

**Chapter
1**

**INTRODUCTION
AND REVIEW**

*Get on the
Fast Track!*

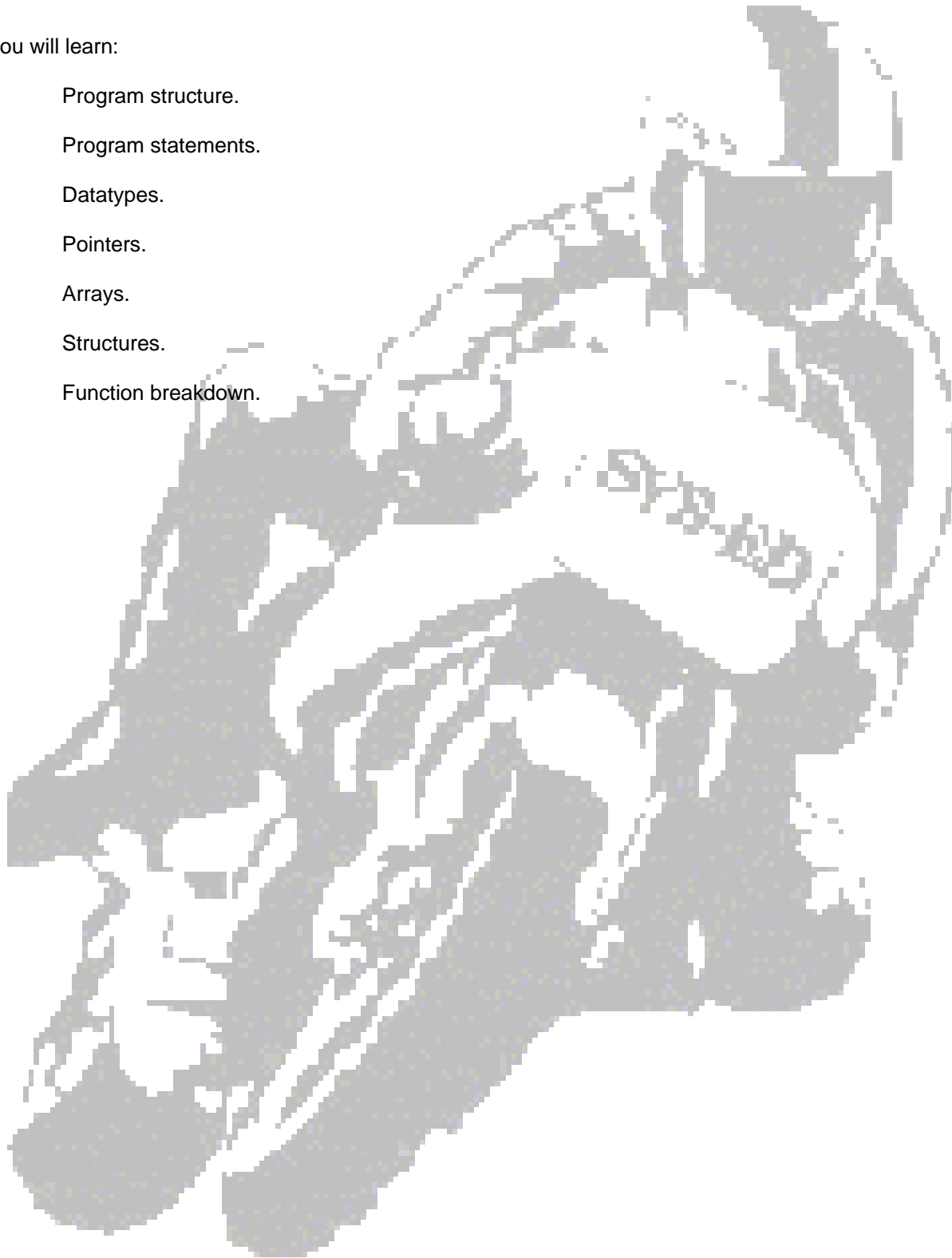


**SYS-ED/
COMPUTER
EDUCATION
TECHNIQUES, INC.**

Objectives

You will learn:

- C Program structure.
- C Program statements.
- C Datatypes.
- C Pointers.
- C Arrays.
- C Structures.
- C Function breakdown.



