

Programming For Engineers

Organizing C Programs

by

Wan Azhar Wan Yusoff¹, Ahmad Fakhri Ab. Nasir²
Faculty of Manufacturing Engineering
wazhar@ump.edu.my¹, afakhri@ump.edu.my²



0.0 Chapter's Information

- **Expected Outcomes**
 - To organize C source, header and library files.
- **Contents**
 - 1.0 Introduction
 - 2.0 C Source file
 - 3.0 C Header file
 - 4.0 C Implementation file
 - 5.0 Create static library
 - 6.0 Create dynamic library
 - 7.0 Summary



1.0 Introduction

- We program with user in mind. When the programming is completed, we deliver executable file to user. User then just double-click our executable file and the program is “running”.
- However, we also program with other programmer in mind. We share our codes and most of the times we develop the codes together in a team. Furthermore, as the codes becomes more complicated (by the time we want to call our codes as “software”), we may easily become disorganized with our large number of functions. Clearly, we need to organize our files.
- Organizing program in C involves using several file types:
 1. Main source file is the location of the main (top-level) program codes. The filename convention is main.c.
 2. Header file is the listing of typedef, constant, function prototypes and derived data types. The file name convention is filename.h.



1.0 Introduction

3. Implementation source file is the location for function implementation codes. The file name convention is filename.c.
 4. Static library file is the machine codes converted from function codes for the purpose of integrating with executable file permanently. The file name convention is filename.a for Windows. The library file is other programmer to use and not for the end user.
 5. Dynamic link library file is the machine codes for the purpose of integrating with executable file when needed. The file name convention is filename.dll. The dynamic link library is needed by the end user when executing the executable file.
 6. The executable file is the program application file. The file name convention is filename.exe. This is the file that starts the program.
- We will learn organizing the program (codes) through an example program. In our example, we will write a simple program to display decimal number in binary format.

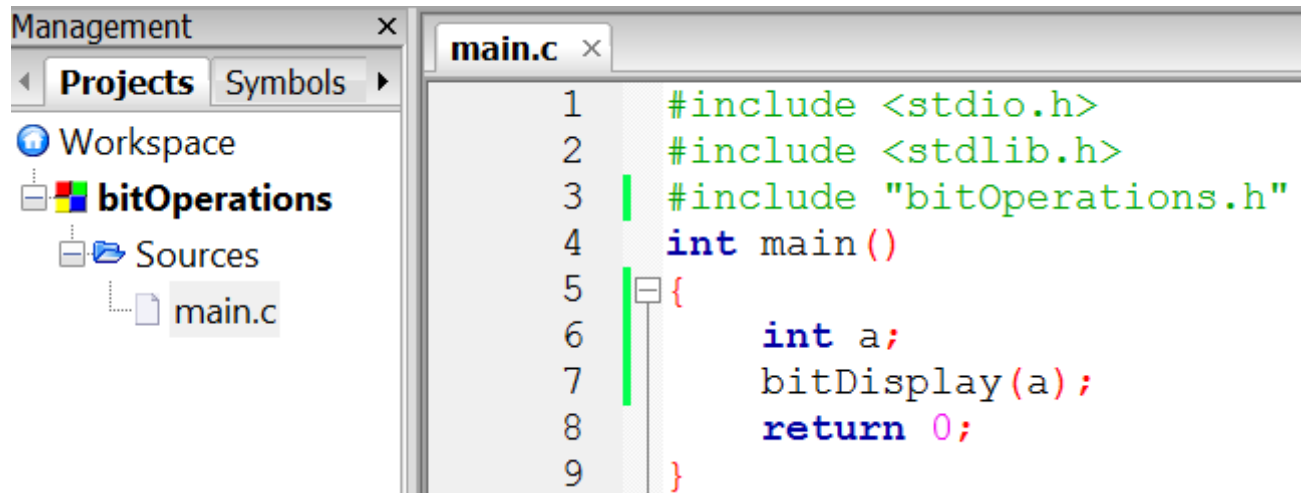


2.0 C Main Source File

- In this file we write codes that are top-level. The codes are inside the main function. The main function is the beginning of a C program. Inside the main function, we code our top-level instruction from the beginning to the end of the program. When the main() return 0 to the operating system, our program exits.
- In this main.c file, we also include all necessary header files that are needed for the main() functions. A familiar example is the `stdio.h` header file.
- Open your CodeBlock IDE and create a new console c project and name it bitOperations. A main.c file should appear after you double-click Sources folder on the left pane. Type the following codes in the main function.



2.0 C Main Source File



```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "bitOperations.h"
4  int main()
5  {
6      int a;
7      bitDisplay(a);
8      return 0;
9  }
```

- This is the main() program. The source file is save as main.c file. In the main program, we just code a top-level code i.e. we want to display a binary number of a decimal number. We create a function called displayBit() with an integer as the input parameter. This is what we call top-level instructions.



