CS 240 Programming in C

Get Started!

September 9, 2019

Schedule

1. Dissection of hello.c

Schedule

- Today's lecture is about getting you started with C programming by introducing you a basic C program structure.
- It is friendly and easy for a new fresh starter.
- It is also important and meaningful to learn the concepts behind it.

hello.c

```
/* Project: HelloWorld
 * Name    : Haoyu Wang
 * Date    : 09/08/2019
 * File    : hello.c
 * Notes   : prints "hello world!"
 */
#include <stdio.h>

int main(void)
{
  printf("hello, world!\n");
  return 0;
}
```

Dissection of hello.c

The program structure of hello.c consists of 3 parts. They are

1. Comments

```
/* Project: helloWorld
 * Name    : Ming Ouyang
 * Date    : 09/06/2018
 * File    : hello.c
 * Notes   : algorithm, pseudo code, etc.
 */
```

2. A C preprocessor directive

```
#include <stdio.h>
```

3. A C function which includes:
   1. C function header

      ```
      int main(void)
      ```

   2. C function body, containing statements enclosed in braces

      ```
      {
        printf("hello, world\n");
        return 0;
      }
      ```

Comments

- All characters between /* and */ are ignored by C compiler
  Do not forget the closing */
- Write comments to make your program easier to understand
- Examples:

```
/* basic comment on its own line */
/*************************************************
* multiline comments sometimes use formatting like
* this to make the commment stand out.
*************************************************/
/* but this works
just as well */
printf("example\n"); /* comments can follow statements */
```

- C++ introduced the double slash for single line comments
- Most newer C compilers recognize this style, but it is not used in K&R

```
// double slash comment goes from slashes to end of line
printf("example\n"); // and can follow statements
```

Preprocessor Directive

- `#include <stdio.h>` is a C preprocessing expression
- `#include` is a preprocessor directive, allows us to share function declarations and macro definitions among several source files.
- For example, in "hello.c" we want to use printf, which is part of the standard I/O library, so we need the header file `stdio.h`
- Note, what actually it does is literally copy the contents to this file from the file it includes at that line position.

Header file

- `stdio.h` is a system library header file to be included. And the form of <stdio.h> has to be used for it.
- A header file has the .h extension
- You can write your own header file with name of "name.h"
- But to include a user defined header file, the statement should be: `#include "name.h"`

- For more knowledge of preprocessor and header file, wait for later.

The Entry Point of a Program: `main`

- main is a function and is the entry point of a program.
- It is the starting address of a block of codes.
- And it is where your program starts execution

Function header

- `int main(void)` is the header of the main function
- A function header is made of three parts:
  1. a function name
  2. the return data type before the function name
  3. the input data types in parentheses following the function name
- Note that K&R use the form `main()`, because the default input and return data type are void and int the if not specified
- Empty parentheses or void indicates the function does not take parameters

Function Body

- The body of your program is between the braces, { }

```
int main(void)
{
  [function body];
}
```

- The standard I/O library provides the function `printf()`, which is defined in stdio.h
- It prints its argument (the text between the parentheses) on the screen
  `printf("Hello, world\n");`
- All C program statements end with a semicolon (;)
- The argument in this statement is: `"Hello, world\n"`
- The quotation marks denote a character string
- `\n` is C notation for the new line character
  When printed, this advances output to the left margin of the next line

GNU C Compiler

- The default C compiler on a Linux system
- The command is gcc or just cc, which is an alias
  `gcc -Wall [sourceFile] -o [executableName]`
- The command `$ gcc -Wall hello.c -o hello` produces an executable file named hello
- `-Wall`, this option causes the compiler to warn you about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning)
- `-o [file]`, this option causes the executable file produced by gcc to be named [file]
  If this option is not used, the filename defaults to `a.out`

Compilation

- The term compile is often used as a high-level way to say "convert source code to a program"
- Specifically, to compile is to "convert source code to object code and then combine the object code into an executable"
- Source code is the human-readable instructions that a programmer writes such as `hello.c`
- Machine code is binary numerical data that can be read by a CPU to execute tasks in its instruction set
- Object code is source code that has been translated to machine code, but has not yet been linked into a complete program
- Object code files use the .o extension

Running a Program

- After you have successfully produced an executable file called `hello`, you can run it by typing:
  `$ ./hello`
- Recall that . denotes the current working directory
- Because the executable file you just created is not included in your PATH environment variable (which tells the shell where to find executable files), you are explicitly specifying where to find `hello`
- On windows, you can run it by typing:
  `$ hello`

Do it!

- Now, it is your turn to write the hello.c in class and run it.
- Think about how many parts it consists of and what they are used of.
- 2 minutes of your hands-on time.

The return of printf!

- We know that printf is a function.
- But what it returns?

The return of printf!

- It returns the number of characters it prints out.
- And note that the new line character "" is also counted as 1 character.
- Let's do a test.

prt.c

```
/* Project: HelloWorld
 * Name    : Haoyu Wang
 * Date    : 09/08/2019
 * File    : prt.c
 * Notes   : prints "hello world!"
 */
#include <stdio.h>

int main(void)
{
  int i=0;
  i = printf("hello, world!\n");
  printf("%d\n", i);
  return 0;
}
```

prt.c

- It prints this out:

```
hello, world!
14
```

prt.c

```
/* Project: HelloWorld
 * Name    : Haoyu Wang
 * Date    : 09/08/2019
 * File    : prt.c
 * Notes   : prints "hello world!"
 */
#include <stdio.h>

int main(void)
{
  int i=0;
  i = printf("hello, world!");
  printf("%d\n", i);
  return 0;
}
```

prt.c

- It prints this out:

hello, world!13

The arguments of main

- We can pass arguments values from the command line into main.
- These values are called command line arguments, and a specific main header should be used.
- int main(int argc, char *argv[])
  1. argc represents how many arguments in total for an command line arguments. It counts the directory and the program name from the beginning.
  2. argv[] is a pointer array which points to each argument passed to the program. Right now let's just see argv[0] is the program name and arg[1] is the first argument and so on.

args.c

```
#include <stdio.h>

int int main( int argc, char *argv[] )
{
  printf("The total arguments is %d\n", argc);
  printf("The program name is %s\n", argv[0]);
  printf("The first argument is %s\n", argv[1]);
  return 0;
}
```

Debugging a C Program

- The types of errors you will encounter are generally divided into three categories:

1. Compile errors
   An error makes the compiler unable to create an executable file
2. Runtime errors
   The program compiles, but performs some illegal operation during execution
3. Logical errors
   The program compiles and runs without generating errors, but it does not provide the desired result

Compiler Error Messages

- May direct you to a specific error in your program
- May be vague about what the error is and why it is an error
- Some compilers are better than others in this regard
- Example: try to compile `hello.c` with the closing brace missing