

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
};

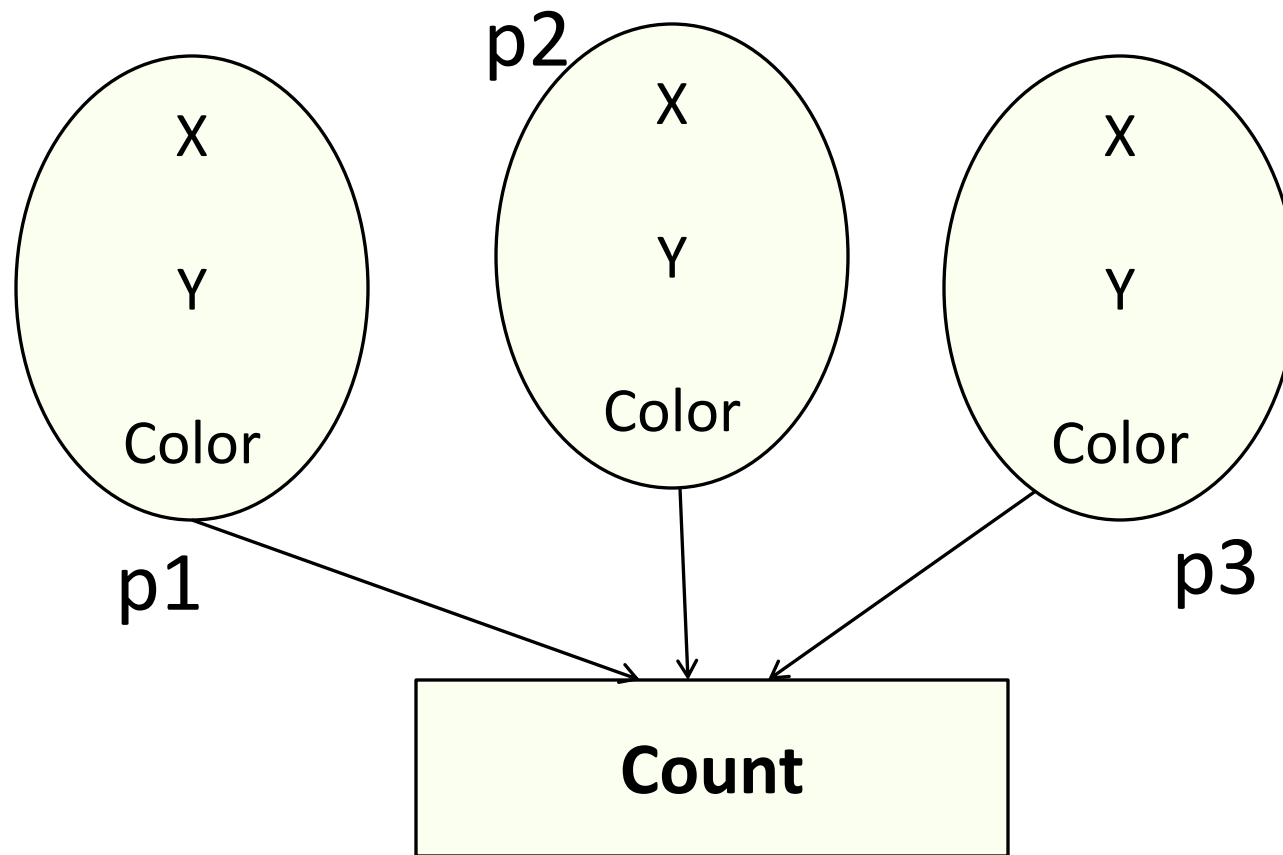
//getInt can not make changes to data member I as it is const function

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 as 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 copy 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.