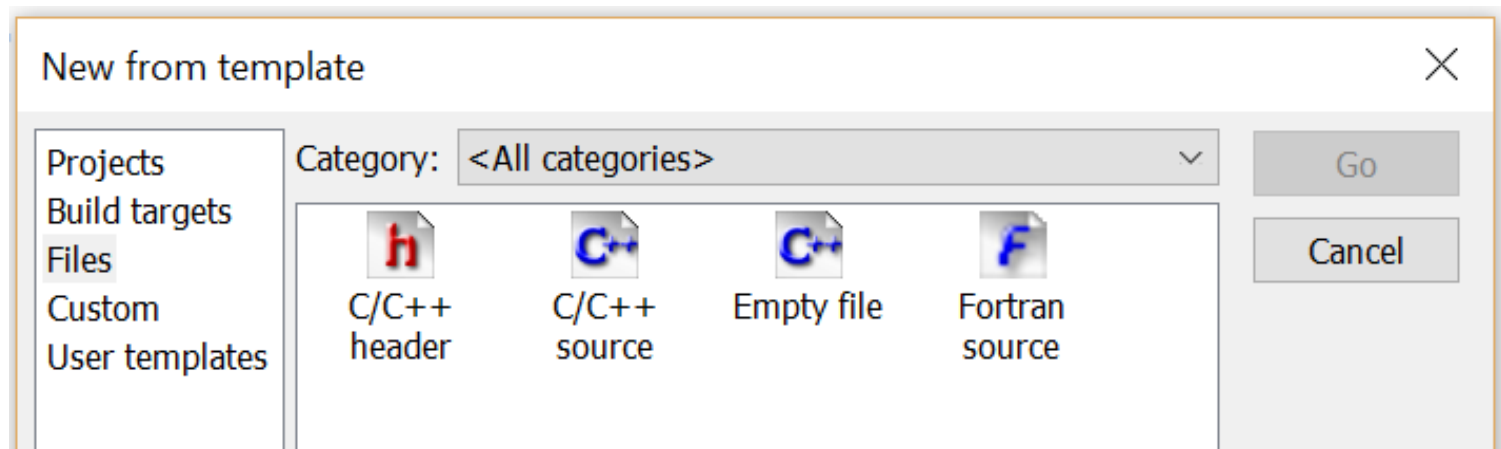
2.0 C Main Source File

- We will not code the function `displayBit()` in the main source program. We will code the “implementation” of function `displayBit()` in a separate file called `bitOperations.c`. But first, we must include a header file in our project. The name of the header file is `bitOperations.h`. That is the reason in our `main.c`, we include a statement `#include “bitOperations.h”` at the top our program together with `<stdio.h>` and `<stdlib.h>`. The reason the `bitOperations.h` is in double quotation mark is because the file location is the same as the `main.c` folder location. If the file is in the standard library folder, we will use `#include <bitOperations.h>` instead of `#include “bitOperations.h”`. The `stdio.h` is in the standard library folder and that is why we write `#include <stdio.h>` in our program.
- Next, we will create the `bitOperations.c` and `bitOperations.h` file.



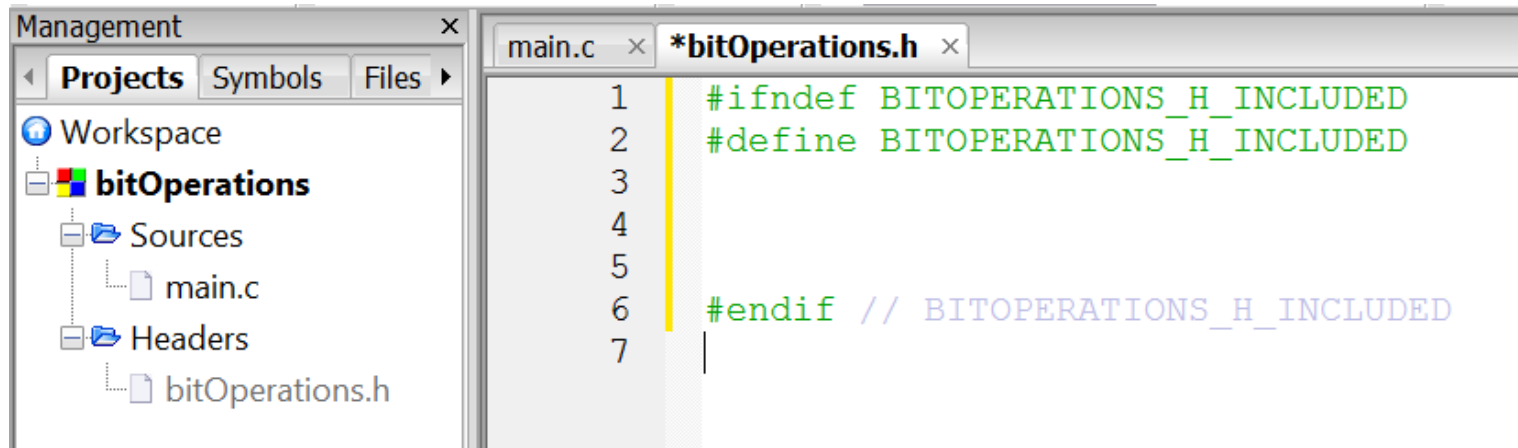
3.0 C Header File

- Next, we add a new header file to the project. Click File>New>File... and a form like below will appear. Choose C/C++ header and name the file bitOperations. You do not need to type .h because it automatically make it .h.



3.0 C Header File

- Under the folder Headers in the left pane, you will find the header file. Double-click the file and you will see something like the following.



