C Data Types

Overview

- Fundamental Data types
- Type Specifiers
- Type Qualifiers
- Constants

2

Objectives

In C we have to specify what type each value is before we use it. By specifying the type of the value the compiler can check if we use the value in the right way. This is called type checking. C is a language with strong type checking. This means that values we assign have to be of the same type or the compiler will generate a warning.

When we use these data type definitions the compiler checks for possible mistakes made in the code when assigning values of different types. Defining a variable of the type integer and assigning a float value to it will generate a compiler warning, the other way around the compiler does not complain because the type is being promoted. This section is devoted to the different types used in a normal C program.

The topics discussed in this Unit

- Fundamental Data Types
- Fundamental Data Types and Their Size
- Constants

Fundamental Data Types

- Types
  - char
  - int
  - float
  - double
- Sizes are machine dependent
- Magnitudes of types in *limits.h*

3

Fundamental Data Types

In this section we give an introduction to the data types which are present in the C language. These data types have equivalents in other high level languages (such as Pascal and Fortran). These types are the building blocks in C and it is possible to create derived versions of them.

Integral Types
The integral types can be divided into two different sub-types:

- char
  This type is used for storing a single character. A char types mostly is one byte large. Typical examples of characters are 'a', 'b'.

- int
  The int type is for storing integral types (whole numbers). The maximum size of this int is compiler and machine dependent, later more about the size.

Float types
C has two types for representing floating point numbers float and double. These two types differ in the size they occupy in memory and their precision. A double has a better precision than a float value and will take up more memory for guaranteeing this precision.

The compiler uses strong type checking in operations on the fundamental data types. It checks to see if the values in the calculations are of the same type.
If the types in the calculation are of different types and there is a possibility of data loss, the compiler issues a warning that values will be converted.
For example, assigning a type to a type with higher range or precision will result in conversion of the type without any warning. The other way around, assigning a type to a type with lower range or precision, the compiler also converts the type but issues a warning at compile time.

Type Magnitudes

- Different types have different boundaries
- Sizes are machine dependent
- Header file `<limits.h>` contains constants for maximum and minimum values of the different types

4

Type Magnitudes

The four data types mentioned in the previous section occupy different sizes of memory. The smaller types take less memory than the bigger types. Because the memory is type dependent the lowest and highest value therefor differs.

The following table shows a list of the four standard data types in order from small to big. These sizes are the smallest certified sizes that are guaranteed. Depending on your machine, they can be larger.

Type	Bits	Limits
char	8	-128 t/m 127
int	16	-32768 t/m 32767
float	32	3.4E ± 38 (7 digits)
double	64	1.7E ± 308 (15 digits)

Type Specifiers and Qualifiers

- Can change size of fundamental types (qualifiers)
  - `short`, short number of bits
  - `long`, lengthens number of bits
- Enlarging by using sign specifiers, only for integer types (specifiers)
  - `signed`, default positive and negative
  - `unsigned`, no negative values

5

Type Specifiers and Qualifiers

Type specifiers and qualifiers give us some control on the values we use in applications.

Qualifiers

The memory sizes of for example the int depends on what kind of machine you are using. It can be a 16 bit or a 32 bit value. In some applications you have to be sure of the size a certain value occupies in memory. The type qualifiers let us explicitly state the size of a certain value. These qualifiers however cannot be used on all types. They can only be used for the int. Except for the long that one can be used for double values as well.

The short qualifier makes sure that the value takes up the least certified memory size. Specifying a value to be short int will make sure that it takes up at most 16 bits and not 32 bits. The long specifier is the opposite.

