Message Passing: It is the dynamic process of asking an object to perform a specific action. The communication between objects is performed via message which is known as message passing. Objects communicate with one another by sending and receiving information to each other.

A message for an object is a request for the execution of a particular function. The receiving object invokes that function and generates the desired results. The object that receives the message is also known as receiver.

Message passing involves specifying the name of the object (receiver), the name of the function (message) and the information to be sent.

Program Example- 78:

```
class Random
{
    int num;
public:
    void get_data(int x){
        num=x;
    }
    void display(){
        cout<<"Value of num: "<<num<<endl;
    }
};

int main()
{
    Random r;
    r.get_data(5);   /** object passing message **/
    r.display();   /** object passing message **/
    return 0;
}
```

Output: Value of num: 5

Difference between message passing and procedure calls:

Procedure call	Message Passing
1. Procedure call is basically executing a block of code to perform a specific action.	1. Message passing is the dynamic process of asking the object to perform a specific action.
2. In a procedure call, there is no designated receiver.	2. In message passing, there is a designated receiver for that message.
3. There is an early binding between the procedure name and the code fragments.	3. There is a late binding between the message and the code fragments used to respond the message.
4. It involves specifying the name of the procedure and the information (arguments) to be sent.	4. It involves specifying the name of the object, the name of the message and the information to be sent.
5. For example: add(4,5)	5. For example: account.addbalance(5000)

Memory mapping and allocation:

Stack: It is a region of computer's memory that stores temporary variables created by each function including the main function. It is limited in size. It is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO (Last In Fist Out) or FILO (First In Last Out). This means, the last element inserted inside the stack is removed first and the one inserted at first is removed at last and remains in the stack for the longest period of time.

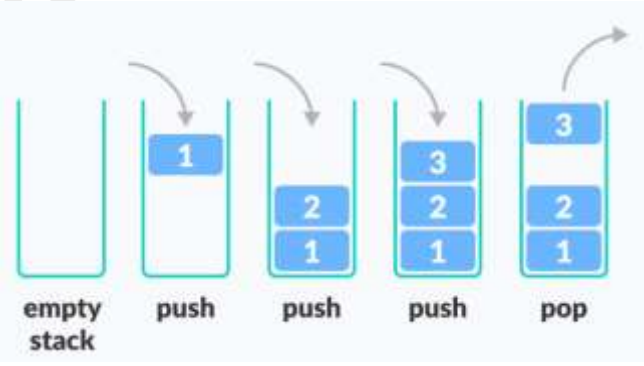


Figure: Push and pop operations on stack

Every time a function declares a new variable, it is pushed (inserted) onto the stack. Every time a function exits, all the variables which are pushed onto the stack are popped (set free or deleted). They are lost forever.

Advantages of stack:

- The variables in stack are stored and deleted automatically. We don't have to explicitly allocate memory or free the memory.
- The stack memory is organized very efficiently the CPU. Reading from and writing to stack is very fast.

Disadvantages:

- The stack is limited in size (dependent on OS).
- The variables cannot be resized.

Heap: It is a large pile of memory space available to programmers for allocation and de-allocation. It is not managed automatically by the CPU. Whenever a programmer declares a variable dynamically, the memory is allocated from the heap section. The reference of this allocated memory is then stored in the stack. This memory has to be set free explicitly if the variables are used up in the program and don't need the allocated space anymore. A 'new' operator is used in C++ to allocate the memory in the heap. And to free up the memory, a 'delete' operator is used.

Unlike the stack, the heap does not have size restrictions on variable size. Reading from and writing to heap is relatively slower. We have to use pointers to access memory on the heap. The variables created on the heap are accessible by any function anywhere in the program. They are global in scope.

Memory allocation for objects: When the instances (objects) of a class are created, the memory is allocated for them. For each object, separate memory spaces are allocated. The size of the memory depends upon the data members and the member functions.

- The memory space for data members are allocated separately. It is because the data members will hold different data values for different objects.
- However, the same member function is used by all the objects belonging to the same class. So, when the member functions are defined they are placed in memory space only once and no separate memory space is allocated for member functions when various objects are created.