1 Features of the C Programming Language

Features of the C Programming language include:

- C A full set of loop, conditional and transfer statements for controlling program flow logically and efficiently.

This serves to encourage structured programming.
- C A large set of operators.

Many of these operators correspond to common machine instructions; thereby allowing a direct translation into machine code. These operators allow the concise specification of different kinds of operations clearly and with a minimum of code.
- C Several integer sizes; this is in addition to single-precision, double-precision and floating-point types.

Complex data types such as arrays and data structures can also be constructed.
- C Declaration of pointers to variables and functions.

Pointers serve to make programs more efficient by allowing reference to be made to items by the machine address. C also supports pointer arithmetic; which allows memory addresses to be accessed and manipulated directly.
- C A preprocessor which acts on the text of files before they are compiled.

The C preprocessor can be used to define program constants, substitute fast macro definitions for function calls, and compile parts of programs based on specified conditions.

2 Program Structure

Functions

All C programs are divided into units called functions. A function in C is similar to a subroutine in BASIC or a function in Pascal.

Every C program consists of at least one function `main()`. The operating system passes control to the `main()`. The `main()` function then becomes the first function to be executed.

Parentheses

All C functions are followed by a set of parentheses. The parentheses inform the compiler that this is to be a function.

Following the function definition are braces that signal the beginning and the ending of the body of the function.

C The opening brace indicates that a block of code (ie. a distinct unit) is about to begin.

C The closing brace on the next line serves to terminate the block of code (ie. delimit).

In addition to their use in functions, the braces serve to delimit blocks of code used in loops and decision making statements.

Within the C programming language these braces have a purpose similar to the `BEGIN` and `END` statements in Pascal; they serve to delimit a section of code.

3 Program Stages

Editor

- C Input from terminal.
- C Output is a source code file containing C source code and possibly preprocessor commands.

Preprocessor

- C Input is source code file.
- C Output is source code file with expanded macros and the additional files generated by preprocessor commands.

Compiler

- C Input is source code file with expanded macros and the additional files generated by preprocessor commands.
- C Output is assembly language source code.

Assembler

- C Input is a assembly language source code file.
- C Output is relocateable object code.

Linker

- C Input is relocateable object code modules from programs and the C library.
- C Output is executable code (e.g. a.out).

4 Syntax

Semi-colon

Every statement in C is followed by a semi-colon.

Whitespace Characters

C ignores any whitespace characters such as the carriage return, space, or tab. There is no restriction on the use of whitespace characters; use as many as are required to make the program readable. The whitespace characters are invisible to the compiler.

Comments

Comments can be written into the C program and are useful for self documentation and improving the readability of the program listing. C ignores all entries made between a slash and an asterisk and the accompanying closing asterisk and slash.

```
/* The following code will be ignored by the C compiler. */
```

Indentation and Readability

Indenting blocks of code enclosed in braces {} is an important aspect of making your C program readable.

Stretching the code out vertically makes the results easier to comprehend. It is important that matching braces be aligned in order to ensure that all opening braces be followed by closing braces.

Case Sensitivity

C is case sensitive. All reserved words in C must be used in the lowercase. Any variation in a word between uppercase and lowercase, will be interpreted by C as two distinct words. For example, Variable, VARIABLE and variable are all different words to C.

Since C is case sensitive, it is important to write all function statements in lowercase letters. PRINTF(), and Printf(), are not the same as printf(). For ease of typing it is common programming practice for all characters to be lowercase letters.

5 Identifiers

Within a C program the names given to constants, data types, variables and functions are known as identifiers.

The identifier is used for constants, variables or function names. All identifiers follow the same rules, the identifiers must be made up of letters and digits only, and the first character must be a letter.

