Session 2 : Lesson 2

* If Else statement.

```
if (condition) {
        True statement
}

else {
        False statement.
}
```

* Prog 1 :- If else statement

```
#include <iostream>
using namespace std;
int main ()
{
   int number;
   cout << "Enter the number ";
   cin >> number;
   if ( number > 0)
   {
      cout << " the no. is positive ";
   }
   else if (number < 0)
   {
      cout << "The no is negative ";
   else
   {
      cout << " the no is zero ";
}
return 0; }.
```

Page No.
Date

* Short hand If else
{ Ternary Operators }

Syntax:-

variable = (condition) ? expression True :
expression False ;

⇒ Prog 2 :- Ternary operators.

```
#include <iostream>
using namespace std;
int main()
{
  int number;
  string result;
  cout << "Enter a number.";
  cin >> number;
  result = (number > 0) ? "the
      number is positive"
      : "the number is negative";
  cout << result;
  return 0;
}
```

* Switch Statements

syntax :-

```
switch (expression)
{
   case X :
      // code Blocks
      break;
   case Y :
      // code Blocks
      break;
   default :
      // code blocks
}
```

=> Prog 3 :- switch statements

```
#include <iostream>
using namespace std;
int main()
{
   int num;
   cout << "Enter a number from 1 - 7 :- ";
   cin >> num;
   switch (num)
   {
      case 1 :
      cout << "Monday";
      break;
```

```
case 2:
cout << "Tuesday";
break;
case 3:
cout << "Wednesday";
break;
case 4:
cout << "Thursday";
break;
case 5:
cout << "Friday";
break;
case 6:
cout << "Saturday";
break;
case 7:
cout << "Sunday";
break;
default:
cout << "Invalid Inputs";
break;
}
return 0;
}
```

LOOPS :-

* while Loop :-

syntax :-

```
while (condition){
        // code block to be executed
}
```

⇒ Prog 4 :- while loop

```
#include <iostream>
using namespace std;
int main()
{
    int i=0;
    while(i<=6)
    {
        cout << i << endl;
        i++;
    }
return 0;
}
```

⇒ output :-

0
1
2
3
4
5

Page No.
Date

Do while Loop :-

```
do {
   // code block
}
while (condition);
```

→ Prog 5 :- Do while Loop

```
#include <iostream>
using namespace std;
int main()
{
    int i = 5;
    do
    {
        cout << i << endl;
    } while (i--);
    return 0;
}
```

⇒ output :-

5
4
3
2
1
0

* For Loop :-

Syntax :-

```
for (statement 1; statement 2;
                    statement 3)
{
    // code block
}
```

statement 1 :- initialisation
statement 2 :- conditions
statement 3 :- updation.

⇒ Prog 6 :- For Loop.

```
#include <iostream>
using namespace std;
int main()
{
    for (int i = 0; i < 6; i++)
    {
        cout << i << "\n";
    }
return 0;
}
```

⇒ output :-

0
1
2
3
4
5

Page No.
Date

* Break statement :-

↳ Break statement is used to jump out of the loop.

⇒ Prog 7 :- Break statement

```
#include<iostream>
using namespace std;
int main()
{
    for(int i=0; i<=5; i++)
    {
        if(i==2)
        {
            break;
        }
        cout<<i<<"\n";
    }
return 0;
}
```

⇒ output :-

```
0
1
```

Page No.
Date

* Continue :-

↳ The continue statement breaks one iteration in the loop, if a specified condition occurs, & continue with the next iteration in the loop.

⇒ Prog 8 :- continue statement

```
#include <iostream>
using namespace std;
int main()
{
    for(int i=0; i<=5; i++)
    {
        if(i%2==0)
        {
            continue;
        }
        cout<<i<<"\n";
        return 0;
    }
```

⇒ output :-

1
3
5

Page No
Date

* Pointers :-

⇒ It is a variable that stores the memory address as it value.

⇒ A pointer variable points a datatype like 'int' or string of the same type, and is created with the * operator. The address of the variables you're working with is assigned to the pointers.

⇒ There are three ways of declare pointers which are as follows :-

```
string * mystring;
string *mystring;
string * mystring;
```

⇒ Prog 1 :- Pointers

```
#include<iostream>
using namespace std;
int main()
{
    int a = 200;
    int **ptr = &a;

    cout << "Variable value:" << a;
    cout << "\n Pointer value " << ptr;
    return 0;
}
```

⇒ Output :-
variable value = 200
Pointer value :- 0x61ff08

→ & → reference operater
* → dereference operater

⇒ Prog 10 → reference & dereference operator

```
#include <iostream>
using namespace std;
int main()
{
    int a = 200;
    int * ptr = & a;

    cout << "Reference Operater :- "
         << ptr;
    cout << "\n Dereference operater :- "
         << * ptr;

    return 0;
}
```

⇒ output :-
Reference operater :- 0x61ff08
Dereference operater :- 200

Page No.
Date

* Functions :-

↳ A function is a code of block which only run when it is called.

↳ We can pass data, known as parameters into a function.

↳ Functions are used to perform certain actions, & they are important for reusing code.

↳ C++ provides some predefined functions such as main() which is use to execute code..

⇒ Syntax :-

```
<datatype> function-name (parameters)
{
    // statements
    // return value
}
```

⇒ Prog 11 :- Functions.

```
#include<iostream>
using namespace std;
int sum (int x, int y) // function declaration
{
    return x+y;
}
```

```
int main()
{
    int a = 10, b = 23;
    cout << "sum :- " << sum(a,b);
return 0;                    // function calls
}
```

⇒ output :-

Sum :- 33

---

* Pass by value or.
Call by value

↳ While calling a function, we pass the values of variables to it. Such functions are known as call by values.

↳ In this method, the value of each variable in the calling functions is copied into corresponding dummy variables of the called functions.

↳ With this method, the changes made to the dummy variables in the called functions have no effect on the values of actual variables in the calling functions.

=> prog 12:- Call by value or pass By value.

```
#include <iostream>
using namespace std;
void swap( int x, int y) // Pass by value
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}
int main()
{
    int a=10, b=20;
    cout << "Before Swapping :- "
         << "a = " << a << " and
             b = " << b << endl;
    swap( a, b);
    cout << "After swapping :- "
         << "a= " << a << " and b="
         << b << endl;
    return 0;
}
```

o/p output :-

```
Before Swapping :- a = 10 and b = 20
After Swapping :- a = 10 and b = 20
```

* Thus actual values of a & b remain unchanged even after exchanging the values of x & y in the function.

** Call by reference :—

↳ While calling a function, instead of passing the values of variables we pass the address of variables to the function known as call by references.

↳ In this method, the address of actual variables in the calling function is copied into the dummy variables of the called functions.

↳ With this method, using address we would have access to the actual variables & hence we would be able to manipulate them.

⇒ Prog 13 :- call by reference

```
#include <iostream>
using namespace std;
void swap (int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
int main()
{
    int a = 10, b = 23;
    cout << "Before Swapping :- "
        << "a= " << a << " and b="
        << b << endl;
    swap( &a, &b);
    cout << "After Swapping :- "
        << "a = " << a << " and b= "
        << b << endl;
    return 0;
}
```

⇒ output :—

Before swapping :- a = 10 and b = 23
After Swapping :- a = 23 and b = 10