Programming with C

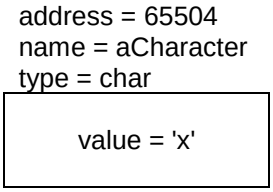
Terry Marris November 2010

2 Strings

Previously we saw that a string constant was just a sequence of characters enclosed within quotation marks. Now we take a look at string variables.

2.1 Variables

A variable is a location in memory. It has an address, a name, a type and stores a value.



The address is assigned by the system. The name is chosen by the programmer. The type is also chosen by the programmer. The value may be chosen by the programmer or by the user.

2.2 Primitive Data Types

Primitive data types include:

char	character	letters of the alphabet, digits, space, symbols such as !,£ and $
int	integer	a whole number such as.-2, -1, 0, 1, 2 and 3
double	double precision floating point number	3.1416, 37.0, -0.001

These data types are known as primitives because they are part of the C language. Programmers can define more complex types. We shall discuss data types in the next few chapters.

There are others primitives, but we shall restrict our attention to just these three since they are sufficient for most purposes.

2.3 Variable Declarations

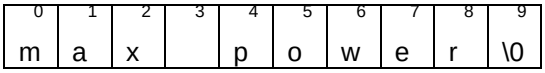
Variables need to be declared before they can be used in a program. A variable declaration starts with a type, followed by a variable, name, and, optionally, an initial or starting value.

```
char gender = 'm';
int age = 24;
double height = 1.78;
```

Variable names chosen by the programmer usually reflect their purpose. There are some restrictions on what the programmer can use for variable names. They must begin with a letter, may contain digits, but spaces are not allowed. Case is significant: *Age* and *age* are two different variable names. And they cannot be a C word. C words we have met so far include *main*, *int*, *return*, *char* and *double*.

2.4 String Input Output

A string is a sequence of characters. The characters are indexed from zero upwards.



The first character is at location zero. The last character is at location nine. The last character in the sequence must be the end-of-string character, *\0*. There are 10 locations altogether. To declare a variable that can hold a name of up to 10 characters, including the *\0* character, we might write:

```
char name[10];
```

This says we have an *array* of 10 elements, indexed from 0 up to 9, where each element can hold a value of type *char*. We shall have more to say about arrays in a later chapter.

What if a name has more than ten characters? One option is to declare the array to be amply large enough.

```
char name[BUFSIZ];
```

where *BUFSIZ*, defined in *stdio.h*, is typically 512.

To get input from the use at the keyboard we display a prompt and then collect their keystrokes.

```
printf("Your name? ");
gets(name);
```

gets() waits for the user to enter a sequence of characters at the keyboard. The *enter* or *return* key-press signals the end of the keyboard input. *gets()*, for get string, is defined in *stdio.h*.

Notice the space following the prompt *Your name?* Without this space the user's input will be hard up against the prompt. For example: *Your name?**Terry Bull***. This is ugly. Notice also that there is no newline character in the prompt - the user supplies this when a value is entered.

Here is the complete program.

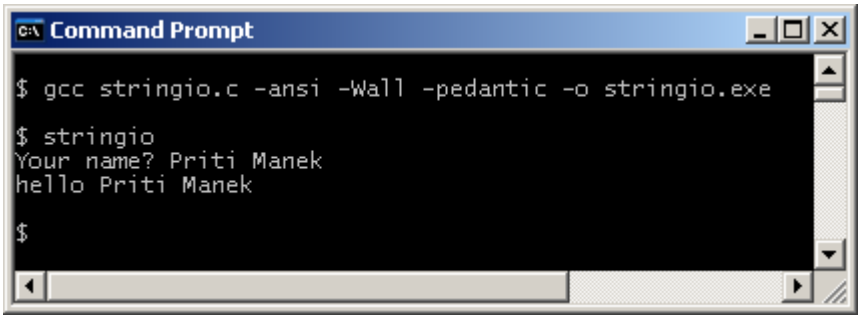
```
/* stringio.c: inputs and outputs a string */

#include <stdio.h>

int main()
{
  char name[BUFSIZ];

  printf("Your name? ");
  gets(name);
  printf("hello %s\n", name);
  return 0;
}
```

Look at the *printf()* statement.

```
printf("hello %s\n", name);
```

The text between the quotation marks, *hello*, is printed exactly as you see it, but the *%s* is not. *%s* is an example of a *conversion specification*. It specifies that *name* is to be printed as a string.

Here, the user has chosen to enter *Priti Manek* for a name.



