C Programming Exercises

Dr. Alun Moon

Jan 2007

Abstract

Instructions: Work through the exercies. The first section is a general set of problems to help you remember what you know about programming. Later sections go into detail about parts of C. If you get stuck in one section, try again, then go on to the next section and come back later.

Important reading includes (McGrath 2006) a good introduction to C. (Kernighan & Ritchie 1988) is *the* book for the C89 standard for ANSI C. (Holmes 1995) is a very good internet tutorial and reference.

1 General examples

These are general exercies of varying complexity. A good way to refresh memory of C and build problem solving skills.

Exercise. 1.1 Write and run the "hello, world!" program. Experiment with different output formats by inserting newlines and tabs in the control string.

Exercise. 1.2 Write a program to output an integer in decimal format and a floating point number in float format. Experiment with outputting an integer in float format and visa versa. Output a character in decimal format (prints the character code e.g. ascii). Fix any problems by using an explicit type coercion or cast.

Exercise. 1.3 Write a program to determine, using trial and error, the minimum value of $196/n + n$ where n is an integer. Don't use a loop, try different values of n. Note that in general $196/n + n$ will be a float quantity.

Exercise. 1.4 Write a program to check that the volume and surface area of a sphere of radius 0.1234567789e2 is approximately 7881.948508, and 1915.313445 respectively. The volume of a sphere is four thirds Pi times the radius cubed and the area is four Pi times the radius squared. Take Pi to be 3.14159265.

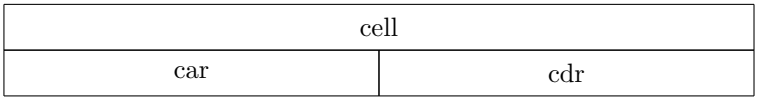
Exercise. 1.5 A number is special if it is divisible (no remainder) by 15. A number is big if it is greater than 999. A number is weird if it is divisible by 5 and 6 but not 18. A number is scary if it is big or weird.

Write a program to check which of the following, 450, 540, 600, and 675 are special but not scary.

Declare four variables called `special`, `big`, `weird`, and `scary` and make suitable assignments to these variables as a number is tested.

Exercise. 1.6 Write a program to express a given number of seconds (which may be a literal in the program) in terms of hours, minutes and seconds and output the result. Use distinct variables for distinct quantities. For example, for representing the total number of seconds which can be greater than 59 and the seconds part of the output which can never be greater than 59.

Exercise. 1.7 Write a program to independently store and extract two short 16 bit integers (called car and cdr) in one 32 bit integer (called cell). All Integers are positive.



The storage or access of one number should not affect the other . For example, if the cell contains a car value of 2 and a cdr value of 3, then the value of the cell is 131075 (hex 00020003). If the car value is subsequently stored in the cell, the cell's value becomes 65539 (hex 00010003).

Begin by declaring suitable integers and initialise them. Write functions to set and extract the values of car and cdr.

Write a version of the previous program using a structure with fields car and cdr.

Test both with various examples of car and cdr.

Exercise. 1.8 Write functions,using conditional expressions,

1. abs, which returns the absolute value of an integer,
   e.g. $3 \mapsto 3$ and $-3 \mapsto 3$
2. zeroCut, which returns the maximum of an integer ans zero
   e.g. $3 \mapsto 3$ and $-3 \mapsto 0$
3. absDiff, which returns the absolute difference of two integers

2 Strings and I/O

Exercise. 2.1 Consider the program

```
#include <stdio.h>
char word[] = "fire";
main()
{
    word[0] = 'h';
    printf("%s\n",word);
}
```

run it with the following substitutions added before the printf,

- word[1]='e';
- word[3]='d';
- word[4]='\n';

what happens in the last case?

Exercise. 2.2 Write a program to write out the string arguments to `main` in reverse order.

Exercise. 2.3 An argument to `main` is either an *option* or a *word*. An option starts with a hyphen character '`-`', the remaining part is the option name.

Write a program to examine each argument and print out whether it is an option or a word, along with the option name or the word value. An example output is shown below.

```
option: version
word: 1.5.a
```

Exercise. 2.4 `scanf` naturally splits its input at whitespace (see the man page). Using this fact write a program that counts words read from the standard input. *Hint:* read the man page for scanf, check the section "RETURN VALUE" for a clue on how to end the loop.

"What loop?" – think about it!

3 Characters and Strings

Computers represent characters with numbers, each number representing a letter of the alphabet. The most common system is ascii, in which 'A' is 65, 'B' is 66, 'C' is... Upper case letters are in the range 65...90, and lower case letters are 32 higher, 97...122. In C individual can be stored in a char data type. The values can be written as the letter in single quotes 'A'. Alternatively the number can be written, in which case the value need to be cast into a char type.

The following statements are equivalent

```
c = 'A';
c = (char)65;
```

arithmetic can be done with characters

```
c = 'A'+5;
b = 'Z'-'A';
```

As well as the printable characters, there are ascii codes for "special" characters, some of these are

0	'\n'	null character (string terminator)
7	'\a'	bell (beep)
9	'\t'	horizontal tab
13	'\r'	return

3.1 Characters

Exercise. 3.1 The following program prints out the alphabet from A to Z. Copy, compile and run the program.

```
#include <stdio.h>
main()
{
    char i;
    for(i='A' ; i<='Z' ; ++i)
        putchar(i);
}
```

Modify the program to print just the first 10 letters of the alphabet.

