PF

Programming Fundamentals

(OR)

Introduction To Programming

Q#02 : What is Program?

Ans : A precise sequence of steps to solve a particular problem is called program.

Q#02 : What is Pseudocode?

Ans : Pseudocode is a program or task written in simple words or statements.

Q#03 : Things Required To write a Program?

Ans :
(i) Source Code
(ii) Text Editor
(iii) Compiler.

Q#04 : What is source code?

Ans : The program instructions written by following the rules of any high-level language are known as program source code. The files you create with your editor are called source files and for C++ they typically are named with the extension .cpp or .cp or .C

Q#05 : What is text Editor?

Ans : This is a type of computer program that has an interface, menus, dialogue boxes and different options to write your desired code.

Q#06 : What is Compiler?

Ans : The language translator program that translates the source program into machine code is called compiler. Machine code is called object code.

—— x ——

Date:

*Programming Language Generations

(01) First Generation of Programming Language (1GL).

→ A first generation programming language (1GL) is a grouping of programming languages that are machine level languages used to perform ~~first~~ generation computers. No translator was used to compile or assemble the first generation language. There is string of 0s and 1s was used.

(02) Second Generation of Programming Language (2GL)

→ A second generation programming language (2GL) is also known as an assembly language. A second generation programing language uses alphabets, letters and numbers. e.g. ADD 12, 8. An assembler was used to converts the assembly language statements into machine language.

(03) Third Generation of Programming Language (3GL).

→ A third generation programming language is called high-level programming language such as C/C++, Pascal or Java etc. These languages are near to English. The translator program are used to translate the high-level language into machine language.

(04) Fourth Generation of Programming Language (4GL)

→ The fourth generation programming language is a upgradation of 3GL. It is designed to be closer to natural language than a 3GL. Fourth generation programming languages are often use for accessing databases. e.g.

```
SELECT NAME FROM EMPLOYEES WHERE SALARY > $1000
```

(05) Fifth Generation of Programming Language (5GL).

→ The fifth generation programming language use a visual or graphical development interface to create source language. IBM, Microsoft, Borland etc. are the examples of fifth generation programming language. Visual Studio. NET, JBuilder and NetBeans etc. are the programs used to create source code.

★ Introduction to C-language.

→ C is a high-level programming language. It was created by Dennis Ritchie at Bell labs in 1972. It was derived from an earlier programming language named B. B-language was developed by Ken Thompson in 1970.

★ Levels of Programming Language.

There are 3 levels of programming languages

(i) High-level Language
(ii) Middle-level Language
(iii) Low-level Language.

★ Uses of C/C++ Programming Language.

→ Areas where C/C++ language can be used are as follow:

- Operating System
- Network Drivers
- Communication Packages
- Databases
- Language Interpreters
- Utilities
- Language Compilers
- Spreadsheets and Text Editors etc.

★ Introduction to C++ Language.

→ C++ is a high-level programming language developed by Bjarne stroustrup at Bell Labs. C++ adds object oriented features to its predecessor, C. C++ is one of the most popular programming language for graphical applications, such as those that run in Windows and Macintosh environments.

★ C/C++ Character Set

→ C/C++ character set comprises of following characters.

- A, B, C, ..., Z (Capital letters).
- a, b, c, ..., z (small letters).
- 0, 1, 2, ..., 9 (Numbers).
- , . ; : ? ! " / ' \ | ~ (characters).
- ( ) [ ] { } < > (Brackets)
- + - # % _ ^ = & * (Operators).

★ White Space Chracters

→ The character that produces blank spaces when printed is called a white space character, e.g. spaces, tabs and new lines.

★ Tokens

→ A group of characters separated by white spaces is known as tokens.

- Keywords
- Identifiers
- Constants
- Variable
- Functions
- String literals
- Operators

★ Constants & Variables

→ Constants (1, 56, 1.89, 'a', 'Z', "BSCS").

→ Variables (X = 5, X = 20).

★ Data Types & Sizes

→ There are three main data types in C/C++ language.

(i) Character (char)
(ii) Integer (int)
(iii) Floating Point (float).

(i) Character Data Type

→ It is represented by "char".
→ It is used for storing a character, digit or special character.
→ A character constant must be enclosed in single quotations i.e. 'A', '2' or '*' etc.
→ It occupies one byte (8 bits) of memory.
→ Character constant can be signed or unsigned.
→ Character data type has 3 types, i.e. char, signed char, unsigned char, e.g.
→ The value range of binary numbers in:
- char -128 to 127 or 0 to 255
- signed char is from -128 to 127
- unsigned char is from 0 to 255.

(ii) Integer Data Type

→ Integer data type is represented by "int".
→ It is used for storing integers, i.e. numeric values without decimal portions.
→ The range of integer data type is from -32,768 to 32,767.
→ It takes 2 or 4 bytes of memory.
→ Integer data type is also represented as
- short.
- Long.

★

Type	Storage Size	Value Range
int	2 or 4 bytes	-32,768 to 32,767 (OR) -2,147,483,648 to 2,147,483,647
short	2 bytes	-32,768 to 32,767
long	4 bytes	-2,147,483,648 to 2,147,483,647

→ There are 9 types of integer data types, i.e.
int, signed int, unsigned int.
short, short signed int, short unsigned int.
long, long signed int, long unsigned int.

★ Float Data Type

→ Float data type is represented by "float".
→ It is used for storing numeric values.
→ It also store fraction or decimal portion.
→ Float data type takes 4 or 8 bytes of memory.
→ A floating point number is expressed in scientific notation.
→ The reason of storing float values in scientific notation is that they can be very large or extremely small. i.e.
- 20000000000 = 2e+10
- 0.000000023 = 2.3e-8

→ A value written as 47e3 means 47×10^3
→ Exponent value ranges from -38 to 38.
→ Another float data type is double that takes 8 bytes of memory.
→ Exponent values in double ranges from -308 to 308...

★ Other Data Types.

→ String is a data type that contains sequence of characters.
→ Boolean is a data type that gives value in True or False.

★ Introduction To C++

→ C++ is a statically typed, compiled, general-purpose, case-sensitive, free-form programming language that supports procedural, object-oriented.

→ C++ is regarded as a middle-level language, as it comprises a combition of both high-level and low level language features.

★ Object-Oriented Programming.

→ C++ fully supports object-oriented programming, including the four pillars of object-oriented development.

- Encapsulation (protective layer that protects data
- Data hiding from outward).
- Inheritance (characteristic of parents into child).
- Polymorphism (shape).

★ Standard libraries

→ Standard libraries are building blocks including variables, data types and literals etc. Standard libraries are rich set of functions manipulating files, strings etc. It is also rich set of methods manipulating data structures etc.

★ Rules For Naming Variables in C/C++

→ The following are the rules for naming variables in C/C++.

- A variable name may consist of letters, digits and underscore (_).
- The first character of a variable name must be a letter or an underscore. Digit cannot be the first character.
- Special characters, such as arithmetic operators, ^, #, ~ etc. (except underscore) cannot be used in a variable name.
- The keywords and function names cannot be used as variable name.

- The blank spaces cannot be used in variable names.
- The max. length of a variable name is up to 40 characters. But it varies from compiler to compiler.
- A variable name declared for one data type, the same name cannot be used for another data type.
- Single underscore cannot be used as variable name.
- Both uppercase and lowercase letters can be used for naming variables.
- C/C++ is a case sensitive language. Thus, variable names with same spellings but different cases are treated as different variable names. For example, variables "ABC", "abc" & "Abc" are three different variables.

★ Types of C/C++ Instructions

→ There are 4 types of C/C++ instructions.

(i) **Declaration Instructions**
   → Variable types and definations etc.

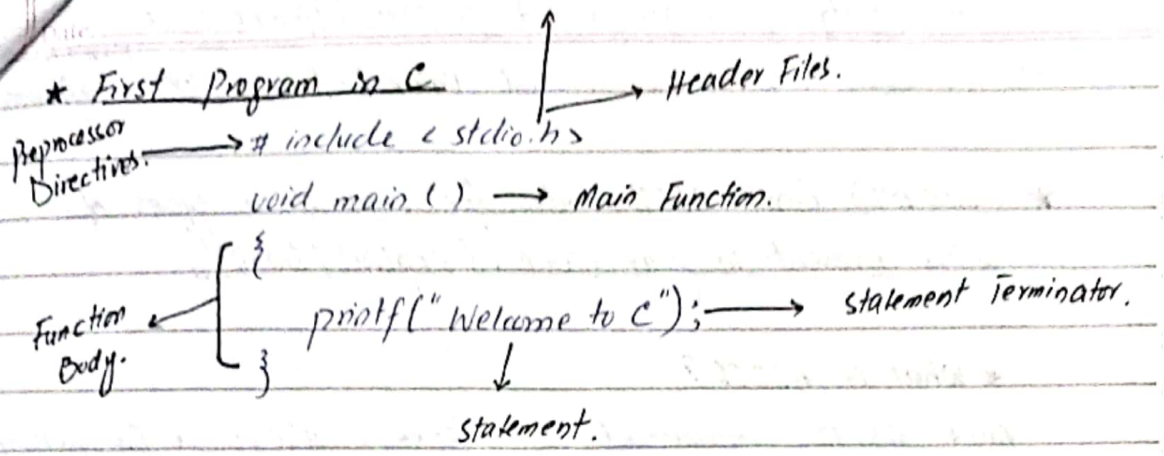
(ii) **Input/Output Instructions**
   → Data Input, Data output/Display, Data write etc.

(iii) **Control Instructions**
   → controls the sequence of execution of the program instructions.

