Assignment - 1

(1) Advantages of object-Oriented programming languages.

↳ Advantages:

→ Modularity
→ Reusability
→ Encapsulation
→ Flexibility and Scalability
→ Code Organization
→ code maintenance
→ code Reusability
→ Better problem Solving

1) Modularity:
→ OOP divides complex systems into smaller components, making the codebase easier to comprehend, create and maintain.

2) Reusability:
→ Inheritance allows code reuse, improving code quality and saving time.

3) Encapsulation:
→ Protects data integrity and privacy by restricting direct access and allowing controlled access through methods.

4) Flexibility and scalability:
→ OOP enables easy addition and modification of features without impacting the entire codebase.

5) code maintenance:
→ changes and bug fixes can be made to specific objects or classes without effecting other parts of the system, reducing errors and improving debugging.

6) Code organization:
→ OOP promotes a structured approach, enhancing collaboration and code redubility.

7) code Reusability:
→ OOP encourages the development of reusable code elements, saving time and improving system reliability.

8) Better Problem Solving:
→ OOP models real-world systems, allowing developers to create intuitive solutions that closely mimic real-world circumstances.

2) Explain OOPs concept in details.

↳ OOPs (Object Oriented Programming) is an approach or a programming pattern where the programs are structured around objects.

→ There are some basic concepts that acts as the building blocks of OOPs :-

- classes & objects
- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

→ Object

→ object can be defined as an entity that has a state and behavior, or in other world, anything that exists physically in the world is called an object.

→ It can represent a dog, a person, a table etc.

→ An object means a combination of data and programs, which further represent an entity.

Date
Page

• Classes:

→ Class can be defined as a blueprint of the object.

→ It is basically a collection of objects which act as building blocks.

→ A class contains data members (variables) and member functions. These member functions are used to manipulate the data members inside the class.

→ for example a class named car contains datamember such as Wheels, Bumpers, Exhaust, and member functions such as Speed(), Engine(), Break() etc.

• Abstraction:

→ Abstraction helps in the data-hiding process. It helps in displaying the essential features without showing the details or the functionality to the user.

→ It avoids unnecessary information or irrelevant details and shows only that specific part of the user want to see.

→ for example Your car. You can start a car by turning the key or pressing the start button. You don't need to know how the engine is getting started, what all components your car has.

• Encapsulation :

→ The wrapping up of data and functions together in a single unit is known as encapsulations.

→ It can be achieved by making the data member's scope private and the member function scope public to access these data members.

→ Encapsulation makes the data non-accessible to the outside world.

→ example of encapsulation in real life is Car. Now, think about the various components that make up the car, such as : tires, wheel, engine, and shafts. All attributes grouped and make a car.

• Inheritance :

→ Inheritance is the process in which two classes have an is-a relationship among each other and objects of one class acquire properties and features

of the other class.

→ The class which inherits the features is known as the child class, and the class whose features it inherited is called the parent class.

→ for example, Class Vehicle is the parent class, and Class car, Bus are child classes.

• Polymorphism :

→ Polymorphism means many forms. It is the ability to take more than one form.

→ It is a feature that provides a function or an operator with more than one definition.

→ It can be implemented using function overloading, operator overload, virtual functions.

→ for example, a person who at the same time can have different characteristics. A man at the same time is a father, a husband, and an employee.

3) Define cin function in details.

↳

→ The "c" in c++ refers to 'character' and "in" refers to 'input'.

→ 'cin' means 'character input'.

→ The c++ cin object belongs to the istream class.

→ It accept input from a standard input device, such as a keyboard and it is linked to stdin, the regular c input source.

Syntax :

```
cin >> var_name;
```

→ Here >> is the extraction operator.

Example :

```
#include <iostream>
using namespace std;
int main() {
    int age;
    cout << "Enter age";
    cin >> age;
    cout << "Your age is : " << age;
    return 0;
}
```

(4) Define 'cout' function in details.

↳

→ The 'c' in cout refers to "character" and "out" refers to the output".

→ The cout means "character out".

→ The cout object is used along with the insertion operator << in order to display a stream of characters.

→ cin and cout aren't applicable to c language because of the conflicting header statements.

