Chapter 1 Summary

1. The first attempt at creating a self-operating computational machine was attempted by Charles Babbage in 1822. The concept became a reality with the Atanasoff-Berry Computer built in 1937 at Iowa State University, which was the first computer to use a binary numbering scheme to store and manipulate data. The earliest large-scale digital computer was the ENIAC, built in 1946 at the Moore School of Engineering at the University of Pennsylvania. This machine, however, required external wiring to direct its operation. The first computer to employ the concept of a stored program was the EDSAC, built at Cambridge University in England. The operating principles used in the design of this machine, developed by the mathematician John Von Neumann, are still used by the majority of computers manufactured today.

2. The physical components used in constructing a computer are called hardware.

3. The programs used to operate a computer are referred to as software.

4. Programming languages come in a variety of forms and types. Machine language programs, also known as executable programs, contain the binary codes that can be executed by a computer. Assembly languages permit the use of symbolic names for mathematical operations and memory addresses. Programs written in assembly languages must be converted to machine language, using translator programs called assemblers, before the programs can be executed. Assembly and machine languages are referred to as low-level languages.

5. Compiler and interpreter languages are referred to as high-level languages. This means that they are written using instructions that resemble a written language, such as English, and can be run on a variety of computer types. Compiler languages require a compiler to translate the program into a machine language form, while interpreter languages require an interpreter to do the translation.

6. An algorithm is a step-by-step sequence of instructions that must terminate and describes how to perform an operation to produce the desired output.

7. The software development procedure consists of the following four phases:
- Specification of the program's requirements
- Design and development
- Documentation
- Maintenance

8. The design and development phase consists of four well-defined steps:
- Analyze the problem
- Select an overall solution algorithm

- Write the program
- Test and correct the program

9. Writing (or coding) a program consists of translating the solution algorithm into a computer language such as C.

10. Four fundamental control structures used in writing a program are:
- Sequence
- Selection
- Iteration
- Invocation

11. Although making copies of a program is not formally a part of the software development process, it is essential that you always keep at least one copy of a program. This copy is referred to as a backup copy, or backup, for short.

Chapter 2 Summary

1. A C program consists of one or more modules called functions. One of these functions must be called main() . The main() function identifies the starting point of a C program.

2. A function is a C language description of an algorithm.

3. Many functions, like printf(), are supplied in a standard library of functions provided with each C compiler.

4. Simple C programs consist of the single function named main().

5. Following the function name, the body of a function has the general form:

```
{
All program statements in here;
}
```

6. An executable statement causes some specific action to be performed by the computer when the program is executed.

7. All executable C statements must be terminated by a semicolon.

8. The printf() function is used to display text or numerical results. The first argument to printf() can be text, which is enclosed in double quotes. The text is displayed directly on the screen and may include newline escape sequences for format control.

9. The two basic numerical data types used almost exclusively in current C programs are

integers and double-precision numbers. (Older C programs used single-precision numbers rather than double-precision.) Compilers typically assign different amounts of memory to store each of these types of data.

10. An expression is a sequence of one or more operands separated by operators. An operand is a constant, a variable, or another expression. A value is associated with an expression.

11. Expressions are evaluated according to the precedence and associativity of the operators used in the expression.

12. The printf() function can be used to display all of C's data types. The conversion control sequence for displaying integer, floating-point (both single-precision and double-precision), and character values are %d, %f, and %c, respectively.

13. Every variable in a C program must be declared as to the type of value it can store, and a variable can only be used after it is declared. Additionally, variables of the same type may be declared using a single declaration statement. Variable declaration statements have the general syntax: dataType variableName(s); Additionally, variables may be initialized when they are declared.

14. A simple C program containing declaration statements has the form:

```
#include <stdio.h>
int main()
{
    declaration statements;
    other statements;
    return 0;
}
```

15. Declaration statements always play the software role of informing the compiler of a function's valid variable names. When a variable declaration causes the computer to set aside memory locations for the variable, the declaration statement is also a definition statement. (All the declarations we have encountered so far have also been definition statements.)

