CS 210 **November 3, 2016**
Day 18

Chapter 12 Separate Compilation

Storage Classes

- auto
  - default storage class of function parameters and local variables
  - storage is automatically allocated on the stack at the time of a function call and deallocated when the function returns
- extern
  - storage class of names know to the linker
- global variable
  - a variable that may be accessed by many functions in a program
- static
  - storage class of variables allocated only once, prior to program execution
- register
  - storage class of automatic variables that the programmer would like to have store in registers

It is possible to write a program in multiple parts, compile these parts separately, and put them together to form a complete project. In this chapter we look at how this is done.

Using Abstraction to Manage Complexity

- procedural abstraction
  - separation of what a function does form the details of how the function accomplishes its purpose
- data abstraction
  - separation of the logical view of a data object (what is stored) from the physical view (how the information is stored)

Using Abstraction to Manage Complexity

- information hiding
  - protecting the implementation details of a lower-level module from direct access by a higher-level module
- encapsulate
  - packaging as unit a data object and its operators

A good example of *abstraction* is a high level language like C. It allows you to use a computer system without knowing much at all about what is going on inside. This is not too different from learning how to drive a car without understanding much at all about the engine or transmission.

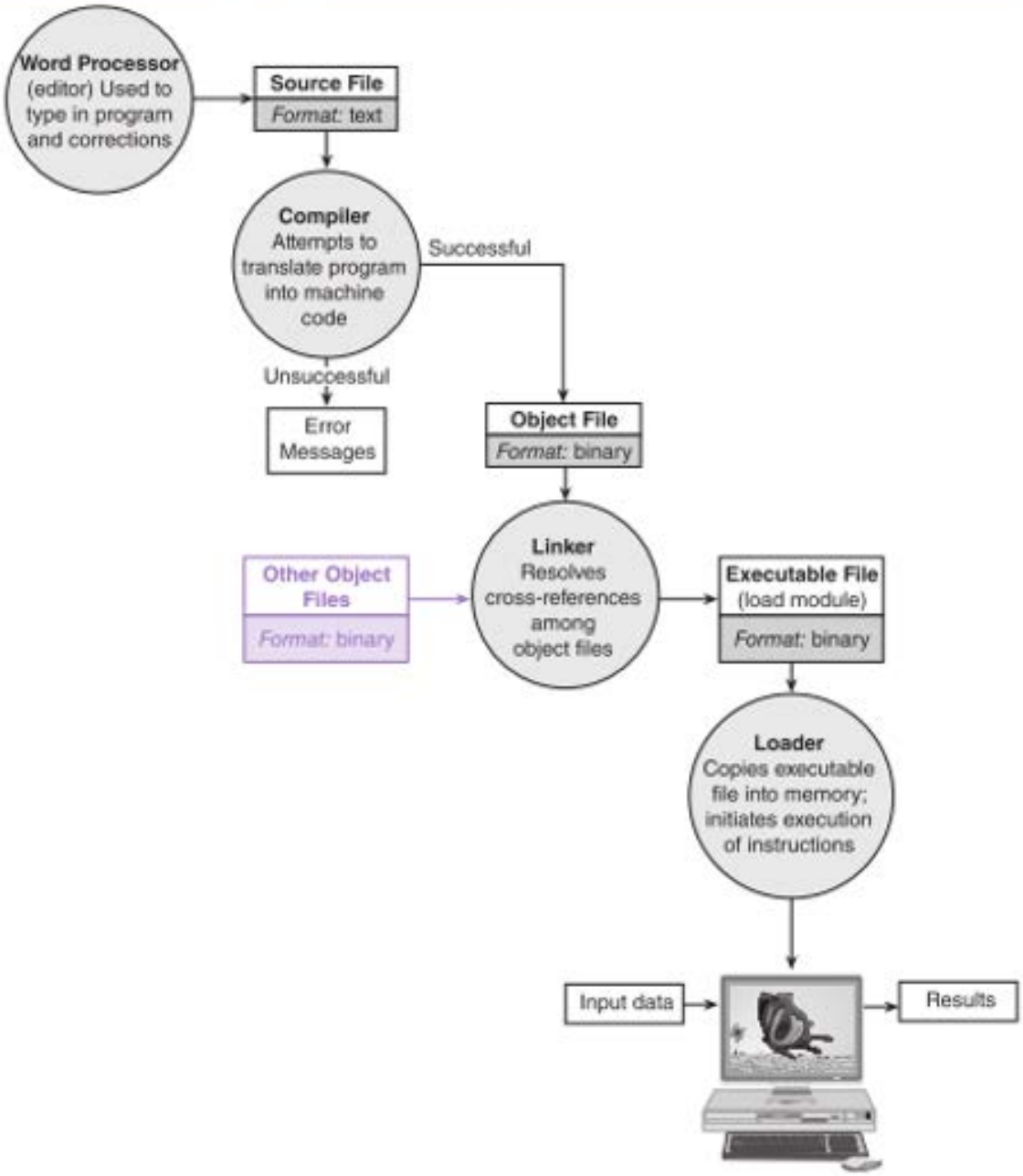
One way for you to do abstraction is to create your own library functions. For example, you might write a function which does file I/O. you could put this function in a personal library and reuse it in many programs. To anyone else looking at he program the library functions would be abstract. This is similar to what you now with stdio.h or math.h.

Personal Libraries

- header file
  - text file containing the interface information about a library needed by a compiler to translate a program system that uses the library or by a person to understand and use the library

The figure on the next page shows what happens when you write and execute a program.

