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 2

GETTING STARTED

Introduction

Mobile phones are so popular in that they are considered as a part of our life. There are a number of ways to call someone: key in a number to dial, dial a number from the phone book, speed dial, or even voice dial. To call Kate from the phone book, her number must have been stored in the phone book. To store Kate's number, memory is needed, either in the mobile phone or in the sim card. If we do not store a number in the phone book, then we need to key in the number each time to dial that number. You may wonder, even if this number does not exist in the phone book, it will appear in the recently dialed list after you have made the call, and there is no need to store it in the phone book. This is true only if this number always remains "recently dialed". In fact, the recently dialed list takes up memory.

In programming, we need to store a piece of information (e.g. a value) for later use. "Variable" and "constant" are terms which refer to this kind of storage. Value associated with a constant cannot be changed, whereas value associated with a variable can be changed during the execution of the program. In this chapter, we are going to learn about storing information to memory, retrieving information from memory, and a few programming skills as a kick start to program writing.

Good programming style

We all know the benefits of good programming styles. In practice, what can we do to achieve this?

I. Comprehensive comments. Make use of comments to help reader know what's going on with your program. Let's look at this program [C02_style_v1.c]:

```
#include <stdio.h>
void main ()
{
```

```
   int classes, people, mark, sum, i;

   scanf ("%d", &classes);
   while (classes > 0) {
      sum = 0;
      scanf ("%d", &people);
      for (i=0; i<people; i++) {
            scanf ("%d", &mark);
            sum += mark;
      }
      printf ("average: %.2f\n", sum*1.0/people);
      classes--;
   }
}
```

Figure 0.1 Program without comment

There is no comment in this program. According to our knowledge in C at this stage, we could not understand. For long programs, even we know all the skills, more time is needed to understand a program without comment. In the program below [C02_style_v2.c], comments are added, if you just follow the comments, you will be able to get a rough idea about what this program is about without knowing all the technical details.

```
#include <stdio.h>
void main ()
{
   int classes;         /* number of classes */
   int people;          /* number of students */
   int mark;            /* the mark of a student */
   int sum;             /* the sum of mark of a class */
   int i;               /* counter */

   scanf ("%d", &classes);  /* get number of classes */
   while (classes > 0) {    /* not every class is processed */
      sum = 0;              /* reset sum to zero */
      scanf ("%d", &people);

     /* perform this for each student in that class */
      for (i=0; i<people; i++) {
            scanf ("%d", &mark);     /* read his/her mark */
            sum += mark;             /* calc sum */
      }

     /* calc and display average mark of that class */
      printf ("average: %.2f\n", sum*1.0/people);
```

```
      classes--;
   }
}
```

Figure 0.2 Program being over commented

When we write comments, avoid writing too much, we are not writing a report! Besides, we assume others know C, so we just use comments to highlight key features of the program or some special ideas we have used in creating the solution. Actually, the comments written in the above program is a bit too much for a simple program. The following [C02_style_v3.c] has enough comments for this purpose:

```
#include <stdio.h>
void main ()
{
   int classes;          /* number of classes */
   int people;           /* number of students */
   int mark;             /* the mark of a student */
   int sum;              /* the sum of mark of a class */
   int i;                /* counter */

   scanf ("%d", &classes);    /* get number of classes */
   while (classes > 0) {
      sum = 0;                /* reset sum to zero */
      scanf ("%d", &people);
      for (i=0; i<people; i++) {
            scanf ("%d", &mark);
            sum += mark;             /* calc sum */
      }

     /* calc and display average mark of that class */
      printf ("average: %.2f\n", sum*1.0/people);
      classes--;
   }
}
```

Figure 0.3 Program having appropriate comments

II. Meaningful names. Meaningful name has similar purpose as comments, it allows reader to comprehend the program within a shorter time. Examples:

- `studentID` is more meaningful than `id`
- `sum` is more meaningful than `x`
- `area` is more meaningful than `a`

Besides these, we can make use of mixture of small letters, CAPITAL letters and digits to create meaningful names. Examples:

- `num2word` `/* convert number to word */`
- `avgMark` `/* average mark */`
- `dayOfWeek` `/* day of week */`