Syntax:

```
cout << "Any String";
or
cout << var_name;
```

→ Here, << is the insertion operator.

Example:-

```
#include <iostream>

using namespace std;
```

```
int main() {
    cout << "Hello World";
    return 0;
}
```

(5) Explain output and Input operator in details.

↳

Output operator:-

→ The output operator is also known as the insertion operator (<<).

→ Here, the treatment of data takes place as a stream of characters by cout.

→ Furthermore, the flow of those characters takes place from the program to cout through the output operator.

→ The working of the output operator takes place on two operands, namely, the expression whose display is to be on its right and the cout stream on its left.

→ Below program demonstrate the working of cin output operator.

```
#include <iostream>
using namespace std;
int main() {
    int a;
    cin >> a;
    a = a+1;
    cout << a;
    return 0;
}
```

* Input operator:

→ The use of the input operator, commonly known as the extraction operator (>>), takes place with the standard input stream, cin.

→ The treatment of the data takes place as a stream of character by cin.

→ The flow of these characters takes place from cin to the

Date
Page

program through the input operator.

→ The working of the input operator is on two operand, namely, a variable on its right and the cin stream on its left.

→ Below program demonstrate the working of an input operator :-

```
#include <iostream>

using namespace std;

int main() {

    int a;

    cin >> a;

    a = a+1;

    return 0;

}
```

(6) Define: variable

↳

The variables are containers for storing data values.

→ In c++, there are different types of variables (defined with different keywords), for example;-

- int
- double
- char
- float
- string
- bool

(7) Define: constant

↳

Constants refers to fixed value that the program may not alter (change). ~~and~~ they are also called as literals.

→ Constants can be of any of the basic data types.

(8) ~~Scop~~ Explain Scope Resolution operator in detail.

↳

→ The scope resolution operato is used to reference the global variable or member function that is out of scope.

→ Therefore, we use the scope resolution operator to access the hidden variable or function of a program.

→ The operator is represented as the double colon (::) symbol.

→ for example, when the global and local ~~or~~ variable or function has the same name in a program, and ~~#~~ when we call the variable, by default it only accesses the inter or local variable without calling the global variable.

→ It this way, it hides the global variable or function.

→ To overcome this situation, we use the scope resolution operator to fetch a program's hidden variable or function.

→ Here is the example of scope resolution operator:

```
#include <iostream>

using namespace std;

int num = 50;
```

```
int main() {
    int num = 100;
    cout << "The value of the
            local variable
            num: " << num << endl;
    cout << "The value of the
            global variable
            num: " << ::num;
    return 0;
}
```

Output:

The value of the local variable num: 100
The value of the global variable num: 50

UNIT-2

Date
Page

→ If we want to make a function in c++ then we must follow these steps :-

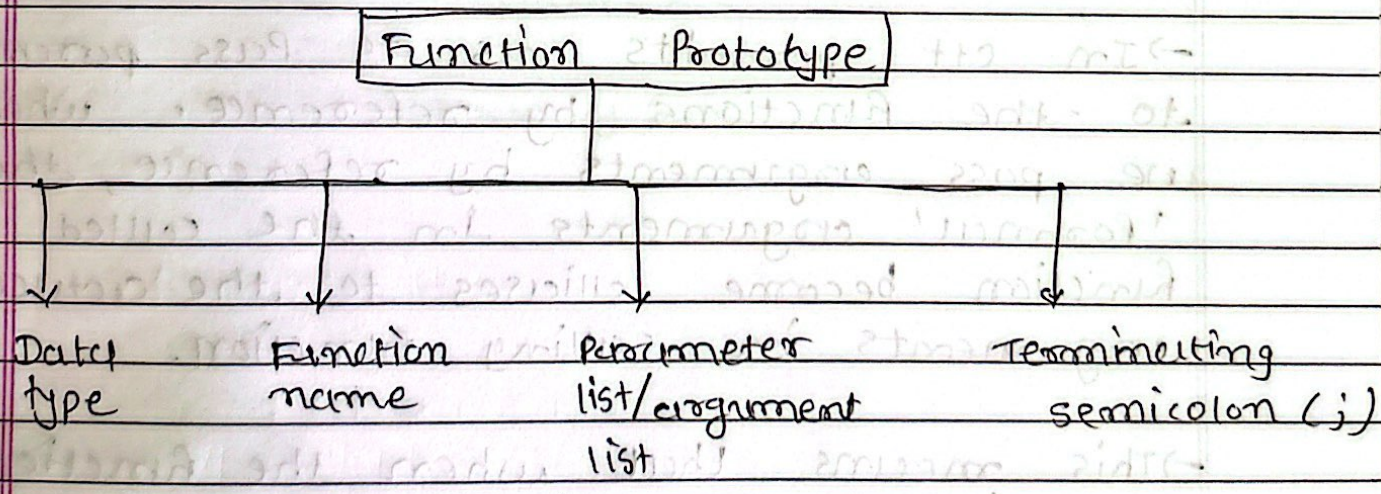
1) Function Prototype
2) Function Definition
3) Function call

