Cout << "Entered array are : ");
     for (i=0; i<n; i++)
      {
          Print Cout << a[i];
      }
       return 0;
        }.
```

⇒ 2-1) Array :

Scanned by CamScanner

→ is_array() : It checks that the specified variable is of the array or not.

=> 2-1) Array:

```
int x[10][20];
int x[i][j];
```

int [2][3] = {1, 2, 3, 4, 5, 6}

2×3 = 6 elements

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Condition:

i) int a[][] = {10, 3, 4, 6, 1, 2}

$$\begin{bmatrix} 10 & 3 & 4 \\ 6 & 1 & 2 \end{bmatrix}_{2\times3} \text{ or } \begin{bmatrix} 10 & 3 \\ 4 & 6 \\ 1 & 2 \end{bmatrix}_{3\times2} \text{ or } \begin{bmatrix} 10 & 3 & 4 & 6 \\ 1 & 2 & 0 & 0 \end{bmatrix}$$

As there are many cases so compiler will give error.

ii) int a[3][] = {10, 2, 4, 6, 1, 2, 4}

$$\begin{bmatrix} 10 & 2 \\ 4 & 6 \\ 1 & 2 \end{bmatrix} \text{ or } \begin{bmatrix} 10 & 2 & 4 \\ 6 & 1 & 2 \\ 0 & 0 & 0 \end{bmatrix}$$

Compiler will give error.

iii) int [][2] = {10, 2, 4, 6, 1, 2}

$$\begin{bmatrix} 10 & 2 \\ 4 & 6 \\ 1 & 2 \end{bmatrix}$$ → No error.

Scanned by CamScanner

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

int main(){

int marks[] = {23, 25, 56, 89};

int mathmarks[4]

mathmarks[0] = 227;
mathmarks[1] = 738;
mathmarks[2] = 378;
mathmarks[3] = 578;

Cout<<" These are math marks"<<endl;
Cout<< mathmarks[0]<<endl;
Cout<<      "      [1]<<" ";
Cout<<      "      [2]<<" ";
Cout<<      "      [3]<<" ";

// You can the change value of an assay.
marks[2] = 455;

    Cout<<marks[2]<<endl;

return 0;
}
```

Scanned by CamScanner

* Linear Search:

```
#include<iostream>
using namespace std;
int main()
{
    int a[50];
    int i, loc=-1, key, n;
    Cout<<"Enter the value of n:";
    cin>>n;
    Cout<<"enter the elements")
    for(i=0; i<n; i++)
    {
     cin>>a[i];
     }
Cout<<"enter the element you want to seam:";
    cin>>key;

 // attempt to locate searchkey in an array is.
 for (i=0; i<n; i++)
    {
     if(a[i]==key)
     {
         loc=i;
         break;
     }
     }
 if(loc!=-1)
```

Scanned by CamScanner

```
{ cout << " element found at :" loc +1;
  }
  else
   {
   cout << "element not found, ";
   }
   }.
```

* Binary Search:

Search in dictionary way.

```
#include<iostream>
using namespace std;
  int main()
  {
int a[50]; n, loc = -1, key, beg, last, mid, i;
cout << " Enter the no. of array element";
  cin >> n;
cout << " Enter array elements : ";
  for(i=0; i<n; i++)
  {
     cin >> a[i]
     }
   beg = 0;
     last = n-1;
cout << "Enter integer value to search in sorted array";
     cin >> key;
```

Scanned by CamScanner

```
while(beg<=last) // loop will run untill unless only one element is not remaining.
{
mid=(beg+last)/2; // determine index of middle element.
if(a[mid]==key)
{
loc=mid; //save the location of element
break;
}
else if(a[mid]>key) //middle element is greater than key.
{
last=mid-1; //If middle element is greater than key, we need to consider left sub array.
}
else if(a[mid]<key) //middle element is less than key.
{
beg=mid+1;
}
}
if(loc!=-1)
{
cout<<"element found at"<<loc+1;
}
else
{
cout<<"element not found";
}
return 0;
}
```

Scanned by CamScanner

Sorting in array:

Arranging in accending or desending order.

```
#include<iostream>
using namespace std;
int main()
{
  int A[100], n, t;
  Cout<<"Enter no. of element; :"
  Cin>>n;
  Cout<<"Enter element:";
  for(int i=0; i<n; i++)
  {
     Cin>>A[i];
  }
  for(int i=0; i<n; i++)
  {
    for(int j=0; j<n-1; j++)
    {
      if(A[j]<A[j+1])
      {
        t = A[j];
        A[j] = A[j+1];
        A[j+1] = t;
      }
    }
  }
  Cout<<"Sorted array is :";
  for(int i=0; i<n; i++)
  {
    Count<<A[i]<<"\t";
  }
}
```

Scanned by CamScanner