Chapter 3 Summary

1. Arithmetic calculations can be performed using either assignment statements or mathematical functions. They can also be performed using expressions in calculating an argument's value supplied to a function.

2. The assignment symbol, =, is an operator. It has a lower precedence than all

mathematical operators (+, -, *, /,%). Because assignment is an operation in C, multiple uses of the assignment operator are possible in the same expression.

3. In addition to the assignment operator, =, C provides the +=, -=, * = and /= assignment operators.

4. The increment operator, ++, adds I to a variable, while the decrement operator, --, subtracts I from a variable. Both of these operators can be used as prefixes or postfixes. In prefix operation, the variable is incremented (or decremented) before its value is used. In postfix operation, the variable is incremented (or decremented) after its value is used.

5. C provides library functions for calculating square root, logarithmic, and other mathematical computations. Each program using one of these mathematical functions must either include the statement # include <math. h> or have a function declaration for the mathematical function before it is called.

6. Mathematical functions may be included within larger expressions.

7. The scanf() function is a standard library function used for data input. The scanf() function requires a control string and a list of addresses. The general form of this function call is
```
scanf("control string", &argl, &arg2, ..., &argn);
```
The control string typically contains only conversion control sequences, such as %, and must contain the same number of conversion control sequences as the argument addresses.

8. When a `scan()` function is encountered, the program temporarily suspends further statement execution until sufficient data has been entered for the number of variable addresses contained in the `scan()` function call.

9. It is good programming practice to display a message, prior to a `scan()` function call, that alerts the user as to the type and number of data items to be entered. Such a message is called a prompt.

10. Field width specifiers can be included with conversion control sequences to explicitly specify the format of displayed fields. This includes both the total width of the output field and, for floating-point and double-precision numbers, the number of decimal digits to display.

11. Each compiled C program is automatically passed through a preprocessor. Lines beginning with # in the first column are recognized as commands to this preprocessor. Preprocessor commands are not terminated with a semicolon.

12. Expressions can be made equivalent to a single identifier using the preprocessor

`#define` command. This command has the form
#define identifier expression
and allows the identifier to be used instead of the expression anywhere in the program after the command. Generally, a `#define` command is placed at the top of a program or at the beginning of the `main()` function.

Chapter 4 Summary

1. Relational expressions, which are also called simple conditions, are used to compare operands. If a relational expression is true, the value of the expression is the integer 1. If the relational expression is false, it has an integer value of 0. Relational expressions are created using the following relational operators:

Relational Operator	Meaning	Example
<	less than	age < 30
>	greater than	height > 6.2
<=	less than or equal to	taxable <= 20000
>=	greater than or equal to	temp >= 98.6
==	equal to	grade == 100
! =	not equal to	number != 250

2. More complex conditions can be constructed from relational expressions using C's logical operators, && (AND), || (OR), and ! (NOT).

3. A one-way if statement has the general form:

```
if (expression)
    statement;
```

4. A compound statement consists of any number of individual statements enclosed within the brace pair { and }. Compound statements are treated as a single unit and can be used anywhere a single statement is called for.

5. An if-else statement is used to select between two alternative statements based on the value of an expression. Although relational expressions are usually used for the tested expression, any valid expression can be used. All of C's selection statements interpret a non-0 value as true and a 0 value as false. The general form of an if-else statement is:

```
if (expression)
    statement1;
else
    statement2;
```

If the expression has a non-0 value, it is considered as true and statement 1 is executed; otherwise statement 2 is executed.

6. An if-else statement can contain other if-else statements. In the absence of braces, each else is associated with the closest unpaired if.

7. The if-else chain is a multiway selection statement having the general form:

```
if (expression1)
    statement1;
else if (expression2)
    statement2;
else if (expression3)
    statement3;
...
else if (expressionm)
    statementm;
else
    statementn;
```

Each expression is evaluated in the order it appears in the chain. Once an expression is true (has a non-0 value), the statement between that expression and the next else if or else is executed, and no further expressions are tested. The final else is optional, and the statement corresponding to the final else is only executed if none of the previous expressions are true.