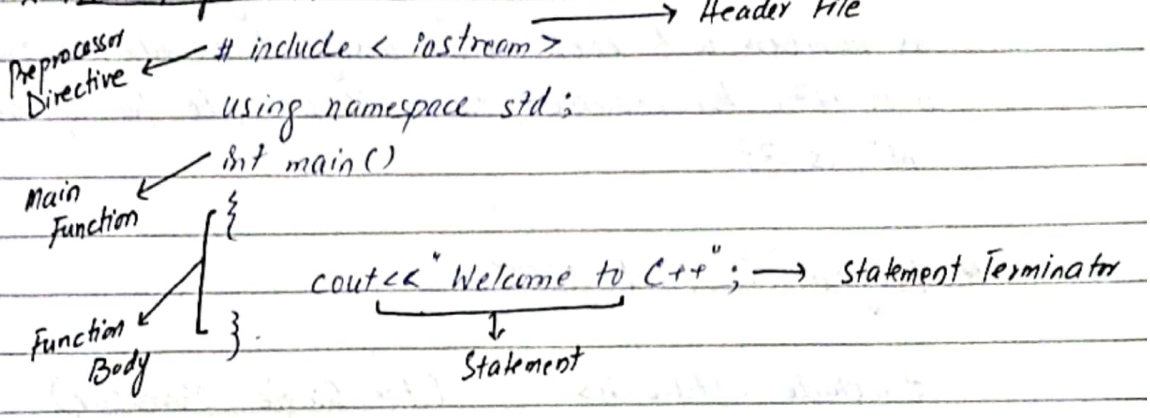
(iv) **Arithmetic Instructions**
   → Arithmetic Operations etc.

Header File Extension (.h)

★ First Program in C



★ First Program in C++



(include)

→ # is called Preprocessor Directives. (The commands that give instructions to the compiler preprocessor directives). There are two types of preprocessor directives.

① The "include".

(The "include" preprocessor directive is used to include the specified header file in the program.)

② The "define"

(The "define" preprocessor directive is used to define a constant known as (constant macro) in the program.

→ void main () (OR) int main () is called Main Function. Main function control the whole program. Without main function the program will not start and execute.

→ int a; (OR) float b; is called Declaration of variable.

→ <iostream> (OR) <stdio.h> are called header files. The header files contain the declaration or definations of standard library functions. The functions are called in the main body of the program to perform different task.

→ a = 5 (OR) b = 10 is called Value Defination (OR) Variable Defination.

★ (Task #1) Find ASCII codes for char data types of letters present in your name. (Capital / Small).

★ What is ASCII ?

Ans: ASCII pronounced as ask-ee. ASCII is the acronym of American Standard code for Information Interchange. It is a code for representing 128 English characters as numbers with each letter assigned a number from 0 to 127. For example, the ASCII code for uppercase "M" is 77.

★ Program To Find ASCII.

```
#include <stdio.h>          (For Single Character)
int main()
{
    char c;
    printf("Enter a character: ");
    scanf("%c", &c);
    printf("ASCII value of %c = %d", c, c);
    return 0;
}
```

(%d displays the integer value of a character
%c displays the actual character).

(First Program in C/C++)

★ Using Comments in The Program. (comments are not shown in output)

→ There are two ways of using coments in C++.

(i) // (ii) /* and */ For example

```
// It is a C Program  &  /* main function */.
```

```
#include <stdio.h>          (For String).
#include <stdlib.h>
#include <conio.h>
int main (int argc, char* argv[])
{
    char str[100];
    int i=0;
    printf("Enter any string to get ASCII value of
            each character\n");
    Scanf("%s", &str);
    printf("ASCII values of each characters of
            given string:\n");
    while (str[i])
        printf("%d\n", str[i++]);
    getch();
}
```

★ ASCII Code of My Name.

humair (small letters)	HUMAIR (capital letters).
h = 104	H = 72
u = 117	U = 85
m = 109	M = 77
a = 97	A = 65
i = 105	I = 73
r = 114	R = 82
humair = 646.	HUMAIR = 454.

* What are bugs?

Ans:- Errors in the program are called bugs.

* What is debugging?

Ans:- The process of finding and removing bugs (errors) in a program is called debugging. In this process, the program is thoroughly checked for errors. Then errors are pointed out and debugged.

* Types of Errors.

→ There are three types of errors which can occur during the program or execution of a program.

(i) Syntax Error

(A syntax error occurs when the program violates one or more gramatical rules of the programming language. These errors are detected at compile time, i.e., when the translator (compiler or interpreter) attempts to translate the program).

(ii) Runtime Error

(The error occur during the execution of the program is called runtime error. A runtime error occurs when the computer is directed to perform an illegal operation by the program such as dividing a number by zero. Runtime errors are the only errors which are displayed immediately during the execution of the program. When a runtime error occurs, the computer stops the execution of program and displays an error message.)

(iii) Logical Error

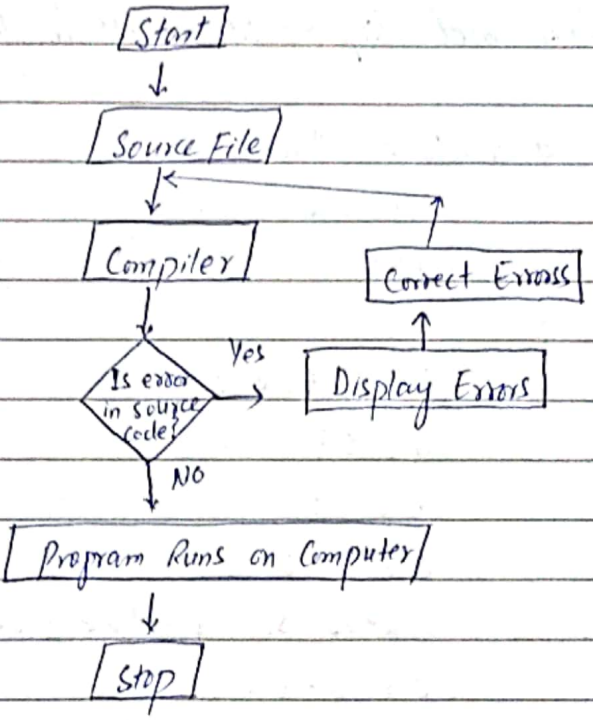
(The errors in the logic of the program are called logical errors. The logical error occurs when a program implements a wrong logic. The translator (compiler or interpreter) does not report any error message for a logical error. These errors are the most difficult to locate)

★ What is flowchart?

Ans:- [redacted]

The flowchart is a pictorial representation of a program which helps in understanding the flow of control and data in the algorithm. For example,

(Flow Chart)



★ What is an Algorithm?

Ans:- An Algorithm is a finite set of steps which, if allowed, accomplish a particular task. An algorithm must be clear, finite and effective.

* Write a Program to display the sum of two integer numbers.

```
#include <iostream>
using namespace std;
int main()
{
    int a, b, sum;
    a = 5;
    b = 10;
    sum = a+b;
    cout<<" Sum is: "<< sum;
}
```

---

* Write a program to sum, subtract, multiply and divide two integers and display its result in third integer number.

```
#include <iostream>
using namespace std;
int main()
{
    int a, b, c;
    a = 5;
    b = 10;
    c = a+b;
    cout<<" Sum is ="<< c << endl;
    c = a-b;
    cout<<" a-b = "<< c << endl;
    c = a*b;
    cout<<" a x b = "<< c << endl;
    c = a/b;
    cout<<" a/b = "<< c << endl;
}
```

→ The blank screen is called console screen.

→ #include <iostream>
#include <conio.h> (Header Files)
#include <stdio.h>
<stdio.h> (standard Input/output)
<conio.h> (console Input/Output).

→ clrscr(); (Functions)
getch();

★ Write a program that will have 3 variables, first, Second and Third variable. First variable will be of character data type having character constant value equal to 65. The second variable will be of integer data type having value 964. Third variable will be of float data type having value 5.543. Display the variables and their values.

```
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
    char first = '65';
    int Second = 964;
    float third = 5.543;
    cout << first << endl << Second << endl << third;
    getch();
}
```

★ Write a program using escape sequence, that has output as follow: (Hello
How
Are
You
Hello
How
Are
You.)

```
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
    clrscr();
    cout<<"Hello\n How\n Are\n You\n";
    cout<<"Hello\n\t How\n\t\tAre\n\t\t\t You";
    getch();
}
```

→ cout<<endl : Inserts a new line and flushes the stream.

→ cout<<"\n" : only inserts a new line.

→ cout<<"\n" seems performance wise better than cout<<endl; unless flushing of stream is required.

→ Escape Sequences.

