Introduction to Programming Language (C)

(EEF 110E)

Dr. Emre DİNCEL

LECTURE NOTES

(WEEK 11)

❖ Writing Large Programs

It is often neither possible nor required to write large programs in a single source file. A common practice is to create function libraries and put several related functions together (most probably in a single file) and then include (or use) them whenever necessary.

First of all, let's talk about the C compiler (pre-processor) directives briefly.

C Compiler (Pre-processor) Directives

- **#include:**

It is possible to include the contents of a file at any point in a C program using the #include directive. For example,

```
#include <stdlib.h>
#include "mylibrary.h"
```

Please note that if an included file could not be found, compilation will cease with an error.

- **#define:**

It is possible to use the #define directive in two different forms: defining a constant or creating a macro.

```
#define token [value]
```

Here, the value field can be empty and if this is the case, the token will be replaced with blank text. This usage is often preferred for the purposes of "#ifdef" and "#ifndef" directives. For example,

```
#define PI 3.1416
#define MAX(a, b) (a > b ? a : b)
#define INCREMENT(x) x++
```

- **#if, #elif, #else, #endif**

The #if directive checks whether the value is true and if so, includes the code until the closing #endif. If not, that code is removed from the copy of the file given to the compiler prior to compilation (but it has no effect on the original source code file).

For example,

```
#if <value>
// Code to be executed if this value is true
#elif <otherValue>
// Code to be executed if this value is true
#else
// Code to be executed otherwise
#endif
```

- **#ifdef, #ifndef**

The #ifdef directive checks whether the given token has been #defined earlier in the file or in an included file. If so, it includes everything between it and the closing #else or, if no #else is present, the closing #endif.

```
#ifdef __cplusplus
// C++ code
#else
// C code
#endif
```

This is especially used together with #define directive to avoid the header files to be included more than once (which causes an error). For example;

```
#ifndef _FILE_NAME_H_
#define _FILE_NAME_H_
/* code */
#endif
```

- **__FILE__**

__FILE__ is a preprocessor macro that expands to full path to the current file. __FILE__ is useful when generating log statements, error messages intended for programmers.

- **__LINE__**

__LINE__ is a preprocessor macro that expands to current line number in the source file, as an integer. __LINE__ is useful when generating log statements, error messages intended for programmers

External Declarations:

Usually it is required to compile libraries separately and then link the object codes. This reduces the compile time considerably and also allows hiding the implementation details from the programmer (which is nice from software engineering point of view).

In order to be able to share functions and (global) variables defined in different libraries, the shared functions and variables must be declared wherever they are used.

Such declarations are done using the keyword **extern** to indicate that the real definition is done somewhere else.

Here mylib.c and main.c are compiled separately and then linked together:

```
/* mylib.c file */

void fswap(float *num1, float *num2){
  float temp;
  temp=*num1;
  *num1=*num2;
  *num2=temp;
}
```

```
/* main.c file */
#include <stdio.h>

extern void fswap(float *num1, float *num2);
int main(){
  float n1=1.0, n2=2.0;
  fswap(&n1,&n2);
  printf("n1 = %3.1f   n2 = %3.1f\n",n1,n2);
  return 0;
}
```

```
n1 = 2.0   n2 = 1.0
```

On the other hand, the better approach is to develop a library structure, which consists of header files, C files and the main program file. Consider the following example:

Library Header File

```
/* mylib.h File */

#ifndef MYLIB_H  // Check if the library is already included or not
#define MYLIB_H  // This prevents multiple declarations!

extern void fswap(float *num1, float *num2);
extern int globalVar;
```

Library Source (C) File

```
/* mylib.c File */

#include "mylib.h"

int globalVar = 5;
void fswap(float *num1, float *num2)
{
   float temp = *num1;
   *num1 = *num2;
   *num2 = temp;
}
```

Main Program File

```
/* main.c File */

#include <stdio.h>
#include "mylib.h"

int main()
{
   float n1 = 1.5, n2 = 4.0;

   fswap(&n1, &n2);
   printf("n1 = %3.1f \t n2 = %3.1f\n", n1, n2);

   return 0;
}
```

In this last example we have separated the library from the main program. This way it is possible to change implementation details of each .c file without affecting the other. The only thing that connects the two source files is the interface of the library (.h file). Library developers can provide .h files and the object file (.obj) to the users of the library to secure the source code.

Creating and Using DLL Files:

A DLL is a library that contains code and data that can be used by more than one program at the same time. By using a DLL, a program can be modularized into separate components. For example, an accounting program may be sold by module. Each module can be loaded into

the main program at run time if that module is installed. Because the modules are separate, the load time of the program is faster. And a module is only loaded when that functionality is requested.

- **Creating DLL File:**

In order to create a DLL file, first of all, it is required to indicate that the function will be exported to the DLL. For this purpose, the following code can be used (in library header file):

```
// Define the DLL export keyword
#define exportToDLL __declspec(dllexport)
// Declare the functions
exportToDLL extern void Hello();
```

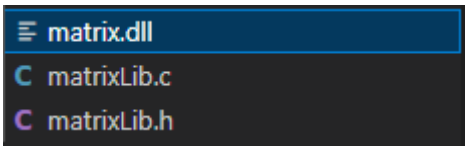
The same keyword should also be put in the function implementation (in library C file):

```
exportToDLL void Hello()
{
   printf("Hello world\n");
}
```

After that it is required to compile the code (in VSCode) as follows:

```
gcc -shared libraryFileName.c -o libraryOutput.dll
```

It is seen that the corresponding DLL file is created in the working folder.



Now, the user can share the DLL file (and the library header file) with anyone.

- **Using DLL File:**

The created DLL file can now be imported and the functions exported to this DLL file can be used as desired. Please, however, note that, it is required to change the library header (.h) file as follows:

```
// Define the DLL import keyword
#define importFromDLL __declspec(dllimport)
// Declare the functions
importFromDLL extern void Hello();
```

After this minor modification in the header (.h) file, the user can include it to main C file using the #include directive. Finally, the main C file (together with the used DLL) is compiled as follows:

```
gcc main.c -o main.exe libraryOutput.dll
```

This will produce an executable file which is ready to be run.