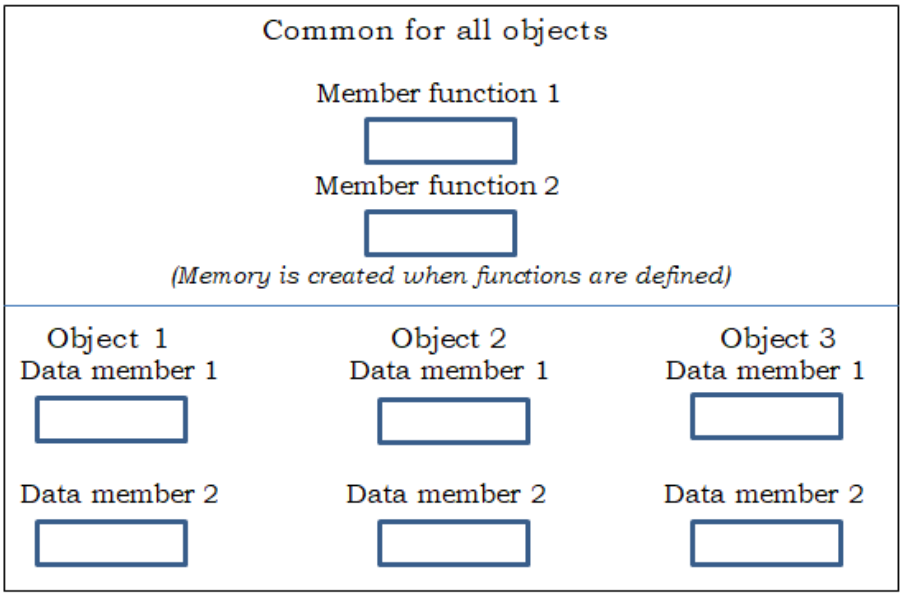


Figure: Memory allocation of objects

Differences between Stack and Heap based Allocation:

Stack	Heap
1. It is used for static memory allocation.	1. It is used for dynamic memory allocation.
2. It is limited in size which is dependent on OS.	2. It is a large pile of memory and does not have a specific limit on memory size.
3. Reading to and writing from stack is very fast.	3. Reading to and writing from heap is relatively slower.
4. The variables allocated in stack are local in scope.	4. The variables allocated in heap have a global scope.
5. Variables cannot be resized.	5. Variables can be resized.
6. Memory for variables are allocated and de-allocated automatically.	6. We have to explicitly allocate and de-allocate memory for variables.
7. We don't have to use pointers for stack allocation.	7. We have to use pointers to access memory on the heap.
8. No any special operators are required for stack memory allocation. The memory is automatically allocated after variables are declared. And they are automatically de-allocated when the function exits.	8. 'new' and 'delete' operators are used to allocate and de-allocate memory from heap.

Subtype, Subclass and Principle of Substitutability:

Subtype: It is a type that is related to another type by some notion of substitutability. It can be regarded as a subclass that can be substituted with its parent class using polymorphism with no observable effect. A subtype can safely be used in a context where its parent type is expected.

The **principle of substitutability** states that- "If we have two classes A and B such that class B is a subclass of A, it should be possible to substitute instances of class B for instances of class A in any situation with no observable effect."

The subtype is used to refer subclass relationship when a child class is inherited from a parent class publicly. If the mode of inheritance is made private, then base class object can't be replaced by the object of the child class.

Example program- 79:

```
class Parent{
    public:
    void display(){
        cout<<"Parent class"<<endl;
    }
};

class Child: public Parent{
    public:
    void display(){
        cout<<"Child class"<<endl;
    }
};

void test(Parent a)
{
    a.display();
}

int main()
{
    Child c;
    test(c); /** object c is substituted for object of Parent class **/
    return 0;
}
```

Output: Parent class

Software reusability: It is the practice of re-using the existing code for building new software or updating extra features on an existing system. The software projects can be built from high-quality, robust, well-tested small software modules whose behavior is well known and well understood. These modules should be safe, secure, reliable, efficient and maintainable.

In object oriented programming, the concept of inheritance provide reusability. The additional features can be added to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will be able to inherit the features of the parent class and will be having its own additional features.

In addition to inheritance, the concept of composition also provides the reusability. In composition, one class contains the object of another class and hence can acquire the features of it.

Advantages of reusability:

- Reusability saves the programmer time and effort.
- The development and maintenance costs are reduced.
- The existing features can be extended with new additional features.
- Reusability enhances reliability.

Q. What is data hiding? How do you achieve data hiding in C++? Explain with a suitable example.

Or,

What is encapsulation? How can encapsulation be enforced in C++? Explain with suitable code.

Example answer:

One of the important characteristics of OOP is **data encapsulation.** That means, the data and functions are wrapped into a single unit. The data are kept private and are not accessible to the outside world. These data are accessed by only the member functions that are present in the same unit (i.e. in the same class). This process is also known as **data hiding** as the data are hidden from outside. This feature of encapsulation or data hiding provides more security to the system.