* Write a program that will perform Temperature conversion. It will take the temperature in Fahrenhiet from the user and convert it into the celsius. (Celsius = (F-32)* 5/9.)

```
#include <iostream>
using namespace std;
int main()
{
    float fah, cel;
    cout<<" Enter temperature in Fahrenhiet";
    cin>> fah;
    cel = (fah-32)* 5/9;
    cout<<" Temperature in celsius is: "<< cel;
}
```

* Write a program that will take value of radius from the user and calculate the area of a circle. (Area = πr^2).

```
#include <iostream>
using namespace std;
int main()
{
    const float pie = 3.14;
    float r, area;
    cout<<"Enter value of radius";
    cin>> r;
    area = pie*r*r;
    cout<<" Area of Circle is :"<< area;
}
```

* Write a program that will take height and weight from user and calculate the BMI. ($BMI = \frac{weight \times 703}{(height)^2}$)

```
#include<iostream>
using namespace std;
int main()
{
    float w, h, bmi;
    cout<<"Enter Weight";
    cin>>w;
    cout<<"Enter Height";
    cin>>h;
    bmi = (w*703)/(h*h);
    cout<<"BMI is:"<<bmi;
}
```

* Write a program that takes the marks of 5 subjects from user and display their average ($Average = \frac{Sum\ of\ All\ Sub.\ mark}{No.\ of\ Subjects}$)

```
#include<iostream>
using namespace std;
int main()
{
    int Eng, math, Urdu, Phy, Che;
    float avg;
    cout<<"Enter marks of English";
    cin>>Eng;
    cout<<"Enter marks of Mathematics";
    cin>>math;
    cout<<"Enter marks of Urdu";
    cin>>Urdu;
    cout<<"Enter marks of Physics";
    cin>>Phy;
    cout<<"Enter marks of Chemistry";
    cin>>Che;
    avg = (Eng+math+Urdu+Phy+Che)/5;
    cout<<"Average is"<<avg;
}
```

* Write a program that will take salary from user and increment the annual salary by adding 5000 in it.

```
#include <iostream>
using namespace std;
int main()
{
    int s, annual_salary, incr;
    cout<<"Enter Salary";
    cin>>s;
    annual_salary = s*12;
    incr = annual_salary + 5000;
    cout<<"Increment is :"<<incr;
}
```

* Write a program That convert the following equation into C++ statement. $\left(P = F\left[\frac{r}{(1+r)^n - 1}\right]\left[\frac{1}{1+r}\right] \right)$

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
    float r, F, P, S, D, E;
    cout<<"Enter value of r";
    cin>>r;
    cout<<"Enter value of F";
    cin>>F;
    E = pow(1+r, 5);
    S = r/E-1;
    D = 1/1+r;
    P = F*S*D;
    cout<<"Value of P ="<<P;
}
```

Date:

★ Write a program that will convert the following statement /equation into C++ statement. Payment = $\frac{PR}{2-(1+R)^{-N}}$

```
#include <iostream>
#include <math.h>
using namespace std;
int main ()
{
    float P, R, E, payment;
    cout<<"Enter value of P";
    cin>>P;
    cout<<"Enter value of R";
    cin>>R;
    E = pow(1+R, -4);
    payment = (P*R)/1-E;
    cout<<"Payment is:"<<payment;
}
```

★Const Qualifier

→ The qualifier const variable is used to indicate that its value will not be changed.

→ A const must be initialised with some value.

★ Type Conversion

→ Type conversion is basically a way of changing an expression of a given data type into another data type.

There are two types of type conversion:

ⓘ Implicit Type Conversion.

ⓘⓘ Explicit Type Conversion.

ⓘ Implicit Type Conversion

→ The type conversion done by the compiler is called implicit type conversion.

ⓘⓘ Explicit Type Conversion

→ The type conversion done by the user is called explicit type conversion. It is also called type casting.

★ Cast

→ Cast is a way through which we change the type of the variable during the execution of the program for a limited time. Because variables previously defined type can not calculate the values correctly due to its low range.

(Type Conversion)

```
#include <iostream.h>
#include <conio.h>
void main()
{
    clrscr();
    int count = 7;
    float weight = 200.5;
    double totalweight = count * weight;
    cout << "Total weight calculated is" << totalweight;
    getch();
}
```

Output: Total weight calculated is 1403.5

Date:

```
#include <iostream.h>
using namespace std;
void main()
{
    int x = 10;
    char y = 'a';      // a=97
    x = x+y;
    float z = x + 1.0;
    cout << "x = " << x << endl << "y = " << y << endl << "z = " << z << endl;
    return 0;
}
```

(Implicit Type Conversion.)

(Output : x=107
y = 'a'
z = 108)

```
#include <iostream.h>
using namespace std;
void main()
{
    double x = 1.2;
    int sum = (int) x + 1;
    cout << "Sum = " << sum;
    return 0;
}
```

(Explicit Type Conversion.)

(Output : Sum = 2)

```
#include <iostream.h>
using namespace std;
int main()
{
    int test = 25000;
    test = (test*10)/10;
    cout << "Result is " << test << endl;
    test = 25000;
    test = (long(test)*10)/10;
    cout << "Result now is " << test;
}
```

(Cast)

(Output : Result is 25000
Result now is 25000)

Date:

★ Arithmetic Operators

→ Following are the basic Arithmetic operators used in C/C++.

(i) + (Addition)
(ii) - (Subtraction)
(iii) * (Multiplication)
(iv) / (Division).

→ Apart from the specified basic operators, there are some other operators used in C/C++. These are as follow:

(v) % (Remainder or Modulus)
(vi) ++ (Increment)
(vii) -- (Decrement)
(viii) += (Increment Assignment)
(ix) -= (Decrement Assignment)
(x) *= (Multiplication Assignment)
(xi) /= (Division Assignment).
(xii) %= (Remainder Assignment).

→ Increment and Decrement operators can be used in two ways, i.e.

(i) Prefix (++ variable, -- variable)
(ii) Postfix (variable ++, variable --)

(Basic Operators)

```
#include <iostream.h>
#include <conio.h>
void main()
{
   clrscr();
   int a=5, b=2;
   cout<<"A=5 and B=2"<<endl<<endl;
   cout<<a<<"+"<<b<<"="<<a+b<<endl;
   cout<<a<<"-"<<b<<"="<<a-b<<endl;
   cout<<a<<"*"<<b<<"="<<a*b<<endl;
   cout<<a<<"/"<<b<<"="<<a/b<<endl;
   cout<<a<<"%"<<b<<"="<<a%b;
   getch();
}
```

(Other Operators)

```
#include <iostream>
#include <conio.h>
void main()
{
    clrscr();
    int a=5, b=2;
    a+=b;
    cout<<"a+=b means value of a is"<<a<<endl;
    a=5, b=2;
    a-=b;
    cout<<"a-=b means value of a is"<<a<<endl;
    a=5, b=2;
    a*=b;
    cout<<"a*=b means value of a is"<<a<<endl;
    a=5, b=2;
    a/=b;
    cout<<"a/=b means value of a is"<<a<<endl;
    a=5, b=2;
    a%=b;
    cout<<a%=b means value of a is"<<a<<endl;
    getch();
}
```

(Prefix Increment)

```
int x=5;
int y=++x;
// x is now equal to 6, and 6 is assigned to y.
```

(Postfix Increment)

```
int x=5;
int y=x++;
// 5 is assigned to y, and x is now equal
   to 6.
```

Date:

```
#include<iostream.h>
#include<conio.h>
void main()
{
   int x=5, y=5;
   cout<<x<<" "<<y<<'\n';
                  // Prefix
   cout<<++x<<" "<<--y<<'\n';
   cout<<x<<" "<<y<<'\n';
                  // Postfix
   cout<<x++<<" "<<y--<<'\n';
   cout<<x<<" "<<y<<'\n';
   getch();
}
```

(Prefix & Postfix Increment & Decrement)

(Output

```
5 5
6 4
6 4
6 4
7 3 )
```

★ Relational Operators

→ A relational operator compares two values. Comparison involved in relation operators can be

(i) < Less than
(ii) > Greater than
(iii) == Equals to
(iv) != Not Equals
(v) <= less than or equals
(vi) >= Greater than or equals

→ if result = 1, means True.
→ if result = 0, means false.

(Relational Operators)

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int number;
    cout<<"Enter a number";
    cin>>number;
    cout<<"number<10 = "<<(number<10)<<endl;
    cout<<"number>10 = "<<(number>10)<<endl;
    cout<<"number==10 = "<<(number==10)<<endl;
    getch();
}
```

(Output:
Enter a number 10.
number<10 = 0
number>10 = 0
number==10 = 1)

(Using Library Functions)

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    clrscr();
    int a;
    cout<<"Enter a number";
    cin>>a;
    cout<<"square Root of "<<a<<" is "<<sqrt(a);
    getch();
}
```

Statements

Date: / /20

Decision Making "Statements"

In decision making statements, the instructions of the program are written in a sequence but statements are used for making decisions. The given condition is tested. If given condition is true then statements that follow the structure, are executed. Otherwise these are ignored and alternative path is followed. There are three types of statements in C/C++.

(i) if statement
(ii) if-else-if statement
(iii) switch statement.

(i) The "if" Statement (OR) "if-else" statement

The "if" statement is the simplest form of statements for decision making. It executes one or more statements when a given condition is true. If condition is false then those statements are ignored and are not executed.

Syntax of "if" statement / "if-else" statement

(if)

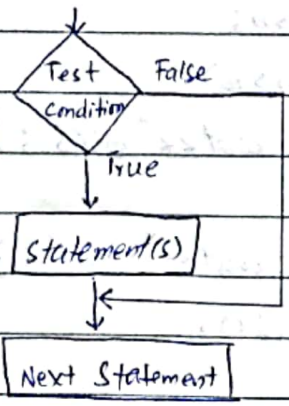
```
if (condition)
    statement-1;
    statement-2;
```

(OR) (if-else)

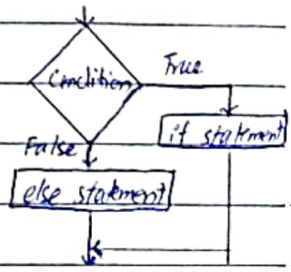
```
if (condition)
{ statement-1; }
else
{ statement-2; }
```

Flowchart of "if" statement / "if-else" statement

(if)



(OR)



(if-else)

Date: __/__/20

★ Write a program that takes a number and check the number if the number is less than 20 then display the message that number is less than 20. otherwise display the value of number.

(if)

```
#include<iostream.h>
void main()
{
    int a;
    cout<<"Enter a number";
    cin>>a;
    if(a<20)
    {
        cout<<"a is less than 20"<<endl;
    }
    cout<<"Value of a is: "<<a<<endl;
}
```

★ Write a program that inputs a number from user, if the number is >=10 then it display "BSCS" otherwise it display "mscs".

(if-else)

