Lecture 9.1

Topics

1. Copy Constructor – Revisited
2. Class Destructor – `ClassName::~ClassName()`
3. Multi-file Compilation – **`Fraction`** Objects
4. String in C++ – **`string`** Class

---

1. Copy Constructor – Revisited

Recall that there can be as many constructors as one would "like" to have. One of these possible constructors is the "**copy constructor**".

In general, an argument for a constructor **must not belong** to the same class that the constructor is defined for. That means a class **`CC`** constructor must not take an argument of type **`CC`**.

However, a **reference argument** is allowed. If a constructor of class **`CC`** takes an argument of type **`CC`**&, that is, **`CC`** reference, then this constructor is called a **copy constructor**.

If the code for the copy constructor is not explicitly provided then the compiler will automatically provide it. The compiler version would have the same effect as,

```
Fraction::Fraction( const Fraction& frOld ); // prototype


// definition
Fraction::Fraction( const Fraction& frOld ) {
  iNum = frOld.iNum;
  iDenom = frOld.iDenom;
}
```

Note that the argument of this constructor is the reference of some existing `Fraction` object (e.g., `frOld` in the above example); and a direct access of the private member (e.g., `frOld.iNum`) is legal for this case.

2. Class Destructor – `ClassName::~ClassName()`

Recall that there is one and only destructor that would be called to clean up after the object just before it is disappeared (i.e., out of scope).

Specifically, for some class –

(1) Each class can have ONE AND ONLY ONE destructor.

(2) A destructor is a function that has

- No argument,
- No return value,
- The same name as that of class preceded by a **~** (tilde).

Syntactically, the destructor will have the following form:

`ClassName::~ClassName()`

3. Multi-file Compilation – `Fraction` Objects

There is a conventional way of designing a computer program so that specification will be separate from implementation. A complete program will be organized in several files as follows,

(a) Specification file – `*.h`

(b) Implementation file – `*.cpp`

(c) Application driver file – `*.cpp`

3.1 Specification File – `*.h` File

All declarations and specifications, such as `struct` or `class`, would be put in some header file.

A header file that contains a class declaration is called a **class specification file**. Usually, the specification is stored as a `*.h` file; for example, `Fraction.h` for a single `Fraction` class.

3.2 Implementation File

The implementation file contains function definitions and member function definitions.

Usually, the implementation file is saved with the same name as the specification file but with different extensions such as `*.c` or `*.cpp`; for example, `Fraction.cpp` corresponding to its specification file of `Fraction.h`.

3.3 Application Driver File

This application driver file is where objects are created and functions are being called to produce the required solution.

This file will include a call to the specification file, implementation, or both.

3.4 Example – Multi-File Setup

```
// FILE #1 - Class Specification File
/**
 * Program Name: myfraction.h
 * Discussion:  Class with and without
 *              Data Encapsulation
 */

#ifndef MYFRACTION_H
#define MYFRACTION_H

class Fraction {
public:
  Fraction();
  Fraction( const Fraction& arg );
  Fraction( int arg );

  ~Fraction();

  int getNum();
  void setNum( int arg );

  int getDenom();
  void setDenom( int arg );

private:
  int iNum;
  int iDenom;
};
```

```
#endif

// FILE #2 - Implementation File
/**
 *Program Name:  myfraction.cpp
 *Discussion:    Implementation File:
 *                 Objects in Functions -
 *                   Fraction Class
 */
#include <iostream>
#include "myfraction.h"
using namespace std;

Fraction::Fraction() {
  iNum = 0;
  iDenom = 1;
}

Fraction::Fraction( const Fraction& arg ) {
  iNum = arg.iNum;
  iDenom = arg.iDenom;
}

Fraction::Fraction( int arg ) {
  iNum = arg;
  if ( arg ) {
    iDenom = arg;
  } else {
    iDenom = 1;
  }
}

Fraction::~Fraction() {
  cout << "\nDestructor Call!" << endl;
}

int Fraction::getNum() {
  return iNum;
}

void Fraction::setNum( int arg ) {
  iNum = arg;
  return;
}

int Fraction::getDenom() {
  return iDenom;
}

void Fraction::setDenom( int arg ) {
  if ( arg ) {
    iDenom = arg;
  } else {
    iDenom = 1;
  }

  return;
}
```

```
// FILE #3 – Application Driver File
/**
 *Program Name:  cis25FractionDriverVer1.cpp
 *Discussion:    Objects in Functions -
 *                 Fraction Class
 */
#include <iostream>
#include "myfraction.h"
using namespace std;

int main( void ) {
  Fraction frA; // What can we do with this prfrA object?

  // print information

  cout << "Numerator : " << frA.getNum() << endl;
  cout << "Denominator : " << frA.getDenom() << endl;

  frA.setNum( 2 );
  frA.setDenom( 3 );

  Fraction* frPtr;

  frPtr = new Fraction();

  delete frPtr;

  frPtr = new Fraction( frA );

  delete frPtr;

  frPtr = new Fraction( 2 );

  delete frPtr;

  return 0;
}
```