III. Organized layout with proper spaces and indentation. How you line up the programming statements will affect the readability of the program. Make use of indentation (i.e. use of tab), space, empty lines to organize the layout, as well as matching of parenthesis. If we make use of all these to organize the messy program in lecture notes, it is more organized (although we still have to work on the naming and commenting). Don't worry about the technical details of this program [C02_organized.c]:

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

char c[27],v[27],r1[26],r2[26];

int sn(char n,int ii)
{
   int i=0;

   while (i<ii) {
      if (c[i] == n)
            return i;
      else
            i++;
   }
   return 0;
}

void main()
{
   long cc;
   int i,s,E,p,r,q;
   char ss[50000],e[50000],C;

   scanf ("%d", &cc);
   while (cc>0) {
      scanf ("%d%d", &r, &q);
      for (i=0; i<r; i++) {
            scanf ("%s%s", ss, e);
            c[i] = e[0];
            v[i] = ss[0];
```

```
      }
      for (i=0; i<q; i++) {
            scanf ("%s%s", ss, e);
            r1[0] = ss[0];
            C = ss[0];
            s = 0;
            while (C != 'R') {
                  p = sn (C, r);
                  C = v[p];
                  s++;
                  r1[s] = C;
            }
            r2[0] = e[0];
            C = e[0];
            E = 0;
            while (C != 'R') {
                  p = sn (C, r);
                  C = v[p];
                  E++;
                  r2[E] = C;
            }
            while (s>=0 && E>=0 && r1[s]==r2[E]) {
                  s--;
                  E--;
            }
            s++;
            for (p=0; p<=s; p++)
                  printf ("%c", r1[p]);
            for (p=E; p>=0; p--)
                  printf ("%c", r2[p]);
            printf ("\n");
      }
      cc--;
      if (cc>0) {
            printf ("\n");
      }
   }
}
```

Figure 0.4 Program being properly organized

Compilation errors

FOLLOW ME

Let's explore some compilation errors using the program L02_bucket_v3.c.

```
Line #
     8  #include <stdio.h>
     9  #define PI 3.141592653589793
    10
    11  void main()
    12  {
    13     double radius, base;
    14     int height;
    15
    16     printf ("How big is your bucket?\n");
    17     printf ("Enter radius and height: ");
    18     scanf ("%lf%d", &radius, &height);
    19
    20     printf ("Base area = PI * r * r");
    21     base = PI * radius * radius;
    22     printf (" = %.2f\n", base);
    23     printf ("Volume = %.2f\n", base * height);
    24  }
```

Figure 0.5 Program L02_bucket_v3.c used to explore compilation errors

You can download this file from the C resources in the course web. Follow the instructions below to explore different compilation errors with this program.

Error #1

Rename `radius` in line 13 to `radii`. Compile the program. What does the compiler say? Note, there are 1 error and 1 warning. Read the error, what does it mean? Double click the error message and the cursor will jump to the error line (line 18). Why does the compiler say `radius` is undeclared? What do you think the warning means?

When the compiler says "undeclared identifier 'radius'", it means it does not understand what `radius` is, because the program does not define (or in technical term: declare) it.

Sometimes, when we declared a variable but did not use it in the program, the compiler will give us warning, in this case we know that maybe that variable (e.g. `radii`) is redundant, or maybe we have made a typing mistake (e.g. `radius` to `radii`) in such a way that compiler misunderstands `radii` is useless.

Error #2

Correct the above error and make the program error-free again. Now, delete the semi-colon (i.e. `;`) at the end of the `scanf ()` function in line 18. Compile the program. What is the error this time?

Compiler will try its best to predict the locations of all errors. In this case, it is looking for `';'`, and keeps searching until the end of line 18. No trace of `';'` is found, instead saw `printf` at the start of line 20. This is what the compilation error tells us.

Error #3

Correct the above error and make the program error-free again. Now, rename `#include` into `#includes`. There are three warnings when you compile, what do they say?

Because of the misspell of include, the compiler does not understand this new preprocessing command. Besides, without the standard IO library, the compiler does not know about `printf ()` and `scanf ()` functions.

Error #4

Correct the above error and make the program error-free again. Now, remove the # from `#include`. How many errors and warnings the compiler give you this time? Try to read each error and warning and see if you understand them all.

Because of one single error, a lot of error / warning messages can be resulted. So, do not panic if you see a lot of error messages, maybe all these are due to a few real errors.

From this exercise, you have some experiment with understanding errors and making corrections. Please aware that the compiler will try its best to understand our program and "guess" the errors, but not every "guess" is absolutely correct. So, you need to look around too! That is, after you double click the error and know where it is, scan through a few lines above and a few lines below that location, so that you will know more about the "guessed error" from the compiler, and be able to make appropriate judgment and corrections.

scanf () - reads user input

`scanf ()` is a function for reading user input via the keyboard, whereas `printf ()` is a function for displaying information to the screen. Both are considered as input / output tools. So, is there any difference or similarity in the use of `scanf ()` and `printf ()`?

	Formatting control string in printf	Formatting control string in scanf
int	%d	%d
char	%c	%c
double	%f	%lf

Table 0.1 Formatting control string in `printf ()` and `scanf ()` for fundamental data types

Actually, `printf ()` and `scanf ()` use the same formatting for all data types in their control strings, except for `double`, which is the only difference. Besides, address operator `&` is needed to read a value into the storage area, and we use this a lot in `scanf ()` function. However, there are exceptions, we will discuss more later in the course.

FOLLOW ME

Try the program C02_money_v1.c and input 2000, note down the response from this program. Now, remove `&` from the argument of `scanf ()`. Compile (there is no error message!) and run this program with the same 2000 as input. What did the program response this time? Why the response is different when `&` is missing?

`&` is the address operator. The programming statement