```
#include<iostream.h>
#include<conio.h>
void main()
{
    int n;
    cout<<"Enter a number";
    cin>>n;
    if(n>=10)
    { cout<<"BSCS"; }
    else
    { cout<<"mscs"; }
    getch();
}
```

Date: __/__/20__

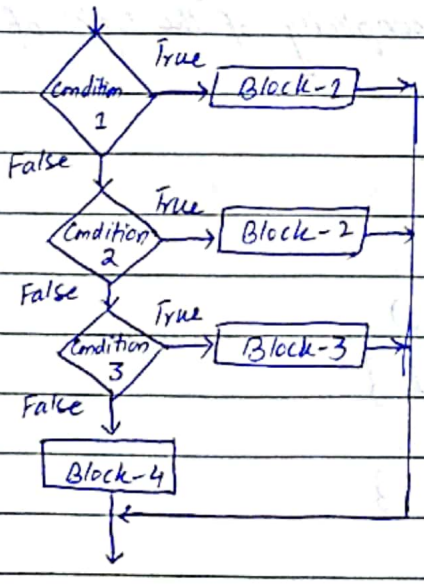
(ii) The "if-else-if" Statement

The "if-else-if" statement is also called multiple "if-else" statement. It is used to execute one block of statement from multiple blocks of statements. In this statement, multiple conditions and multiple block of statements are given. When any given condition is true, the statements associated with condition are executed and all other block of statements are ignored.

Syntax of "if-else-if" statement

```
if (condition-1)
{
    Block-1 }
else if (condion-2)
{
    Block-2 }
else if (condion-3)
{
    Block-3 }
else
{ Block-4 }
```

Flowchart of "if-else-if" statement



Date: / /20

☆ Write a program that inputs a number and finds whether a number is positive, negative or zero using "if-else-if" statement.

```
#include <iostream.h>
#include <conio.h>
void main()
{
    int n;
    cout<<"Enter a number";
    cin>>n;
    if (n>0)
    { cout<<"The number is positive";}
    else if(n<0)
    { cout<<"The number is negative";}
    else
    { cout<<"The number is zero";}
    getch();
}
```

★ The Nested "if" Statement

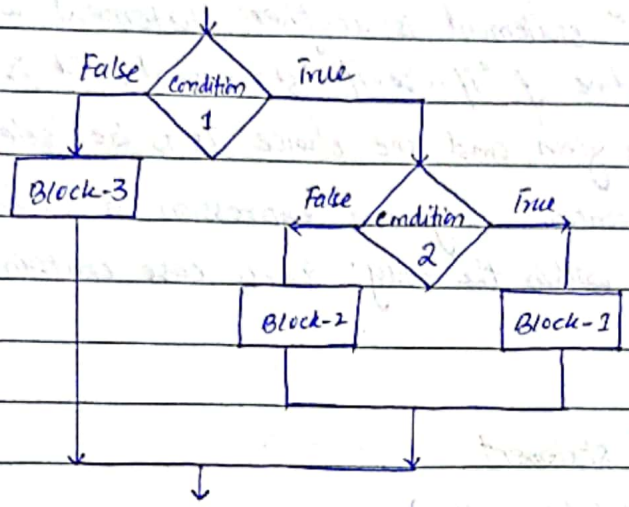
An "if" statement within another "if" statement is called nested "if" statement. The nested "if" statements are used when multiple conditions have to check. Nesting can be up to any level. However, it increases the complicity of the logic of program.

Syntax of "Nested if" statement

```
if (condition-1)
{
    if (condim-2)
    { Block-1 }
    else
    { Block-2}
}
else
{ Block-3}.
```

Date: __/__/20

Flowchart of "Nested-if" statement



★ Write a program that inputs three numbers and displays the largest number using nested "if" structure.

```
#include <iostream.h>
#include <conio.h>
void main()
{
    float x, y, z, max;
    cout << "Enter First number";
    cin >> x;
    cout << "Enter Second number";
    cin >> y;
    cout << "Enter Third number";
    cin >> z;
    if (x > y)
    {   if (x > z)
        { max = x; }
        else
        { max = z; }
    }
    else
    {   if (y > z)
        { max = y; }
        else
        { max = z; }
    }
    cout << "Maximum number is:" << max;
}   getch();
```

Date: __/__/20

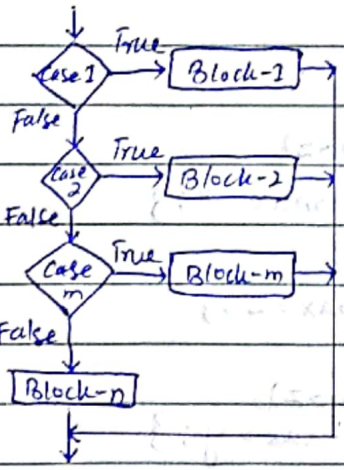
(iii) The switch statement.

The "switch" statement is another statement which can be used as alternative of "if-else-if" statement. It is used when multiple choices are given and one choice is to be selected. The "switch" statement contains only one expression at its beginning and multiple cases within the body. Each case contain set of statements.

Syntax of "switch" statement

```
switch (expression)
{
    case 1:
        set of statements-1
        break;
    case 2:
        set of statements-2
        break;
    .....
    .....
    default:
        set of statements-n
}.
```

Flowchart of "switch" statement



Date: __/__/20__

* Write a program that inputs a character and check whether if the character is "b" then display "BSCS" and if the character is "m" then display "MCS" otherwise display "BCS and MCS".

```
#include<iostream.h>
#include<conio.h>
void main()
{
    char ch;
    cout<<"Enter a character";
    cin>>ch;
    switch(ch)
    {
        case 'b':
            cout<<"BSCS"<<endl;
            break;
        case 'm':
            cout<<"MCS"<<endl;
            break;
        default:
            cout<<"BSCS and MCS";
    }
    getch();
}
```

* Write a program that inputs a character and check it is vowel or consonant.

```
#include<iostream.h>
#include<conio.h>
void main()
{
    char ch;
    cout<<"Enter a character";
    cin>>ch;
    switch(ch)
    {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':
            cout<<"It is a vowel";
            break;
        default:
            cout<<"It is a consonant";
    }
    getch();
}
```

* **The "break" statement**

The "break" statement is used to exit from the body of the statement or loop structure. It is also used to terminate the loop. Whenever, we use "break" statement the compiler is exit from statement or loop to the main function.

```
break;
```

Date: __/__/20

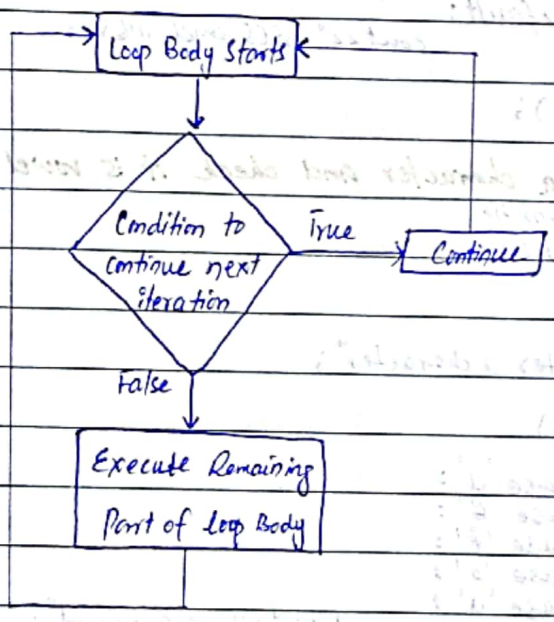
★ "Continue" Statement

"Continue" is also a loop control statement. It forces the loop to execute the next iteration of the loop. When the "continue" statement is executed in the loop, the code inside the loop following the "continue" will be skipped and next iteration of the loop will begin.

Syntax of "continue" statement

```
loop
{
    continue;
    (Body of Loop)
}.
```

Flowchart of "continue" statement



★ Write a program that displays numbers from 1 to 10 by skipping number 6 using "continue" statement.

```
#include<iostream.h>
#include<conio.h>
void main()
{
    int n;
    for(n=1; n<=10; n++)
    {
        if(n==6)
        continue;
        cout<<n<<" ";
    }
    cout<<endl;
    getch();
}.
```

Loops

Date:

* Loops +

→ Loops cause a section of program to be repeated certain number of lines. In computer programming, a loop is a sequence of instructions that is continuously repeated until a certain condition is reached.

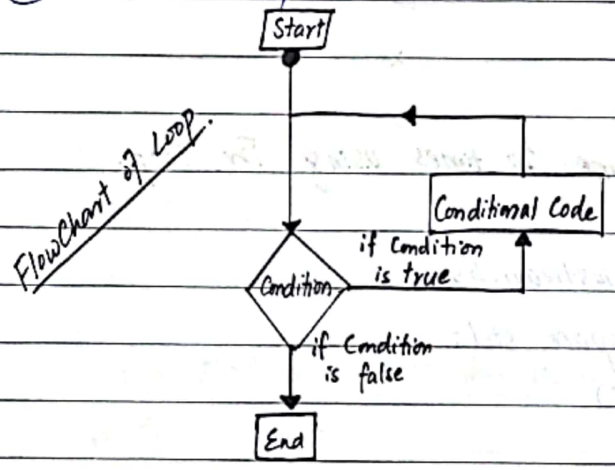
→ As long as condition remains true, the repitition continues, when the condition becomes false, the loop ends and the control passes to the statement following the loop.

→ There are three kinds of loops in C/C++.

(i) For Loop.

(ii) While Loop.

(iii) do-while Loop.



(i) For Loop:

→ The "For Loop" executes a section of code for a fixed number of times.

→ "For Loop" is normally used when we know, (before entering the loop), that how many times we want to execute the code.

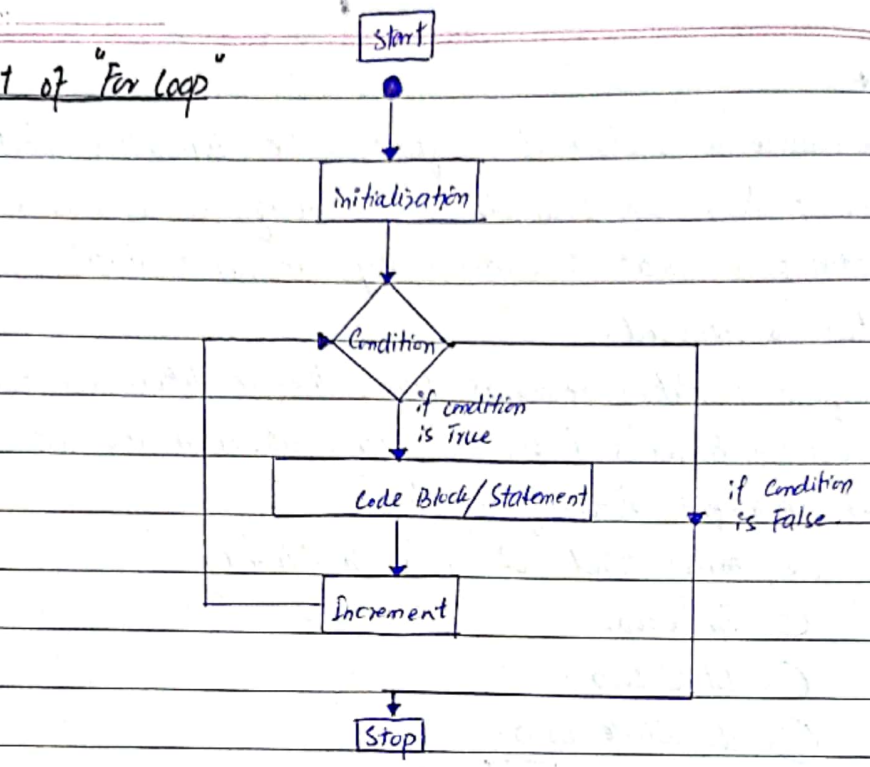
Syntax of "For Loop"