- C Blanks inside identifiers are not permitted.
- C The underscore `_` can serve as a letter and is frequently used for separating the parts of long descriptive identifiers.

Valid Characters for C

- C Uppercase and lowercase letters ('A'-'Z', 'a'-'z').
- C The underscore character ('_').
- C Digits ('0'-'9').
- C Identifiers are case sensitive.
- C The first 31 characters of an identifier are significant.

Valid Names or Identifiers

- C An identifier must begin with a letter.
- C Do not use an underscore as a first letter of an identifier.
Identifiers beginning with an underscore can cause conflicts with the names of system routines or variables and produce errors. Programs containing names beginning with leading underscores are not guaranteed to be portable.
- C Digits are not allowed as the first character of an identifier.

Many C programmers use only lowercase letters and underscores for their identifiers. There is one major exception to this practice. By convention, identifiers containing only uppercase characters indicate that the identifier is associated with a constant value.

An identifier may have any number of characters, but only the first eight are retained by many of the older C compiler.

- C ANSI standard for C Language requires compilers to accept 31 characters for a identifier.
- C Microsoft C complies with the ANSI standard and will read the entire identifier.

Reserved Words

Reserved words are the words that have a special meaning to the compiler.

C is strict with respect to these reserved words and will not allow their use as identifiers.

When naming the program, most compilers need all C source code files to end with the suffix .c(.

6 Program Statements

A program statement is an instruction for the compiler to create machine-language code in order to perform a certain action.

C contains a variety of statements that regulate the flow of a program. Every complete statement in C must be terminated with a semicolon.

Code Block

A series of statements, enclosed within braces, or {}, that execute together as a unit.

7 Datatypes

7.1. Character (char)

Character (char)

Characters occupy one byte and have a range from -128 to 127 (-80 to 7F hex). Unsigned characters have a range from 0 to 255 (0 to FF hex).

```
char ch;          /* declare a character variable      */
unsigned char k; /* declare unsigned character variable */
char c1 = 'a';   /* initialize character variable      */
char c2 = 97;    /* initialize using decimal value     */
char c3 = 0x61;  /* initialize using hexadecimal value */
```

```
/* special character constants */
'\n' /* newline (linefeed); 0x0A */
'\b' /* backspace; 0x08 */
'\r' /* carriage return; 0x0D */
'\f' /* formfeed; (0x0C) */
'\t' /* tab; 0x09 */
'\v' /* vertical tab; 0x0B */
'\' ' /* backslash; 0x5C */
'\'' /* single quote; 0x27 */
'\'' /* double quote; 0x22 */
'\0' /* null; 0x00 */
```

7.2. Short Integer (short)

Short integers occupy two bytes and have a range from -32768 to 32767 (-8000 to 7FFF hex).

Unsigned short integers have a range from 0 to 65535 (0 to FFFF hex).

```
short x;          /* declare short integer          */
short y =12;     /* initialize short integer          */
unsigned short c; /* declare unsigned short integer   */
12              /* decimal const (no initial 0)    */
0x0c           /* hex constant (initial 0x)       */
014            /* octal constant (initial 0)      */
```

7.3. Integer (int)

Integers are the same as type short or type long depending on the machine used by the compiler.

- C For 8088, 8086, and 8026 processors, the Microsoft C compiler treats integers as two bytes (ie. similar to type short).
- C For 80386 processors and higher, the Microsoft C compiler treats integers as four bytes (ie. similar to long).

```
int x;           /* declare an integer              */
int y = 12;      /* initialize an integer           */
unsigned int c;  /* declare unsigned integer        */
               /* constants same as for short int */
```

7.4. Long Integer

Long integers occupy four bytes in memory and have a range from -2,147,483,648 to 2,147,483,647 (-80000000 to 7FFFFFFF hex).