OUTPUT

```
Numerator : 0
Denominator : 1

Destructor Call!

Destructor Call!

Destructor Call!

Destructor Call!
```

4. Strings in C++ – `string` class

C++ has a standard `string` class where the header `<string>` is required to access this class. This `string` class provides many supporting methods to manipulate these `string` objects.

A `string` object is dynamic in nature; it can change content and size but not the completely remove itself after defined.

Note that, in this particular case, the name of the class starts out with a lower case character '**s**'; but **string** is the name of the given class.

4.1 `string` Class – Skeleton

A definition with simplified view of `string` class is given below.

```
class string {
public:
  string(); // Empty string
  string( const string & ); // Copying
  string( const char * ); // Constructing with character strings
  ~string( void );

  String & operator=( const string & ); // Assigning string
  String & operator=( const char * ); // Assigning C++ string
  String & operator=( const char ); // Assigning char

  String & operator+=( const string & ); // Appending string
  String & operator+=( const char ); // Appending char

  char & operator[]( int ); // Accessing specific char

  String & insert( int pos1, const string & str );

  String & remove( int pos = 0, int n );

  int length() const;

  int find( const string & ) const; // Finding 1-st occurrence
                                    // of substring
  int find( char ) const; // Finding 1-st occurrence
                          // of char
  int find( const char * ) const; // Finding 1-st occurrence
                                  // of C++ string

  string substr( int pos, int length ) const; // Substring
};

// I/O Operations
ostream & operator<<( ostream & , const string & );
istream & operator>>( istream & , string & );
istream & getline( istream & , string & , char sentinel = '\n' );

// Comparison
bool operator==( const string & lhs, const string & rhs );
               //Operators -- !=, <, <=, >, >=

// Concatenation
string operator+( const string & lhs, const string & rhs );
string operator+( const string & str, char ch );
string operator+( char, const string & str );
```

4.2 Example

Example 1

```
/**
 *Program Name: cis25L0911.cpp
 *Discussion:   Use of string Class
```

```
 */
#include <iostream>
#include <string>
using namespace std;

int main( void ) {
  string sObj1( "Some old" );
  string sObj2( "string" );
  string sObj3( sObj1 + sObj2 );

  cout << "\nsObj3 : " << sObj3 << endl;
  cout << "\nLength of sObj3 : " << sObj3.length() << endl;
  cout << "\nFirst character in sObj3 : " << sObj3[ 0 ] <<endl;
  cout << "\nIs there \"string\" in sObj3 (starting location)? : "
       << sObj3.find( "string" ) << endl;

  cout << "\nA substring of sObj3 : " << sObj3.substr( 5, 9 )
       << endl;

  sObj3.insert( 8, " " );
  cout << "\nsObj3 : " << sObj3 << endl;

  string stLine;

  cout << "\nType a line ended with a period '.' + ENTER ...:\n\t";
  getline( cin, stLine, '.' );

  cout << "\nstLine : " << stLine << endl;

  return 0;
}
```

OUTPUT

```
sObj3 : Some oldstring

Length of sObj3 : 14

First character in sObj3 : S

Is there "string" in sObj3 (starting location)? : 8

A substring of sObj3 : oldstring

sObj3 : Some old string

Type a line ended with a period '.' + ENTER ...:
        This is OOP.

stLine : This is OOP
```