

A Rudimentary Intro to C programming

Wayne Goddard

School of Computing, Clemson University, 2008

Part 1: Variables and Data-types

1	Getting Started	A1
2	Some Basics	A4
3	A Worked Example	A8
4	Floating-Point Numbers	A10
5	Chars and Operators	A13

Getting Started

1.1 Computer Programming

A C program is a list or sequence of instructions for the computer. C is a *high-level* programming language: it is written in characters and words that represent concepts meaningful to humans. However, the CPU/chip understands *machine code*. The *compiler* (and its helpers) converts the one to the other. So you write a program using an editor, then run the compiler, then run the program (all being well). Compared to other high-level programming languages, C is noted for the fact that it facilitates direct access to the machine level. This is both an advantage and a disadvantage.

1.2 Goals

The goal of the course is for you to:

be able to produce and understand a short C program

Though it is not the focus of the course, you should somewhat bear in mind the goals of good programming. These include clarity and efficiency. For this course, a good program:

- works for all the cases it is supposed to
- in some manner deals with all other cases
- is self-describing
- is flexible

We will come back to these ideas later in the course.

1.3 Hello World

Here is a sample program. It is traditional for the first program to be one that displays the statement “Hello world”.

```
//This simple program displays "hello world" to the command line.
#include <stdio.h>

//Main function prints "hello world"
int main(void) {
    printf("hello world\n");
    return 0;
}
```

The *syntax* of the language gives the rules for the format of statements—what goes where. Common syntax errors include missing semi-colons, missing quotation marks and incorrect brackets/curly braces.

The line starting with `//` is a comment, and is ignored by the compiler.

A *string* is a sequence of characters. There are a few dozen *keywords* (or reserved words) such as `int`, `main` and `return` which are in-built in the language. The word `printf` is a function call that is added by *including* the file `stdio.h`.

1.4 Running your Program

On our system, the code is compiled using the program called `gcc`. If you have saved your code in file `myprog.c`, then you can type

```
gcc myprog.c
```

But you should immediately get in the habit of

```
gcc -Wall myprog.c
```

which will print all warning messages. (It will help later in the course.) Then to run the code you use

```
./a.out
```

The compiler produces the executable file `a.out`. The `./` part tells the operating system to look in the current directory. (Why it doesn't automatically is another story.)

Be aware that there are variations of C. However you develop your code, it must run on the CS system using the above process.

1.5 Rules of C

Brian Brown's summary:

<http://gd.tuwien.ac.at/languages/c/programming-bbrownc/start.htm>

- program execution begins at `main()`
- keywords are written in lower-case
- statements are terminated with a semi-colon
- text strings are enclosed in double quotes
- C is case sensitive, and we use mainly lower case
- `\n` means position the cursor on the beginning of the next line
- `printf()` can be used to display text to the screen
- the curly braces `{}` define the beginning and end of a program block

Some Basics

We look at variables, assignments and input/output.

2.1 Program Structure

To start with, our programs will only have a `main` section. Thus they will all start:

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

and end with:

```
    return 0;
}
```

We will discuss later why things are this way. (A preview is that the `main` part is a special case of a *function*, and the `return 0` sends a message to the operating system that all went well.)

2.2 Variables

Values are stored in memory via *variables*. Variables are *declared* before use:

```
int score;
int fred;
```

This code does two things for each variable:

- it reserves memory for the variable, which will store a whole number; and
- it tells the compiler that any other use of the word `fred` refers to this variable.

Once a variable has been declared, it can be *assigned* a value:

```
score=2;
fred=score;
```

Variables can be *initialized* at declaration. For example:

```
int count = 0;
```

Note that C does NOT automatically initialize variables; until you assign them a value, they can have any value (whatever was left in memory last time that cell of memory was used).

There are rules about what can be used as a variable name. Most of the time one just uses lower-case letters, with a sprinkling of numbers, upper-case letters and the underscore character `_`. Note that *reserved words* such as `int` that have a specific meaning cannot be used as variable names.

2.3 Arithmetic Operators

For doing calculations with numbers there are many operators. The first four are the standard arithmetic operators:

+ is addition, - is subtraction, * is multiplication, / is division

Note that the division of integers always produces another integer—what elementary-school kids call the quotient. For example, the following code will cause `x` to have the value 7:

```
x = 22/3;
```

Then there is the modulus operator:

% gives the remainder of integer division

For example, the following code will cause `x` to have the value 2:

```
x = ( 2 * ( 3 + 5 ) ) % 7 ;
```

The modulus operator is useful for getting values to wrap around. For example, if we had a variable storing the hour on a 24-hour clock, it might be changed by:

```
hour = (hour + 1) % 24 ;
```

If the `hour` is less than 23, this just increments it by 1; but if `hour` is 23, this resets it to 0.

There are rules for *operator precedence*: which operator is executed before which. Most of the time these rules behave just as one would expect. For example, multiplication has precedence over addition.