Unsigned long integers have a range from 0 to 4,294,967,295 (0 to FFFFFFFF hex).

```
long int bignum;      /* declare long integer      */
long bignum;         /* alternative form          */
long bn1 = 12L;      /* initialize long integer   */
10L                  /* decimal constant         */
0x0AL                /* hexadecimal constant     */
012L                 /* octal constant           */
```

7.5. Floating Point (float)

Floating point numbers occupy four bytes of memory. The exponent has a range of 10^{-39} to 10^{38} . The mantissa has up to six digits of precision. The keyword to declare a floating-point variable is `float`.

These numbers are referred to as floating point numbers because the value is represented in three parts.

- C A sign.
- C Numerical value (the fractional part of which is referred to as the "mantissa").
- C Exponent to indicate where the decimal point is placed.

The number 250.450 can be represented in the following ways:

250.450e0	25.0450e1
2.50450e2	2505.50e-1
25045.0e-2	

e0	moves decimal zero places.
e1	moves decimal one place to the right.
e2	moves decimal two places to the right.
e-1	moves decimal one place to the left.
e-2	moves decimal two places to the left.

When representing this number the exponent makes the decimal point float.

C allocates 32 bits to variables of type float. Because numerical values are represented in binary format, rounding errors can occur in numerical calculation. These errors can become significant; particularly if a high volume of calculations are being performed.

```
float flnumb;          /* declaring floating point number    */
float fnb = 37.42     /* initializing floating point number  */
99.99                 /* constant in decimal notation       */
999E-2                /* constant in exponential notation    */
                      /* floating point constants always    */
                      have type double */
```

7.6. Double-Precision Floating Point (double)

Double-precision floating point numbers occupy eight bytes of memory.

The exponent has a range of 10^{-308} to 10^{309} . The mantissa has up to 15 digits of precision.

```
double verybig;      /* declaring double precision
                    floating point number          */
long float verybig   /* alternative form for declaration          */
double vb = 7.1416   /* initializing double precision number      */
320000              /* constant in decimal notation             */
3.2E5               /* constant in exponential notation         */
```

8 C Reserved Words

Keywords in C

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

In addition to the above reserved words, a specific compiler may provide some additional words.

9 Pointers

A pointer to an item corresponds to the item's machine address.

The address is called a pointer to the variable.

Pointers are a fundamental part of the C programming language. Along with data types that mark a variable as a character, an integer, or a floating point number, C also supports a pointer data type. This means that there can be a variable that is a pointer to a variable of a particular data type. Arithmetic can be performed on these pointers which takes into account the type of variable to which the pointer is pointing.

C has two basic operators for working with pointers:

&	The ampersand (&) obtains the address of a variable. <code>Ptr = &x;</code>
*	The asterisk (*) is known as the indirection operator and is used to access the contents of the memory location to which the pointer is pointing. <code>*Ptr = 2;</code>

These two statements taken together are equivalent to:

```
x = 2;
```

In either case the value of 2 is stored in a memory location pointed to by the address of x.

The C language provides that certain collections of data must be organized in a predictable manner.

The elements of an array are stored sequentially in memory, thereby allowing a program to access these elements using a pointer.

10 Arrays

An array is a collection of data items of the same type, whose elements form an ordered sequence and which are placed contiguously in memory. The elements of an array can be of any data type.

- C An array may be accessed by an index.
- C Arrays may have one or more dimensions, and multidimensional arrays typically represent tables.
- C A linear array is a single row of values.

A two-dimensional array or matrix is an array of arrays. There are two elements or rows in a two-dimensional array.

Declaring Arrays

An array is declared like any other data object with one exception. Square brackets [] are added after the name, usually with a number inside which indicates the number of elements in the array.

An array of 5 integers named 'name' would be declared:

```
int name[5]
```

In this example, int specifies the type of variable. The type of variable is followed by the name of the variable. The brackets inform the compiler that an array is being dealt with. The number within the brackets tells the compiler how many variables of type int will be contained within the array.

11 Structures

A structure consists of a number of data items, which need not be of the same type, grouped together.

A structure differs from an array in that it contains information of different data types. Structures are useful, not only because they can hold different types of variables, but also because they can form the basis for more complex data constructions, such as linked lists.

