# Concept of const, static, friend & inner class

# Const Data Member in Class

- Data members of a class could be made constant.
- Constant data members must be initialized through constructors using constructor initialization list.
- Use of constant inside a class means, “It will be constant for the lifetime of the Object”
- Each object will have a different copy of the constant data member probably having different value.

# Initializing Const Data Member of Class

```
class Data {
    int val;
    const int someConst;
public:
    Data() : someConst(0) { // constructor initializer syntax
        val = 0;
    }
    Data( int val, int c ) : someConst(c) {
        this->val  = val;
    }
};
```

# Const Object & Const Member Function

- To create constant object use const keyword
  - const Pixel p1(2,3); // statement in main
- No data members of Constant object could be changed.
- const objects can invoke const member function, which guarantees that no data members of the object will be changed.
- Characteristic of const functions is it is 'Read Only' function.
- Constant functions can not write data members.
- To make a function constant place the const specifier after argument list in definition and declaration of function.
- Any function which is not going to modify data members should be made constant.
- Constant functions can be invoked by non constant objects also.
- Constructor and destructor can not be made const.

# Example

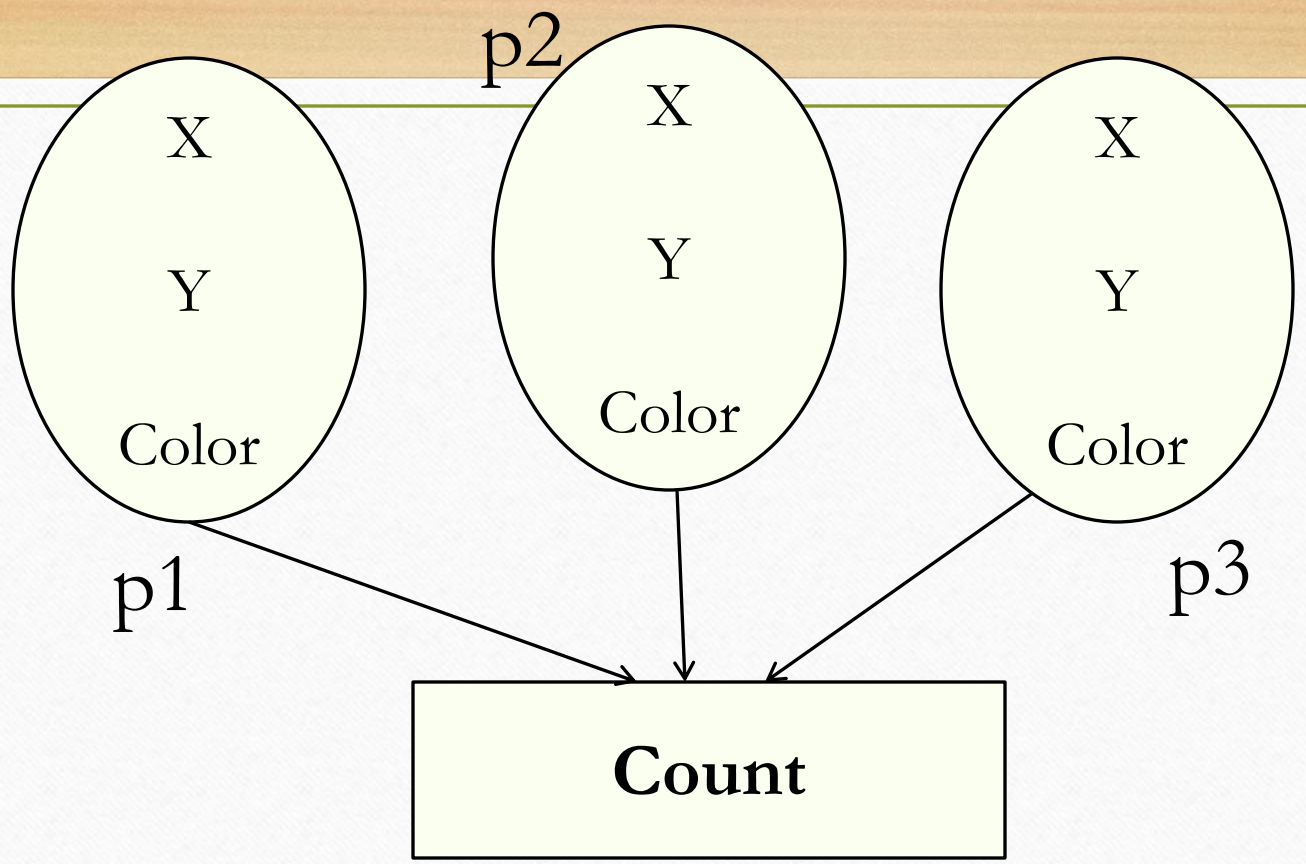
```
class Integer {
    int i;
public:
    Integer( int i ) {
        this->i = i;
    }
    void setI( int i ) {
        this->i = i;
    }
    int getInt() const; // const keyword makes the function constant
};

//getInt can not make changes to data member I as it is const function

int Integer::getInt() const { //const keyword must be used in def also
    return i;
}
int main() {
    const Integer i1( 10 );
    i1.getInt();
    //i1.setInt(); Error can not execute non const fun on const object
    return 0;
}
```

