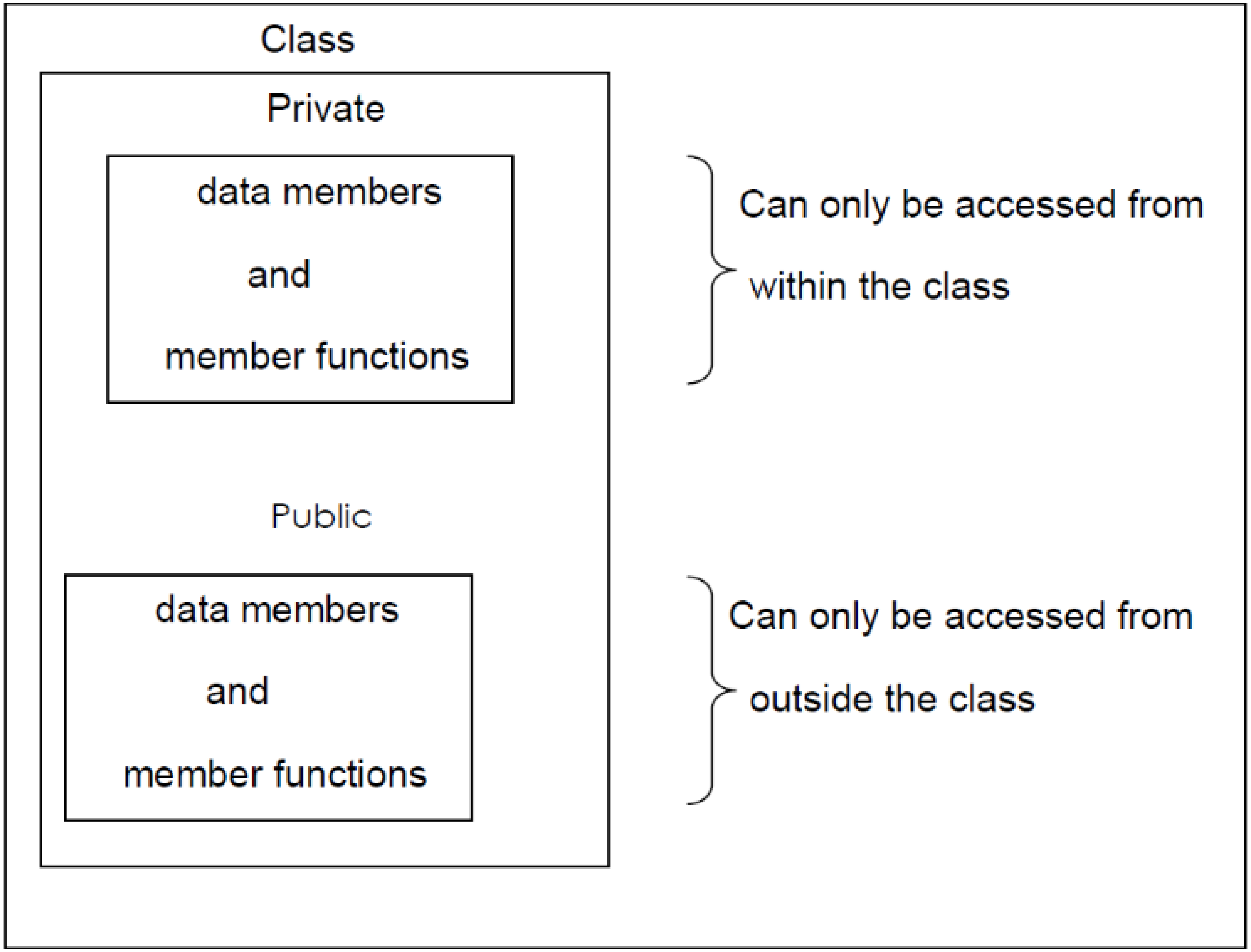
DEFINITION AND DECLARATION OF A CLASS



Class

Private

data members

and

member functions

Can only be accessed from within the class

Public

data members

and

member functions

Can only be accessed from outside the class

The syntax of a class definition is shown below :

```
Class name_of _class
{
private : variable declaration; // data member
                   Function declaration; // Member Function (Method)
protected: Variable declaration;
                    Function declaration;
public : variable declaration;
                  Function declaration;
} ;
```