1) Function Prototype :-

→ Function Prototype is one of the major improvement added to c++ functions.

→ The Prototype describe in four parts :

1) Function type
2) Function name
3) Parameter list or Argument list
4) Terminating Semicolon (;)



→ Function prototype is a declaration statement in calling program and the following form:

Date
Page

Declaration Statement:

Type function_Name (parameter list);

Ex:

int add (int x, int y);

void (dd (void);

↓

In c++ this means that function does not pass any parameter and and return value.

→ Formal Parameter is called reference variable.

Call By Reference:

→ In c++ permits us to pass parameter to the functions by reference. When we pass arguments by reference, the 'formal' arguments in the called function become aliases to the 'actual' arguments in calling function.

→ This means that when the function is working with its own arguments, it is actually working on the original data.

Ex:

```
void swap ( int &a, int &b)
{
    int t = a;
    a = b;
    b = t;
}
```

Return By Reference

→ A function can also return a reference.

ex:

```
int & max (int &x, int &y)
{
    if (x > y)
        return x;
    else
        return y;
}
```

Date
Page

Inline Functions:

→ When a function is likely to be called many times. To eliminate the cost of calls to small functions, C++ proposes a new feature called 'Inline Function'.

→ An Inline Function is a function that is expanded in lines when it is invoked.

→ That is, the compiler replaces the function call with the corresponding function code.

syntax:

```
inline function-header
{
    function-body
}
```

Ex.

```
inline double cube(double a)
{
  return (a*a*a);
}
```

Date
Page

Default Arguments:

→ c++ allows us to call a function without specifying all its arguments.

→ In such cases, the function assigns a default value to the parameter which does not have matching arguments in the function call.

ex.

```
float amount(float principal, int period,
             float rate = 0.15);
```

call: `value = amount(5000, 7);`

const Arguments:

→ In c++, an argument to a function can be declared as const as shown below:

```
int strlen(const char *p);
```

Function overloading:

→ overloading refers to the use of same thing for different purposes.

→ c++ also permits overloading of functions. This means that we can use the same function name to create

functions that perform a variety of different tasks.

→ this is also known as function polymorphism in OOP.

ex:
```
// function declarations
int add(int a, int b);
int add(int a, int b, int c);
double add(double x, double y);

// function calls
cout << add(5,10);
cout << add(5.5,10.0);
cout << add(5,10,15);
```

~~Friend Functions~~

→

Date
Page

class :

→ A class is a way to bind the data and its associated functions together.

→ We can also say that, class is a combination of data members and member functions in a single unit.

→ The class declaration describes the type and scope of its members.

↳ general form of class declaration is :

```
class class_name
{
    private :
        variable declaration;
        function declaration;
    Public :
        variable declaration;
        function declaration;
};
```

→ By default, all the members of class are private.

Ex.

```
class item
{
    int number;
    float cost;
  pubic:
        void getdata(int a, float b);
        void putdata(void);
};
```

Date
Page

Object:

→ Object is an instance of class.

→ Once a class has been declared, we can crete variables of their type by using the class name.

for ex.

```
item x;
// creates a variable of type item.
// In ctt, the class variable are
// known as objects.
```

Accessing class members:

→ The private data of a class can be accessed only through the member functions of their class.

