Lecture 10.1

Topics

1. Multi-file Compilation – Revisited
2. Pointers as Member Data – Revisited

---

1. Multi-file Compilation – Revisited

Recall that a complete program will be organized in several files as follows,

(a) Specification file – `*.h`

(b) Implementation file – `*.cpp`

(c) Application driver file – `*.cpp`

1.1 Specification File – `*.h` File

- All declarations and specifications, such as `struct` or `class`, should 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.

1.2 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`**.

1.3 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 for Class `Fraction`

```
// FILE #1 - Class Specification File
/**
 * Program Name: fraction.h
 * Discussion:   Class with and without
 *               Data Encapsulation
 */
#ifndef FRACTION_H
#define FRACTION_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:  fraction.cpp
 *Discussion:    Implementation File:
 *                 Objects in Functions - Fraction Class
 */
#include <iostream>
#include "fraction.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;
  }
}

// FILE #3 – Application Driver File
/**
 *Program Name:  cis25FractionDriver1001.cpp
 *Discussion:    Objects in Functions -
 *                 Fraction Class
 */
#include <iostream>
#include "fraction.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!
```

2. Pointers as Member Data – Revisited

Recall that a pointer can also be made as a member data of a class. The following descriptions will show a simple example.

```
// Class Specification File
/**
 * Program Name: dynamicArray.h
 * Discussion:  Class with a pointer member data
 *                Pointer member data is used to create
 *                 a dynamic array of int's
 */
#ifndef DYNAMICARRAY_H
#define DYNAMICARRAY_H

class DynamicArray {
public:
  DynamicArray();
  DynamicArray( const DynamicArray& arg );

  ~DynamicArray();

  int* getDataPtr();
  void setDataPtr( int* arg );

private:
  int size;
  int* dataPtr;
};

#endif
```

And,

```
// Implementation File
/**
 *Program Name:  dynamicArray.cpp
 *Discussion:    Class with dynamic array of int's
 */
#include <iostream>
#include "dynamicArray.h"
using namespace std;

DynamicArray::DynamicArray() {
  // TODO code
}

DynamicArray::DynamicArray( const DynamicArray& arg ) {
  // TODO code
}

DynamicArray::~DynamicArray() {
  // TO DO code
}

int DynamicArray::getSize() {
  //TODO code

  return 0;
}

void DynamicArray::setSize( int arg ) {
  // TODO code
```

```
}

int* DynamicArray::getDataPtr() {
  // TODO code

  return 0;
}

void DynamicArray::setDataPtr( int* arg ) {
  // TODO code
}
```

Explanations will be provided in class.