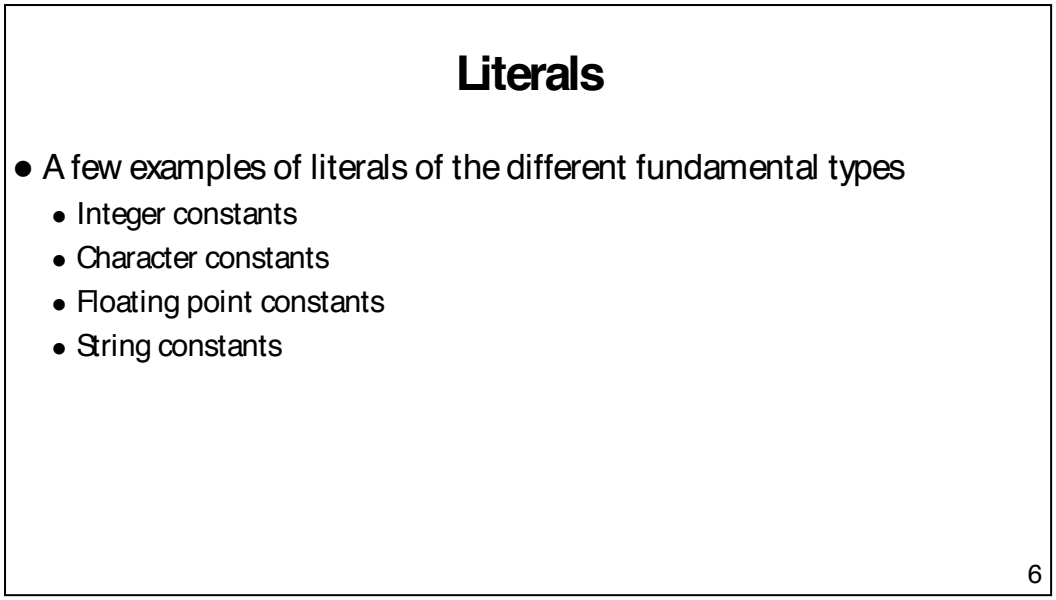
Specifiers

All values used of any type are by default signed. This means that one bit is used to specify if the value is negative or positive. It is possible to specify that certain integer types are unsigned and so increasing the magnitudes by a factor of two.

The next page shows a table with all C data types.

The C data types:

Type	Other name	Bits	Limits
char		8	-128 to 127
unsigned char		8	0 to 255
short int	short	16	-32768 to 32767
unsigned short int	unsigned short	16	0 to 65535
int		16 or 32	-32768 to 32767 or -2147483648 to 2147483647
unsigned int		16 or 32	0 to 65535 or 0 to 4294967295
long int	long	32	-2147483648 to 2147483647
unsigned long int	unsigned long	32	0 to 4294967295
float		32	10E-38 to 10E38 (7 digits)
double		64	10E-308 to 10E308 (15 digits)
long double		64	10E-308 to 10E308 (15 digits)



Literals

When using literal numbers without suffixes the following rules apply:

- Integer numbers are by default int
- Floating point numbers are by default double

If you want to use literals that are other than double or int you have to use suffixes to specify what kind of literal it is. The *l* or *L* for example specifies that it is a long value. The *f* or *F* suffix specifies that it is a float value.

Integer literals

The most used basic literal is the integer literal, mostly used when initialising variables. Integer literals can be divided into different mathematical systems. The integer numbers can be presented in the decimal, octal or hexadecimal system. Following are a few examples of the different types of values.

Integer literals:

1234	Decimal
0232	Octal
0x402	Hexadecimal

Just as with using the fundamental data types the integral literals have also a signed value or unsigned value. By default the literals are signed. To specify if a literal is an unsigned value or a long value, suffixes are used. For an unsigned value the suffix *u* and for a long value the suffix *l* is added to the end of the literal. These suffixes can be upper or lower case (thus they are not case sensitive).

123u	unsigned integer constant
345671	signed long constant

Character literals

Character literals are values corresponding to elements of the ASCII character set. A character literal is always a single character between single quotation marks. For example the character a is as a constant ‘a’. It is not allowed to put multiple characters between the single quotation marks. For example ‘aa’ is not allowed.

Escape characters
Except for the printable characters the ASCII table has non printable characters as well. These characters include the new line, spaces, etc. To use these characters in a C program we have to use the escape characters as already mentioned in module 1. An escape character is placed between single quotation marks as normal characters. These special marks are the only exception to the rule that only one character is allowed between the quotation marks. (In fact, escape character evaluate to one character.)

Some examples:
newline '\n'
tab '\t'
return '\r'

```
#include <stdio.h>

void main()
{
       printf("First line\nSecond line\nA backslash \\");
}
```

Output:
```
First line
Second line
A backslash \
```

For the complete table of these marks we refer to the last page of module 1.

Floating point literals
These are the literals used for float and double values. When using floating point literals in a program, the compiler always assumes that these are double values. So the value 123.567 is assumed to be a double by the compiler.

When you want to specify that a floating point literal is to be used as a float value instead of a double value, the suffix *f* is added at the end of the value. You can use both the lower and upper case *f*.

234.08	Double literal
2347.889f	Floating point literal

String literals
The C language has no type for strings. Later in the book we shall see that the way for representing strings is an array of characters. For now we will assume it is not possible to use strings in a C program. It is however possible to use string literals for example in strings used when displaying messages. The strings are entered between double quotation marks, for example: "This is a string". In these string literals, which are really a group of characters connected to each other, it is possible to use escape characters. If the string "First line\nSecond line" is printed, the '\n' creates a newline character, so the single string is displayed as two lines on the screen. String variables are explained in a later chapter.