Copy Constructor

A copy constructor is a member function which initializes an object using another object of the same class.

```
ClassName (const ClassName & old_obj);
```

```
class Point
{
private:
    int x, y;
public:
    Point(int x1, int y1) { x = x1; y = y1; }

    // Copy constructor
    Point(const Point &p2) {x = p2.x; y = p2.y; }

    int getX()          {  return x; }
    int getY()          {  return y; }
};
  int main()
  {
      Point p1(10, 15); // Normal constructor is called here
      Point p2 = p1; // Copy constructor is called here

      // Let us access values assigned by constructors
      cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();
      cout << "\np2.x = " << p2.getX() << ", p2.y = " << p2.getY();

      return 0;
  }
```

```
p1.x = 10, p1.y = 15 p2.x = 10, p2.y =
15
```

Static class members

- A class member can be declared as keyword ***static***
- Only one copy of a ***static*** variable exists – no matter how many objects of the class are created
  - All objects share the same variable
- It can be private, protected or public
- A ***static*** member variable exists before any object of its class is created
- In essence, a ***static*** class member is a global variable that simply has its scope restricted to the class in which it is declared
- It is visible only within the class, but its lifetime is the entire program

Static class members

- When we declare a ***static*** data member within a class, we are not defining it
- Instead, we must provide a definition for it elsewhere, outside the class
- To do this, we re-declare the ***static*** variable, using the scope resolution operator to identify which class it belongs to
- All ***static*** member variables are initialized to **0** by default

```
Class student
{
Static int count; //declaration within class
-----------------
-----------------
-----------------
};
```

The static data member is defined outside the class as :

```
int student :: count; //definition outside class
```

The definition outside the class is a must.

We can also initialize the static data member at the time of its definition as:

int student :: count = 0;

Static variables in a Function:

```
#include <iostream>
#include <string>
using namespace std;
 void demo()
{
    // static variable
    static int count = 0;
    cout << count << " ";
       // value is updated and
    // will be carried to next
    // function calls
    count++;
}
```

```
int main()
{
    for (int i=0; i<5; i++)
        demo();
    return 0;
}
```

Output:

```
0 1 2 3 4
```

Static variables in a class

```
class GfG
{
public:
    static int i;

    GfG()
    {
        // Do nothing
    };
};

int GfG::i = 1;
```

```
int main()
{
    GfG obj;
    // prints value of i
    cout << obj.i;
}
```

```
    Output:
    1
```

Static variables in a class

As the variables declared as static are initialized only once as they are allocated space in separate static storage so, the static variables **in a class are shared by the objects.**

```
class GfG
{
   public:
     static int i;

     GfG()
     {
        // Do nothing
     };
};
```

```
int main()
{
  GfG obj1;
  GfG obj2;
  obj1.i =2;
  obj2.i = 3;

  // prints value of i
  cout << obj1.i<<" "<<obj2.i;
}
```

```
//int GfG::i=1;
```

```
3
3
```

we have tried to create multiple copies of the static variable i for multiple objects. But this didn’t happen.

```
// creating a class
class myClass{
  public:
  static int var;
} ;

// initializing
int myClass:: var = 5;
int main()
{
    // creating two objects of myClass
    myClass obj1, obj2;

    cout << "Value of var: " << obj1.var << endl;

    // changing value of var by using obj2
            obj2.var = 7;

    cout << "New value of var: " << obj1.var << endl;
    return 0;
}
```

```
5
7
```