FIGURE 12.1 Preparing a Program for Execution



There are several ways to create personal libraries. A very easy way to do this is to create a set of functions in a separate c-language file and include them in your project.

Here is a normal project (as we have been writing it). The program uses two points (x_1, y_1) and (x_2, y_2) and finds the length of a line between them as well as the slope of line from the first point to the second.

```
#include<stdio.h>
#include<math.h>
double FindDistance(double x1, double y1, double x2, double y2);
double FindSlope(double x1, double y1, double x2, double y2);
int main()
{
    double x1, y1, x2, y2;
    double dist, slope;
    x1 = 0; y1 = 0;
    x2 = 1.0; y2 = 1.0;
    dist = FindDistance(x1, y1, x2, y2);
    slope = FindSlope(x1, y1, x2, y2);
    printf("Line from (%4.1f, %4.1f) to (%4.1f, %4.1f)\n",
                x1, y1, x2, y2);
    printf("has a length of %6.2f\n", dist);
    printf("and a slope of %6.2f\n", slope);
    return 0;
}
double FindDistance(double x1,double y1,double x2,double y2)
{return sqrt(pow(x1-x2, 2)+pow(y1-y2, 2));
}
double FindSlope(double x1,double y1,double x2,double y2)
{return (y2 - y1)/(x2 - x1);
}
```

This program can be broken into two programs – one has the main program and the second has the functions.

LineNormal2.c

```
#include<stdio.h>
#include<math.h>
extern double FindDistance(double x1,
        double y1, double x2, double y2);
extern double FindSlope(double x1,
        double y1, double x2, double y2);
int main()
{
    double x1, y1, x2, y2;
    double dist, slope;
    x1 = 0; y1 = 0;
    x2 = 1.0; y2 = 1.0;
    dist = FindDistance(x1, y1, x2, y2);
    slope = FindSlope(x1, y1, x2, y2);
    printf("Line from (%4.1f, %4.1f) to
                (%4.1f, %4.1f)\n",
                   x1, y1, x2, y2);
    printf("has a length of %6.2f\n", dist);
    printf("and a slope of %6.2f\n", slope);
    return 0;
}
```

LineNormalLib.c

```
#include<math.h>

double FindDistance(double x1,
             double y1,double x2,double y2)
{return sqrt(pow(x1-x2, 2)+pow(y1-y2, 2));
}

double FindSlope(double x1,double y1,
             double x2,double y2)
{return (y2 - y1)/(x2 - x1);
}
```

Note that we have two source programs. The main program no longer contains the function definitions. Instead it contains only the function prototypes and these are declared **extern**.

The second program called LineNormalLib.c contains the function definitions.

You can also put the function prototypes in a separate header file. To do this right click on Header Files in the solution explorer and add a header file to the program.

```c
#include<stdio.h>
#include<math.h>
#include "LineNormalLib.h"
int main()
{
    double x1, y1, x2, y2;
    double dist, slope;
    x1 = 0; y1 = 0;
    x2 = 1.0; y2 = 1.0;
    dist = FindDistance(x1, y1, x2, y2);
    slope = FindSlope(x1, y1, x2, y2);
    printf("Line from (%4.1f, %4.1f) to
       (%4.1f, %4.1f)\n", x1, y1, x2, y2);
    printf("has a length of %6.2f\n",
                                  dist);
    printf("and a slope of %6.2f\n",
                                  slope);
    return 0;
}
```

LineNormal2.c

```c
#include<math.h>

double FindDistance(double x1,
          double y1,double x2,double y2)
{return sqrt(pow(x1-x2, 2)+pow(y1-y2, 2));
}
double FindSlope(double x1,
          double y1,double x2,double y2)
{return (y2 - y1)/(x2 - x1);
}
```

LineNormalLib.c

```c
#pragma once
extern double FindDistance(double x1,
         double y1, double x2, double y2);
extern double FindSlope(double x1,
         double y1, double x2, double y2);
```

LineNormalLib.h

In this version there are three files. LineNormal2.c is the main program. LineNormalLib.c contains our "library" functions, and LineNormalLib.h has the prototypes which are all declared **extern**.

The compiler will place the header file information in the main program and compile it. It will also compile, separately, the program with the function definitions. The linker will fill in the addresses so that the two programs work together.

A second way to create a personal library of functions is to write and compile a set of functions in a separate project. You can then compile this project, add it to a project that has your main code, and add in a header file.

To do this you need to add the library project to your main project, include the header file, and tell the main project where your library project is located.

This is explained in detail in SeparateCompilationNotes.pdf located on the web site.

Arguments to function main

- command-line arguments
  - options specified in the statement that activated a program

```
int
main(int  argc,    /* input - argument count (including
                              program name)                    */
     char *argv[]) /* input - argument vector                  */
```

Run the .exe file for this program

```
#include<stdio.h>
int main(int argc, char *argv[])
{
    char s2[80];
    printf("Number of arguments %d\n", argc);
    printf("Argument 0 is %s\n", argv[0]);
    printf("Argument 1 is %s\n", argv[1]);
    scanf_s("%s", s2, sizeof(s2)); //Requires buffer size
}
/*
c:\>ArgsToMain abc
Number of arguments 2
Argument 0 is ArgsToMain
Argument 1 is abc
*/
```

CS 210 **November 3, 2016**

Separately Compiled Programs

Complete the example in SeparateCompilationNotes.pdf on the web site.