

Programming in C

© 2001, 2005 Rex Jaeschke. All rights reserved.

1. The Basics

In this chapter, we will learn about a number of fundamental constructs and language elements.

Goals of this Chapter

Once the reader has read the information provided in this chapter, and solved the related programming exercises, they should be able to do the following:

- Understand the use of white space, comments, and blocks, and to identify individual source tokens.
- For user-defined names, know the rules regarding their spelling, case sensitivity, and length of significance.
- Know how to use the basic I/O facilities to write to the screen and read from the keyboard, using simple formatting via manipulators.
- Understand the size and value range of the arithmetic data types, and how their representation can change from one compiler to another.
- Use enumerated data types.
- Use the type qualifier `const` to provide read-only access to variables.
- Know how to write simple expressions and statements.
- Know how to write constants and be able to tell their type.
- Know how to define and initialize local variables.
- Understand and apply operator precedence and associativity.
- Know when and how to use implicit and explicit type conversion.

2. Looping, Testing, and Branching

C provides several different kinds of looping constructs and a number of ways of implementing conditional and unconditional branching. In this chapter, we will look at these constructs as well as a number of arithmetic, relational, equality, and logical operators.

Goals of this Chapter

Once the reader has read the information provided in this chapter, and solved the related programming exercises, they should be able to do the following:

- Use the while, for, and do-while looping constructs.
- Make decisions using if-else and switch.
- Understand when and how to use the branching statements break, continue, and goto.
- Understand the purpose of, and problems associated with, the null statement.
- Start to appreciate certain aspects of programming style with respect to white space usage and indenting.
- Understand the prefix and postfix increment and decrement operators.
- Know the family of relational and equality operators.
- Understand that assignment is an operator, and the implications thereof, especially with regard to embedded assignment and the potential for confusion with equality.
- Know about the basic compound-assignment operators.

3. Arrays

In this chapter, we will learn how to define and use arrays. While only a few types are used in the examples, the principles can be applied to arrays of objects of any data type.

Goals of this Chapter

Once the reader has read the information provided in this chapter, and solved the related programming exercises, they should be able to do the following:

- Define, initialize, and use single and multi-dimensional arrays.
- Understand that each dimension's elements start at index 0.
- Know how arrays are stored in memory.
- Know how to store and manipulate strings, and understand the special role of the null character.
- Understand the purpose of the character testing and conversion library functions, declared in the header ctype.h.
- Understand the purpose of the basic string manipulation library functions, declared in the header string.h.
- Know how to use the library functions, declared in the header stdlib.h, to convert strings containing numbers to actual numbers.
- Know when and how to use the sizeof operator.

4. Functions

Functions are the backbone of C programs. A typical program is made up of one or more user-written functions and calls to functions provided in the standard and other libraries.

Goals of this Chapter

Once the reader has read the information provided in this chapter, and solved the related programming exercises, they should be able to do the following:

- Know how to pass arguments to functions, and the difference between passing by value and by address.
- Know how to return a value from a function, and the difference between returning by value and by address.
- Understand function prototypes and their role in allowing the compiler to check function calls for correctness.
- Have a basic understanding of recursion.
- Have a basic understanding of the logical and bit-manipulation operators.

5. Storage Classes

In this chapter, we will learn how to declare local and global functions and variables. We achieve these things by using one of a number of storage class keywords and/or by placing declarations in one of several places.

Goals of this Chapter

Once the reader has read the information provided in this chapter, and solved the related programming exercises, they should be able to do the following:

- Understand and use static and automatic storage duration to control the lifetime of identifiers.
- Understand and use block and file scope to control the visibility of identifiers.
- Understand and use external, internal, and no linkage to control the sharing of resources.
- Have a feel for how one might package libraries of functions.

6. The Preprocessor

When we compile a source file, the first phase of compilation involves the *preprocessor*, a program that transforms our source file into another source file, called the *intermediate file*, on the basis of the *preprocessing directives* found in the original source. In this chapter, we will learn about these directives. Using them, we can define symbolic constants, share common declarations among separate source files, and conditionally compile source lines on the basis of various criteria.

Goals of this Chapter

Once the reader has read the information provided in this chapter, and solved the related programming exercises, they should be able to do the following:

- Understand the purpose of the preprocessor.
- Define object- and function-like macros.
- Effectively use conditional compilation.
- Know how to use some of the basic predefined macros.

7. Pointers and Addresses

So far, we have learned how to do things in C that we already know how to do in some other language. In this chapter, we will be exposed to those capabilities of C that make it different from higher-level languages.

Goals of this Chapter

Once the reader has read the information provided in this chapter, and solved the related programming exercises, they should be able to do the following:

- Have a clear understanding of how memory is conceptually laid out when a program executes.
- Understand how variables, called pointers, can contain other variables' and functions' addresses, and how multiple pointers can point to the same thing.
- Have a good idea of how pointers can be used to solve interesting problems.
- Know how to define and initialize pointer variables, and how to dereference pointer expressions.
- Know the rules for reading and writing pointer expressions.
- Understand the ways in which the const qualifier can be used with pointers.
- Understand address arithmetic, and the relationship between pointer operations and array subscripting.
- Know how to access arguments specified on the command line when a program is executed.
- Understand the basics of pointers to functions, and calling functions other than by name.
- Know how to allocate and free memory at runtime, under the programmer's control.

8. Input and Output

We have learned how to perform basic I/O via printf and scanf. In this chapter, we will get a more formal introduction to these (and other) facilities.

Goals of this Chapter

Once the reader has read the information provided in this chapter, and solved the related programming exercises, they should be able to do the following:

- Have a basic understanding of console stream.
- Know how to perform basic I/O using files.
- Know how to encode values into strings and how to decode them from such strings.

9. Structures, Bit-Fields, and Unions

In this chapter, we will learn how to deal with a set of objects as a group. We will also see how to redefine an area of memory so that it can be used for multiple purposes, one at a time. An introductory discussion of linked lists is also provided.

Goals of this Chapter

Once the reader has read the information provided in this chapter, and solved the related programming exercises, they should be able to do the following:

- Understand, define, and use group items called structures, and the members they contain.
- Know the difference between the two member selection operators, dot (.) and arrow (->).
- Understand the basics of intermediate and advanced data structures, and how they can be implemented using combinations of arrays, pointers, and structures.
- Know how a given structure's physical layout can vary from one system to the next.
- Be able to define small integer variables, called bit fields, to maximize storage.
- Know how to use a union to redefine an area of memory for multiple purposes.

10. Sundry Issues

In this chapter, we look at a number of specialized and/or esoteric topics that don't fit into any of the other chapters.

Goals of this Chapter

Once the reader has read the information provided in this chapter, and solved the related programming exercises, they should be able to do the following:

- Know when and how to use the comma operator.
- Understand that a pointer to an array is quite different to a pointer to the first element of that array.
- Read and write arbitrarily complex declarations.
- Understand sequence points and identify them in source code.
- Know the difference between modifiable and non-modifiable lvalues.
- Define a function taking a variable number of arguments.