Static class members

- The principal reason ***static*** member variables are supported by C++ is to avoid the need for global variables
- Member functions can also be ***static***
  - Can access only other ***static*** members of its class directly
  - Need to access ***non-static*** members through an object of the class
  - Does not have a ***this*** pointer
  - Cannot be declared as ***virtual***, ***const*** or ***volatile***
- ***static*** member functions can be accessed through an object of the class or can be accessed independent of any object, via the class name and the scope resolution operator
  - Usual access rules apply for all ***static*** members

```
class Demo
 {
 private:

                int X =10;
               int m=7;
public:
                void fun()
               {
               cout<<"Value of X: " << X  << endl;
               }
};

 int main()
{
       Demo Y;
       Y.fun();
       return 0;
}
```

Output: 10

Access private data member with in class

```
class Demo
 {
 private:

                 int X =10;
                int m=7;
public:
                 void fun()
                {
                cout<<"Value of X: " << X << endl;
                }
};

 int main()
{
        Demo Y;
        Y.fun();
        cout<<"Value of X: " << Y.X << endl;
        return 0;
}
```

ERROR

```
class Demo
 {
 public:

              int X =10;
             int m=7;
public:
              void fun()
             {
             cout<<"Value of X: " << X << endl;
             }
};

 int main()
{
       Demo Y;
       Y.fun();
       cout<<"Value of X: " << Y.X << endl;
       return 0;
}
```

```
10

10
```

```
class Demo
 {
 public:

                static int X = 10;
                int m=7;
public:
                 void fun()
                {
                cout <<"Value of X: " << X  <<  endl;
                }
};

 int main()
{
        Demo Y;
        Y.fun();
        return 0;
}
```

Error static data member initialization

Here static data member is accessing through the member function:

```
class Demo
 {
 public:

                static int X;
                int m=7;
public:
                 void fun()
                {
                cout<<"Value of X: " << X  << endl;
                }
};
Int Demo:: X = 10;

 int main()
{
        Demo Y;
        Y.fun();
        return 0;
}
```

```
10
```

```
class Demo
 {
 private:

                static int X;
                int m=7;
public:
                 void fun()
                {
                cout<<"Value of X: " << X << endl;
                }
};
Int Demo:: X=10;

 int main()
{
        Demo Y;
        Y.fun();
        cout<<"Value of X: " << Y.X << endl;
        return 0;
}
```

error

```
class Demo
 {
 public:

                static int X;
                int m=7;
public:
                 void fun()
                {
                cout<<"Value of X: " << X << endl;
                }
};
Int Demo:: X =10;

 int main()
{
        Demo Y;
        Y.fun();
        cout<<"Value of X: " << Y.X << endl;
        return 0;
}
```

10   10

Here static data member is accessing through the static member function:

```
class Demo
 {
 private:
           static int X;
          int m=7;
public:
           static void fun()
           {
           cout <<"Value of X : " << X << endl;
           }
}; //defining
int Demo :: X =10;

 int main()
{
     Demo Y;
     Y.fun();
     return 0;
}
```

```
Value of X : 10
```

Here non static data member is accessing through the static member function:

```
class Demo
 {
 private:
          int X;
          int m=7;
public:
           static void fun()
           {
           cout <<"Value of X: " << X << endl;
           }
}; //defining
int Demo :: X =10;

 int main()
{
     Demo Y;
     Y.fun();
     return 0;
}
```

error

Accessing static data member without static member function

A static data member can also be accessed through the class name without using the static member function

```
class Demo
{
public:
        static int ABC;
}; //defining
int Demo :: ABC =10;
int main()
{
        cout<<"\nValue of ABC: "<<Demo::ABC;
         return 0;
}
```

```
Value of ABC: 10
```

Write a Program to print through object and class names of static data members

```
class Demo
 {
 private: //static data members
            static int X;
            static int Y;
public:
            //static member function
            static void Print()
            {
            cout<<"Value of X: " << X << endl;
            cout<<"Value of Y: " << Y << endl;
            }
 };
 //static data members initializations
 int Demo :: X =10;
 int Demo :: Y =20;
```

```
Printing through object name:
Value of X: 10
Value of Y: 20
 Printing through class name:
 Value of X: 10
Value of Y: 20
```

```
int main()
{
Demo OB;
//accessing class name with object name
cout<<"Printing through object name:"<<endl;
OB.Print();
 //accessing class name with class name
cout<<"Printing through class name:"<<endl;
Demo::Print();
return 0;
 }
```