↔ A variable declared as public can be access by the objects directly.

ex.

```
class xyz
{
    int x;
    int y;
public:
    int z;
};
xyz p;
```

P.x = 10; // error, x is private
P.z = 10; // OK, z is public

Implementing class methods:

→ Member functions can be defined in two places:

(1) outside of the class definition.
(2) Inside of the class definition.

(1) outside of the class definition:

→ Their definitions are very much like the normal function.

→ the general form of a member function definition is:

```
return-type class-name :: func.-name
                    (argument decl.)
{
    function Body
}
```

ex.

```
void item :: getdata (int a, float b)
{
    number = a;
    cost = b;
}
```

(2) Inside the class definition:

→ Another method of defining a member function is to replace the function declaration by the actual function definition inside the class.

for ex:

```
class item
{
    int number;
    float cost;
public:
    void getdata(int a, float b);
    // inline function
    void putdata(void)
    {
        cout << "number" << endl;
        cout << cost << endl;
    }
};
```

Date
Page

Friend Function:

→ The private members cannot be accessed from outside the class.

→ However, C++ allows the common function to be made friendly with both the classes, thereby allowing the function to have access to the private data of these classes. Such a function need not be a member of any of these classes.

→ To make an outside function "friendly" to a class, we have to simply declare this function as friend of the class as shown below:

```
class ABC
{
    ....
    ....
public:
    ....
    ....
    friend void xyz(void);
};
```

Constructor

→ A constructor is a special member function whose task is to initialize the objects of its class.

→ It is special because its name is the same as the class name.

→ The constructor is invoke whenever an object of its associated class is created.

→ It is called constructor because it constructs the values of data members of the class.

Types of constructors:

(1) Default constructor
(2) Parameterized constructor
(3) copy constructor

(1) Default constructor:

→ Default constructor is a type of constructor which is special member function and invoked automatically whenever object of that class is created.

Ex:

```
class integer
{
    int m, n;

    public:
        integer(void);
        ....
        ....
};

integer :: integer(void)
{
    m = 0;
    n = 0;
}
```

(2) ~~Defa~~ Parameterized constructor:

→ Parameterized constructor is a type of constructor which is special member function and invoked automatically whenever objects of that class is created.

→ Parameterized constructor is similar to default type but it accept parameters.

Ex:

```
class integer
{
    int m, n;
    public:
        integer(int x, int y);
        ....
};      ....
```

(3) Copy constructor:

→ copy constructor is a type of constructor which is special type of member function and invoked automatically whenever object of their class is created.

→ A copy constructor takes a reference to an object of the same class as itself as an argument.

Ex.

```
class integer
{
    int m, n;
    public:
    integer(integer &i);

};
```

Date
Page

Destructor :

→ A destructor, as the name implie is used to destroy the objects tha have been created by a constructo

→ Like, a constructor, the destructor i a member function whose name is the same as the class name but is preceded by a tilde. (~).

ex.

```
~integer();
```

→ A destructor never takes any argumen nor does it return any value.

→ It will be invoked implicitly by the compiler upon exit from the program to clean up storage.

ex.

```
class alpha()
{
    public:
        alpha() {
            cout << "constructor called" << endl;
        }
        ~alpha() {
            cout << "destructor called" << endl;
        }
};
```

Date
Page

UNIT-3

Operator Overloading & Type casting

Operator overloading :-

→ Operator overloading is a one of the most important feature in ctt which is a type of polymorphism.

→ operator overloading provides a flexible option for the creation of new definition for most of the ctt operators.

→ We can overload all the ctt operators except the following:

1. class member access operator (. , .*)
2. Scope Resolution operator (::)
3. size operator (sizeof)
4. Conditional operator (? :)

→ The general form of an operator function is:

```
return-type classname :: operator op(arglist)
{
    function body
}
```

Ex.

```
class sum
{
    int a;
    public:
        void getdata()
        {
            cout<<" Enter no.: ";
            cin>>a;
        }
        void putdata() {
            cout<<"\n value : "<<a;
        }

        sum operator + (sum bb)
        {
            sum cc;
            cc.a = a+ bb.a;
            return cc;
        }
};

int main() {
    sum aa, bb, cc;
    aa.getdata();
    bb.getdata();
    cc = aa + bb;
    aa.putdata();
    bb.putdata();
    cc.putdata();
    return 0;
}
```