Individual items or components of the structure are called members.

The keyword `struct` is used to declare the structure.

A structure is a data type whose format must be defined by the programmer.

Defining a Structure

```
struct name (known as tag name)
{
int num          /* elements of members of the structure */
char ch          /* are surrounded by braces */
};               /* semi-colon terminates whole statement */
```

11.1. Declaring a Structure Type

Since a structure may contain any number of elements of different types, the programmer must define to the compiler what a particular structure is going to look like. This needs to be done before using variables of that type. This is referred to as declaring a structure type.

This differs from how other fundamental data types are used in C. For example, the int and float data types are predefined by the compiler.

The following statement declares the structure type:

```
struct easy
{
  int num;
  char ch;
};
```

- C The keyword struct introduces the statement.
- C The name easy is referred to as the "tag". It designates the kind of structure being defined. The tag is not a variable name, since a variable is not being declared; it is a type name.
- C The elements of the structure are surrounded by braces, and the entire statement is terminated by a semi-colon.

11.2. Defining Structures

After declaring a structure, storage is allocated for the structure. This can be done at the time of the declaration.

To define variables during declaration, list the identifiers for the structures you want to define. This list should come between the right brace that ends the structure template and before the semi-colon that ends the statement.

```
struct tag_name
{
    int member1;
    int member2;
}var1, var2;
```

Structure variables can be defined separately from the declaration; this assumes that the declaration has already been made.

```
struct tag_name
{
    int member1;
    int member2;
};

struct tag_name var1, var2;
```

To define the variables separately, it first will be necessary to specify the variable's tag, which in this example is the tag_name. The variable names are then specified, as when defining other types of variables.

11.3. Initializing Structures

Similar to simple variables and arrays, structure variables can be initialized and given specific values at the beginning of a program. The format used is comparable to that used for initializing arrays.

```
/* initstruct.c */
/* demonstrates initialization of structures */

struct personnel          /* defines data structure */
{
char name[30];           /* name */
int empnum;              /* code number */
};

struct personnel empnum1 = /* initializes struct variable */
{ "Harrison Ford", 012 };

struct personnel empnum2 = /* initializes struct variable */
{ "Sean Connery", 007 };

main()
.
.
.
```

After the structure type is declared, the structure variables are defined and initialized at the same time. As with array initialization, the equal sign is used, followed by braces enclosing a list of values, with the values separated by commas.

12 Functions

C programs consist of functions.

These functions are similar to subroutines in other programming languages. Functions consist of statements. Statements serve to define or express an action to be executed.

As with independent subroutines, functions promote modular program creation.

The main program will contain calls to functions, which then perform operations, return values, call other functions and generally act as a simple stand-alone programs.

Extensive use of functions increases portability.

A 'C' function is a statement written to perform a specific action that can be compiled by the 'C' compiler.

Format:

```
function-name (argument list, if any)
argument declarations
{
    statements comprising
body of function
}
```

12.1. Function Breakdown

Command	Explanation
function-name	<p>C Indicates the beginning of function.</p> <p>C Must be coded.</p> <p>C First 6 characters must be unique.</p>
()	<p>C Includes the argument list passed to function, if any.</p> <p>C Must be coded after function name.</p>
argument	<p>C Declares variables being passed to declarations function.</p> <p>C Must be coded, if there are arguments passed.</p>
{	<p>C Indicates beginning of function statement.</p> <p>C Must be coded after function argument list.</p>
body of	<p>C Statements which describe action to be performed by the function.</p> <p>C Two types of statements: declarations - defining variables to be used within the function. test - commands to execute the action needed.</p>
}	<p>C Indicates end of function statements.</p> <p>C Must be coded.</p>

12.2. main() Function

C programs consist of functions. Functions are the basic operational entities of any C program.

The function called main() is the one to which control is passed when the program is executed.

Regardless as to how many functions there are in a C program, the main() function is the one to which control is passed from the operating system when the program is run.

Main() function is always performed first. It governs the program by giving the actual order in which the processes occur.