```
for ( initialization; condition; increment)
{
    statement(s);
}
```

2901

Date:

Flow Chart of "For Loop"



★ Display your name 20 times using For Loop.

```
# include < iostream.h >
using namespace std;
void main()
{
    for ( int a=1 ; a<=10 ; a++)
    {
        cout<< " My name is Humair Shoukat \n " ;
    }
}.
```

★ Print Numbers from 1 to 10 using For Loop.

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

* Write a program that display numbers from 1 to 10 in integer a and from a tab space display (11-a) and then display the sum of numbers from 1-10.

```
#include <iostream.h>
#include <conio.h>
using namespace std;
void main()
{
    clrscr();
    int sum = 0;
    for (int a=1; a<=10; a++)
    {
        cout<< a << "\t" << (11-a) << "\n";
        sum+ = a;
    }
    cout<< "Sum is " << sum;
    getch();
}
```

* Write a program that will take the value from user and display its table.

```
#include <iostream.h>
#include <conio.h>
void main()
{
    int n;
    cout<<" Enter A Number ";
    cin>> n;
    for( int i=1 ; i<=10 ; i++)
    {
        cout<<n<<" * "<< i<<" = "<< n*i<<"\n";
    }
    getch();
}
```

★ Sum of numbers using "for Loop".

```
#include <iostream.h>
using namespace std;
void main()
{
    int sum = 0;
    int num = 1;
    for (int i=1 ; i<=5 ; i++)
    {
        sum+ = num;
        cout<<"Sum is" << sum<<"\n";
    }
    cout<<"Result is" << sum;
}
```

★ Dry Run

sum = 0	num = 1	i = 2 (initialization)	i <= 5 (condition)	sum = sum+num (statement)	Sum is	i++
0	1	1	1<=5 (True)	0+1 = 1	1	2
		2	2<=5 (True)	1+1 = 2	2	3
		3	3<=5 (True)	2+1 = 3	3	4
		4	4<=5 (True)	3+1 = 4	4	5
		5	5<=5 (True)	4+1 = 5	5	6
		6	6<=5 (False) (condion Fake)			

(Result is 5.)

Date:

* Write a program that will take a number from user and display its Factorial!

```
#include <iostream.h>
#include <conio.h>
void main()
{
    int n, fact = 1;
    cout << "Enter a Number";
    cin >> n;
    for( int a = n ; a > 0 ; a-- )
    {
        fact *= a;
    }
    cout << " Factorial of " << n << " is: " << fact;
    getch();
}
```

TASK

(A) Vary the control variable from 100 down to 1 in increments of -1.

```
for( int i = 100 ; i >= 1 ; i-- )
{
    cout << i << "\n";
```

(B) Vary the control variable from 7 to 77 in steps of 7.

```
for( int i = 7 ; i <= 77 ; i+7 )
{
    cout << i << "\n";
}
```

Syntax of

Date:

Date:

(C) Vary the control variable from 20 down to 2 in steps of -2.

```
for(i=20; i>=2; i-2)
{
    cout<<i<<"\n";
}
```

(D) Vary the control variable over the following sequence of values.

2, 5, 8, 11, 14, 17, 20.

```
for(i=2; i>=20; i+3)
{
    cout<<i<<"\n";
}
```

(E) Vary the control variable over the following sequence of values.

99, 88, 77, 66, 55, 44, 33, 22, 11, 0.

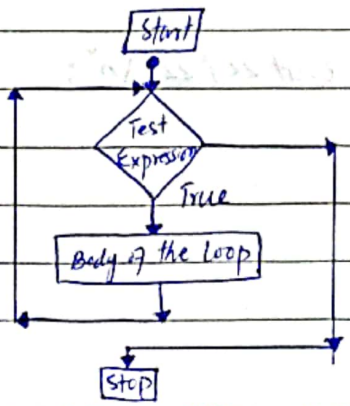
```
for(i=99; i>=0; i-11)
{
    cout<<i<<"\n";
}
```

(ii) **While Loop**

→ While Loop is used when we have (even before starting Loop) no idea that how many times a section of code or statement will be executed.

→ Like for Loop, while loop also contain a test expression but there is no initialization or increment/decrement expression etc

FlowChart of While Loop



Syntax of "While Loop"

```
while ( condition)
{
    statement (s);
    increment / decrement ;
}
```

---

* Write a program that display the output as follow:

```
0     100
25    75
50    50
75    25
```

```
#include < iostream.h>
# include < conio.h>
void main()
{
    int a =0 ;
    while ( a!=100)
    {
      cout<< a << "\t" << (100-a) << "\n";
      a = a+25 ;
    }
    getch();
}
```

Date:

(iii) **Do-While Loop :**

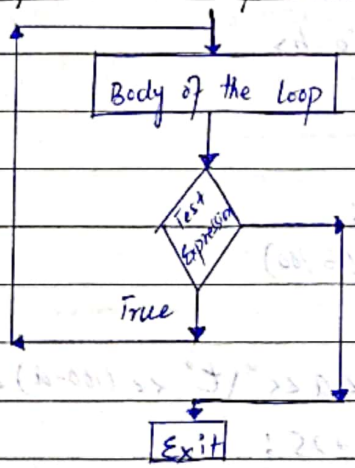
→ do while loop is used when we want to guarantee that the loop body should execute at least once, whatever the initial state of the test expression contains.

→ In do-while loop, the text expression or condition is placed at the end of the loop.

Syntax of do-while Loop

```
do
{
    statement(s);
    increment/decrement;
} (condition/Text expression);
```

FlowChart of do-while Loop



★ Write a program that display the value from 1 to 10 and also display their square and cube.

```
#include< iostream.h>
#include< conio.h>
void main()
{
    int a = 1;
    cout<<"value\t Square\t Cube"<<"\n";
    do
    {
        cout<<a<<"\t"<< a*a <<"\t"<< a*a*a<<"\n";
        a++;
    }while(a<=10);
    getch();
}
```

Date: __/__/20

ASK.

* Write a program that input a number n and print series from 1 to n using for-loop, while-loop and do-while loop respectively.

For-Loop...

```
#include <iostream.h>
#include <conio.h>
void main()
{
    int n;
    cout<<"Enter a number";
    cin>>n;
    for(int i=1; i<=n; i++)
    {
        cout<<i<<" ";
    }
    getch();
}
```

While-Loop

```
#include <iostream.h>
#include <conio.h>
void main()
{
    int i,n;
    cout<<"Enter a number";
    cin>>n;
    while(i<=n)
    {
        cout<<i<<" ";
        i++;
    }
    getch();
}
```

Date: / /20

flowchart

do-while Loop

```
#include <iostream.h>
#include <conio.h>
void main()
{
    int i,n;
    cout<<"Enter a number";
    cin>>n;
    do
    {
        cout<<i<<" ";
        i++;
    } while(i<=n)
    getch();
}
```

(iv) The Nested Loop :

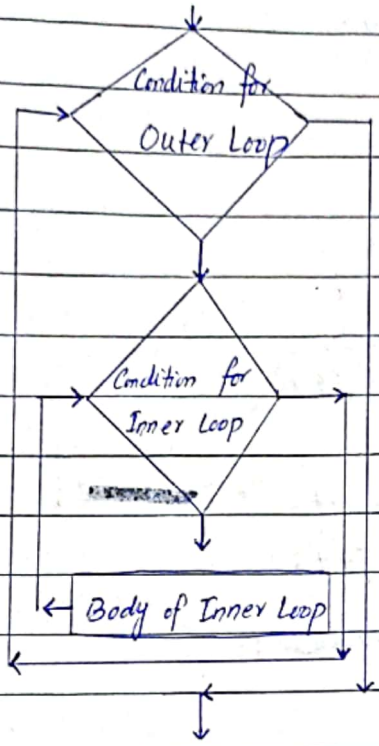
A loop inside the body of another loop is called nested loop. The nesting can be done up to any level. However, as the level of nested/nesting increases, the nested loop becomes more complex. For example, a "for" loop can be placed inside the body of "while" loop and vise versa.

Syntax of "Nested Loop"

```
Outer_Loop
{
    Statement(s)-1
    Inner_Loop
    {
        Body of inner Loop
    }
    statement(s)-2
}
```

Date: __/__/20__

Flowchart of "Nested Loop"



* Write a program that displays the following shape using nested for loop, while loop and do-while loop.

