// Name: Hasibul Hasan
// Roll: 2003127
Lab No - 4

Module-2

GitHub Link: https://github.com/hasibul-hasib29/cse-1204/blob/master/module_2.cpp

```
#include<iostream>
using namespace std;
class Test{
private:
   int x;
   int y;
   static int z;
   static int sumX;
   static int temp;
   static int yInd;

public:


   Test (int a=0, int b=0){
      x=a; y =b;
      z++;
      sumX+=x;


      if(y>temp){
         yInd =z;
      }
      else temp =y;

   }
   Test(const Test &p){
      x = p.x;
      y = p.y;
      z++;
      sumX +=x;

      if(y>temp){
         yInd =z;
      }
      else temp =y;
   }

   void Initilize(int a, int b){
      x =a;
      y =b;
      sumX +=x;
      if(y>temp){
         yInd =z;
      }
      else temp =y;

   }

   static void getZ(){                    // for display data member z
      cout<<"z = "<<z<<endl;
   }

   void getAll(){
      cout<<"x = "<<x<<endl<<
         "y = "<<y<<endl<<
         "z = "<<z<<endl;
   }

   void getConstData ()const{
      cout<<"x = "<<x<<endl
         <<"y = "<<y<<endl
         <<"z =  "<<z<<endl;
   }
   static void SumOfx() {
      cout<<"the sum of all x = "<<sumX<<endl;
   }

   static void YmaxInd(){
      cout<<"Y's maximum value is class no : "<<yInd<<endl;
   }



};
int Test::z = 0;
int Test::sumX =0;
int Test::yInd=0;
int Test::temp =0;
int main(){
   Test a;
   Test b(4,5);
   Test c(b);
   Test d(8, 7);
   Test e;
   e.Initilize(7, 9);
   e.getConstData();

   Test::SumOfx();
   Test::YmaxInd();


   return 0;
}
```

OUTPUT:
x = 7
y = 9
z = 5
the sum of all x = 23
Y's maximum value is class no : 5

MODULE-3

Table: inheritance among Father-->Son -→Grandson class

Class		In son class			In Grandson class		
Son	Grandson	money	gold	land	Money	Gold	Land
public	public	NO	YES	YES	NO	YES	YES
protected	public	NO	YES	YES	NO	YES	YES
private	public	NO	YES	YES	NO	NO	NO
public	protected	NO	YES	YES	NO	YES	YES
protected	protected	NO	YES	YES	NO	YES	YES
private	protected	NO	YES	YES	NO	NO	NO
public	private	NO	YES	YES	NO	YES	YES
protected	private	NO	YES	YES	NO	YES	YES
private	private	NO	YES	YES	NO	NO	NO

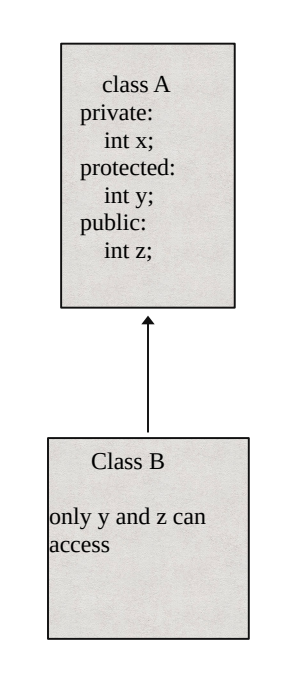
Topic: Types of Inheritance

1. Single Inheritance:

```
class A{
private:
    int x;
protected:
   int y;
public:
  int z;
}
```

```
class B:public A{
// x is not accessible
  void getz(){ cout<<”z is “ <<z<<endl;

 void gety(){ cout<<”y is “        <<y<<endl;
}
```

```
int main(){
B b;
b.getz();
b.gety();
return 0;
}
```



