C Programming
a Q&A Approach

H.H. Tan
T.B. D'Orazio
S.H. Or
Marian M. Y. Choy

Additional Reading Material

Prepared by Marian M. Y. Choy

Chapter 1

AN OVERVIEW TO C PROGRAMMING

From program to executable file

There are 3 main steps to transform a computer program into an executable file. We are going to explore each of these with the analogy of baking a cake.

Step 1. (Preparation Step) Programmer uses an editor (eg. notepad, Pelles C) to write a C program. We call this the source code or the source file, this file has .c as the file extension. One thing interesting to observe is that this file is readable by the programmer. This action is similar to pouring the cake mix into the bowl.

Step 2. (Compilation Step) The source file from step 1 is being compiled. During compilation, the file is first passed to a preprocessor, which will make some changes to this file (eg. replace some of the content, add in other contents). This action of the preprocessor is similar to the step where ingredients are added into the cake mix bowl. Now, the source file is said to be complete (c.f. the bowl is filled with all the ingredients). Next, the compiler translates the complete source file into an object file with .obj as file extension. One thing interesting to observe is that this file is readable by the computer only. This action is similar to stirring and mixing everything in the bowl.

Step 3. (Linking Step) Now, after mixing the cake mix with other ingredients, it's ready to bake the cake. Similarly, the loader is ready to produce an executable file (.exe) from all the object files. Each source file corresponds to one object file. When we have a big problem to solve, usually we'll divide it into smaller pieces, say we write *n* source files, in this case n object files will be generated by the compiler. However, no matter how many object files we had, only one executable file will be generated for each problem.

What will happen if the computer get stuck at one of the steps? i.e. there errors are detected by the compiler. Unfortunately, it is possible to experience errors, maybe during compilation or during linking. In this case, the compiler will display error messages to us, we need to correct the error and then perform step 2 and 3 for a number of times until an exe file is generated

successfully.

Writing comments

The use of comments allows us to recap what we have written and comprehend others' programs effectively. With the huge among of examples available on-line, you might wonder what is the difference between:

```
/* this type of comments */
```

and

```
// this type of comments?
```

Can I use both in my program?

In C language, the syntax for writing comments is:

```
/* this type of comments */
```

whereas

```
// this type of comments
```

is for C++ language, another programming language. Pelles C and a lot of C compilers support both formats, however, as we are learning a standard C language, not a mixture of C and C++, therefore you are advised to stick with the proper style of writing C comments.

Preprocessing directive

This refers to those special instructions for the preprocessor to perform during the first stage of program compilation. There are 10+ preprocessing directives in C, each with a different name. An instruction like `#include` requires a specific toolbox to be included in the source program, whereas `#define` is used to map a name to a value. You might have noticed the # symbol in front of the preprocessing directive. Only preprocessing directive has this # symbol to start with, therefore it is very easy for us and for the computer to spot out those special instructions for the preprocessor. We are not going to learn all of them at this stage, it is easier to get more familiar with preprocessing directives when we start to program.

#include - opens the tool box

FOLLOW ME

Let's try to locate the standard I/O library file (stdio.h). Since this is a library file, we write

`#include <stdio.h>` and use `<>` to enclose the filename. Suppose you have installed Pelles C in C drive (installation guide is available under C resources in the course web). Go to the folder: C:\Program Files\PellesC\Include and you will see a lot of .h files and one of them being stdio.h. Drag this file to Pelles C to open it. Be careful not to modify anything in this file! This file contains all the library functions related to standard input and output. See if you can locate the function printf. .h is a header file, and its purpose is to tell us (i.e. programmers) the necessary information in order to use the functions, i.e. what kind of data to pass to each function and what kind of data to expect from each function. Operation details are not included in .h files.

Functions

Malvin inserts a $10 coin to the drink vending machine, and then selects his favorite drink, a bottled drink will rolled out to the dispensary hole. How does this work internally? We don't know. Can we use it without knowing how it works? Of course. The drink vending machine is just like a black box. To use it, we only need to know what to put in and what to expect from it. Function in a computer program can be treated as a black box. To make use of a function, we only need to know what information it expects and what information it will produce. How it works internally is not a big matter if we just want to use it and do not care about the performance. On the other hand, when we design a function, we want to develop it in a way that other programmers can use it conveniently, just like the vending machine.

`main` is considered a function in C. What makes `main` a special function?

main - program entry point

One day, as Karen was about to enter the lecture theatre, a visitor came up to her and asked, "How to get to the sport complex?" How are you going answer this question? Somehow, our answer will guide the visitor to the entrance of the sport complex. As a C programmer, we want to know the entrance to a C program too. `main` is the entrance to a C program. Some buildings might have more than one entrance, but C program only allows a single entry point. The computer will start executing the program at this entry point. `main` is also a reserved word (or keyword) in C language, therefore we cannot use `main` as the name of other things, it can only be referred to as the program entrance. Can we use another name for the entrance? No. Each program is expected to have one and only one `main`.