Static class members

1 5 2 10

```
class myclass {
   static int x;
public:
   static int y;
   int getX () { return x; }
   void setX (int x) {
      myclass::x = x;
   }
};
int myclass::x = 1;
int myclass::y = 2;
```

```
void main ( ) {
   myclass ob1, ob2;
   cout << ob1.getX () << endl; // 1
   ob2.setX (5);
   cout << ob1.getX () << endl; // 5
   cout << ob1.y << endl; // 2
   myclass::y = 10;
   cout << ob2.y << endl; // 10
   // myclass::x = 100;
}
```

// will produce compiler error x is private

In-line Functions

- ► Main objective of using functions in a program is to save memory.

- ► Every time function is called, it takes a lot of time in executing a series of instructions for tasks.

- ► Ex: jumping to the function, saving registers, pushing arguments into stack and returning into calling function.

Solution: Macros

- ► They are not really functions, therefore the error checking does not occur during the compilation

Solution: Inline function

In-line Functions

Functions that are not actually called but, rather, are expanded in line, at the point of each call.

Advantage

- Have no overhead associated with the function call and return mechanism.
- Can be executed much faster than normal functions.

Disadvantage

If they are too large and called too often, the program grows larger.

In-line Functions

```
inline int even(int x)
{
    return !(x% 2);
}

int main()
{
    if(even(10)) cout << “10 is even\n”;
    // becomes if(!(10% 2))

    if(even(11)) cout << “11 is even\n”;
    // becomes if(!(11% 2))

    return 0;
}
```

- The **inline** specifier is a request, not a command, to the compiler.
- Some compilers will not in-line a function if it contains
  - A **static** variable
  - A **loop**, **switch** or **goto**
  - A **return** statement
  - If the function is **recursive**

Automatic In-lining

- Defining a member function inside the class declaration causes the function to automatically become an in-line function.
- In this case, the **inline** keyword is no longer necessary.
  - However, it is not an error to use it in this situation.
- Restrictions
  - Same as normal in-line functions.

```
// Automatic in-lining
class myclass
{
    int a;
public:
    myclass(int n) {  a =  n; }
    void set_a(int n) {  a =  n; }
     int get_a() {  return a; }
};
```

```
// Manual in-lining
class myclass
{
    int a;
public:
    myclass(int n);
    void set_a(int n);
    int get_a();
};
inline void myclass::set_a(int n)
{
    a =  n;
}
```

An EMPLOYEE class is to contain the following data members and member functions: Data members: EmployeeNumber (an integer), EmployeeName (a string of characters), BasicSalary (an integer), All Allowances (an integer), IT (an integer), NetSalary (aninteger).

Member functions: to read the data of an employee, to calculate Net Salary and to print the values of all the data members. (AllAllowances = 123% of Basic; Income Tax (IT) = 30% of the gross salary (= basic Salary +AllAllowance); Net Salary = Basic Salary + All Allowances – IT)

1. Create Class Employee.
2. Class Employee Contains following data members
   a. Employee_Number as integer
   b. Employee_Name as String
   c. Basic_Salary as integer
   d. All_Allowances as integer
   e. IT as integer
   f. Net_Salary as integer
   g. Gross_Salary as integer

1. Class Employee Contains following members functions
   a. Create function as getdata for accepting information of employee. Like employee name,employee number and basic salary etc.
   b. Create function Net_salary_Calculation to calculate gross salary.
   c. Create function displayInformation to display information about employee.
2. Create main function to call this function of class Employee.

Write a program that uses a class where the member functions are defined inside a class. (Try with different access specifiers)

Write a program that uses a class where the member functions are defined outside a class.

Try with local and global objects

Try with different constructors and definition inside & outside of the class

Try with distractor

Try with function overloading

Write a program using inline function inside and outside of the class

(accessing data members with objects and member functions)

Write a program to demonstrate the use of static data members