

Data Types

Objectives:

Having read this section you should be able to:

1. declare (name) a local variable as being one of C's five data types
2. initialise local variables
3. perform simple arithmetic using local variables

Now we have to start looking into the details of the C language. How easy you find the rest of this section will depend on whether you have ever programmed before - no matter what the language was. There are a great many ideas common to programming in any language and C is no exception to this rule.

So if you haven't programmed before, you need to take the rest of this section slowly and keep going over it until it makes sense. If, on the other hand, you have programmed before you'll be wondering what all the fuss is about It's a lot like being able to ride a bike!

The first thing you need to know is that you can create variables to store values in. A variable is just a named area of storage that can hold a single value (numeric or character). C is very fussy about how you create variables and what you store in them. It demands that you declare the name of each variable that you are going to use and its type, or class, before you actually try to do anything with it.

In this section we are only going to be discussing local variables. These are variables that are used within the current program unit (or function) in a later section we will looking at global variables - variables that are available to all the program's functions.

There are five basic data types associated with variables:

- int - integer: a whole number.
- float - floating point value: ie a number with a fractional part.
- double - a double-precision floating point value.
- char - a single character.
- void - valueless special purpose type which we will examine closely in later sections.

One of the confusing things about the C language is that the range of values and the amount of storage that each of these types takes is not defined. This is because in each case the 'natural' choice is made for each type of machine. You can call variables what you like, although it helps if you give them sensible names that give you a hint of what they're being used for - names like sum, total, average and so on. If you are translating a formula then use variable names that reflect the elements used in the formula. For example, $2\pi r$ (that should read as "2 pi r" but that depends upon how your browser has been set-up) would give local variables names of pi and r. Remember, C programmers tend to prefer short names!

Note: all C's variables must begin with a letter or a "_" (underscore) character.

Integer Number Variables:

The first type of variable we need to know about is of class type int - short for integer. An int variable can store a value in the range -32768 to +32767. You can think of it as a largish positive or negative whole number: no fractional part is allowed. To declare an int you use the instruction:

```
int variable name;
```

For example:

```
int a;
```

declares that you want to create an int variable called a.

To assign a value to our integer variable we would use the following C statement:

```
a=10;
```

The C programming language uses the "=" character for assignment. A statement of the form a=10; should be interpreted as take

the numerical value 10 and store it in a memory location associated with the integer variable a. The "=" character should not be seen as an equality otherwise writing statements of the form:

```
a=a+10;
```

will get mathematicians blowing fuses! This statement should be interpreted as take the current value stored in a memory location associated with the integer variable a; add the numerical value 10 to it and then replace this value in the memory location associated with a.

Decimal Number Variables:

As described above, an integer variable has no fractional part. Integer variables tend to be used for counting, whereas real numbers are used in arithmetic. C uses one of two keywords to declare a variable that is to be associated with a decimal number: float and double. They each offer a different level of precision as outlined below.

float

A float, or floating point, number has about seven digits of precision and a range of about 1.E-36 to 1.E+36. A float takes four bytes to store.

double

A double, or double precision, number has about 13 digits of precision and a range of about 1.E-303 to 1.E+303. A double takes eight bytes to store.

For example:

```
float total;
```

```
double sum;
```

To assign a numerical value to our floating point and double precision variables we would use the following C statement:

```
total=0.0;
```

```
sum=12.50;
```

Character Variables:

C only has a concept of numbers and characters. It very often comes as a surprise to some programmers who learnt a beginner's language such as BASIC that C has no understanding of strings but a string is only an array of characters and C does have a concept of arrays which we shall be meeting later in this course.

To declare a variable of type character we use the keyword char. - A single character stored in one byte.

For example:

```
char c;
```

To assign, or store, a character value in a char data type is easy - a character variable is just a symbol enclosed by single quotes. For example, if c is a char variable you can store the letter A in it using the following C statement:

```
c='A'
```

Notice that you can only store a single character in a char variable. Later we will be discussing using character strings, which has a very real potential for confusion because a string constant is written between double quotes. But for the moment remember that a char variable is 'A' and not "A".

Assignment Statement:

Once you've declared a variable you can use it, but not until it has been declared - attempts to use a variable that has not been defined will cause a compiler error. Using a variable means storing something in it. You can store a value in a variable using:

```
name = value;
```

For example:

```
a=10;
```

stores the value 10 in the int variable a. What could be simpler? Not much, but it isn't actually very useful! Who wants to store a known value like 10 in a variable so you can use it later? It is 10, always was 10 and always will be 10. What makes variables useful is that you can use them to store the result of some arithmetic.