Unary operator overloading

→ A unary operator is just takes one operand.

ex:

```
class space {
    int x;
    int y;
    int z;
  public:
    void getdata(int a, int b, int c);
    void display(void);
    void operator-();
};
void space :: getdata(int a, int b, int c)
{
    x = a;
    y = b;
    z = c;
}
void space :: display(void)
{
    cout << x << " ";
    cout << y << " ";
    cout << z << "\n";
}
void space :: operator-()
{
    x = -x;
    y = -y;
    z = -z;
}
```

```
int main(){
    Space s;
    s.getdata(10, -20, 30);
    cout << "S : ";
    S.display();

    -S;
    cout << "S : ";
    S.display();

    return 0;
}
```

Binary operator overloading :

→ eg:

```
class complex
{
    float x;
    float y;
public:
    complex(){}
    complex(float real, float imag){
        x = real;
        y = imag;
    }
    complex operator+(complex c){
        complex temp;
        temp.x = x + c.x;
        temp.y = y + c.y;
        return(temp);
    }
```

```
void display(void) {
    cout << x << " + j" << y << "\n";
}
};

int main() {
    complex c1, c2, c3;
    c1 = complex(2.5, 3.5);
    c2 = complex(1.6, 2.7);
    c3 = c1 + c2;

    cout << "c1 = ";
    c1.display();
    cout << "c2 = ";
    c2.display();
    cout << "c3 = ";
    c3.display();

    return 0;
}
```

Output:

```
c1 = 2.5 + j3.5
c2 = 1.6 + j2.7
c3 = 4.1 + j6.2
```

Date
Page

Type Conversion

→ one datatype converts into another datatype is known as type conversion.

→ The type ~~conversion~~ of data to the right of an assignment operator is automatically converted to the ~~right~~ type of the variable on the left.

ex.

```
int m;
float x = 3.14;
m = x;
```

Types of Type conversion:

→ There are mainly three types of type conversion:

(1) Conversion from Basic type to class type.

(2) Conversion from class type to basic type.

(3) Conversion from one class type to another class type.

(1) Basic type to class type conversion:

→ In this type of conversion the source type is Basic type and destination type is class type. means basic datatype is converted into the class datatype.

→ for ex. we have a class Employee and one object of employee class is 'emp' and suppose we want to assign the employee code of 'employee emp' by any integer variable say 'code' then statement below is the example of conversion from basic to class type.

```
class employee
{
    public:
        int code;
};

int main()
{
    employee emp;
    int x = 101;
    emp.code = x;
    return 0;
}
```

(2) class to Basic type conversion:

→ In this type of conversion, the source type is class type and the destination type is basic type. means class datatype is converted into basic datatype.

→ for ex. we have a class 'time' and an object of this class 't' and suppose we want to assign total time of object 't' into any integer variable say 'duration'. The statement below is example of type conversion of class to Basic type:

```
duration = t;
```

→ 't' is an object and duration is basic datatype. here the assignment will be done by converting 't' object which is class type into basic or primary type. It requires special casting operator function for class type to Basic type conversion.

```
class time1 {
    int hours;
    int minutes;
public:
    time1(int t) {
        hours = t/60;
        minutes = t % 60;
    }
};
```

```
int add () {
    int m;
    cout << "The time is = ";
    cout << hours << "hrs " << minutes
         << "mnts";
    m = minutes;
    return m;
  }
};

int main() {
    int x = 9.0;
    time1 t1(x);
    int j;
    j = t1.add();
    cout << "The value of j is " << j;

    return 0;
}
```

(3) class to another class type conversion :

→ In this type of conversion both the type that is source and destination type are class type means source type is class type and destination type is class type.

→ In other words, one class datatype is converted into another class datatype.

→ For ex. we have two classes one for the computer and another for mobile. Suppose if we wish to assign price of computer to mobile then it can be achieved by statement below which is the example of conversion from one class to another class type.

```
mob = comp ;
```

→ 'mob' and 'comp' are objects of mobile and computer class respectively.

→ Here the assignment will be done by converting 'comp' object which is class type into 'mob' which is another class type.

—X—