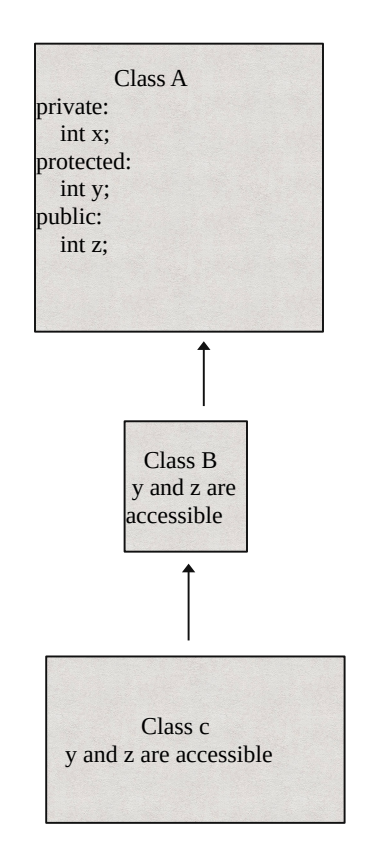
2. Multi-level Inheritance:

```
class A{
private:
   int x;
protected:
   int y;
public:
   int z;
}
```

```
class B:public A{

}
```

```
class C:public
B{
// x is not accessible
   void getz(){ cout<<”z is “
<<z<<endl;

 void gety(){ cout<<”y is “
<<y<<endl;
}
```

```
int main(){
C c;
c.getz();
c.gety();
return 0;
}
```



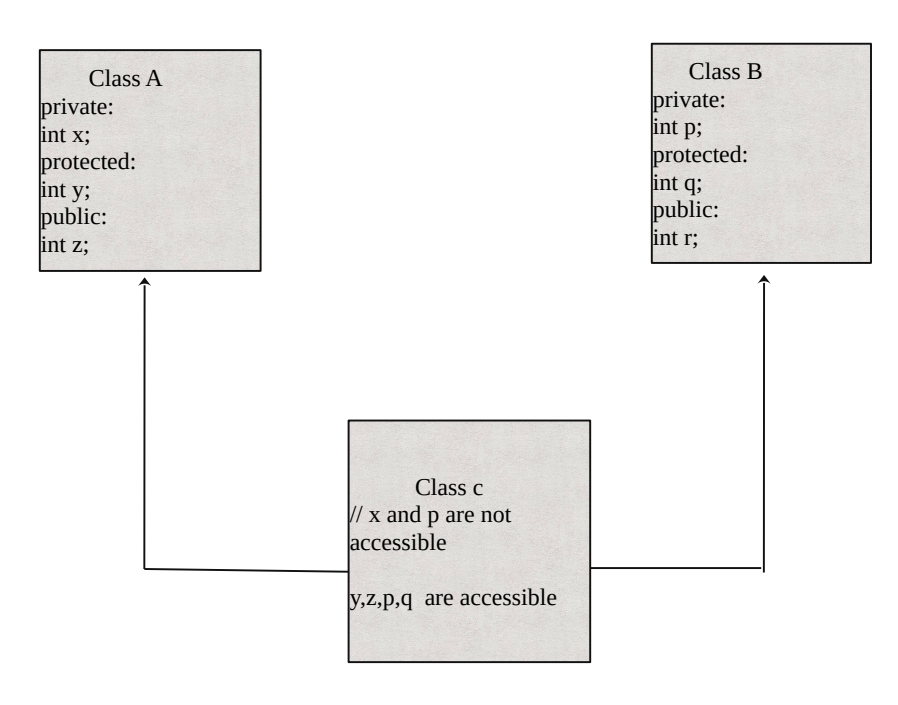
3. Multiple Inheritance:

```
class A{
private:
int x;
protected:
int y;
public:
int z;
}
```

```
class B{
private:
int p;
protected:
int q;
public:
int r;
}
```

```
class C:public A,
Public B{
 void getz(){ cout<<”z is “
<<z<<endl;

 void gety(){ cout<<”y is “
<<y<<endl;

void getq(){ cout<<”q is “
<<q<<endl;

 void getr(){ cout<<”r is “
<<r<<endl;

// x and p are not accessible
}
```

```
int main(){
C c;
c.getz();
c.gety();
return 0
}
```