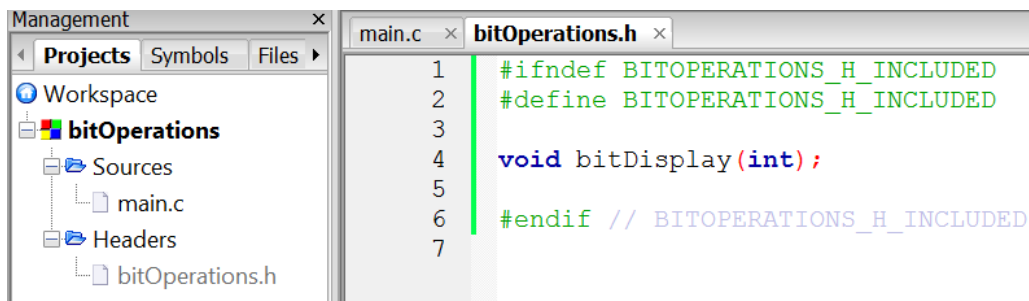
The screenshot shows an IDE window with a project named 'bitOperations'. The left pane shows the project structure with folders 'Sources' and 'Headers'. The 'Sources' folder contains 'main.c' and the 'Headers' folder contains 'bitOperations.h'. The right pane shows the content of 'bitOperations.h' with the following code:

```
1  #ifndef BITOPERATIONS_H_INCLUDED
2  #define BITOPERATIONS_H_INCLUDED
3
4
5
6  #endif // BITOPERATIONS_H_INCLUDED
7  |
```



3.0 C Header File

- The green statements (the `#if` and `#endif`) in the header file are automatically created. The purpose is to ensure that when a header file is included more than once, the compiler will ignore after the first one.
- Inside the header file we will put our function prototype. We type `void bitDisplay(int);` as shown below.



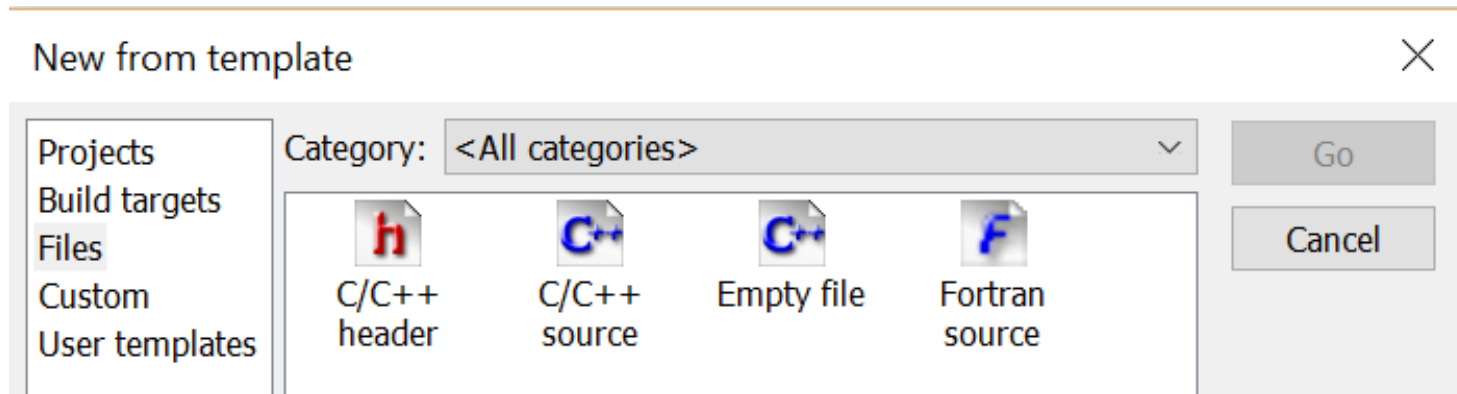
```
1  #ifndef BITOPERATIONS_H_INCLUDED
2  #define BITOPERATIONS_H_INCLUDED
3
4  void bitDisplay(int);
5
6  #endif // BITOPERATIONS_H_INCLUDED
7
```

- Notice that we design our function to return void (nothing) and to receive input of type integer. The semicolon at the end of the statement is necessary.