```
*
* *
* * *
* * * *
* * * * *
```

For-Loop.

```
#include<iostream.h>
#include<conio.h>
void main()
{
    for(int i=1; i<=5; i++)
    {
        for(int j=1; j<=i; j++)
        {
            cout<<"* ";
        }
        cout<<endl;
    }
    getch();
}
```

Date: / /20

While-Loop

```
#include<iostream.h>
#include<conio.h>
void main()
{
  int i=1;
  while(i<=5)
  {
    int j=1;
    while(j<=i)
    {
      cout<<"* ";
      j++;
    }
    cout<<endl;
    i++;
  }
  getch();
}
```

do-while Loop

```
#include<iostream.h>
#include<conio.h>
void main()
{
  int i=1;
  do
  {
    int j=1;
    do
    {
      cout<<"* ";
      j++;
    }while(j<=i);
    cout<<endl;
    i++;
  }while(i<=5);
  getch();
}
```

Functions

Date: / /20

FUNCTIONS :

"A group of statements that perform a specified operation or task is called a function". There are 3 things that are important, related to a function. These are Function Declaration, Function Definition and Function Calling. A function declaration tells the compiler about a function's name, return type and parameters. A function definition provides the actual body of the function. Function calling is the process of calling the function in the main function by writing its function name. Function code is stored in only one place in the memory. The Reason for creating is that a complex or bigger program code is divided into different functions due to which it becomes easy to manage the program. There are two types of functions in C++.

(i) Built-in Functions
(ii) User Defined Functions.

Syntax of Function

```
return_type function_name (parameter)
{
    // C++ statements
}
```

(i) Built-in Functions :

Built-in functions are also called library functions. These are the functions that are provided by C++. These functions are placed in the header files of C++. For example, <cmath> and <string> are the headers that have built-in math functions and strings functions respectively.

Syntax of Built-in Functions

```
#include <iostream.h>
#include <math.h>   } (// Built-in Functions).
#include <string.h> }
void main()
{
    ..... (pow(2,5);
    // C++ statements  Sqrt(4);).
}
```

Function

Date: __/__/20__

#include

#include

(ii) User Defined Functions

C++ allows programmer to define their own functions. A user-defined function is a group of codes used to perform a specific task and that group of code is given a name (identifier). When the function is invoked from any part of program, it all executes the codes defined in the body of function.

Syntax of User-Defined Functions

```
# include < iostream.h>
void function_name( )
{
  ..................
  ..................  }

Int main( )
{  ..................
   function_name( );
   ..................
}
```

Examples:

(01)

```
# include < iostream.h>
# include < conio.h>
void PrintMessage( );  // Function Declaration
void PrintMessage()    // Function Defination
{
    string name;
    cout<< "What is your name?";
    cin>> name;
    cout<< "\n Hello" << name << endl;
}
Int main()
{
    PrintMessage( );  // Function Calling
    getch( );
}
```

Output

```
What is your name? Humair
Hello Humair
```

Date: __/__/20__

```
#include <iostream.h>
#include <conio.h>
void line(); // Function Declaration
void line() // Function Defination
{
    for(int n=1; n<=20; n++)
    {
        cout<<"*";
    }
    cout<<endl;
}
int main()
{
    line(); // Function Calling 1
    cout<<"Hello\n";
    line(); // Function Calling 2
    cout<<"We are studing functions"<<endl;
    line(); // Function Calling 3
    getch();
}
```

Output

```
**********
Hello
**********
We are studing functions
**********
```

Eliminating The Declaration :

The second approach for using a function into a program is to eliminate the function declaration and place the function defination (or you can say the functionality of function itself). In follows example function declaration is not mentioned whereas function definition is written. So, we can say that it is not necessary to write function declaration in the program.

Example:

```
#include<iostream.h>
#include<conio.h>
void line() // Function Definition with Function Declaration.
{
    for(int n=1; n<=20; n++)
    {
        cout<<"*";
    }
    cout<<endl;
}
```

Date: __/__/20

```
int main()
{   line();
    cout<<" Hello "<<endl;
    line();
    cout<<" We are studing functions"<<endl;
    line();
    getch();
}
```

Passing by Value :-

→ Passing by values means that the actual value is passed on. Or we can say that this is the concept of in which the value is copied into the value of the corresponding function parameter.

→ In passing by values, the values passes from program to function.

→ These passed values can be used by the function according to the requirements.

→ There are two ways in Passing by value,

(i) Passing Constants to Functions

(ii) Passing Variables to Functions.

(i) Passing Constants to Functions

In passing constants to functions, a character, integer or float constant is actually passed as values passes from program to function. This is the concept in which the constant value is copied into the value of the corresponding functin parameter.

Parameter :- A parameter is a value which is passed from main function through function calling to the user defined function.

Date: __/__/20__

Ex ⑪.

```
#include <iostream.h>
#include <conio.h>                    (Parameters)
int sum (int num1 , int num2).
{
    int num3 = num1 + num2;
    cout << num3;
}
int main()
{                                     (Parameters)
    sum (1, 99);
    getch();
}
```

Output
100

Example ⑫.

```
#include <iostream.h>
#include <conio.h>                    (Parameter)
void line (char ch)
{
    for (int n=1 ; n<=20 ; n++)
    {
        cout << ch;
    }
    cout << endl;
}
void main()
{
    line('*');  // Function Calling 1
    cout<<" Hello "<<endl;
    line('-');  // Function Calling 2
    cout<<" We are studing functions "<< endl;
    line('*');  // Function Calling 3
    getch();
}
```

Output
****
Hello

We are studing functions
****

Date: __/__/20

* Syntax of Passing By Values.

```
# include < iostream.h>
# include < conio.h>
return_type function_name ( parameter(s)) // Function Definatio
{
  // C++ statements
}
int main()
{
    // C++ statements
    function_name (parameter) ;  // Function Calling
    // C++ statements
}
```

Example (03).

```
#include < iostream.h>
# include < conio.h>
void myFunction (string name)
{
    cout << name << "\n";
}
int main()
{
    myFunction ("Humair");
    myFunction ("Yahya");
    myFunction ("Irtiza");
    myFunction ("Maleek");
}
```

Output.
Humair
Yahya
Irtiza
Maleek

Date: / /20

Example ⑭. Write a program to check whether any entered number is even or not (number is odd). (Hint: use 1 parameter).

Passing Variables To Functions

```
#include <iostream.h>
#include <conio.h>
void myFunction (int n)
{
    if (n%2==0)
    { cout<<"Number is Even"; }
    else
    { cout<<"Number is odd"; }
    cout<<endl;
}
int main()
{
    int a;
    cout<<"Enter any number";
    cin>>a;
    myFunction(a);
    getch();
}
```

Output

Enter any number 10
Number is Even
(OR)
Enter any number 11
Number is Odd

(ii) Passing variables To Functions

Passing variables to functions and passing constants to functions are same. The only difference is that in passing variables to functions, the variables shown as parameters and the value of parameters/variables passed to function.

Date: __/__/20

Example #(02).

```
#include <iostream.h>
#include <conio.h>
int cubeofnumber(int n)
{
    int result;
    result = n*n*n;
    cout << "Cube of number is" << result << endl;
}
int main()
{
    int number;
    cout << "Enter any number";
    cin >> number;
    cubeofnumber(number);
    getch();
}
```

Output.

```
Enter Any number 5
Cube of number is 125
```

Example #(03)

```
#include <iostream.h>
#include <conio.h>
void line(char ch, int n)
{
    for(int a=1; a<=n; a++)
    { cout << ch; }
    cout << endl;
}
void main()
{
    char ch;
    int n;
    cout << "Enter a character";
    cin >> ch;
    cout << "Enter a value";
    cin >> n;
    line(ch, n); // Character and Integer variable passed.
    cout << "Hello" << endl;
    line(ch, n);
    cout << "We are studing functions";
    line(ch, n);
    getch();
}
```

Output.

```
Enter a character *
Enter a value 10
**********
Hello
**********
We are studing functions
**********
```

Date: __/__/20__

Function Overloading

The method of using the same function name for different functions in a program is called function overloading. Function overloading means that more than one function with the same name exists in the program but different number of arguments / parameters present in it. When the function will be called, then the number of arguments (parameters) will decide that which function will be actually called.

→ For example.

(Same function name but different parameters):

- sum(int num1, int num2)
- sum (int num1, int num2, int num3)
- sum (int num1, double num2)

→ (Both concepts passing by variable and passing by constants can be used in function overloading).

→ Function overloading is achieved by varing following points:

- The number of parameters
- datatype of the parameters
- The order of the apperance of parameters.

→ For example,

* Different Types of Parameters	* Different Numbers of Parameters
void print (int a);	void area (float r);
void print (double b);	void area (float a, float b);
void print (char c);	void area (float a, float b, float c);

Date: __/__/20

Example #3

Example #1.

```
#include <iostream.h>
#include <conio.h>
void print (int i)
{   cout << " Here is int " << i << endl;
}
void print ( float f)
{   cout << "Here is float " << f << endl;
}
void print ( char ch)
{   cout << "Here is char" << ch << endl;
}
int main()
{
   print(10);
   print (10.10);
   print ("ten");
   getch();
}
```

Output

```
Here is int 10
Here is float 10.10
Here is char "ten"
```

Example #2

```
#include <iostream.h>
#include <conio.h>
int sum (int x, int y)
{   cout << x+y;
}
int sum ( int x , int y, int z)
{   cout << x+y+z ;
}
int main()
{
   sum(10, 20);
   sum (10, 20, 30);
   getch();
}
```

Output.