8. The switch statement is a multiway selection statement. The general form of a switch statement is:

```
switch (integer-expression) {
    case value1: // terminated with a colon
        statement1;
        statement2;
        break;
    case value2: // terminated with a colon
        statementm;
        statementn;
        break;
    case value3: // terminated with a colon
        statementw;
        statementx;
        break;
    default: // terminated with a colon
        statementaa;
        statementbb;
        } // end of switch and compound statement
```

For this statement, the value of an integer expression is compared to a number of integer or character constants or constant expressions. Program execution is transferred to the first matching case and continues through the end of the switch statement unless an optional break statement is encountered. The cases in a switch statement can appear in any order, and an optional default case can be included. The default case is executed if none of the other cases is matched.

Chapter 5 Summary

1. A section of repeating code is referred to as a loop. The loop is controlled by a repetition statement that tests a condition to determine whether the code within the loop will be executed. Each pass through a loop is referred to as a repetition or iteration. The tested condition must always be explicitly set prior to its first evaluation by the repetition statement. Within the loop, there must always be a statement that permits altering of the condition so that the loop, once entered, can be exited.

2. The three C repetition statements, which are used to construct their corresponding loop types, are:

a. while
b. for
c. do-while

The while and for statements are used to create while and for loops, respectively. Both of these loop types are pretest or entrance-controlled loops. In this type of loop, the tested condition is evaluated at the beginning of the loop, which requires that the tested condition be explicitly set prior to loop entry. If the condition is true, loop repetitions begin; otherwise, the loop is not entered. Iterations continue as long as the condition remains true.

The do-while statement is used to create a do-while loop, which is a posttest or exit-controlled loop that tests its condition at the end of the loop. This type of loop is always executed at least once and, as long as the tested condition remains true, do-while loops continue to execute.

3. Loops are also classified as to the type of tested condition. In a counter-controlled loop, the condition is used to keep track of how many repetitions have occurred. In a condition-controlled loop, the tested condition is based on encountering one or more specific values.

4. The most commonly used syntax for a while loop is:

```
while (expression) {
	statements;
}
```

The expression contained within parentheses is the condition tested to determine if the statement following the parentheses, which is generally a compound statement, is executed. The expression is evaluated in exactly the same manner as that contained in an if-else statement; the difference is how the expression is used. In a while statement, the statement following the expression is executed repeatedly as long as the expression retains a non-value, rather than just once, as in an if-else statement. An example of a while loop is:

```
count = 1;
while (count <= 10) {
	printf ("%d ", count);
	count++;
}
```

5. A for statement performs the same functions as the while statement, but uses a different form. In many situations, especially those that use a fixed-count condition, the for statement format is easier to use than its while statement equivalent. The most commonly used form of the for statement is:

```
for (initializing list; expression; altering list) {
	statements;
}
```

Within the parentheses of the for statement are three items, separated by semicolons. Each of these items is optional but the semicolons must be present. The initializing list is used to set any initial values before the loop is entered; generally it is used to initialize a counter. Statements within the initializing list are only executed once. The expression in the for statement is the condition that is tested at the start of the loop and prior to each iteration. The altering list contains loop statements that are not contained within the compound statement: generally it is used to increment or decrement a counter each time the loop is executed. Multiple statements within a list are separated by commas. An example of a for loop is:
```
```

```
for (total = 0, count = 1; count < 10; count++)
{
    printf("Enter a grade: ");
    scanf("%f", &grade);
    total += grade;
}
```

6. The for statement is extremely useful in creating counter-controlled loops. This is because the initializing statements, the tested expression, and statements affecting the tested expression can all be included in parentheses at the top of a for loop for easy inspection and modification.

7. The do-while statement is used to create posttest loops because it checks its expression at the end of the loop. This ensures that the body of a do loop is executed at least once. Within a do loop, there must be at least one statement that alters the tested expression's value or is a break statement.