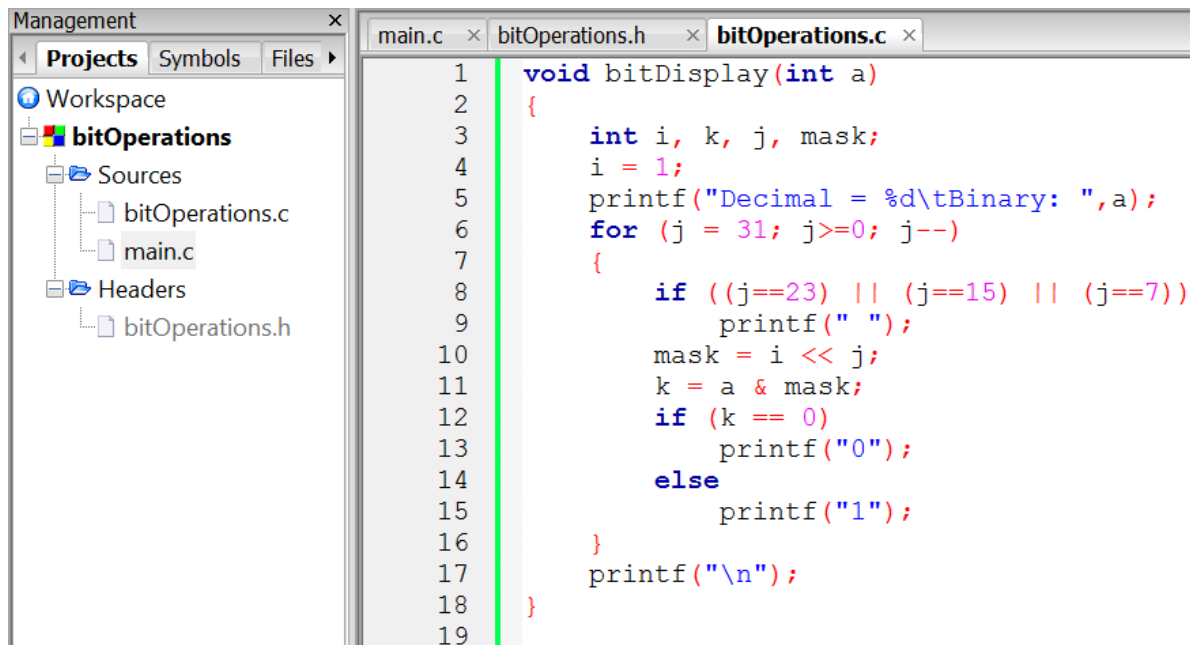
4.0 C Implementation File

- Now, we want to code our function `displayBit()`. We will code in another file called `bitOperations.c`. Thus, we create the implementation file. Click `File>New>File...` and a form like below will appear again. This time select C/C++ source and name it `bitOperations` (no need `.c` extension).



4.0 C Implementation File

- We code the details of the `bitDisplay()` function in the `bitOperations.c`. Do not worry if you are still unclear what the codes are all about. We will learn bit-wise operations later. Type what is shown below.



The screenshot shows a code editor with three tabs: `main.c`, `bitOperations.h`, and `bitOperations.c`. The `bitOperations.c` tab is active, displaying the following C code:

```
1 void bitDisplay(int a)
2 {
3     int i, k, j, mask;
4     i = 1;
5     printf("Decimal = %d\tBinary: ", a);
6     for (j = 31; j >= 0; j--)
7     {
8         if ((j == 23) || (j == 15) || (j == 7))
9             printf(" ");
10        mask = i << j;
11        k = a & mask;
12        if (k == 0)
13            printf("0");
14        else
15            printf("1");
16    }
17    printf("\n");
18 }
19
```

The left sidebar shows a project structure for `bitOperations` with subfolders for `Sources` (containing `bitOperations.c` and `main.c`) and `Headers` (containing `bitOperations.h`).

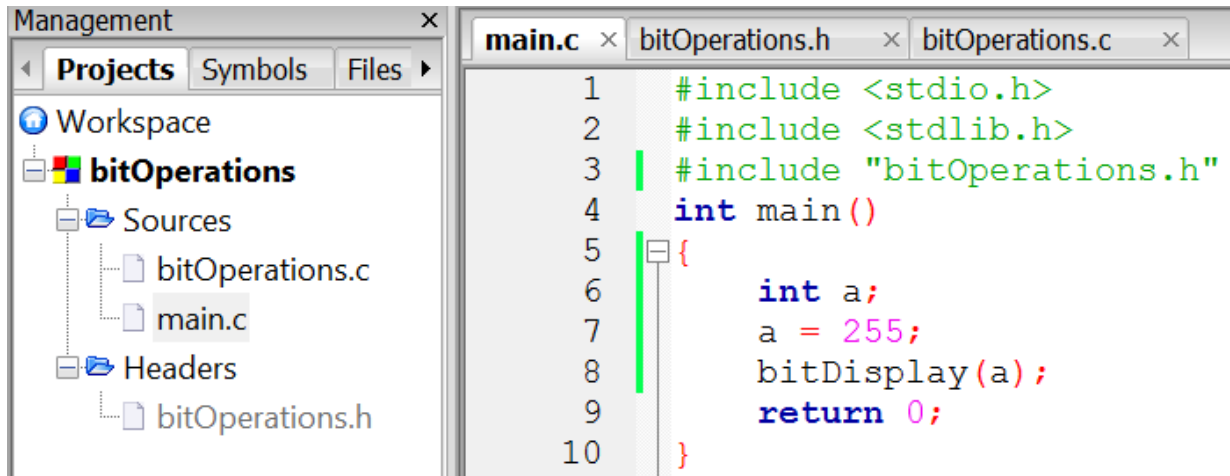


4.0 C Implementation File

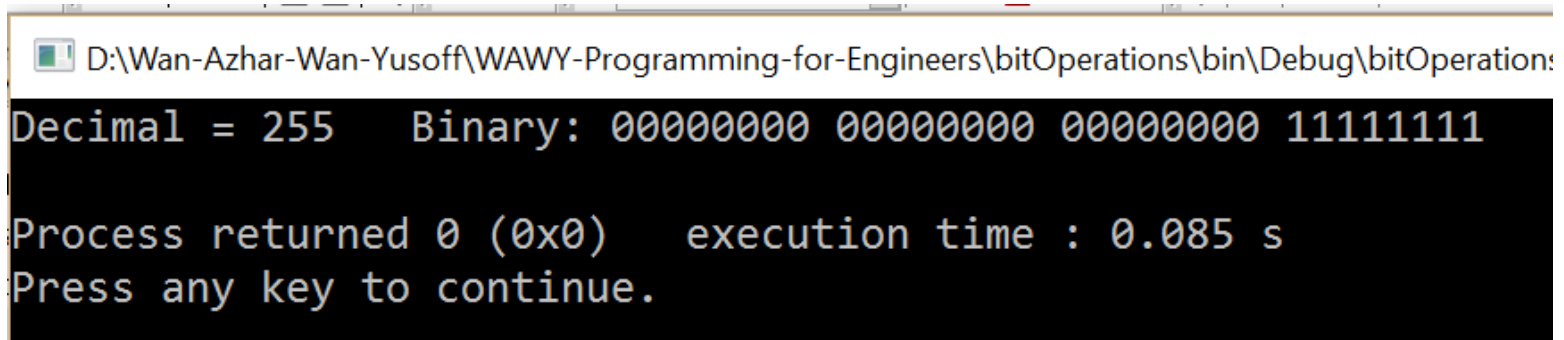
- Now we have completed our program organization using 3 files. We have successfully:
 1. Separate the function prototype declarations from the function implementations. We put our function prototypes in the header file i.e. bitOperations.h file, while we put our function implementations in the bitOperations.c file.
 2. We do our top level programming in the main.c file.
- Now, let's execute our program. Add the statement `a = 255` in the main program as shown. Then, compile and run. If there is error, correct first and run again. Your program and output, should look something like this.