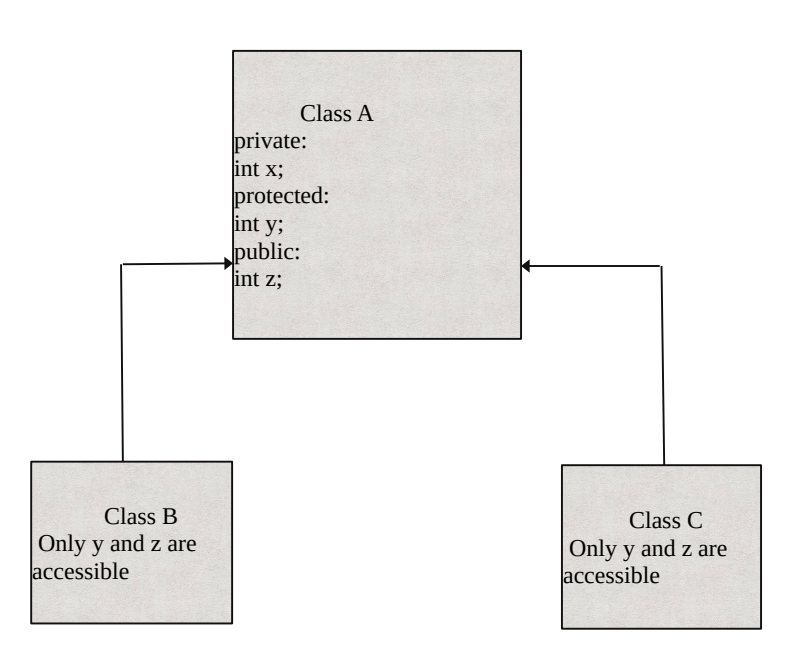
4. Heirarchical inheritance:

```
class A{
private:
int x;
protected:
int y;
public:
int z;
}
```

```
class B:public A
{
// x is not accessible
 void getz(){ cout<<”z is “
<<z<<endl;

 void gety(){ cout<<”y is “
<<y<<endl;
}
```

```
class C:public A
{
// x is not accessible
 void getz(){ cout<<”z is “
<<z<<endl;

 void gety(){ cout<<”y is “
<<y<<endl;
}
```

```
int main(){
B b;
C c;
b.getz();
b.gety();
c.getz();
c.gety();

return 0;
}
```



5. Hybrid Inheritance:

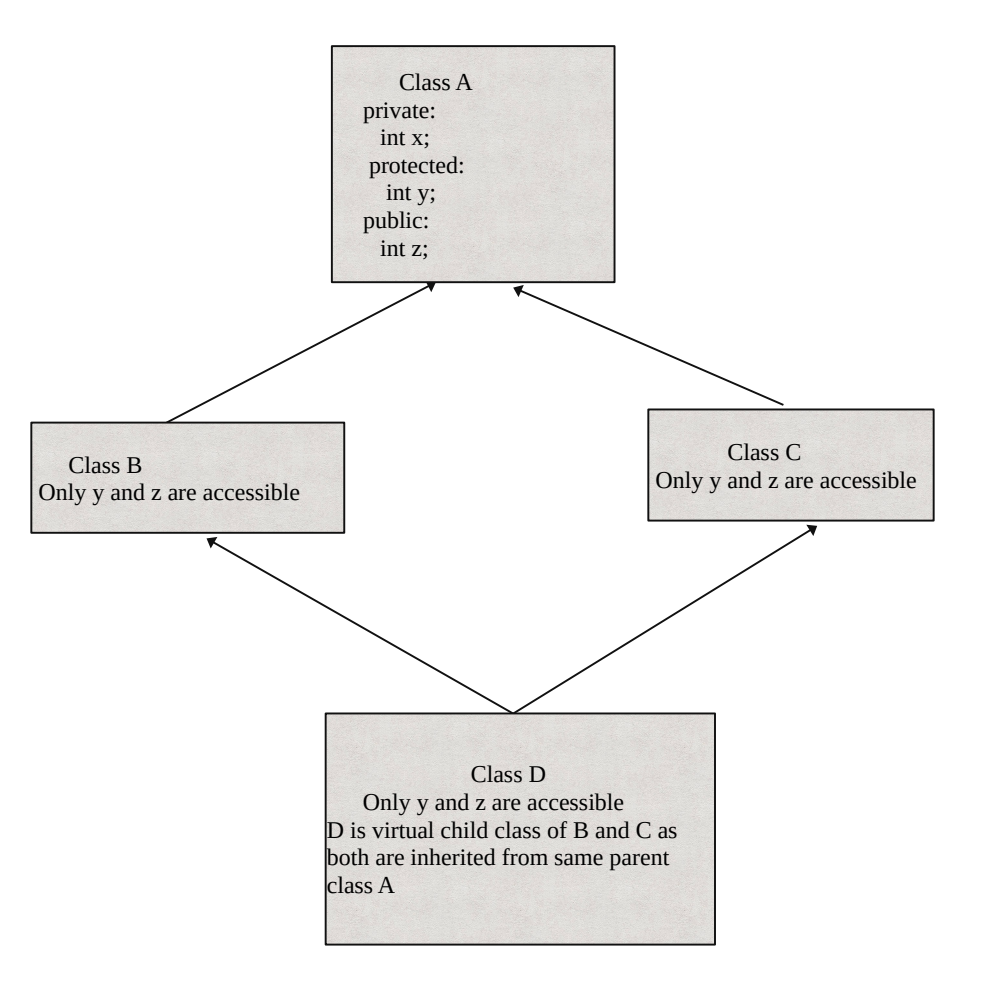
```
class A{
private:
int x;
protected:
int y;
public:
int z;
};
```

```
class B:  virtual
public A
{
// without virtual , d
will be ambiguous
};
```

```
class C:  virtual public
A
{
// without virtual , d
will be ambiguous
};
```

```
class D:public
B, public C {
// x is not accessible
 void getz(){ cout<<”z is
“ <<z<<endl;

 void gety(){ cout<<”y is
“          <<y<<endl;
};
```

```
int main(){
D d;
d.gety();
d.getz();
return 0;
}
```



