CSE-4301
Object Oriented Programming
2022-2023

Week-13

Multi-file Program

-Faisal Hussain
Department Of Computer Science And Engineering, IUT

Contents

- Reasons for Multifile Programs
- Organization and conceptualization
- Inter file communication
- Inter file variables, function, class
- Include header file
- Namespace
- Typdef

Reasons for Multifile Programs

- Class Libraries
  - Vendors provide well furnished library of functions
  - Cpp contains library of classes which is better in modeling
  - It is composed of two component
    - Interface
    - Implementation

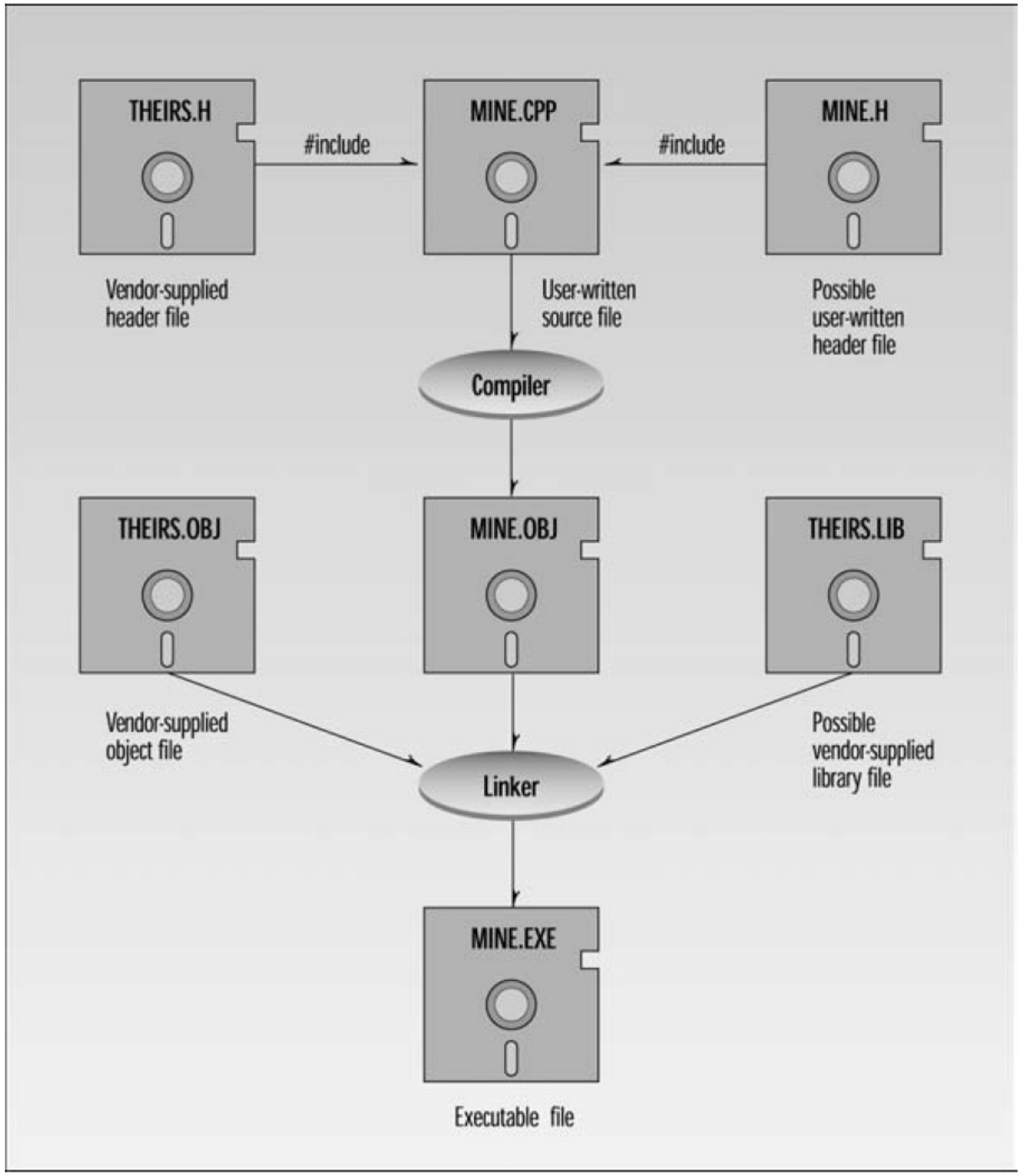
Interface

- Class developer develop the class
- Programmer uses the class in the application

- Programmer needs to access various declaration of the class
  - These are the part of public part of the class
  - It is presented in the header file.
- Why do the programmer needs to access
  - No different description is required/ source code says it all.
  - Main purpose is so that those part of the program element can be called. Programmers don't need to know the actual implementation. Thus these are known as interfaces.

Implementation

- inner workings of the member functions of the various classes don’t need to be known by the programmer
- So the source code is not released.
- Rather .obj file or .lib file is distributed


THEIRS.H
#include
MINE.CPP
#include
MINE.H
Vendor-supplied
header file
User-written
source file
Possible
user-written
header file
Compiler
THEIRS.OBJ
MINE.OBJ
THEIRS.LIB
Vendor-supplied
object file
Possible
vendor-supplied
library file
Linker
MINE.EXE
Executable file

Creating Multi file program

- Header file
- Directory
- Projects

Interfile Communication

- Cpp files are separately compiled
- Its important to understand how these files communicated with each other
- How header files play its role

Distinction between declaration and definition

- Declaration is kind of announcement that this program element is available but how that program element will function is not present in declaration.
- Definition means that how a program element will function.