4.0 C Implementation File



```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "bitOperations.h"
4  int main()
5  {
6      int a;
7      a = 255;
8      bitDisplay(a);
9      return 0;
10 }
```



```
D:\Wan-Azhar-Wan-Yusoff\WAWY-Programming-for-Engineers\bitOperations\bin\Debug\bitOperations
Decimal = 255 Binary: 00000000 00000000 00000000 11111111
Process returned 0 (0x0) execution time : 0.085 s
Press any key to continue.
```



4.0 C Implementation File

- Let's modify the main program as shown below. On line 13, we create a loop in order to display the decimal number from 0 to 15. Compile and run. You should get something similar to the output below.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "bitOperations.h"
4 int main()
5 {
6     int a;
7     for (a=0; a<15; a++)
8         bitDisplay(a);
9     return 0;
10 }
```

```
D:\Wan-Azhar-Wan-Yusoff\WAWY-Programming-for-Engineers\bitOperations\bin\Debug\bitOperations.exe
Decimal = 0    Binary: 00000000 00000000 00000000 00000000
Decimal = 1    Binary: 00000000 00000000 00000000 00000001
Decimal = 2    Binary: 00000000 00000000 00000000 00000010
Decimal = 3    Binary: 00000000 00000000 00000000 00000011
Decimal = 4    Binary: 00000000 00000000 00000000 00000100
Decimal = 5    Binary: 00000000 00000000 00000000 00000101
Decimal = 6    Binary: 00000000 00000000 00000000 00000110
Decimal = 7    Binary: 00000000 00000000 00000000 00000111
Decimal = 8    Binary: 00000000 00000000 00000000 00001000
Decimal = 9    Binary: 00000000 00000000 00000000 00001001
Decimal = 10   Binary: 00000000 00000000 00000000 00001010
Decimal = 11   Binary: 00000000 00000000 00000000 00001011
Decimal = 12   Binary: 00000000 00000000 00000000 00001100
Decimal = 13   Binary: 00000000 00000000 00000000 00001101
Decimal = 14   Binary: 00000000 00000000 00000000 00001110









Process returned 0 (0x0)   execution time : 0.145 s
Press any key to continue.
```



4.0 C Implementation File

- Now, we inspect our program directory. In my case, my directory is in bitOperations folder. Inside there, I have the main.c file, bitOperations.c file and bitOperations.h file and few other CodeBlock specific files.

D (E:) > PFE Sem 1 2016-2017 > Week04 > bitOperations >

<input type="checkbox"/> Name	Date modified	Type
 bin	16/10/2016 7:03 PM	File folder
 obj	16/10/2016 7:03 PM	File folder
 bitOperations.c	16/10/2016 8:50 PM	C source file
 bitOperations.cbp	16/10/2016 9:08 PM	project file
 bitOperations.depend	16/10/2016 9:03 PM	DEPEND File
 bitOperations.h	16/10/2016 8:45 PM	Header file
 bitOperations.layout	16/10/2016 9:21 PM	LayOut Document
 main.c	16/10/2016 9:02 PM	C source file



4.0 C Implementation File

- So, where is our executable file? We have to look into the bin folder. In my case, it is under the Release folder. Usually it is under Debug folder if you compile under debug mode. So, bitOperations.exe is our “software”. This is the file that we give to our friend - not the source code.

n 1 2016-2017 > Week04 > bitOperations > bin > Release

Name	Date modified	Type
 bitOperations.exe	16/10/2016 9:14 PM	Application

- We can execute a console file by executing under MS Command Prompt program. Copy our bitOperations.exe file to our Desktop location. Open Command Prompt program. Below is the Command Prompt window.



4.0 C Implementation File

```
Command Prompt
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\WanAzhar>cd Desktop

C:\Users\WanAzhar\Desktop>bitOperations
Decimal = 0      Binary: 00000000 00000000 00000000 00000000
Decimal = 1      Binary: 00000000 00000000 00000000 00000001
Decimal = 2      Binary: 00000000 00000000 00000000 00000010
Decimal = 3      Binary: 00000000 00000000 00000000 00000011
Decimal = 4      Binary: 00000000 00000000 00000000 00000100
Decimal = 5      Binary: 00000000 00000000 00000000 00000101
Decimal = 6      Binary: 00000000 00000000 00000000 00000110
Decimal = 7      Binary: 00000000 00000000 00000000 00000111
Decimal = 8      Binary: 00000000 00000000 00000000 00001000
Decimal = 9      Binary: 00000000 00000000 00000000 00001001
Decimal = 10     Binary: 00000000 00000000 00000000 00001010
Decimal = 11     Binary: 00000000 00000000 00000000 00001011
Decimal = 12     Binary: 00000000 00000000 00000000 00001100
Decimal = 13     Binary: 00000000 00000000 00000000 00001101
Decimal = 14     Binary: 00000000 00000000 00000000 00001110

C:\Users\WanAzhar\Desktop>
```

- We can execute a console file by executing under MS Command Prompt program. Copy our bitOperations.exe file to our Desktop location. Open Command Prompt program. Below is the Command Prompt window.



4.0 C Implementation File