Chapter 6 Summary

1. A function is called by giving its name and passing any data to it in the parentheses following the name. If a variable or expression is one of the arguments in a function call, the called function receives a copy of the variable's or expression's value, respectively.

2. The commonly used form of a user-written function is:

```
returnType functionName (parameter list)
{
    declarations;
    statements;
    return (expression);
}
```

The first line of the function is called the function header. The opening and closing braces of the function and all statements in between these braces constitute the function's body. The parameter list must include the names of all parameters and their data types.

3. A function's return type is the data type of the value returned by the function. If no type is explicitly declared, the up function is assumed to return an integer value. If the function does not return any value, it should be declared as a void type.

4. Functions can directly return at most a single value to their calling functions. This value is the value of the expression in the return statement.

5. Functions can be declared to all calling functions by means of a function prototype.

The prototype provides a declaration for a function that specifies the data type returned by the function, its name, and the data types of the parameters expected by the function. As with all declarations, a function prototype is terminated with a semicolon and may be specified as a global declaration or included within a function's local variable declarations. The most common form of a function prototype is:
returnDataType functionName (parameter data types);

If the called function is placed physically above the calling function, no further declaration is required because the function's definition serves as a global declaration to all following functions.

6. Arguments passed to a function provide a means of evaluating any valid C expression. It is the expression's value that is then passed to the called function.

7. A set of preprogrammed functions for mathematical calculations, character input and output, character processing, and numerical conversions are included in the standard library provided with each C compiler. To use one of these functions, you must know the name of the function, the arguments expected by the function, the data type of the returned value (if any), and a description of what the function does. You must also include the specific header file containing the function's prototypes and definition.

Chapter 7 Summary

1. Every variable used in a program has scope, which determines where in the program the variable can be used. The scope of a variable is either local or global and is determined by where the variable's definition statement is placed.

A local variable is defined within a function and can only be used within its defining function or block. A global variable is defined outside a function and can be used in any function following the variable's definition. All non-static global variables are initialized to 0 and can be shared between files using the keyword extern.

2. Every variable has a class. The class of a variable determines how long the value in the variable will be retained.

Automatic (auto) variables are local variables that exist only while their defining function is executing. Register variables are similar to automatic variables but are stored in a computer's internal registers rather than in memory. Variables that are static can be either global or local and retain their values for the duration of a program's execution. Static variables are also set to 0 or blanks when they are defined if they are not explicitly initialized by the user.

3. Every variable has a data type, a value, and an address. In C, the address of a variable

can be obtained by using the address operator, &.

4. A pointer is a variable or parameter that is used to store the address of another variable. Pointers, like all C variables and parameters, must be declared. The indirection operator, *, is used to both declare a pointer and to access the variable whose address is stored in a pointer. For example, the statement int * datePtr declares that the identifier named datePtr is a pointer to an integer value. This is commonly read as "datePtr points to an int."

5. If a parameter or variable is a pointer, then the indirection operator, *, must be used to access the variable whose address is stored in the pointer. For example, if `datePtr` has been declared as a pointer, then the value pointed to by this pointer is accessed by the expression `*datePtr`. This can be read as "the thing (actually the value) pointed to by `datePtr`."

6. The address of a variable can be passed to a function. The parameter receiving this address must be declared as a pointer. Passing an address is referred to as a pass by reference.

7. When a called function receives an address, it has the capability of directly accessing the respective calling function's variable. Using passed addresses permits a called function to effectively return multiple values.

8. A recursive solution is one in which the solution can be expressed in terms of a "simpler" version of itself. A recursive algorithm must always specify:
- The first case or cases
- How the nth case is related to the (n-1) case

9. If a problem solution can be expressed repetitively or recursively with equal ease, the repetitive solution is preferable because it executes faster and uses less memory. In many advanced applications, recursion is simpler to visualize and the only practical means of implementing a solution.