```
30 60
```

Date: __/__/20__

Example #3.

```
#include <iostream.h>
#include <conio.h>
void line (void);
void line (int);
void line (char);
void line (int, char);
void line (char, int);
void main( )
{
    line (10);
    line ( );
    line ('=', 15);
    line ('*');
    line (20,'-');
    getch();
}
void line (void)
{
  for( int a=1 ; a<=10; a++)
    { cout<< "*";}
  cout<<endl;
}
void line (int n)
{
  for(int a=1 ; a<=n ; a++)
    { cout << "*";}
  cout <<endl;
}
void line (char c)
{
  for(int a=1 ; a<=10 ; a++)
    { cout << c ; }
  cout << endl;
}
void line ( char c , int n)
{
  for( int a=1 ; a<=10 ; a++)
    { cout << c ; }
  cout<<endl;
}
void line ( int n , char c)
{
  for( int a=1 ; a<n ; a++)
    { cout<<c ; }
    cout<<endl;
}
```

Output.

```
**********
**********
=============
**********
--------------------
```

Date: __/__/20__

Inline Functions.

Inline function instruct compiler to insert complete body of the function whenever that function got used in code. It does not require function calling overhead.

For Example

```
#include <iostream.h>
#include <conio.h>
inline float p2k(float pounds)  // inline function
{
    return 0.453592 * pounds;
}
void main()
{
    int pounds;
    cout<<"Enter weight in pounds";
    cin>>pounds;
    cout<<pounds<<" Pounds = "<<p2k(pounds);
    cout<<"kilograms";
    getch();
}
```

Output.

```
Enter weight in pounds 180
180 pounds = 81.646561 kilograms
```

Arrays

Date: __/__/20__

* Arrays :

Array is a collection of variables of the same data type placed closely in memory. Arrays are used to store multiple values in a single variables. Arrays can be used to store basic data types, i.e. int, char and float etc. The items or elements of array are accessed by index number.

Array Size = Total number of elements present in an array is called array size.

Index = Location of particular element in an array

* Declaring Array

→ data type arrayName [arraysize];

→ [] brackets are the clear indication of an array.

* Initializing Array

→ datatype arrayName [arraysize] = { n elements };

→ All the data present in an array is of same data type.

Another Method (type arrayname [] = { elements }).

* Accessing Array Elements

→ In order to access array elements we will use index or location. The location of array elements start from location 0. Any element can be accessed by writing index inside [] brackets. i.e.

arrayName [index];

Lab 4

Date: / /

* Accessing Array Elements

```
int balance[5] = {10, 20, 30, 40, 50};
```

→ balance[0] means 10.
→ balance[1] means 20.
→ balance[2] means 30.
→ balance[3] means 40.
→ balance[4] means 50.

Note: If array elements are n but array index will start from 0 to n-1.

* How To Insert, print and change Array elements?

Ans:

```
int balance[5] = {10, 20, 30, 40, 50};
```

(i) Take input from user and insert in third element.

```
cin >> balance[2];
```

(ii) Print first element of the array.

```
cout << balance[0];
```

(iii) change 4th element to 10 (means 40 to 10)

```
balance[3] = 10;
```

---

* Write C++ program to store entered numbers and display them using arrays.

```
#include <iostream>
#include <conio.h>
int main()
{
    int numbers[5];
    cout << "Enter 5 numbers ";
    for (int i=0; i<5; i++)
    {
        cin >> numbers[i];
    }
    cout << "Entered numbers are" << endl;
    for (int i=0; i<5; i++)
    {
        cout << numbers[i] << "\n";
    }
    getch();
}
```

Output

```
Enter 5 numbers: 5 6 7 8 9
Entered numbers are
5
6
7
8
9
```

Date: / /20

* Write a C++ program to store and calculate the sum of 5 numbers entered by the user using arrays.

```
#include <iostream>
#include <conio.h>
int main()
{
    int numbers[5];
    int sum = 0;
    cout<<" Enter 5 numbers: ";
    for (int i=0; i<5 ; i++)
    {
        cin>>numbers[i];
        sum = sum + numbers[i];
    }
    cout <<" Sum = " << sum << endl;
    getch();
}
```

Output

```
Enter 5 numbers: 5 6 7 8 9
Sum = 35
```

---

* Accessing Array Elements Using Loop

```
#include <iostream>
#include <conio.h>
int main()
{
    int arr[] = {11, 22, 33, 44, 55};
    int n = 0;
    while (n<=4)
    {
        cout<< arr[n] << endl;
        n++;
    }
    getch();
}
```

Output

```
11
22
33
44
55
```

Date: / /

```
#include <iostream>
#include <conio.h>
int main()
{
    int foo[] = { 16, 2, 77, 40, 12071 };
    int n, result = 0;
    for (n=0; n<5; n++)
    {
        result = result + foo[n];
    }
    cout << result;
    getch();
}
```

Output
72206

* Arrays of String

→ As like arrays deals with numbers, they can also deal with strings. You can create an array of string. For example, if there are 4 strings in an array which means there are 4 elements of type string. All these elements can be accessed just like numbers.

Example:

```
#include <iostream>
#include <conio.h>
int main()
{
    string colour[4] = { "Blue", "Red", "Orange", "Yellow" };
    for(int i=0; i<4; i++)
    {
        cout << colour[i] << "\n";
    }
    getch();
}
```

Output
Blue
Red
Orange
Yellow

Date: __/__/20

* Copying Array Elements

→ Array elements can be copied from one array to another in a simple way. Procedure to copy elements of one array to another in C++ is as follows:

(i) Create an empty array.
(ii) Insert the elements.
(iii) Create a duplicate empty array of the same size.
(iv) Start for Loop i=0 to i = arraylength
(v) newarray[i] = oldarray[i]
(vi) End for Loop.

→ (Copying Integer Array)

```
#include <iostream>
#include <conio.h>
int main()
{
    int arr1[10] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
    int arr2[10];
    for(int i=0; i<10; i++)
    {
        arr2[i] = arr1[i];
    }
    cout<<"\n Copy Array Elements are: ";
    for(int i=0; i<10; i++)
    {
        cout<<arr2[i]<<" ";
    }
    getch();
}
```

Output

Copy Array Elements are: 10 20 30 40 50 60 70 80 90 100

(copying String Array)

```
#include <iostream>
#include <conio.h>
int main()
{
    string str[4] = {"Good", "Bad", "Positive", "Negative"};
    string s[4];
    for(int i=0; i<4; i++)
    {
        s[i] = str[i];
    }
    for(int i=0; i<4; i++)
    {
        cout<<"The elements"<<i+1<<" of copied array = "<<s[i]
        <<" is same as"<<str[i]<<endl;
    }
    getch();
}
```

Passing Arrays To Functions

→ You can pass an array as an argument to a function just like you pass variables as arguments.

→ In order to pass array to the function, you just need to mention the array name during function call.

→ function_name(arrayname);

```
void function_name(data_type array_name[])
{
    ..........
    ..........
}
```

(Passing Arrays To Functions)

```
int main()
{
    int array_name[] = {.....}
    function_name(array_name);
}
```

Date: / /20

Example #3

```
#include <iostream>
#include <conio.h>
void sum(int arr2[], int arr1[])
{
    int temp[5];
    for(int i=0; i<5; i++)
    {
        temp[i] = arr2[i] + arr1[i];
        cout<< temp[i] << endl;
    }
}
int main()
{
    int a[5] = {10, 20, 30, 40, 50};
    int b[5] = {1, 2, 3, 4, 5};
    sum(a, b);    // Passing Arrays To Function
}
```

Output
11
22
33
44
55

---

*Note :

(01) When an array is passed as an argument to a function, only the name of an array is used as argument.

(02) Also notice the difference while passing array as an argument rather than a variable.

Date: __/__/20__

Passing Multi

Example #2

```
#include <iostream>
#include <conio.h>
int main()
{
    int marks[5] = {88, 76, 90, 61, 59};
    display(marks);
    getch();
}
void display (int m[5])
{
    cout << "Display marks" << endl;
    for (int i=0; i<5; i++)
    {
        cout << "Student" << i+1 << ":" << m[i] << endl;
    }
```

Output
Student1 : 88
Student2 : 76
Student3 : 90
Student4 : 61
Student5 : 59

Example #3

```
#include <iostream>
#include <conio.h>
float average (float a[])
{
    float avg, sum=0;
    for (int i=0; i<8; i++)
    {
        sum += a[i];
    }
    avg = sum/8;
}
int main()
{
    float n[] = {20.6, 30.8, 5.1, 67.2, 23, 2.9, 4, 8};
    float b = average(n);
    cout << "Average of numbers = " << b << endl;
    getch();
}
```

Output
Average of numbers = 20.2

Date: / /20

Passing Multiple Arrays To Functions

* Write a program to Print Maximum Number using Passing Multiple Arrays to Function method.