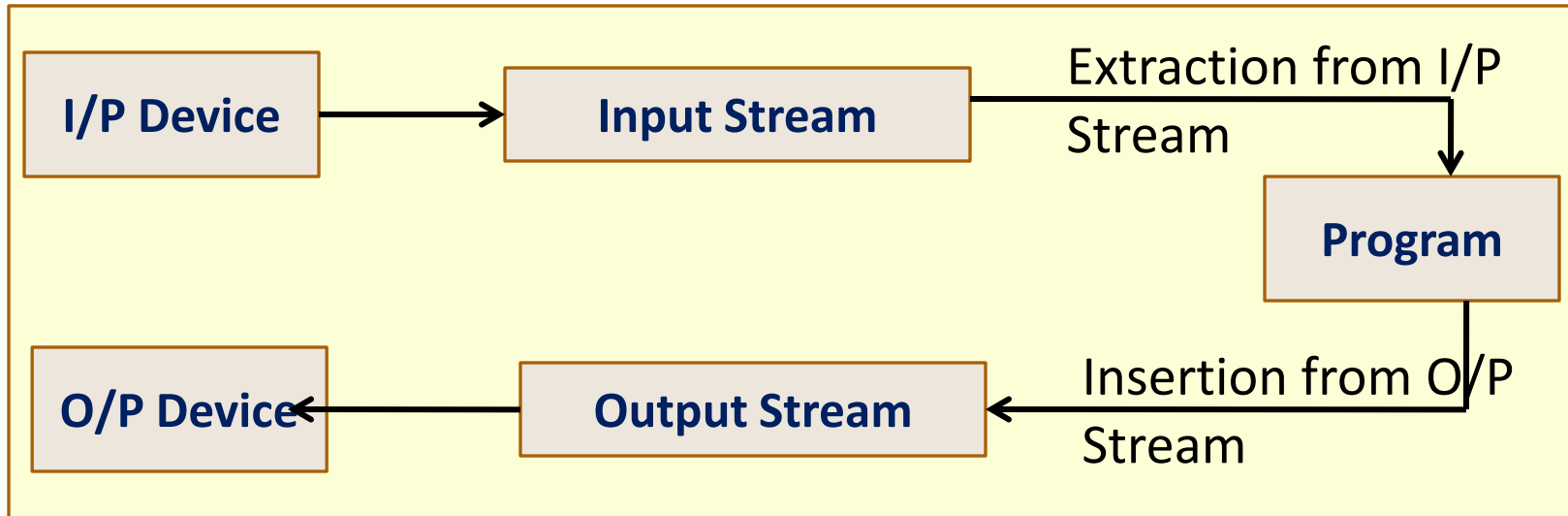
The members of an outer class have no special access to members of inner class, the usual access rules shall be obeyed.

```
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;
}
```

I/O and File Handling



Data Stream- Console IO

```
include<iostream>  
using namespace std;
```

File

A file is a collection on information, usually stored on a computer's disk. Information can be saved to files and then later reused.

File Name-

- All files are assigned a name that is used for identification purposes by the operating system and the user.

File Name and Extension

- MYPROG.BAS
- MENU.BAT
- INSTALL.DOC
- CRUNCH.EXE
- BOB.HTML
- 3DMODEL.JAVA
- INVENT.OBJ
- PROG1.PRJ
- ANSI.SYS
- README.TXT

File Contents

BASIC program
DOS Batch File
Documentation File
Executable File
HTML (Hypertext Markup Language) File
Java program or applet
Object File
Borland C++ Project File
System Device Driver
Text File

File Handling Steps

Using a file in a program is a simple three-step process

- The file must be opened. If the file does not yet exist, opening it means creating it.
- Information is then saved to the file, read from the file, or both.
- When the program is finished using the file, the file must be closed.