Chapter 8 Summary

1. A single-dimensional array is a data structure that can be used to store a list of values of the same data type. Such arrays must be declared by giving the data type of the values that are stored in the array and the array size. For example, the declaration int num[1001]; creates an array of 100 integers. A preferable approach is first to use a named constant for the array size and then use this constant in the definition of the array. For example, #define MAXSIZE 100 and int num[MAXSIZE];

2. Array elements are stored in contiguous locations in memory and referenced using the

array name and a subscript, for example, num[22]. Any nonnegative integer value expression can be used as a subscript and the subscript 0 always refers to the first element in an array.

3. Single-dimensional arrays may be initialized when they are declared. This is accomplished by listing the initial values, in a row-by-row manner, within braces and separating them with commas. For example, the declaration int nums[] = {3, 7, 8, 15};

4. Single-dimensional arrays are passed to a function by passing the name of the array as an argument. The value actually passed is the address of the first array storage location. Thus, the called function receives direct access to the original array and not a copy of the array elements. Within the called function, a parameter must be declared to receive the passed array name. The declaration of the parameter can omit the size of the array.

5. A two-dimensional array is declared by listing both a row and a column size with the data type and name of the array. For example, the declarations

```
#define ROWS 5
#define COLS 7
int mat[ROWS][COLS];
```

create a two-dimensional array consisting of five rows and seven columns of integer values.

6. Two-dimensional arrays may be initialized when they are declared. This is accomplished by listing the initial values, in a row-by-row manner, within braces and separating them with commas. For example, the declaration

```
int vals[ROWS][COLS] = { {1, 2, 3},
                         {4, 5, 6},
                         {7, 8, 9},
                         {10, 11, 12},
                         {13, 14, 15} };
```

produces the following five-row-by-three-column array:

```
1 2 3
4 5 6
7 8 9
10 11 12
13 14 15
```

As C uses the convention that initialization proceeds in row-wise order, the inner braces can be omitted. Thus, an equivalent initialization is provided by the statement

```
int vals[ROWS][COLS] = {1, 2, 3, 4, 5, 6};
```

7. Two-dimensional arrays are passed to a function by passing the name of the array as an argument. Within the called function, a parameter must be declared to receive the passed array name. The declaration of the parameter can omit the row size of the array.

Chapter 9 Summary

1. A string is an array of characters that is terminated by the NULL ('\0') character.

2. Character arrays can be initialized using a string assignment of the form:
   ```
   char arrayName[] = "text";
   ```
   This initialization is equivalent to:
   ```
   char arrayName[] = {'t', 'e', 'x', 't', '\0'};
   ```

3. Strings can always be processed using standard array-processing techniques. The input and display of a string, however, always require reliance on a standard library function.

4. The gets(), scanf(), and getchar() library functions can be used to input a string. The scanf() function tends to be of limited usefulness for string input because it terminates input when a blank is encountered.

5. The puts(), printf(), and putchar() functions can be used to display strings.

6. Many standard library functions exist for processing strings as a complete unit. Internally, these functions manipulate strings in a character-by-character manner, and are included in the string.h header file.

7. The standard C library also includes individual character-handling functions in the ctype.h header file.

8. One of the major uses of strings in programs is validating user input, which is an essential part of any program. Strings are extremely useful for this purpose because they easily permit the inspection of each individual character that is entered by the user.

9. The conversion routines atoi() and atof() are provided in the stdlib.h header file for converting strings to integer and double-precision numeric values. These routines are typically applied to user-input data after the data has been validated for the input data type expected by the application.

Chapter 10 Summary

1. A data file is any collection of data stored together in an external storage medium under a common name.

2. Data files can be stored as either character-based or binary files. A character-based file, also referred to as a text file, stores each individual digit, letter, and symbol with a separate code. This permits the file to be viewed and manipulated with a word processor or editor program. A binary file stores numbers using the internal binary code (typically two's complement) used by the computer's internal processing unit.

3. A data file is opened using the fopen() standard library function. This function connects a file's external name with an internal pointer name. After the file is opened, all subsequent accesses to the file require the internal pointer name. The default file type in C is text. Binary files are designated by including a "b" in the mode indicator, which is the second argument required by fopen().