```
Command Prompt
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\WanAzhar>cd Desktop

C:\Users\WanAzhar\Desktop>bitOperations
Decimal = 0      Binary: 00000000 00000000 00000000 00000000
Decimal = 1      Binary: 00000000 00000000 00000000 00000001
Decimal = 2      Binary: 00000000 00000000 00000000 00000010
Decimal = 3      Binary: 00000000 00000000 00000000 00000011
Decimal = 4      Binary: 00000000 00000000 00000000 00000100
Decimal = 5      Binary: 00000000 00000000 00000000 00000101
Decimal = 6      Binary: 00000000 00000000 00000000 00000110
Decimal = 7      Binary: 00000000 00000000 00000000 00000111
Decimal = 8      Binary: 00000000 00000000 00000000 00001000
Decimal = 9      Binary: 00000000 00000000 00000000 00001001
Decimal = 10     Binary: 00000000 00000000 00000000 00001010
Decimal = 11     Binary: 00000000 00000000 00000000 00001011
Decimal = 12     Binary: 00000000 00000000 00000000 00001100
Decimal = 13     Binary: 00000000 00000000 00000000 00001101
Decimal = 14     Binary: 00000000 00000000 00000000 00001110

C:\Users\WanAzhar\Desktop>
```



4.0 C Implementation File

- If you do not know where the location of Command Prompt, search the program under windows. Open the Command Prompt like the previous figure. Go to the location of your copy of bitOperations.exe. In my case, I copied to Desktop directory. Then type bitOperations plus enter. I get the output shown in the previous figure. That is how you execute a console program.
- Your friend who has a copy of your bitOperations.exe file can execute (run) the program under his MS Command Prompt without the need of your source or header file. Everything the program need is already integrated inside the bitOperations.exe.



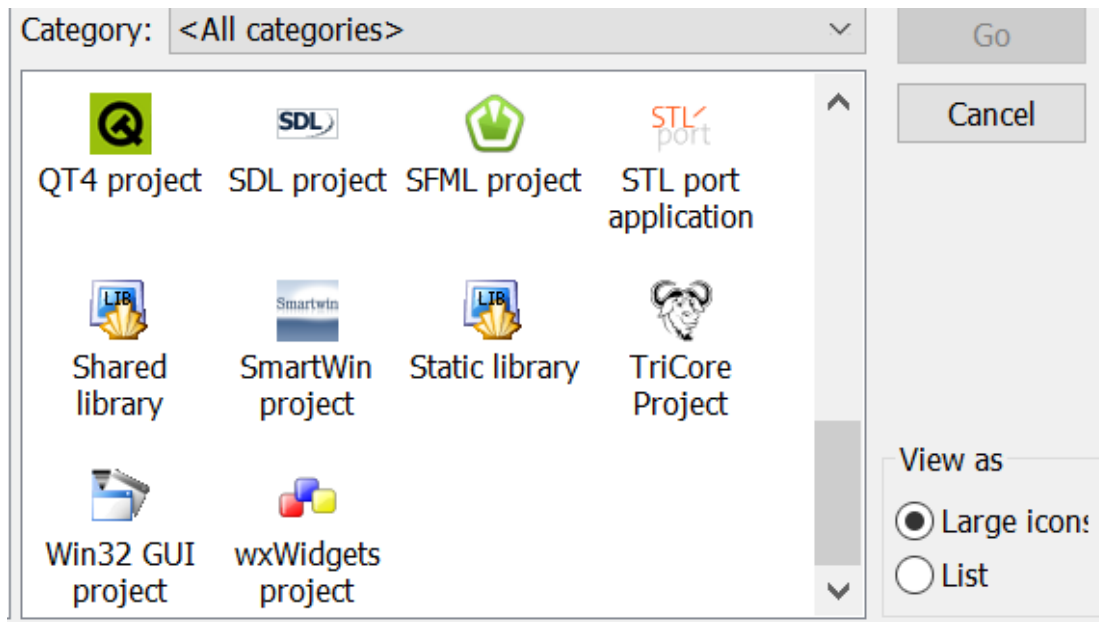
5.0 Creating Static Library

- The executable file that we created is a collection of main.c, bitOperations.c and the required library in one executable file. For example, we use the printf() function by invoking stdio.h because the library for printf() is a part of standard library. The “code” of printf() is integrated into our bitOperations.exe. The library for printf() is a standard static library. We do not create the function printf(), we just use it.
- We can also create our own library. We create a library file so that we can have modular functions. Also, other programmer can use our library for their purpose but without our protected source code. This is consistent with the idea that our function implementation file is our intellectual properties. We can sell our library to other programmer so that they can make executable file for the end user.



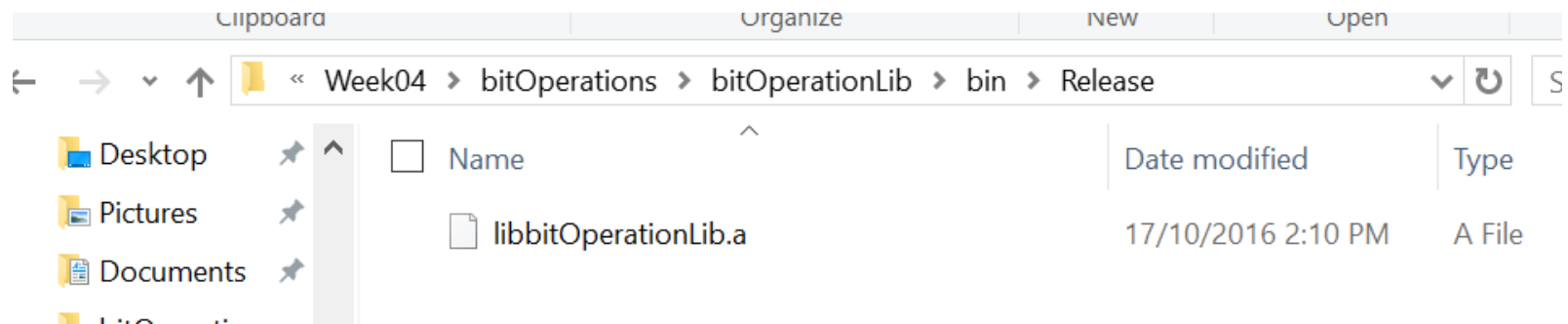
5.0 Creating Static Library

- Open CodeBlock IDE. Click Project> New>Project... and a window similar like below will appear. Choose Static library. Give the file name bitOperationsLib.



5.0 Creating Static Library

- In the menu, click Project>Add file...> and select our bitOperations.c which is our function implementation file.
- Remove the main.c file by right clicking and selecting Remove file from project. You should have only bitOperations.c file.
- Compile the program (not execute the program). If there is no error, the library is created. You should find under the Release folder the file name libbitOperationsLib.a file.



5.0 Creating Static Library

- The .a extension is the static library file. You give this file and the bitOperations.h file to other programmer who want to use your program. This ensure that your codes is protected from being manipulated by other programmer. For example, I give these two files to my programmer friend: bitOperations.h and libbitOperationLib.a files. How is he going to use this?
- First, he opens the header file (bitOperations.h) and then he knows already what the function is doing by reading the prototype statement.