It can be illustrated by the following program example.

```
class Customer
{
    int id;    //data memeber
    string name; //data member
  public:
    void getData(int x, string y){   //member function
        id=x;
        name=y;
    }
    void display(){    //member function
        cout<<"Id: "<<id<<endl;
        cout<<"Name: "<<name<<endl;
    }
};
int main()
{
    Customer c;

    c.id;   /** can't be accessed **/
    c.name; /** can't be accessed **/

    c.getData(105, "Manoz"); /** Accessed via member function **/

    cout<<c.id<<endl;  /** can't be accessed **/
    cout<<c.name<<endl; /** can't be accessed **/

    c.display();              /** Accessed via member function **/

    return 0;
}
```

In above program example, the data members 'id, name' and member functions 'getdata(), display()' are encapsulated in a class 'Customer'. The data members are kept private and are hidden from the outside world. They cannot be accessed from anywhere except from the member function getdata(), and display() member functions. If we want to access the data members from outside, they have to be accessed via the member functions (which are made public). So, member functions acts as an interface between the data and the outside world. In this way, encapsulation is enforced and data hiding is achieved in C++.

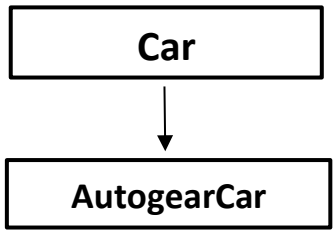
Forms of inheritance: Inheritance can be used in variety of ways depending on user's requirements. Various forms of inheritance are:

1) *Sub classing for specialization:* The child class is the special case of the parent class. In other words, the child class is a subtype of parent class.
2) *Sub classing for specification:* The parent class defines behavior that is implemented in child class but not in parent class.
3) *Sub classing for construction:* The child class makes use of behavior provided by the parent class but is not a subtype of parent class.
4) *Sub classing for generalization:* The child class modifies or overrides some of the methods of the parent class but is not a subtype of parent class.
5) *Sub classing for extension:* The child class adds new functionality to the parent class but does not change any inherited behavior.
6) *Sub classing for limitation:* The child class restricts the use of some of the behavior inherited from parent class.
7) *Sub classing for variance:* The child class and parent class are variants of each other and the class-subclass relationship is arbitrary.
8) *Sub classing for combination:* The child class inherits features from more than one parent class. This is the kind of multiple inheritance.

Some of the forms of inheritance *(Elaborated):*

1) Sub classing for specialization: It is the most common form of inheritance. In sub classing for specialization, the new class is a specialized form of the parent class but satisfies the specifications of the parent class in all relevant aspects. So, in this form, the principle of substitution is maintained and the child class can be regarded as the subtype of the parent class.

An example of sub classing for specialization is discussed below.

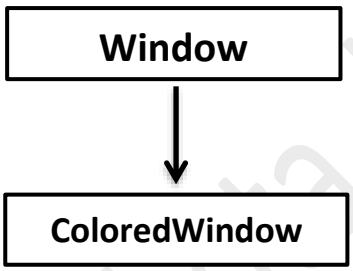


Consider a class Car provides general operations such as changing gears, accelerating, changing direction etc. A specialized sub class AutoGearCar inherits the operations of Car class and in addition, provides the facilities that allow Car to change the gear in auto mode. As the AutoGearCar satisfies all the

properties we expect from a Car, we recognize the situation as an example of sub classing for specialization.

2) Sub classing for generalization: It is opposite to the sub classing for specialization. In sub classing for generalization, the new class extends the behavior of the parent class to create a more general kind of object. It is applicable when we build on a base of existing classes that we do not wish to modify, or cannot modify.

An example of sub classing for generalization is discussed below.



Consider a graphics display system in which a class Window has been defined for displaying on a simple black-and-white background. A subtype ColoredWindow is created that lets the background color to be something other than black and white. This is implemented by adding an additional data member to store the color and overriding the display function of the parent class Window. This overridden function specifies the background to be drawn in that specific color. As ColoredWindow contains the data member that is not necessary in the parent Window class, we recognize the situation as an example of sub classing for generalization.

STL: It stands for Standard Template Library. A set of general purpose template class and functions were developed in order to help the C++ programmers in generic programming. The collection of these generic classes and functions is called the Standard Template Library (STL). It is used as a standard approach of storing and processing of data. It saves the time and effort of the programmer and thus helps them to produce high quality programs.