2.4 Simple Output

Almost all programs have some output. We have already seen `printf`, the function used to print output to the terminal or screen. This has a general form of

```
printf( formatString, zeroOrMoreDataValues);
```

The format string contains both normal characters and special codes. These codes start with % or \. For example, a %d in the format string stands for a number (d for decimal), and a \n means to advance the cursor to the start of the next line. The data can be in the form of actual values or variables.

```
printf("Message for the day is as follows\n");  
int four = 4;  
printf("Today I learned that %d + %d is %d", 2, 2, four);
```

will produce the output

```
Message for the day is as follows
Today I learned that 2 + 2 is 4
```

2.5 Simple Input

The simplest input function is getting a value from the user at the console. This is provided by the `scanf` function. To get one value from the user, use the format

```
scanf( formatString, addressOfVariable);
```

For example, the following reads the value of `number` from the user:

```
int number;
scanf("%d", &number)
```

The ampersand tells the compiler to use the *address* in memory of the variable (to be discussed later). If you leave this out, the code will compile but you will very probably get the famous “Segmentation fault” error. Note that if you omit the ampersand but are compiling with warnings enabled, `gcc` will warn that “format argument is not a pointer”.

2.6 Sample Code: `adder.c`

Here is code that reads in two numbers, adds them together, and prints out the answer.

Commentary: the program starts with the other form of C comments: in this form, the comment starts at `/*` and finishes at `*/`. The remaining stars are just a standard style thing to help the reader realize that the comment continues.

```
/* adder.c
 * This simple program adds two user-supplied numbers
 * wdg 2008
 */
#include <stdio.h>

//Main function
int main(void) {
    int num1, num2, sum;
    printf("Input first number: ");
    scanf("%d", &num1);
    printf("Input second number: ");
```

```
scanf("%d", &num2);  
sum = num1+num2;  
printf("The sum of %d and %d is %d\n", num1, num2, sum);  
return 0;  
}
```

A Worked Example

Consider the problem of making change with US coins for a specific amount. The goal is a C program to do this. But first we introduce constants.

3.1 Constants

If you have the same fixed value recurring throughout the program, you should use a constant at the top of the program. The “new” C style is to use the reserved word `const`:

```
const int NUM_STATES = 50;
```

For the rest of the program, `NUM_STATES` is just a normal variable, except that its value cannot be changed. It is common style to use uppercase for constants.

3.2 A Program for Making Change

With any programming task, you should think how **you** would do it (as a human computer). And then plan the steps.

In this case, the key question is what recipe (what computer science calls an **algorithm**) should be used for making change. But cashiers know: take as many quarters as are at most the amount; then as many dimes so that still at most the amount; and so on.

The first plan for a program might be:

- 1) Input number
- 2) Calculate number of quarters
- 3) Calculate number of dimes
- 4) Calculate number of nickels
- 5) Calculate number of pennies
- 6) Output results

Calculating the number of quarters uses integer division. How?

Suppose input is stored in `amount`, an `int` variable. Then the number of quarters is

```
amount / 25
```

And you need to make change for the remainder, which is

```
amount % 25
```

And now for a program, called `makeChange.c`

```
/* making change with coins
 * wdg 2008
 */
#include <stdio.h>

const int QUARTER = 25;
const int DIME = 10;
const int NICKEL = 5;

int main( void ) {
    int amount, quarters, dimes, nickels, pennies;

    printf("Enter amount of change: ");
    scanf("%d", &amount);

    quarters = amount / QUARTER;
    amount = amount % QUARTER;
    dimes = amount / DIME;
    amount = amount % DIME;
    nickels = amount / NICKEL;
    pennies = amount % NICKEL;

    printf("%d quarters, %d dimes, %d nickels, %d pennies\n",
           quarters, dimes, nickels, pennies);

    return 0;
}
```

Note that the call to `printf` is spread over two lines; most white space is just there for the human reader and is ignored by the compiler. Strings are an exception: they cannot be broken over a line.

Floating-Point Numbers

We explore the use of real numbers in C.

4.1 Floating-Point Numbers

A floating-point number is one that can have a fractional part. C has two main *data types* that store floating-point number: `float` and `double`. Note that any fraction has a fixed size of memory for storage; you can think of it as a fixed number of decimal places (though this is not quite correct). The data type `double` usually has more precision than `float`. Bowing to convention, we will use `float` throughout.

```
const float LB_PER_KG = 2.2046;
float weightInPounds = 191.6;
float weightInKilos = weightInPounds / LB_PER_KG;
```

Type conversion occurs automatically: you can set a `float` equal to an `int`, and a `float` to a `double` without compiler complaint. Of course, going from `double` to `float` might cause some loss of accuracy.