Exercise. 3.2 Copy, compile and run the following program. It takes its input in the same way cat does (except it does not read files). If reading from the keyboard it needs at least one carriage return, and a control-D to end input. This is due to the way the kernel buffers input, not the way the program reads data.

```
#include <stdio.h>
main()
{
    char c;
    c = getchar();
    while (c != EOF ) {
        if ('A' <= c && c <= 'Z')
            c = c + 32;
        putchar(c);
        c = getchar();
    }
}
```

What does the program do?

Exercise. 3.3 Compare the last program with the following

```
#include <stdio.h>
#include <ctype.h>
main()
{
    char c;
    while ((c=getchar()) != EOF) {
        if (isupper(c))
        c = toupper(c);
        putchar(c);
    }
}
```

3.2 Strings

Strings in C are stored as a sequence of ascii codes (array of char) terminated by a null character `\0`. The string "hello-world" has 11 visible characters, but needs 12 storage locations for the 11 characters and the null character.

Exercise. 3.4 Copy and try this program

```
#include <stdio.h>
main()
{
    char a[27];
    int i;
    for ( i=65 ; i<91 ; ++i)
        a[i-'A'] = i;

    a[26] = '\0';
    printf(a);
}
```

Now remove the line a[26] = '\0';, what is the effect?

The program will still try to print the letters 'A'...'Z'. With the zero, the output stops. Without the zero the program keeps going, printing whatever is in memory – random characters, until a zero byte is reached.

In a C program a string value can be written using *double* quotes `"`, the compiler will allocate the memory and add the zero byte. The input functions `fgets`, and the `%s` field in `scanf`, also add the zero byte. There are a number of functions to provide information about, or manipulate strings. To use these the `string.h` header file is needed.

- `strlen(char s)` returns the length of the string (not including the zero byte).
- `strcat(char s,char cs)` will copy the characters from the second string onto the end of the first. If the first string does not have enough space strcat will keep going overwriting whatever is in memory.

Exercise. 3.5 What does the following do

```
#include <stdio.h>
main()
{
    enum { BUFSIZE = 100 };
    char buf[BUFSIZE];
    int i;
    fgets(buf, BUFSIZE, stdin);
    buf[strlen(buf)-1]='\0';/*delete the newline character*/
    for (i = 0; buf[i]; ++i)
        if (islower(buf[i]))
            printf("%c is lower case\n", buf[i]);
        else if (isupper(buf[i]))
            printf("%c is upper case\n", buf[i]);
        else
            printf("'%c' is neither\n", buf[i]);
}
```

how can the program be improved (for clarity) by including more braces and ?

Exercise. 3.6 The following program will read words from the standard input, and concatenate them together. It stops when either the end of the file (control-D) is reached or the buffer is full. The `scanf` function returns an `EOF` value if it tries to read past the end of a file.

```
#include <stdio.h>
#include <string.h>
main()
{
    enum { BUFSIZE = 1000, INPUTSIZE = 100 };
    const char SEP[] = " ";
    char s[BUFSIZE];
    char cs[INPUTSIZE];
    *s = '\0';
    while (scanf("%s",cs)!=EOF && strlen(s) < (BUFSIZE - INPUTSIZE)) {
        strcat(s, SEP);
        strcat(s, cs);
    }
    printf("%s",s);
}
```

- experiment with changing the SEP string. Try `", "` or `"\t"`.

Exercise. 3.7 3. Write a program that reads a users first and last name, on separate lines (hint use `fgets` it stops at the end of a line, `scanf("%s")` stops at a space). The output is the two names as a single string, for example "smith, john" keeping the case of the input.

Exercise. 3.8 Write a similar program that outputs the name in the form "SMITH,John", this time use the case in the example string.

References

Eckel, B. (2000). *Thinking in C++, Volume 1: Introduction to Standard C++ (2nd Edition)*, Prentice Hall.
`http://mindview.net/Books/TICPP/ThinkingInCPP2e.html`

Hanly, J. R., Koffman, E. B. & Horvath, J. C. (1997). *C Program Design for Engineers*, Addison-Wesley. classmark 518.567.5 HAN.

Holmes, S. (1995). C programming, Internet tutorial, University of Strathclyde Computer Centre.
`http://www.its.strath.ac.uk/courses/c/`

Hunt, A. & Thomas, D. (1999). *The Pragmatic Programmer*, Addison-Wesley.

Kernighan, B. W. & Ritchie, D. (1988). *C Programming Language (2nd Edition)*, Prentice Hall PTR.

Knuth, D. E. (1984). *TeXbook*, Addison-Wesley Professional.

Knuth, D. E. (1999). *The Art of Computer Programming, Volumes 1-3 Boxed Set*, Addison-Wesley Professional.

McGrath, M. (2006). *C programming*, In Easy Steps, Computer Step.
`http://www.ineasysteps.com/`

Pike, R. (n.d.). Notes on programming in c.
`http://www.lysator.liu.se/c/pikestyle.html`

Pozo, R. & Remington, K. (n.d.). C++ programming for scientists, NIST.
`http://math.nist.gov/~RPozo/c++class/`

Raymond, E. S. (1999). *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary (O'Reilly Linux)*, O'Reilly.

Raymond, E. S. (2004). *The Art of Unix Programming*, Professional Computing Series, Addison-Wesley. http://catb.org/esr/writings/taoup/html/.

van der Linden, P. (1994). *Expert C Programming: Deep C Secrets*, Prentice Hall.