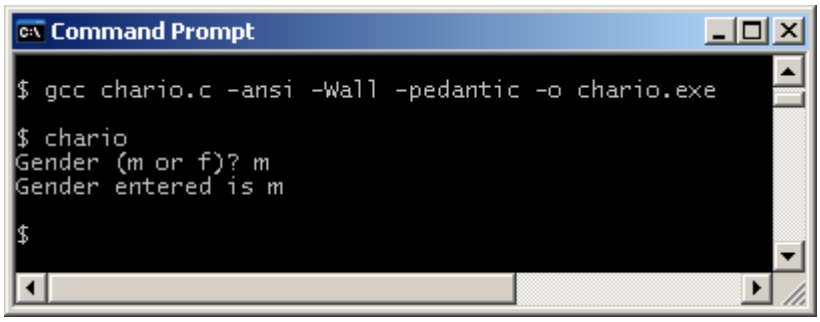
There is a problem with using *BUFSIZ* to declare the maximum size of a string. Usually, a lot of memory reserved for the string is unused. Memory is a finite resource. But if we attempt to store a value larger than the string can take, we have an error. We need to define a string size that is just right, no matter what is input. We address this problem in the chapter on functions

We use string input as the basis for character and number input.

2.5 Character Input Output

The character input output program shown below prompts the user to enter a single character, *m* or *f*, representing a gender. It stores the input in a string and copies the first character in that string to the *char* variable, *gender*.

```
/* chario.c: inputs and outputs a single character */

#include <stdio.h>

int main()
{
  char string[BUFSIZ];
  char gender;

  printf("Gender (m or f)? ");
  gets(string);
  gender = string[0];
  printf("Gender entered is %c\n", gender);
  return 0;
}
```



The first character in the *string*, at location zero, is copied into *gender*.

```
gender = string[0];
```

Its conversion specification is *%c*.

```
printf("Gender entered is %c\n", gender);
```

What happens if the user enters nothing by just pressing the enter key, or if the user enters *myob*? We shall deal with these problems in the chapter on validation.

2.6 Number Input Output

The principle behind number input is to input a string, and then convert that string into a number.

```
gets(string);
integer = atoi(string);
```

converts the string input into an integer. *atoi()*, for ASCII to integer, is defined in *stdlib.h*.

ASCII stands for American Standard Code for Information Interchange. ASCII is a table that shows the number equivalent of characters. So, for example, A is 65, Z is 90, and space is 32. It is important because only numbers can be stored in a computer's memory.

```
gets(string);
decPointNumber = atof(string);
```

converts the string input into a double precision floating point number - a number with a decimal point if you like. *atof()* is defined in *stdlib.h*.

The conversion specification for an integer is *%d*.

```
printf("Your integer is %d\n", integer);
```

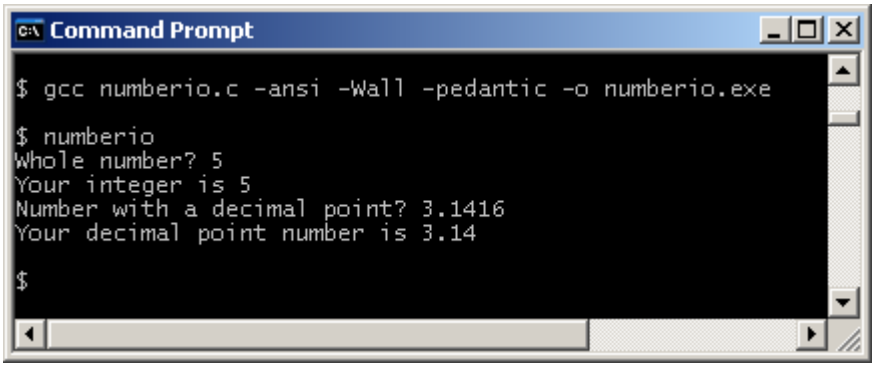
If we want to display a floating-point number with just two digits after the decimal point we use the conversion specification *%0.2f*.

```
printf("Your decimal point number is %0.2f\n", decPointNumber);
```

The *.2* in the conversion specification specifies the two digits. The *0* in *0.2f* means left justified.

Here is the complete program and its run.

```
/* numberio: inputs and outputs numbers */

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

int main()
{
  char string[BUFSIZ];
  int integer;
  double decPointNumber;

  printf("Whole number? ");
  gets(string);
  integer = atoi(string);
  printf("Your integer is %d\n", integer);

  printf("Number with a decimal point? ");
  gets(string);
  decPointNumber = atof(string);
  printf("Your decimal point number is %0.2f\n", decPointNumber);
  return 0;
}
```



There are limits to the size of numbers that can be stored in a variable. We explore these limits in the next few chapters. And, of course, the number input output program fails if the user enters non-numeric values. We address this problem in the chapter on validation.

Exercise 2.1

1. Try out the program, shown above, that inputs a name and then outputs a personalised greeting.

2. Design and write a program that prompts for, and inputs a single character code for a coat size: S for small, M for medium and L for large. Output the code that was input. Discuss what can go wrong with your program.

3. Design, write and test a program that inputs an integer representing the maximum number of students in a class, and then outputs that number. What happens if the user enters the letters IO instead of the number 10?

4. Design, write and test a program that inputs a patient's body temperature in degrees Celsius, and then outputs that temperature correct to one decimal place.

We have seen how to declare, input and print string and number variables.

Next we see how to use some string handling functions.

Bibliography

Kernighan B and Ritchie D *The C Programming Language* Prentice Hall 1988
Mark Williams Company *ANSI C A Lexical Guide* Prentice Hall 1988