

Lecture 2.1

Topics

1. Data Types – Continued
2. Sizes of Data Types
3. Data Output with `printf()` and `sizeof()` operator – Continued

1. Data Types in C – Continued

In general, C has many data types that can be grouped in three categories as follows,

- (i) Simple Data Type
- (ii) Structured Data Type
- (iii) Pointers

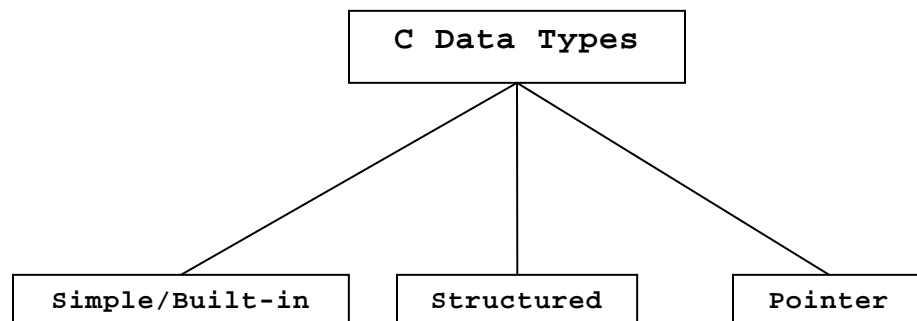


Figure 1 C's data types

Simple or built-in data types will be considered in the next several lectures. Structured and pointer data types will be considered in later discussions.

Built-in types are depicted in the following figures.

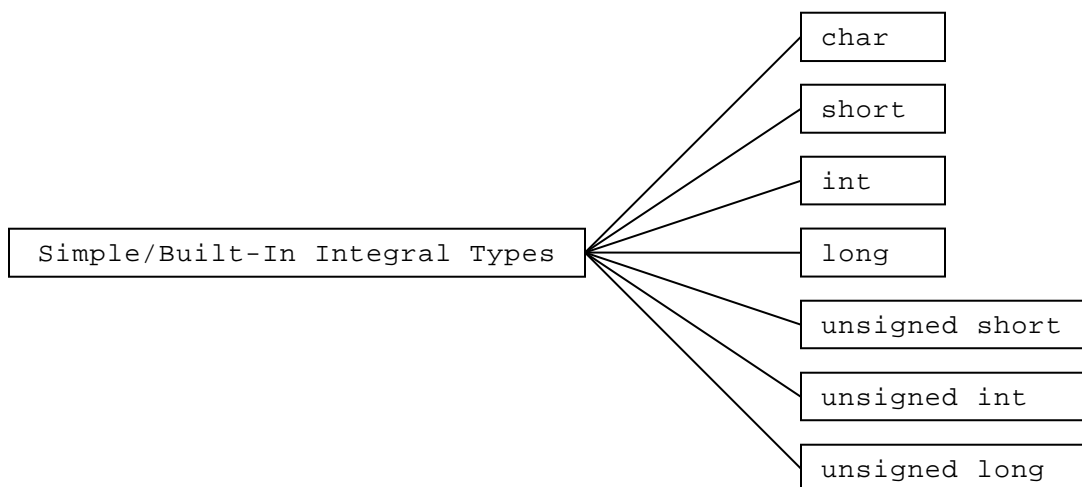
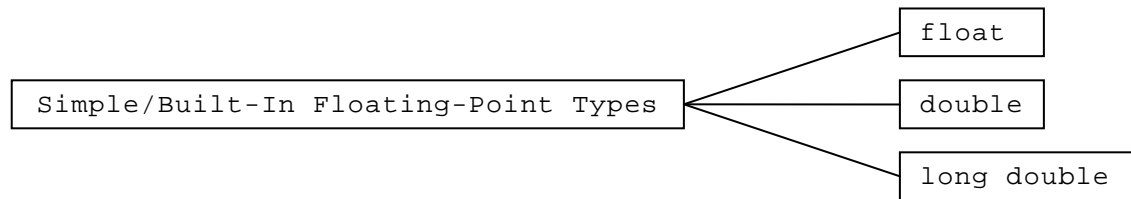