```
void bitDisplay(int);
```



5.0 Creating Static Library

- He knows that if he can use the function by calling `bitDisplay(a)` whereas he knows that `a` must be integer value. That is the use of prototypes and the header files. It just tell us what is required to use a function. We do not need the details.
- Next, he copy the `bitOperations.h` to his `main.c` file directory.

m 1 2016-2017 > Week04 > bitOperations > friendProgram ▼ ↺

Name	Date modified	Type
 friendProgram.cbp	17/10/2016 2:25 PM	project file
 main.c	5/11/2015 4:09 AM	C Source file
 bitOperations.h	16/10/2016 8:45 PM	Header file



5.0 Creating Static Library

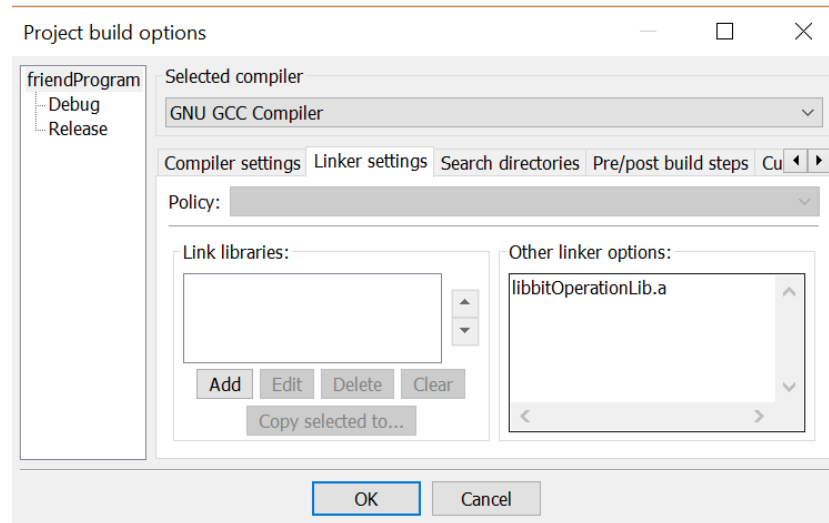
- Now he is ready to use the static library. For example, his main program can be like this:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "bitOperations.h"
4
5  int main()
6  {
7      int a;
8      printf("Please enter your decimal number:\n");
9      scanf("%d",&a);
10     bitDisplay(a);
11     return 0;
12 }
13
```




5.0 Creating Static Library

- Remember, he has to write `#include "bitOperations.h"` and the header must be in the same directory as his `main.c` file. Next, he has to add the static library to his program. Click `Project>Build option...` then select `Linker Setting`. In `Other linker options` type `libbitOperationLib.a`. click `OK`. When he compiles, it will integrate his `main.c` program with the static library program. The dialog box is something like below.



5.0 Creating Static Library

- Compile. If there is no error, execute the program. In my case, I get something like below.

 "E:\PFE Sem 1 2016-2017\Week04\bitOperations\f... —

```
Please enter your decimal number:
21
Decimal = 21    Binary: 00000000 00000000 00000000 00010101

Process returned 0 (0x0)   execution time : 4.204 s
Press any key to continue.
```

- He can distribute his .exe file to the end user. Users will use MS Command Prompt to execute the program. Remember, he only has to distribute the .exe file because the function is already integrated. Now, you understand that any programmer can use your function without knowing the details of your functions.