Standard program layout

The layout provides a skeleton for the programs we are going to write. Besides the given template, other components of good programming style will be introduced later. To train ourselves to be successful programmer, please observe proper programming discipline and follow the guidelines strictly.

In C, we need to declare all the variables at the beginning of a function, that is, before

writing any programming statement. When the program executes, the computer will first assign storage spaces to variables, and then execute the programming statement. This is neat to group all variable declarations at the start of a function. This is also a thoughtful way to plan and layout the program. The environment we are working on, Pelles C, is quite relax in this rule. Even though we might be messy to put declarations along with programming statements, the program runs ok in Pelles C. However, you are not encouraged to do so. Our aim is to become a successful programmer, not a messy programmer!

#define - gives a name to a value

Consider the following program [C01_interest_v1.c] which calculates the compound interest:

```
#include <stdio.h>

void main ()
{
   int principal=1000;
   double newValue;

   printf ("initial money: $%.1f\n", principal);

   newValue = principal * (1 + 0.0056);
   printf ("after 1 month: $%.1f\n", newValue);
   newValue = newValue * (1 + 0.0056);
   printf ("after 2 months: $%.1f\n", newValue);
   newValue = newValue * (1 + 0.0056);
   printf ("after 3 months: $%.1f\n", newValue);
   newValue = newValue * (1 + 0.0056);
   printf ("after 4 months: $%.1f\n", newValue);
   newValue = newValue * (1 + 0.0056);
   printf ("after 5 months: $%.1f\n", newValue);
}
```

Figure 1.1 Calculation of compound interest

What is the monthly interest rate? Where is the monthly interest rate? Paul wrote this program to find out how much he will get after five months if he puts $1000 in fixed term deposit at a bank. Some times afterwards, he found out there is a promotion in another bank, giving new customer a 0.61% monthly interest rate. He needs to revise his program to cope with the new interest rate. Isn't this troublesome to replace all 0.0056 with 0.0061?

After learning `#define`, Paul used this preprocessing directive in his program, making the program easier to cope with different interest rates. His new program [C01_interest_v2.c] now becomes:

```
#include <stdio.h>

#define rate 0.0056  /* monthly interest rate */

void main ()
{
   int principal=1000;
   double newValue;

   printf ("initial money: $%.1f\n", principal);

   newValue = principal * (1 + rate);
   printf ("after 1 month: $%.1f\n", newValue);
   newValue = newValue * (1 + rate);
   printf ("after 2 months: $%.1f\n", newValue);
   newValue = newValue * (1 + rate);
   printf ("after 3 months: $%.1f\n", newValue);
   newValue = newValue * (1 + rate);
   printf ("after 4 months: $%.1f\n", newValue);
   newValue = newValue * (1 + rate);
   printf ("after 5 months: $%.1f\n", newValue);
}
```

Figure 1.2 A modifiable way to calculate compound interest

Which style of writing do you prefer? What makes the program with `#define` a better choice? With `#define`, we can associate a name to a value, therefore whenever the computer sees the name (e.g. rate), it will replace the name with the value 0.0056 during the preprocessing stage. Now, if Paul want to check out other interest rate, he simply revise the value in `#define`, compile and then run the program to get the updated result. Of course, this is not the only solution for this kind of calculation. However, according to what we have learnt up to this stage, this is the best we can have.

printf () - displays information to the screen

FOLLOW ME

Run Pelles C. Under the menu bar, select Help. A window will pop out, click the Index tab, and then type **printf** to browse the help for "printf function". The information tells you briefly about `printf ()`, do not worry about `wprintf ()`, it is another display function and we are not going to use this. Click on the link formats for output, and the help will show you different kinds of formatting we can use inside the control string for `printf ()`. The list is long, just get a rough idea is ok at this stage, there is no need to go through all the details.

Your challenge

101 Give num a value and check out the meaning of these `printf ()`:

```
printf ("--%4d--", num);    /* what does %4d mean? */

printf ("--%04d--", num);   /* what does %04d mean?
                               Can I use another number
                               instead of 0? */

printf ("%d%%", num);       /* what does %% mean? */
```

102 Try to compile the following program:

```
void main()
{
   int num;

   printf ("Hi there!");
   #include <stdio.h>
}
```

What error message(s) the compiler display to you? Can you explain the error message(s)?

103 Follow the program layout in the examples to write a program which produce the following output:

```
Hi, I am 'C', say "Hello" to me.
```

What are the compilation errors you get when using " in `printf ()`? Try to replace `"Hello"` with `'Hello'`, what is the output? Now, change `"Hello"` to `\"Hello\"`, you will be able to produce the required output. Why? What does \" mean? Check out the help in Pelles C: Help >> Contents >> C99 language reference >> Elements of C >> Constants >> character constants to find out the use of \.