```
scanf ("%d", &hkDollar);
```

will store the user input into the variable `hkDollar`. Where is this variable located? Inside the computer memory. Where about in the computer memory is this variable located? At the location `&hkDollar`. The address operator will find out the address of this variable, so that the computer knows where to store the value.

Let's investigate the `&` operator in this `scanf ()`. Suppose, the address of `hkDollar` is 5000 (i.e. `&hkDollar` is 5000).

```
scanf ("%d", &hkDollar);
```

The user enters 2000 at the prompt. After executing the above `scanf ()`, the memory becomes:

Memory location	Memory content
4999	...
5000 (address of `hkDollar`)	2000 (value stored at this address)
5001	...

On the other hand, if we have this program segment:

```
hkDollar = 1999;           /* just give it a value */
scanf ("%d", hkDollar);    /* & is missing */
```

The user again enters 2000 at the prompt. After executing the above statement, the memory becomes:

Memory location	**Memory content**
1999	2000
2000	...
:	...
5000 (address of `hkDollar`)	1999

After the first statement, `hkDollar` becomes 1999. Then, in the second statement, the value 2000 from the user is stored at address 1999, which is a wrong location!! Why? It is because `&` is missing when calling the `scanf ()` function. Without the `&` operator, the computer will treat the value of `hkDollar` (i.e. 1999) as the address, and stores the user input in that location.

CHECK THIS OUT

To read two numbers, what is the difference between "`%d%d`", "`%d %d`", and "`%d-%d`" in the control string of `scanf ()` function?

When we use "`%d%d`" in the control string, it will read two integers, one after the other. Normally, the user will type two integers with a space in between, and then hit the Enter key (we are going to use [Enter] to denote the Enter key in the rest of this workbook). The computer is capable to differentiate the numbers from the space. Therefore, "`%d %d`" is not preferred.

"`%d-%d`" is used when we expect the user to input two integers with formatting information, e.g. we want the user to enter a date in dd-mm format, like 14-7. In this case, by including `-` in the control string, only one `scanf ()` is needed.

Now, consider the following program [C02_seat.c]:

```
#include <stdio.h>
void main()
{
   int row;
   char seat;
   scanf ("%d%c", &row, &seat);
   printf ("===%d%c===", row, seat);
}
```

Figure 0.6 A program displaying the input information

FOLLOW ME

Run the program and input 20A, what's the output? Now, duplicate the two programming statements and make this program performs the same action two times. Run the program and input 20A[Enter]30B. How does the program response? Now, swap the input order in the second `scanf ()` to something like this:

```
#include <stdio.h>

void main()
{
   int row;
   char seat;

   scanf ("%d%c", &row, &seat);
   printf ("===%d%c===", row, seat);

   scanf ("%c%d", &seat, &row);   /* change the input order */
   printf ("===%d%c===", row, seat);
}
```

Figure 0.7 A modified program to display the input information

Run the program and input 20A[Enter]B30, how does the program response this time? The computer does not response correctly in the last try. A second 20 is displayed instead of 30, and B is missing. Why did 30B get eaten up by the program? In the last try, we have entered:

```
20A[Enter]B30
```

through the keyboard. The enter key on the keyboard is considered as a character too, therefore, data are being captured like this:

```
20A[Enter]B30
```

`%d` of first `scanf ()` - tries to capture the number 20
`%c` of first `scanf ()` - tries to capture the character A
`%c` of second `scanf ()` - tries to capture the character [Enter]
`%d` of second `scanf ()` - tries to capture the character B

The first `scanf ()` works ok. However, [Enter] is considered a character, therefore it is captured by `%c` of the second `scanf ()`. Following this, the character B is captured by `%d` of the second `scanf ()`. The computer cannot understand B as an integer, therefore nothing is stored in the variable row. As a result, the value of row remains unchanged and stays as 20.

Tokens of C

FOLLOW ME

Let's check out other tokens of C from help. Open Help window in Pelles C: Help >> Contents >> C99 language reference >> Elements of C >> Keywords. You will see a number of keywords used by C. We are going to learn a lot, but not all of them in the course. For your interest, you might like to look inside a few that you are familiar with and check out the details.

More on operators

CHECK THIS OUT

What is the result of 16 / 3? To answer this in C, we need to ask, "What do you want? Do you want an integer result or do you want a real number?" Let's say,

```
int ansInt;
ansInt = 16/3;
```

Both 16 and 3 are integers, therefore the answer from this division is an integer. So, is `ansInt` = 5 or 6 after the division? Should be 5, because this is an integer division, therefore, the computer will take care of the integer result only.

What if we make the variable `ansReal` a `double` instead of an `int`, the program segment becomes:

```
double ansReal;
ansReal = 16/3;
```

This time, `ansReal` is a real number, what will `ansReal` be after the division? Should be 5 again. Why?? Both numbers involved in the division are integers, therefore result is definitely an integer, no matter what kind of data type we use to store it.

How can I get the answer in real number? Try this:

```
double ansReal;
ansReal = 16.0/3;
```

By changing 16 to 16.0, we have a real number divides an integer, and the result will become a real number. Actually, there is another way to do this, we will learn more in the second half of the course.

The reasoning for 16 / 3 applies to 16 * 3 as well. As for 16 % 3, this is a modulus operator for the calculation of remainder, it only works for integers (why?). So we do not need to consider the case for real numbers.

Increment and decrement operators

Be extra careful when using `++` or `--` operators. Just `a++` alone is easy to understand, however, when we have `b = a++`, it starts to get complex. From lecture, we know that this is indeed two steps combined into one programming statement. First action: cover `++` with your fingertip and the programming statement will perform according to what you see now (i.e. `b=a;`). Second action: computer will then perform `a++`. The same goes with `--` operator.

CHECK THIS OUT

Assume all `a`, `b`, `c` below are integers. Find out their values after each programming statement. You can write a program similar to the one in the lecture notes to check your answer.

Program segment	**Value of** `a`	**Value of** `b`	**Value of** `c`
`c = 5;`			
`a = c++;`			
`b = 3 * (a--) - c;`			
`C = (b++) + a;`			

Your challenge

201 Refer to the program [C02_seat.c] discussed above, if the user input remains unchanged:

```
20A[Enter]B30
```

and we want the output to look like this:

```
===20A===
===30B===
```

Can you suggest a way to modify the program so that the output is as expected above?

202 Explore those programs we have learnt up to this stage, see if you can discover other possible compilation errors and explain the linkage between the error and the error message reported by the compiler.

203 With the use of scanf, construct a flexible program for Paul's investment queries. He wants to find out how much he will get after five months under different investment conditions.