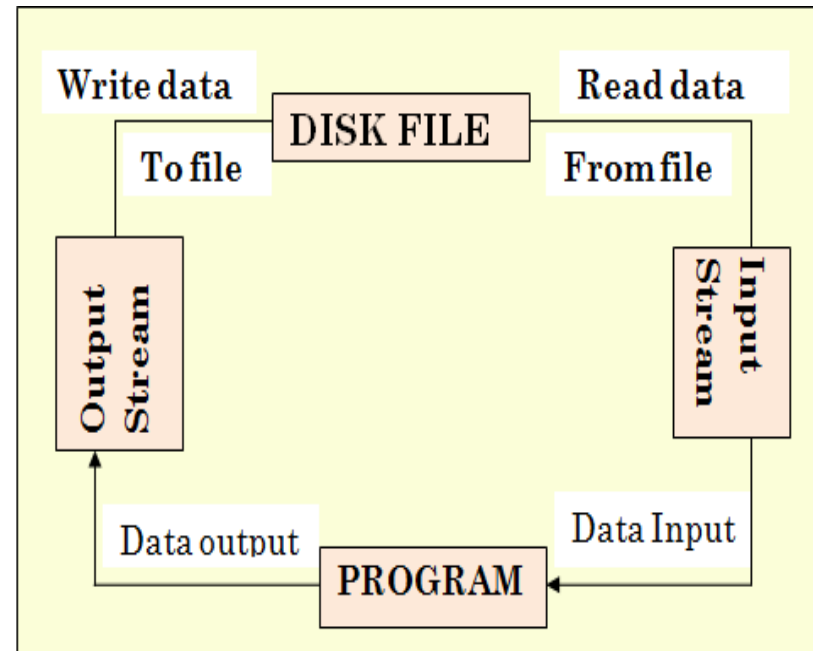
Using Input / Output Stream

Streams act as an interface between files and programs. In C++ . A stream is used to refer to the flow of data from a particular device to the program's variables.

Every stream is associated with a class having member functions and operations for a particular kind of data flow.

- File -> Program (Input stream) - reads
- Program -> File (Output stream) – write

All designed into `fstream.h` and hence needs to be included in all file handling programs.



Using Input / Output File

stream - a sequence of characters

- interactive (iostream)
 - **istream** - input stream associated with **keyboard**.
 - **ostream** - output stream associated with **display**.
- file (fstream)
 - **ifstream** - defines new input stream (normally associated with a file).
 - **ofstream** - defines new output stream (normally associated with a file).

- Stream of bytes to do input and output to different devices.
- Stream is the basic concepts which can be attached to files, strings, console and other devices.
- User can also create their own stream to cater specific device or user defined class.

Opening and Closing File

Before data can be written to or read from a file, the file must be opened.

- `ifstream inputFile;`
`inputFile.open("customer.dat");//opens file in read mode`
- `ofstream outputFile;`
`outputFile.open("customer.dat");//opens file in write mode`
- `fstream iofile;`
`iofile.open("customer.dat", filemode);// opens file in specified mode`

Closing File- A file should be closed when a program is finished using it

- `inputFile.close();`
- `outputFile.close();`
- `Iofile.close();`

File Modes

<code>ios::app</code>	Append mode. If the file already exists, its contents are preserved and all output is written to the end of the file. By default, this flag causes the file to be created if it does not exist.
<code>ios::ate</code>	If the file already exists, the program goes directly to the end of it. Output may be written anywhere in the file.
<code>ios::binary</code>	Binary mode. When a file is opened in binary mode, information is written to or read from it in pure binary format. (The default mode is text.)
<code>ios::in</code>	Input mode. Information will be read from the file. If the file does not exist, it will not be created and the open function will fail.
<code>ios::nocreate</code>	If the file does not already exist, this flag will cause the open function to fail. (The file will not be created.)
<code>ios::noreplace</code>	If the file already exists, this flag will cause the open function to fail. (The existing file will not be opened.)
<code>ios::out</code>	Output mode. Information will be written to the file. By default, the file's contents will be deleted if it already exists.
<code>ios::trunc</code>	If the file already exists, its contents will be deleted (truncated). This is the default mode used by <code>ios::out</code> .

Writing/Reading Data From/To File

Writing data to file-

- The stream insertion operator (<<) may be used to write information to a file.
 `outputFile << "I love C++ programming !"`
- File output may be formatted the same way as screen output.

Reading Data From File-

- The stream extraction operator (>>) may be used to read information from a file.

End of File Detection-

- The eof() member function reports when the end of a file has been encountered.
 `if (inFile.eof())`
 `inFile.close();`

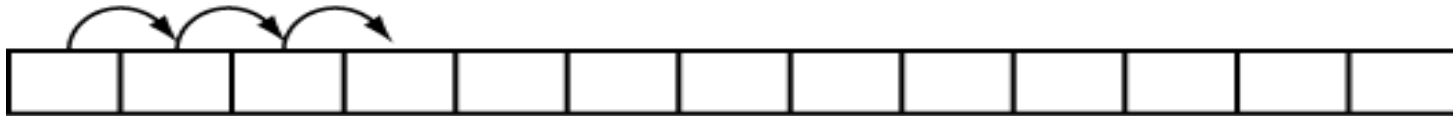
Random Access File

Random Access means non-sequentially accessing information in a file.