Topic: [Constructor and Destructor in Inheritance]

1.Single Inheritance:

```
class A
{
private:
    int ax;

public:
    A(int a)
    {
        ax = a;
        cout << "constructor A"
<< endl;
    }
    int getax()
    {
        return ax;
    }
    ~A()
    {
        cout << "destructor A"
<< endl;
    }
};
```

```
class B : public A
{
private:
    int bx;

public:
    B(int b) : A(b)
    {
        bx = b;
        cout << "constructor B"
<< endl;
    }
    void sum()
    {
        cout << "sum = " <<
getax() + bx << endl;
    }
    ~B()
    {
        cout << "Destructor B"
<< endl;
    }
};
```

```
int main()
{
    B b(10);
    b.sum();
    return 0;
}
```

OUTPUT:
constructor A
constructor B
sum = 20
Destructor B
destructor A

Sequence of the execution : Constructor A() → constructor B() → void sum() → Destructor B() → Destructor A.

2. Multilevel Inheritance:

```
class A
{
private:
    int ax;

public:
    A(int a)
    {
        ax = a;
        cout <<
"constructor A" <<
endl;
    }
    int getax()
    {
        return ax;
    }
    ~A()
    {
        cout << "destructor
A" << endl;
    }
};
```

```
class B : public A
{
private:
    int bx;

public:
    B(int b) : A(b)
    {
        bx = b;
        cout <<
"constructor B" <<
endl;
    }
    void sum()
    {
        cout << "sum = "
<< getax() + bx <<
endl;
    }
    ~B()
    {
        cout <<
"Destructor B" << endl;
    }

    int getbx(){
        return bx;
    }
};
```

```
class C: public B{
private:
    int cx;
public:
    C(int c):B(c){
        cx = c;
cout << "constructor C"
<< endl;
    }

    void sum(){
        cout<<"sum  =
"<<getax()+getbx()
+cx<<endl;
    }

    ~C(){
        cout<<"Destructor
C"<<endl;
    }
};
```

```
int main()
{
    C c(5);
    c.sum();
    return 0;
}
```

OUTPUT:
constructor A
constructor B
constructor C
sum = 15
Destructor C
Destructor B
destructor A

Execution sequence: constructor A → Constructor B → constructor C → c.sum() → Destructor C → Destructor B → Destructor A.