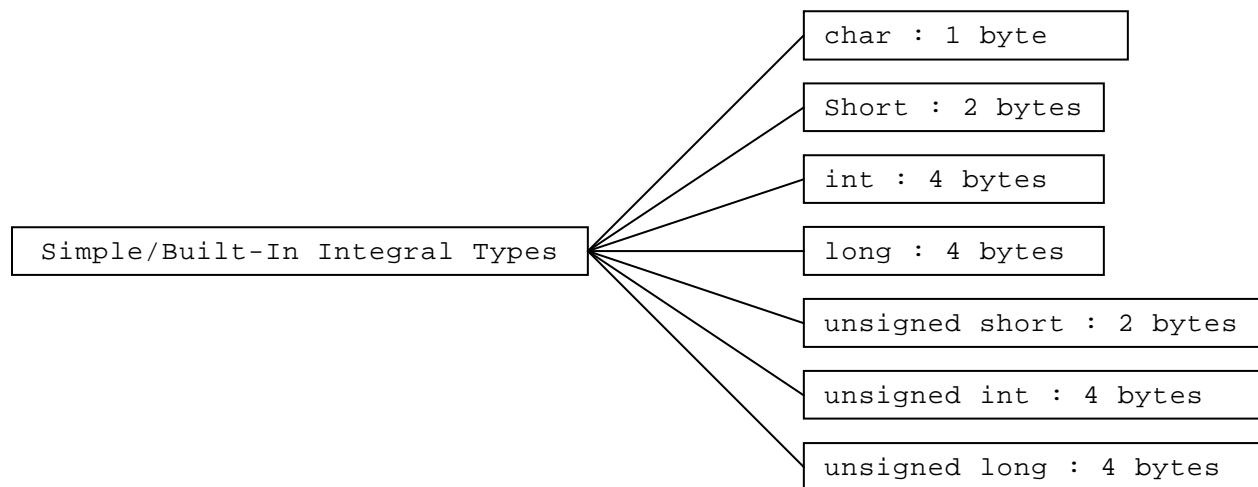
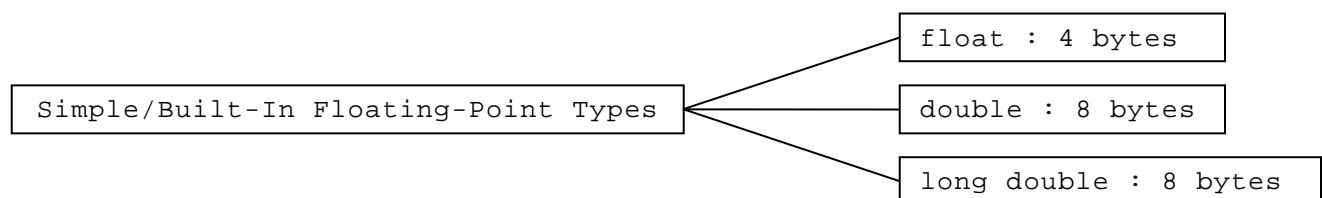
Figure 2 Built-in integral data types

**Figure 3** Built-in floating-point data types

2. Size of Data Types

C types are used to indicate how a computer must take on a value, translate, and store in memory. The simplest interpretation of the storage is just to refer to the size of each data type (or the size of the memory block reserved for a value of a given type).

The sizes are given as follows,

**Figure 4** Data sizes for integral types**Figure 5** Data sizes for floating-point types

3. Data Output with `printf()` – Continued

To output data on screen, one can use `printf()` function. There are many different ways for data (also called arguments/parameters) being passed to this function.

Let's look at some of the calls to `printf()`.

Example 1

```
/**
 *Program Name:   cis26L0211.c
 *Written By:     T. T. Nguyen
 *Discussion:     Sizes of Built-in Types
 */

#include <stdio.h>

int main( void ) {
    printf( "Size of a char %d\n", sizeof( char ) );
    printf( "Size of a short %d\n", sizeof( short ) );
    printf( "Size of an int %d\n", sizeof( int ) );
    printf( "Size of a long %d\n", sizeof( long ) );
    printf( "Size of an unsigned short %d\n",
            sizeof( unsigned short ) );
    printf( "Size of a unsigned int %d\n",
            sizeof( unsigned int ) );
    printf( "Size of a unsigned long %d\n",
            sizeof( unsigned long ) );

    printf( "\nSize of a float %d\n", sizeof( float ) );
    printf( "Size of a double %d\n", sizeof( double ) );
    printf( "Size of a long double %d\n",
            sizeof( long double ) );

    return 0;
}
```

OUTPUT

```
Size of a char 1
Size of a short 2
Size of an int 4
Size of a long 4
Size of a unsigned short 2
Size of a unsigned int 4
Size of a unsigned long 4

Size of a float 4
Size of a double 8
Size of a long double 8
```

In general, one can use `printf()` function to send data to screen. This `printf()` function requires two things:

- (1) The format instruction, and
- (2) The data to be printed.

Putting things together, the syntax of `printf()` is as follows,

```
printf( "format string/instruction", data1, data2, ... );
```

The first argument "**format string**" contains the constant text string to be sent to screen and zero or more **field specifications**.

Field Specification

The **field specification** describes how the data should be printed. Each field specification begins with a percent sign (%) for each data point; such as `data1`, `data2`, etc.

For `printf()`, the syntax of a field specification is given as follows,

`%<flag><minimum_width><precision><size>conversion_code`

We will elaborate on the above specification in the next lectures.