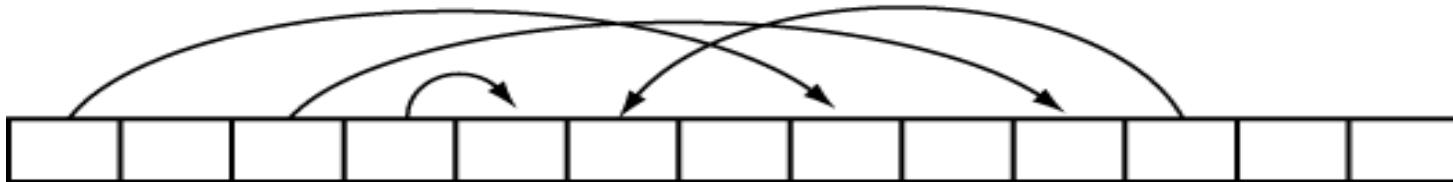
Different modes to operate Random AccessFile-

- `ios::beg` : The offset is calculated from the beginning of the file.
- `ios::end` : The offset is calculated from the end of the file.
- `ios::cur` : The offset is calculated from the current position.

Sequential Access



Random Access



seekp() and seekg()

Statement	How it affects the position
<code>File.seekp(32L, ios::beg);</code>	Sets the write position to the 33 rd byte (byte 32) from the beginning of the file.
<code>file.seekp(-10L, ios::end);</code>	Sets the write position to the 11 th byte (byte 10) from the end of the file.
<code>file.seekp(120L, ios::cur);</code>	Sets the write position to the 121 st byte (byte 120) from the current position.
<code>file.seekg(2L, ios::beg);</code>	Sets the read position to the 3 rd byte (byte 2) from the beginning of the file.
<code>file.seekg(-100L, ios::end);</code>	Sets the read position to the 101 st byte (byte 100) from the end of the file.
<code>file.seekg(40L, ios::cur);</code>	Sets the read position to the 41 st byte (byte 40) from the current position.
<code>file.seekg(0L, ios::end);</code>	Sets the read position to the end of the file.

tellp() and tellg()

tellp returns a long integer that is the current byte number of the file's write position.

tellg returns a long integer that is the current byte number of the file's read position.

```
#include<fstream>
#include<iostream>
using namespace std;
int main()
{
    char ch;
    ofstream f1("first.txt");

    f1<<"Hi";
    f1<<" Hello";
    f1<<" Good Afternoon";
    f1.put('C');

    f1.close();
```

```
ifstream f2("first.txt");
while(!f2.eof())
{
    ch=f2.get();
    cout<<ch;
}
f2.close();
}
```

Program to write to file

```
#include<fstream>
#include<iostream>
using namespace std;
int main()
{
    char arr[]="Infoway Technologies Pvt. Ltd., Pune";
    ofstream f1("text1.txt");
    f1<<arr;
    /* or
    for(int i=0;arr[i]!='\0';i++)
    {
        f1.put(arr[i]);
    }
    */ f1.seekp(5,ios::beg);
    f1.put('P');
    f1.put('p');
}
```

Program to read from file

```
#include<fstream>
#include<iostream>
using namespace std;
int main()
{
    char ch;
    ifstream f1("text1.txt");
    f1.seekg(2,ios::beg);
    while(f1)
    {
        f1.get(ch);
        cout<<ch;
        /* if(ch=='E')
            f1.seekg(1,ios::cur);
        f1.get(ch);
        cout<<ch;
        */
    }
}
```

Binary / Record File

```
#include<fstream>
#include<iostream>
using namespace std;
class student{
    int rno;
    char name[15];
public:
    void set_data()
    {
        cout<<"Enter Rno";
        cin>>rno;
        cout<<"\nEnter Name";
        cin>>name;
    }
    void display()
    {
        cout<<"Rno: "<<rno;
        cout<<"Name: "<<name<<endl;
    }
};
```

```
int main()
{
    ofstream f1;
    ifstream f2;
    f1.open("mydata.dat");
    student obj1,obj2;
    obj1.set_data();
    f1.write((char *)&obj1,sizeof(obj1));
    obj1.display();
    f1.close();

    f2.open("mydata.dat");
    f2.read((char *)&obj2,sizeof(obj2));
    obj2.display();
    f2.close();
}
```