# Static Data Member

- Useful when all objects of the same class must share a common item of information.
- Data to be shared by all objects is stored in static data members.
- There is single piece of storage for static data members.
- It's a class variable. Static data members could be made private, public or protected.
- Static data members are class members, they belong to class and not to any object.
- The static data member should be created and initialized before the main() program begins.




A single of a static member is created per class. All objects share the same copy of static variable.
Compiler will not allocate memory to static for each object.
If static data members are declared and not defined linker will report an error.
Static data members must be defined outside the class. Only one definition for static data members is allowed.

# Static Member Function

- Static member functions can access static data members only.
- Static member function is invoked using class name
  - *class name : : function name()*
- Pointer **this is never passed to a static member function**
- Can be invoked before creation of any object.

```
class ABC
{
   private:
      static int ref;
      static ABC* self;
      ABC()
      {
            cout<<"\n constructor..";
      }

   public:
      static ABC* getInstance();
      static int getref();
};

int ABC::getref()
{
      return ref;
}
```

```
ABC* ABC::getInstance()
{
   if (self == 0)
   {
     //  ref=1;
            self = new ABC();
   }
   ref=ref+1;
   return self;
}


ABC* ABC::self;
int ABC::ref=0;
```

```
int main()
{
   //ABC obj;
   ABC* ptr = ABC::getInstance();
   ABC* sptr = ABC::getInstance();
   ABC* pptr = ABC::getInstance();

   std::cout << ptr << std::endl;
   std::cout << sptr << std::endl;
   std::cout << ABC::getref();
}
```

# Friend class and functions

A class grants access privileges to its friends.
Friend class can access private data members of class.

```
class A{
  int data;
  public:
     void setdata() {
            cout<<"Enter a number";
            cin>>data;
      }
      void display() {
            cout<<"Value of A class Data from class
A function "<<data<<endl;
      }
      friend class B;
};
```

```
class B{
      public:
            void display(A obj)
            {
                  cout<<"Value of A Class
data from class B function
"<<obj.data<<endl;
            }
};
```

```
int main() {
      A obj;
      B obj1;
      obj.setdata();
      obj.display();
      obj1.display(obj);
}
```

```
class B;
class A {
 int Adata;
 public:
  void getdata()
  {
    cout<<"Enter value";
    cin>>Adata;
  }
  void display()
  {
    cout<<"Value of Adata "<<Adata<<endl;
  }
  void friend swap(A &, B &);
};
```

```
class B{
      int Bdata;
      public:
            void getdata()
            {
                  cout<<"Enter value of B class";
                  cin>>Bdata;
            }
            void display()
            {
                  cout<<"Value of Bdata "<<Bdata<<endl;
            }
            void friend swap(A &obj1, B &obj2);
};
```

```
void swap(A &obj1, B &obj2)
{  int t;
   t=obj1.Adata;  obj1.Adata=obj2.Bdata;  obj2.Bdata=t;
   obj1.display();
}
```

# Friendship rules

The privileges of friendship aren't inherited.

> Derived classes of a friend aren't necessarily friends. If class **myClass** declares that class **Base** is a friend, classes derived from **Base** don't have any automatic special access rights to **myClass** objects.

The privileges of friendship aren't transitive.

> A friend of a friend isn't necessarily a friend. If class **Tom** declares class **Jerry** as a friend, and class **Jerry** declares class **Tuffy** as a friend, class **Tuffy** doesn't necessarily have any special access rights to **Tom** objects.

The privileges of friendship aren't reciprocal.

> If class **Tom** declares that class **Jerry** is a friend, **Jerry** objects have special access to **Tom** objects but **Tom** objects do not automatically have special access **Jerry** objects.

# Inner class / Nested class

An inner class is a class which is declared in another outer class.
The inner class is a member and as such has the same access rights as any other member.
The members of an outer class have no special access to members of inner class, the usual access rules shall be obeyed.

```
class outer {     int outer_i;
 public:
     outer() { outer_i=20;}
     void disp_outer() { cout<<"\nouter_i = "<<outer_i;}

     class inner {   int inner_i;
      public :
      inner() { inner_i = 30;}
      void disp_inner(outer o) {
          cout<<"\n outer_i from inner class = "<<o.outer_i;
          cout<<"\n inner_i = "<<inner_i; }
     };
};
```

```
int main()
{
    outer o;
    outer::inner obj;
    obj.disp_inner(o);
    return 0;
}
```