Components of STL: The STL contains several components. But there are three key components of STL at its core. They are:

a) Containers

b) Algorithms
c) Iterators

These three components work in conjunction with one another to provide support to a variety of programming solutions. Algorithms employ iterators to perform operation stored in containers. The relationship between the three components is shown in figure below.

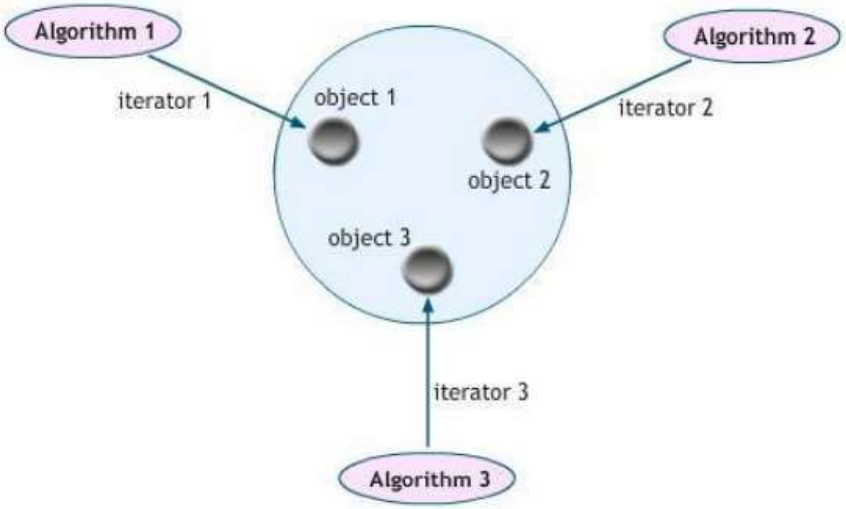


Figure: Components of STL

a) Containers: These are the objects that store the data. The STL containers are implemented by template classes and therefore can be easily modified to hold various types of data.

b) Algorithms: These are the procedures that are used to process the data contained in the containers. The algorithms are implemented by template functions. The STL includes many kinds of algorithms to provide support to tasks such as initializing, coping, searching, sorting, etc.

c) Iterators: These are the objects (like pointers) that point to an element in a container. Iterators can be used to move through the contents of the containers. Iterator connect algorithm with containers and play a vital role in the manipulation of data stored in the containers.

Q. Advantage and disadvantages of inheritance.

Advantages of inheritance:

- Inheritance supports reusability. The features that are defined at parent class can be reused in child class.
- Inheritance will avoid duplication of code.
- The codes of the base class will already be tested and debugged. Hence, reusability via inheritance will enhance reliability.
- The methods of the base class can be overridden in the child class and hence better implementation of the base class methods can be implemented in the derived class.
- The development and maintenance costs will be reduced.
- The time and efforts to develop the software will also be reduced.
- The codes are easy to debug.
- It is easy to partition the work.

Disadvantages of inheritance:

- One of the major drawbacks of inheritance is that the base and child classes are tightly coupled. One cannot be used independently of each other.
- A change in base class will affect the child class.
- The execution speed is reduced as it takes time to jump across various levels of inheritance.
- The data members in the base class are often left unused which leads to memory wastage.
- Inappropriate use of inheritance will cause complexity in the program.

Q. Advantages and disadvantages of friend function:

Advantages of friend function:

- It can access the private data member of class from outside the class.
- It acts as the bridge between two classes by operating on their private data.
- It allows sharing private class information by a non-member function.
- It is able to access members without the need of inheriting classes.

Disadvantage of friend function:

- It violates the law of data hiding principle by allowing access to private members of the class from outside the class.
- The data integrity is breached.
- It is conceptually complex and messy.
- Size of the memory occupied by the objects will be high.

Old/Model/Important Questions:

1. **Explain the message passing formalism with syntax in C++.**
2. **What is data hiding? How do you achieve data hiding in C++? Explain with a suitable example.**
   Or, What is encapsulation? How can encapsulation be enforced in C++? Explain with suitable code.
3. **How inheritance does support reusability?**
   Or, Inheritance supports the reusability characteristics of OOP, Justify your answer.
4. **Explain the different forms of inheritance.**
5. **Define reusability. What are the advantages of software reusability in OOP design.**
6. **List out the advantages and disadvantages of inheritance.**
7. **What are the advantages and disadvantages of using friend function.**
8. **State the principle of substitutability. Explain sub classing for specialization and generalization.**
9. **Differentiate between subclass and subtype.**
10. **What is stack versus heap memory allocation?**
11. **Write short notes on:**
    i. **Standard Template Library**