3. Multiple inheritance:

```
class A
{
private:
    int ax;

public:
    A(int a)
    {
        ax = a;
        cout << "constructor
A" << endl;
    }
    int getax()
    {
        return ax;
    }
    ~A()
    {
        cout << "destructor
A" << endl;
    }
};
```

```
class B
{
private:
    int bx;

public:
    B(int b)
    {
        bx = b;
        cout << "constructor
B" << endl;
    }

    ~B()
    {
        cout << "Destructor
B" << endl;
    }

    int getbx(){
        return bx;
    }
};
```

```
class C:public A,  public
B{
private:
    int cx;
public:
    C(int c):A(c) , B(c) {
        cx = c;
        cout<<"constructor
C"<<endl;
    }

    void sum(){
        cout<<"sum  =
"<<getax()+getbx()
+cx<<endl;
    }

    ~C(){
        cout<<"Destructor
C"<<endl;
    }
};
```

```
int main()
{
    C c(5);
    c.sum();
    return 0;
}
```

OUTPUT:
constructor A
constructor B
constructor C
sum = 15
Destructor C
Destructor B
destructor A

Execution sequence: constructor A → Constructor B → constructor C → c.sum() → Destructor C → Destructor B → Destructor A.

4. Heirarchical inheritance:

```
class A
{
private:
    int ax;

public:
    A(int a)
    {
        ax = a;
        cout << "constructor A"
<< endl;
    }
    int getax()
    {
        return ax;
    }
    ~A()
    {
        cout << "destructor A"
<< endl;
    }
};
```

```
class B: public A
{
private:
    int bx;

public:
    B(int b):A(b)
    {
        bx = b;
        cout << "constructor B"
<< endl;
    }

    ~B()
    {
        cout << "Destructor B"
<< endl;
    }

    int getbx(){
        return bx;
    }
};
```

```
class C:public A{
private:
    int cx;
public:
    C(int c):A(c){
        cx = c;
        cout<<"constructor
C"<<endl;
    }

    void sum(){
        cout<<"sum  =
"<<getax()+cx<<endl;
    }
//bx is inaccessible
    ~C(){
        cout<<"Destructor
C"<<endl;
    }
};
```

```
int main()
{   B b(7);
    C c(5);
    c.sum();
    return 0;
}
```

OUTPUT:
constructor A
constructor B
constructor A
constructor C
sum = 10
Destructor C
destructor A
Destructor B
destructor A

Execution sequence: constructor A → constructor B → constructor A → constructor C → c.sum() → destructor C → destructor A → Destructor B → destructor A .

5. Hybrid (Diamond) inheritance [virtual class] :

```
class A
{
private:
    int ax;

public:
    A(int a)
    {
        ax = a;
        cout <<
"constructor A" <<
endl;
    }
    int getax()
    {
        return ax;
    }
    ~A()
    {
        cout <<
"destructor A" <<
endl;
    }
};
```

```
class B:virtual public
A
{
private:
    int bx;

public:
    B(int b):A(b)
    {
        bx = b;
        cout <<
"constructor B" <<
endl;
    }

    ~B()
    {
        cout <<
"Destructor B" <<
endl;
    }

    int getbx(){
        return bx;
    }
};
```

```
class C:virtual public
A{
private:
    int cx;
public:
    C(int c):A(c){
        cx = c;

cout<<"constructor
C"<<endl;
    }

    int getcx(){
        return cx;

    }

    ~C(){

cout<<"Destructor
C"<<endl;
    }
};
```

```
class D: public B,
public C{
private:
    int dx;
public:
    D(int d):A(d),
B(d), C(d){
        dx = d;

cout<<"constructor
D"<<endl;
    }
    void sum(){
        cout<<"sum of
ax + bx + cx + dx =
"<<getax()+getbx()
+getcx()+dx<<endl;
    }

    ~D(){

cout<<"destructor
D"<<endl;
    }

};
```

```
int main()
{   D d(6);
    d.sum();

    return 0;
}
```

OUTPUT:
constructor A
constructor B
constructor C
constructor D
sum of ax + bx + cx + dx = 24
destructor D
Destructor C
Destructor B
destructor A

Execution sequence: constructor A → constructor B → constructor C → constructor D → d.sum() → destructor D → destructor C → destructor B → destructor A