- Declaration and Definition of
  - Variable
  - Function
  - Class

Interfile Variables

```
int someVar;          //declaration and also definition
extern int someVar;   //declaration only
```

As you might expect, a global variable can be defined in only one place in a program.

```
//file A
int globalVar;  //definition in file A

//file B
int globalVar;  //illegal: same definition in file B
```

```
//file A
int globalVar;    //defined

//file B
globalVar = 3;    //illegal, globalVar is unknown here
```

Interfile Variables

```
//file A
int globalVar;          //definition

//file B
extern int globalVar;   //declaration
globalVar = 3;          //now this is OK
```

```
extern int globalVar = 27;  //not what you might think
```

```
//file A
static int globalVar;   //definition; visible only in file A

//file B
static int globalVar;   //definition; visible only in file B
```

Interfile Function

```
//file A
int add(int a, int  b)    //function definition
   { return a+b; }        //(includes function body)

//file B
int add(int, int);        //function declaration (no body)
. . .
int answer = add(3, 2);   //call to function
```

```
//file A
int add(int, int);  //declaration
int add(int, int);  //another declaration is OK
```

Like variables, functions can be made invisible to other files by declaring them `static`.

```
//file A
static int add(int a, int  b)    //function definition
   { return a+b; }

//file B
static int add(int a, int  b)    //different function
   { return a+b; }
```

This code creates two distinct functions. Neither is visible in the other file.

Interfile Classes

A class declaration is simply a statement that a certain name applies to a class. It conveys no information to the compiler about the members of the class.

```
class someClass;                //class declaration
```

A class definition contains declarations or definitions for all its members:

```
class someClass                 //class definition
   {
   private:
      int memVar;               //member data definition
   public:
      int memFunc(int, int);    //member function declaration
   };
```

Why class definition is required

Why does a class need to be defined in every file where it's used? The compiler needs to know the data type of everything it's compiling. A declaration is all it needs for simple variables because the declaration specifies a type already known to the compiler.

```
//declaration
extern int someVar;              //if it sees this, the compiler
someVar = 3;                     //can generate this
```

```
//declaration
int someFunc(int, int);          //if it sees this, the compiler
var1 = someFunc(var2, var3);     //can generate this
```

However, for a class, the entire definition is necessary to specify the types of its member data and functions.

```
//definition
class someClass                  //if it sees this, the compiler
   {
   private:
      int memVar;
   public:
      int memFunc(int, int);
   };
someClass someObj;               //can generate this
v1 = someObj.memFunc(v2, v3);    //and this
```

Header files

```
//fileH.h
extern int gloVar;                  //variable declaration
int gloFunc(int);                   //function declaration
```

```
//fileA.cpp
int gloVar;                         //variable definition
int GloFunc(int n)                  //function definition
   { return n; }

//fileB.cpp
#include "fileH.h"
. . .
gloVar = 5;                         //work with variable
int gloVarB = gloFunc(gloVar);      //work with function
```

```
//fileH.h
class someClass                              //class definition
   {
   private:
      int memVar;
   public:
      int memFunc(int, int);
   };

//fileA.cpp
#include "fileH.h"
int main()
   {
   someClass obj1;                           //create an object
   int var1 = obj1.memFunc(2, 3);            //work with object
   }

//fileB.cpp
#include "fileH.h"
int func()
   {
   someClass obj2;                           //create an object
   int var2 = obj2.memFunc(4, 5);            //work with object
   }
```

Multiple Include Hazard

```
//file headtwo.h
int globalVar;

//file headone.h
#include "headtwo.h"

//file app.cpp
#include "headone.h"
#include "headtwo.h"
```

Prevention of Multiple Includes

```
#if !defined( HEADCOM )          //if HEADCOM not defined,
#define HEADCOM                  //define it

int globalVar;                   //define this variable


int func(int a, int b)           //define this function
   { return a+b; }

#endif                           //end condition
```

Namespace

- To avoid writing unique identifier name
- Namespace can be declared multiple times
- Namespace is normally used in header file

```
namespace geo
   {
   const double PI = 3.14159;
   }  // end namespace geo

//(some other code here)

namespace geo
   {
   double circumf(double radius)
      { return 2 * PI * radius; }
   }  //end namespace geo
```

typedef

- Using typedef you can rename any types
  - Type of variable
  - Class name

```
typedef unsigned long unlong;

unlong var1, var2;
```

```
typedef int FLAG;        //int variables used to hold flag values
typedef int KILOGRAMS;   //int variables used to hold values in kilograms
```

If you don't like the way pointers are specified in C++, you can change it:

```
int *p1, *p2, *p3;       //normal declaration
typedef int* ptrInt;     //new name for pointer to int
ptrInt p1, p2, p3;       //simplified declaration
```

This avoids all those pesky asterisks.

```
class GeorgeSmith_Display_Utility           //class definition
   {
   //members
   };

typedef GeorgeSmith_Display_Utility GSdu;   //rename the class

GSdu anObj;                                 //create object using new name
```