Consider four very simple mathematical operations: add, subtract, multiply and divide. Let us see how C would use these operations on two float variables a and b.

```
add
  a+b
subtract
  a-b
multiply
  a*b
divide
  a/b
```

Note that we have used the following characters from C's character set:

```
+   for add
-   for subtract
*   for multiply
/   for divide
```

BE CAREFUL WITH ARITHMETIC!!! What is the answer to this simple calculation?

```
a=10/3
```

The answer depends upon how a was declared. If it was declared as type int the answer will be 3; if a is of type float then the answer will be 3.333. It is left as an exercise to the reader to find out the answer for a of type char.

Two points to note from the above calculation:

- 1.C ignores fractions when doing integer division!
- 2.when doing float calculations integers will be converted into float. We will see later how C handles type conversions.

Arithmetic Ordering:

Whilst we are dealing with arithmetic we want to remind you about something that everyone learns at junior school but then we forget it. Consider the following calculation:

```
a=10.0 + 2.0 * 5.0 - 6.0 / 2.0
```

What is the answer? If you think its 27 go to the bottom of the class! Perhaps you got that answer by following each instruction as if it was being typed into a calculator. A computer doesn't work like that and it has its own set of rules when performing an arithmetic calculation. All mathematical operations form a hierarchy which is shown here. In the above calculation the multiplication and division parts will be evaluated first and then the addition and subtraction parts. This gives an answer of 17.

Note: To avoid confusion use brackets. The following are two different calculations:

```
a=10.0 + (2.0 * 5.0) - (6.0 / 2.0)
a=(10.0 + 2.0) * (5.0 - 6.0) / 2.0
```

You can freely mix int, float and double variables in expressions. In nearly all cases the lower precision values are converted to the highest precision values used in the expression. For example, the expression $f*i$, where f is a float and i is an int, is evaluated by converting the int to a float and then multiplying. The final result is, of course, a float but this may be assigned to another data type and the conversion will be made automatically. If you assign to a lower precision type then the value is truncated and not rounded. In other words, in nearly all cases you can ignore the problems of converting between types.

This is very reasonable but more surprising is the fact that the data type char can also be freely mixed with ints, floats and doubles. This will shock any programmer who has used another language, as it's another example of C getting us closer than is customary to the way the machine works. A character is represented as an ASCII or some other code in the range 0 to 255, and if you want you can use this integer code value in arithmetic. Another way of thinking about this is that a char variable is just a single-byte integer variable that can hold a number in the range 0 to 255, which can optionally be interpreted as a character. Notice, however, that C gives you access to memory in the smallest chunks your machine works with, i.e. one byte at a time, with no overheads.

Something To Declare:

Before you can use a variable you have to declare it. As we have seen above, to do this you state its type and then give its name. For example, `int i;` declares an integer variable. You can declare any number of variables of the same type with a single statement. For example:

```
int a, b, c;
```

declares three integers: a, b and c. You have to declare all the variables that you want to use at the start of the program. Later you will discover that exactly where you declare a variable makes a difference, but for now you should put variable declarations after the opening curly bracket of the main program.

Here is an example program that includes some of the concepts outlined above. It includes a slightly more advanced use of the `printf` function which will be covered in detail in the next part of this course:

```
/*
/*
Program#int.c

Another simple program
using int and printf
*/

#include

main()
{
    int a,b,average;
    a=10;
    b=6;
    average = ( a+b ) / 2 ;
    printf("Here ");
    printf("is ");
    printf("the ");
    printf("answer... ");
    printf("\n");
    printf("%d.",average);
}
```

More On Initialising Variables:

You can assign an initial value to a variable when you declare it. For example:

```
int i=1;
```

sets the int variable to one as soon as it's created. This is just the same as:

```
int i;
i=1;
```

but the compiler may be able to speed up the operation if you initialise the variable as part of its declaration. Don't assume that an uninitialised variable has a sensible value stored in it. Some C compilers store 0 in newly created numeric variables but nothing in the C language compels them to do so.

Summary:

Variable names:

- should be lowercase for local variables
- should be UPPERCASE for symbolic constants (to be discussed later)
- only the first 31 characters of a variables name are significant
- must begin with a letter or _ (under score) character