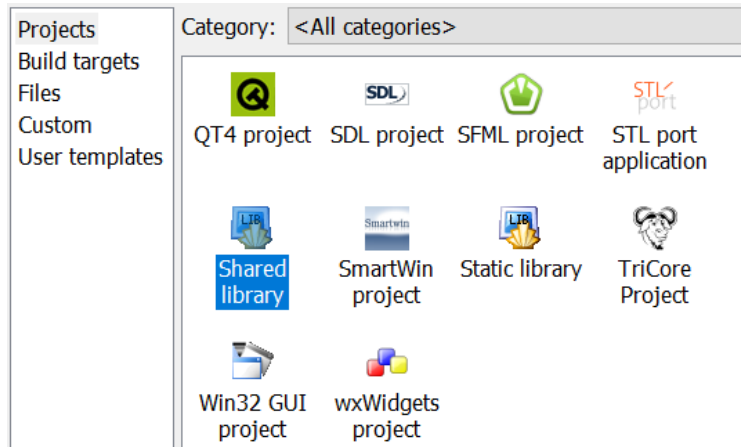


6.0 Creating Dynamic Link Library

- When our project is large, our .exe file can be very large because we integrate many library to our .exe program. We should only link together the library that we really need for our program. Dynamic library Link on the other hand will link to executable file when needed i.e when the program is being executed. In this case, the library must be distributed together with the executable file. Without the dynamic link library, we cannot execute the .exe file
- Some of the process is similar with the static library. First, we create new project and select shared library as our type of project.



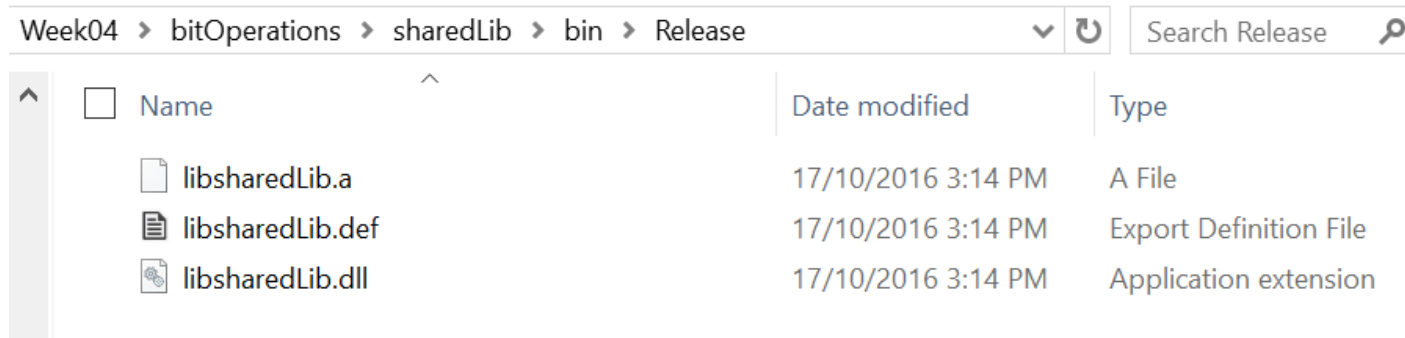
6.0 Creating Dynamic Link Library



- Give whatever name you want for the project (I give sharedLib). Remove the main.c file (Remove file from project) and add the bitOperations.c file (Project>Add files..) just like we did for the static library project. Just compile the program without executing. Your dynamic library should be created.



6.0 Creating Dynamic Link Library

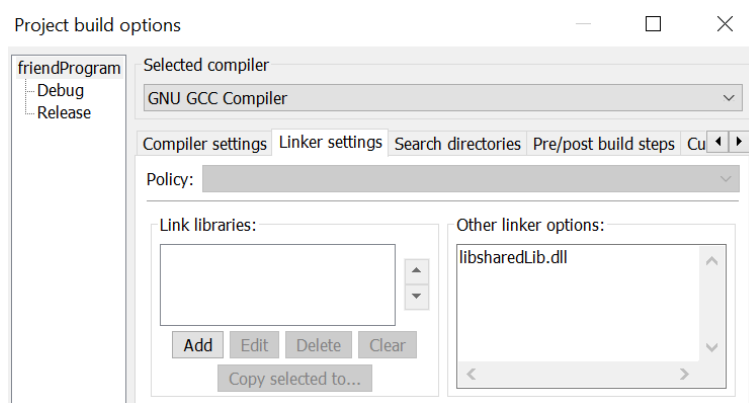


Name	Date modified	Type
libsharedLib.a	17/10/2016 3:14 PM	A File
libsharedLib.def	17/10/2016 3:14 PM	Export Definition File
libsharedLib.dll	17/10/2016 3:14 PM	Application extension

- The libsharedLib.dll is our dynamic link library. To use this we type libsharedLib.dll in the Other linker option pane just like the static library.



6.0 Creating Dynamic Link Library



- If you compile and execute without error you should get the .exe file.
- So what is the difference between using static library versus dynamic library? Using static library, you application program integrate the library. Using dynamic library, you have to supply the dynamic library together with your .exe program. Otherwise, your .exe program cannot run.
- Advantage of using dynamic library is that your .exe file is not large. Also, other .exe program created by other programmer also can use your dynamic library. That is why we call it shared library.



7.0 Summary

- In this note, we organize our c program for proper program development:
 1. We created main.c file and code our top-level instructions inside it. We include our header file also inside main.c file.
 2. We created header file and declare all our function prototypes inside it. We protect it with `#ifndef`, `#define` and `#endif` so that our header file is not included more than once.
 3. We created implementation file and code all our function details (our intellectual properties) inside it.
 4. We created static library and we give it to our friends without giving the source codes. He can integrate our static library with his .exe file.



7.0 Summary

5. We created dynamic library and give it to our friends without giving the source codes. But our friends cannot integrate our dynamic library with his .exe file. He has to supply our dynamic library to the end users before they can use the .exe program. It means we share the library.

