Tutorial 2: Single & Multiple Source-File Programs

Instructions

This tutorial consists of two simple tasks with the second task requiring uploading of your completed source code. How'll your grades be assigned? You'll have to upload your source code no later than the deadline specified on the assignment web page.

Topics

- Practice programming single and multiple source-file programs
- Learn to interpret warning and error messages from the compiler.
- Learn the development environment to become more efficient and productive.

References

1. Lecture 3 & 4 notes.
2. The world wide web.

Task 1

1. Begin by creating a directory called `tut02/task1` as a container and workspace for Task 1 files. Transcribe the following code in source file `main.c`:

   ```c
   #include <stdio.h>

   int sq(int x) {
       return x * x;
   }

   int main(void) {
       int i = 2;
       printf("Square of %d is %d", i, sq(i));
       return 0;
   }
   ```

2. Compile and link `main.c` as follows:

   1. In WSL go to the directory where `main.c` is saved.

3. Issue the following command:

   ```
   gcc main.c -pedantic-errors -Wstrict-prototypes -Wall -Wextra -Werror -
   std=c11 -o ex1
   ```

   The above is to ensure that your source file compiles without any warnings using the full slate of warnings options: `-Wall` ,`-Werror`, and `-Wextra`. We compile with the ISO C11 standard with option `-std=c11`. The compiled program is saved in a file called `ex1`.

4. Run the program by entering:

   `./ex1`

Task 2

1. Having `sq` defined in the same file as `main` is not so useful if we want other programs to use `sq` too. This is because of the following reason: Another program would have its own `main` function. Since `sq` is already with the Task 1 `main`, `sq` cannot be compiled with any other `main`.
2. To allow other programs to use `sq`, we can put `sq` in another file called `calc.c`:

   ```c
   /* calc.c */
   int sq(int x) {
       return x * x;
   }
   ```

   Text surrounded by '`/*`' and '`*/`' are comments for human readers that are ignored by the compiler.
3. Before creating `calc.c` we create a file containing a brief description of `sq`. This description includes the input information of `sq`, i.e. `int x`, and the return datatype `int`. This description is called a declaration of `sq`. We call the file containing the declaration '`calc.h`':

   ```c
   /* calc.h */
   int sq(int x);
   ```

4. We then include `calc.h` in `calc.c` as follows:

   ```c
   /* calc.c */
   #include "calc.h"
   int sq(int x) {
       return x * x;
   }
   ```

5. Next we include `calc.h` in the file where `main` is so that, when `main` is compiled, the compiler knows there is a function `sq` with the input and return information described in `calc.h`:

   ```c
   /* main2.c */
   #include <stdio.h>
   #include "calc.h"
   int main(void) {
       int i = 2;
       printf("Square of %d is %d", i, sq(i));
       return 0;
   }
   ```

6. Complete and submit `calc.h` and `calc.c` below so each function has a declaration in `calc.h` and a definition in `calc.c`.

```c
/* calc.h */

int sq(int x);

double cube(/* Add input parameter here */);

/* Add declaration for minus here */
```

```c
/* calc.c */

#include "calc.h"

int sq(int x) {
    return x * x;
}

double cube(double x) {
    return /* Calculate cube of x here */;
}

double minus(double x) {
    /* Add code to return negation of x */
}
```

7. To create the executable file we compile each `.c` file to produce a corresponding object file. Such a file contains the machine code for the associated `.c` file. The object filename suffix is changed to '`.o`'. Ensure the `.c` files are in the same folder and issue the following commands in the folder to compile the `.c` files and produce the object files:

```
gcc -c calc.c -pedantic-errors -Wstrict-prototypes -Wall -Wextra -Werror -
std=c11 -o calc.o
gcc -c main2.c -pedantic-errors -Wstrict-prototypes -Wall -Wextra -Werror -
std=c11 -o main2.o
```

8. Next we issue the following command to link the `.o` files to create a single executable file called `ex2`:

```
gcc calc.o main2.o -o ex2
```

9. Run `ex2` and check the output values are right.

File-Level Documentation

All source files must contain *file-level* documentation block at the top of each file. Here is a *template* of a file-level documentation block:

```c
/*!
@file     tut1.c
@author   Nicolas Pepe  (nicolas.pepe@digipen.edu)
          Rob Holding   (rob.holding@digipen.edu)
```

```
          Mezut Ozil    (mezut.ozil@digipen.edu)
@course   CS 120
@section  A
@tutorial Tutorial 1
@date     02/02/2020
@brief    This file contains code that puts a salmon on a cedar
          plank and smokes the fish for three hours. Remove the
          plank with the fish, throw away the fish, and enjoy the
          plank.
*//*_____________________________________________________________*/
```

Make modifications to the template so that it identifies your information correctly. Add this file-header at the top of source file `calc.c` that you will author and submit together with `calc.h`.

Deliverables and Submission

You must submit files `calc.h` and `calc.c` in Moodle by using the submission user interface there. After clicking on 'Submit', click on 'Continue' to grade your submission.