```
#include <iostream>
#include <conio.h>
void printMax(int arr[5])
{
    int max = arr[0];
    for(int i=0; i<5; i++)
    {
        if(max < arr[i])
        {
            max = arr[i];
        }
    }
    cout << "Maximum element is: " << max << "\n";
}
int main()
{
    int arr1[5] = {25, 10, 54, 15, 40};
    int arr2[5] = {12, 23, 44, 67, 54};
    printMax(arr1);
    printMax(arr2);
    // Passing Multiple Arrays To Function.
    getch();
}
```

Output
Maximum element is: 54
Maximum element is: 67

Date: __/__/20

* Passing Array To Multiple Functions

```
#include<iostream>
#include<conio.h>
void display1(int m[5])
{
    cout<<"Function1"<<endl;
    for(int i=0; i<5; i++)
    {
        cout<<"Student"<<i+1<<":"<<m[i]<<endl;
    }
}
void display2(int n[5])
{
    cout<<"Function2"<<endl;
    for(int i=0; i<5; i++)
    {
        cout<<"Student"<<i+1<<": "<<n[i]<<endl;
    }
}
int main()
{
    int marks[5]={88, 76, 90, 61, 59};
    display1(marks);
    display2(marks);
    getch();
}
```

Output

```
Function 1
student1 : 88
student2 : 76
student3 : 90
student4 : 61
student5 : 59
Function 2
Student 1 : 88
Student 2 : 76
Student 3 : 90
Student 4 : 61
Student 5 : 59
```

Structures

Date: __/__/20__

* Structures

Structure is a collection of variables that can be of the same data types or different data types. The data items in a structure are called the **members** of the structure. A user can define its own data types using structures.

* Structure Declaration

```
struct structure_name
{
    data-type member-1;
    data-type member-2;
    data-type member-3;
};
```

(**Note**: Semicolon will be present at the end of structure).

Structure will be declared outside the main. Inside the main there will be an **object** of structure, this name will be same as the name of structure. It will be written as follows:

```
Structure_name object_name;
```

→ This object is declared to access the members of of structure. The method to access the members of structures is:

```
object_name.member_name;
```

(Dot operator will be used).

For example:

```
struct Record // Structure Name
{
    int x=7; //member
};

int main()
{
    Record s; // object
    cout << "Value is " << s.x;
}
// Accessing Object
```

Structure 2

Date: __/__/20

Monday

(Structure Can Have Multiple Objects)

```
struct product
{
    int weight = 10;
    double price = 19.36;
};
int main()
{
    product apple; // object 1
    product banana; // object 2
    product melon; // object 3
    cout << apple.weight;
    cout << banana.weight;
    cout << melon.weight;
}
```

* Defining Structure Variables

→ A structure variable is defined the same way as a variable of any built-in data type i.e.

```
int age; // a variable with built-in datatype (Integers)

Record r; // r is a variable of structure type Record
```

* Accessing structure Members

→ The members of a structure variable are accessed by a dot operator, i.e.

```
r.age = 23;
r.salary = 20000;
```

Date: __/__/20__

* Another way to Initialize structure variable.

```
Record r = {6244, 23};
```

(means salary = 6244 and age = 23)

Example #1

```
struct database
{
    int id;
    int age;
    long salary;
};
int main()
{
    database employee;
    employee.age = 22;
    employee.id = 1;
    employee.salary = 20000;
    cout << employee.age << "\n" << employee.id << "\n"
    << employee.salary;
}
```

Example #2: Write a program that declares a structure to store roll no, marks, average and grade of a student. The program should define a structure variable, inputs the values and then display these values.

```
#include <iostream>
#include <conio.h>
using namespace std;
```

Example #3

Date: __/__/20__

```
struct student
{
    int rollno;
    int marks;
    float avg;
    char grade;
};
int main()
{
    student s;
    cout << "Enter roll no: ";
    cin >> s.rollno;
    cout << "Enter marks: ";
    cin >> s.marks;
    cout << "Enter Average: ";
    cin >> s.avg;
    cout << "Enter grade: ";
    cin >> s.grade;
    cout << "You entered the following details: \n";
    cout << "Roll No: " << s.rollno << endl;
    cout << "Marks: " << s.marks << endl;
    cout << "Average: " << s.avg << endl;
    cout << "Grade: " << s.grade << endl;
    getch();
}
```

Date: __/__/20

Example #3 : Write a program that declares a structure to store day, month and year of the birth. It takes three values and display date of birth in dd/mm/yy format.

```
#include < iostream >
#include < conio.h >
using namespace std;
struct birth
{
    int day;
    int month;
    int year;
};
int main()
{
    birth b;
    cout << " Enter day of birth : ";
    cin >> b.day;
    cout << " Enter month of birth : ";
    cin >> b.month;
    cout << "Enter year of birth : ";
    cin >> b.year;
    cout << "\n Your date of birth is " << b.day << "/" <<
    b.month << "/" << b.year;
    getch();
}
```

x

Date: __/__/20

Example #4

Example #4 : Write a program that declares a structure to store bookID, price and page of book. If define two structure variables and inputs the values, it display the record of most costly book.

```
#include <iostream>
#include <conio.h>
using namespace std;
struct Book
{
    int id;
    int pages;
    float price;
};
int main()
{
    Book b1, b2;
    cout<<"Enter id, pages and price of book 1: ";
    cin>>b1.id>>b1.pages>>b1.price;
    cout<<"Enter id, pages and price of book 2: ";
    cin>>b1.id>>b1.pages>>b1.price;
    cout<<"\n The most costly book is as follow: \n";
    if(b1.price > b2.price)
    {
        cout<<"BookID: "<<b1.id<<endl;
        cout<<"Pages: "<<b1.pages<<endl;
        cout<<"Price: "<<b1.price<<endl;
    }
    else
    {   cout<<"BookID: "<<b2.id<<endl;
        cout<<"Pages: "<<b2.pages<<endl;
        cout<<"Price: "<<b2.price<<endl;
    }
    getch();
}
```

Date: __/__/20__

Example #5: Write a structure that stores three parts of phone number ie. national code, area code and number seperatly. Create two variables of a structure. Initialize one variable and gets input from the user in other variable and then displays both.

```
#include <iostream>
#include <conio.h>
using namespace std;
struct phone
{
    int ncode;
    int acode;
    long number;
};
int main()
{
    phone p1 = {92, 41, 9220083};
    phone p2;
    cout << "Enter national code: ";
    cin>> p2.ncode;
    cout << "Enter area code: ";
    cin >> p2.acode;
    cout << "Enter phone number : ";
    cin >> p2.number;
    cout << "Phone Number 1 : + ";
    cout << p1.ncode << "-" << p1.acode << "-" << p1.number << endl;
    cout << "phone Number 2 : + ";
    cout << p2.ncode << "-" << p2.acode << "-" << p2.number << endl;
    getch();
}
```

Date: __/__/20__

✲ Array and Structure in C++

We can use structure and array as:

- Array within structure
- Array of structure

✲ Array within Structure

Syntax:

```
struct structure_name
{
    datatype var1;          // normal variable
    datatype array[size];   // array variable
    ------------
    ------------
    datatype varN;
};
```

Example

```
#include <iostream>
#include <conio.h>
struct Student
{
    int Roll;
    char Name[25];
    int Marks[3];
    int Total;
    float Avg;
};
void main()
{
    int i;
    Student S;
```

Date: __/__/20

```
cout << "\n Enter Student Roll No : ";
cin >> S.Roll;
cout << "\n Enter Student Name : ";
cin>> S.Name;
S. Total = 0 ;
for ( i=0 ; i<3 ; i++)
 {
    cout << "\n Enter marks" << i+1 << " : ";
    cin>> S.marks[i];
    S. Total = S. Total + S.marks[i];
 }
S. Avg = S.Total/3;
cout<< "\n Roll: " << S.Roll << "\n Name: " << S.Name ;
cout << "\n Total: " << S.Total << "\n Average : " << S.Avg;
getch();
}
```

---

* Array of Structure

Structure is a collection of different data types. An object of structure represents a single record in memory. If we want more than one record of structure type, we have to create an array of structure or object.

* Syntax For Declaring Structure Array

```
struct structure_name
{   datatype var1;
    datatype var2;
    datatype var3;
};
structure_name object [size];
```

Date: __/__/20

Example #1

```
#include <iostream>
#include <conio.h>
using namespace std;
struct Employee
{
    int Id;
    char Name [25];
    int Age;
    long Salary;
};
void main()
{
    Employee Emp[3]; // array of structure.
    for(int i=0 ; i<3; i++)
    {
    cout<<"\n Enter details of "<< i+1 <<" Employee ";
    cout<<"\n Enter Employee Id: ";
    cin>> Emp[i].Id ;
    cout<<"\n Enter Employee Name : ";
    cin>> Emp[i].Name ;
    cout<<"\n Enter Employee Age : ";
    cin>> Emp[i].Age ;
    cout<<"\n Enter Employee Salary : ";
    cin>> Emp[i].Salary ;
    }
    cout<<"\n Details of Employees";
    for (int i=0; i<3 ; i++)
    {
    cout <<"\n" << Emp[i].Id <<"\t" << Emp[i].Name
    <<"\t" << Emp[i].Age <<"\t" << Emp[i].Salary ;
    }
    getch();
}
```

Date: / /20

Example #2.

```
#include <iostream>
#include <conio.h>
using namespace std;
struct Customer
{
    int cid;
    string name;
};
int main()
{
Customer Record[2];
Record[0] = {25, "Bob Jones"};
Record[1] = {26, "Jim Smith"};
cout << Record[0].cid << " " << Record[0].name << "\n";
cout << Record[1].cid << " " << Record[2].name << "\n";
getch();
}
```

Date: __/__/20__

Example #3

```
#include <iostream>
#include <conio.h>
using namespace std;
struct CandyBar
{
    string name;
    float weight;
    int calories;
};
int main()
{
    CandyBar Candy[3];
    for(int i=0; i<3; i++)
    {
    cout << "Enter the name of Candy: ";
    cin>> Candy[i].name;
    cout << "Enter the weight of Candy: ";
    cin >> Candy[i].weight;
    cout << "Enter the calories of Candy: ";
    cin >> Candy[i].calories;
    }
    for (int i=0; i<3; i++)
    {
      cout << Candy[i].name << " " << Candy[i].weight
      << " " << Candy[i].calories << "\n";
    }
    getch();
}
```