One famous trap is the integer division trap: the following code sets `ratio` to 0.0 because the integer division $2/5$ is evaluated as equal to 0:

```
double ratio;
ratio = 2/5;
```

If at least one of the values is neither a whole number nor an integer variable, then normal division is performed. The following codes sets `ratio` to 0.4.

```
ratio = 2.0/5;
```

4.2 Input and Output

In `scanf` and `printf`, floating-point numbers are done by the `%f` field. For reading floating-point numbers, just use `%f` by itself:

```
printf("Enter annual interest rate: ");
scanf("%f", &rate);
```

For printing out floating-point numbers, one can specify the space taken and the number of decimal places (amongst other things). For example, `%6.2f` means to output the floating-point number using 2 decimal places and a total of 6 character positions (adding white space before if needed). This enables one to align a list of amounts. Of course, the program can only do its best: it might be that the number really cannot fit into 6 positions. This type of alignment can also be used with `ints`.

```
printf("The class average is %.1f\n", 85.47);
```

will print out the float with 1 decimal place:

```
The class average is 85.5
```

Practice. Write code to read 5 float values from the user, and calculate their average. For example, if the data is 11.2, 22.3, 33.4, 44.5 and 0.0, then the answer is 22.28.

4.3 Mathematics

If you `#include` the `math.h` file, then you have access to mathematical functions such as `sqrt`. But note that you have to get `gcc` to load the math library:

```
gcc file.c -lm
```

4.4 Sample Program: `goldBall.c`

A program that prompts the user for the diameter of a golden ball, then calculates the volume, surface area and mass.

```
/* goldBall.c */
#include <stdio.h>
#include <math.h>

int main(void) {
    const float PI = acos(0.0); // or you could hardcode 3.14159 etc
    const float DENSITY = 19.3; // grams per cubic cm
    float diameter, radius, volume, surfaceArea, mass;

    printf("Enter diameter of gold ball in cm ");
    scanf("%f", &diameter);
    radius = diameter/2;

    volume = 4 * PI * radius*radius*radius / 3 ;
    surfaceArea = 4 * PI * radius*radius ;
    mass = volume * DENSITY;

    printf("Your gold ball of diameter %.2f has\n", diameter);
    printf("volume %.2f cm^3,\n surfaceArea %.2f cm^2,\n and mass %.2f g\n",
           volume, surfaceArea, mass);
}
```

```
    return 0;
}
```

4.5 Sample Program: converter.c

A program that converts from dollars to another currency, rands.

```
// converter.c - wdg 2008
#include <stdio.h>

int main(void) {

    // declarations
    float dollars, rands, exchangeRate;

    // input
    printf("Enter dollar amount ");
    scanf("%f", &dollars);
    printf("Enter rands per dollar ");
    scanf("%f", &exchangeRate);

    // processing
    rands = dollars * exchangeRate;

    //output
    printf("%.2f dollars is %.2f rands\n", dollars, rands);

    return 0;
}
```

Chars and Operators

We explore further the operators and char type.

5.1 Character Variables

The other in-built data type is `char`. This stores a character. For example:

```
char letterGrade = 'A';
```

The sequence `\n` is considered a single character, sometimes called a *special* character.

Type conversion occurs: you can set a `char` equal to an `int`, and an `int` to a `char` without compiler complaint. Indeed some people prefer to think of `char` as a subtype of `int`.

5.2 Input and Output

In `scanf` and `printf`, chars are done by the `%c` field. For example,

```
printf("The threshold for an %c is %.1f\n", 'A', 89.23);
```

will print out:

```
The threshold for an A is 80.2
```

Reading characters is trickier because of issues with the operating system and invisible characters. One possibility is simply

```
char response = getChar();
```

But beware mixing character input and other input.

Escape sequences: we have seen that `\n` in a string produces a newline. If you really want a backslash in a string, you need to use `\\`. A percentage sign needs `%`.

5.3 More Operators

C has the increment and decrement operators written after the variable:

```
++ is increment; -- is decrement
```

For the time being, we will use them only as simple single statements. The following code will cause `counter` to have the value 6:

```
int counter = 4;
counter++;
counter++;
```

The increment and decrement operators are usually used with `int` variables, but they work with `char` variables too.

There are also short-cut operators. One can easily survive without them, but they can enhance the readability. They are used when the variable on the left is the first argument on the right. For example, the above code with `counter` could be written as `counter = counter + 2`. But one might write is as:

```
counter += 2;
```

There are also `*=`, `-=`, `/=`, and `%=`.

5.4 Sample Program: `alphabet.c`

The following program reads a position from the user and prints out the corresponding letter of the alphabet and the next one. Note that it fails if user puts bad value of position, or even if user puts in 26.

```
// alphabet - wdg 2008
#include <stdio.h>

int main( void ) {

    int pos;
    printf("Enter position in alphabet: ");
    scanf("%d", &pos);

    char upper = 'A' +(pos-1);
    char lower = 'a' +(pos-1);
    printf("Position %d is %c in upper and %c in lower\n", pos, upper, lower );

    upper++;
    lower++;
    printf("Next comes %c in upper and %c in lower\n", upper, lower );

    return 0;
}
```