4. A file can be opened for reading, writing, or appending. A file opened for writing creates a new file and erases any existing file having the same name as the opened file. A file opened for appending makes an existing file available for data to be added to the end of the file. If the file does not exist, it is created. A file opened for reading makes an existing file's data available for input.

5. An internal filename must be declared as a pointer to a FILE. This means that a declaration similar to `FILE *fileName;` must be included with the declarations in which the file is opened. In this declaration, `fileName` can be replaced with any user-selected variable name.

6. In addition to any files opened within a function, the standard files `stdin`, `stdout`, and `stderr` are automatically opened when a program is run. The symbolic constant `stdin` is the pointer name of the physical file used for data entry by `scanf()`, `stdout` is the pointer name of the physical file device used for data display by `printf()`, and `stderr` is the pointer name of the physical file device used for displaying system error messages.

7. Data files can be accessed randomly using the `rewind()`, `fseek()`, and `ftell()` functions.

8.Table 10.7 lists the standard file library functions.

.

Table 10.7 Standard File Library Functions

Name	Purpose
fopen()	Open or create a file
fclose()	Close a file

fgetc()	Character input
getchar()	Character input from stdin
fgets()	String input
gets()	String input from stdin
fscanf()	Formatted input
scanf()	Formatted input from stdin
fputc()	Character output
putchar()	Character output to stdout
fputs()	String output
puts()	String output to stdout
fprintf()	Formatted output
printf()	Formatted output to stdout
fseek()	File positioning
rewind()	File positioning to the beginning
ftell()	Position reporting

Chapter 11 Summary

1. An array name is a pointer constant. The value of the pointer constant is the address of element zero in the array. Thus, if `val` is the name of an array, `val` and `&val[0]` can be used interchangeably.

2. Any access to an array element using subscript notation can always be replaced using pointer notation. That is, the notation `a[i]` can always be replaced by the notation `*(a + i)`. This is true whether `a` was initially declared explicitly as an array or as a pointer.

3. Arrays are passed to functions by address, not by value. The called function always receives direct access to the originally declared array elements.

4. When a single-dimensional array is passed to a function, the parameter declaration for the array can be either an array declaration or a pointer declaration. Thus, the following parameter declarations are equivalent: `double a[]` and `double *a`.

5. In place of subscripts, pointer notation and pointer arithmetic are especially useful for manipulating string elements.

6. String storage can be created by declaring an array of characters or a pointer to be a character. A pointer to a character can be assigned a string directly. String assignment to an array of characters is invalid except when done in a declaration statement.

7. Pointers can be incremented, decremented, and compared. Numbers added to or subtracted from a pointer are automatically scaled. The scale factor used is the number of bytes required to store the data type originally pointed to.

Chapter 12 Summary

1. Each variable in a structure is accessed by its structure name, followed by a period, followed by its individual variable name. Another term for a structure is a record. The general form for declaring a structure is:

```
struct {
    individual member declarations;
} structureVariableName;
```

2. A structure type name can be used to create a generalized structure type describing the form and arrangement of elements in a structure. This declaration has the syntax:

```
struct StructureTypeName {
    individual member declarations;
};
```

Individual structure variables may then be defined as this StructureTypeName. By convention, the first letter of a StructureTypeName is always capitalized.

3. Structures are particularly useful as elements of arrays. Used in this manner, each structure becomes one record in a list of records.

4. Individual members of a structure are passed to a function in the manner appropriate to the data type of the member being passed. Most ANSI C compilers allow complete structures to be passed, in which case the called function receives a copy of each element in the structure. The address of a structure may also be passed, which provides the called function with direct access to the structure.

5. Structure members can be any valid C data type, including structures, unions, arrays, and pointers. When a pointer gis is included as a structure member, a linked list can be created. Such a list uses the pointer in one structure to "point to" (contain the address of) the next logical structure in the list.

6. Unions are declared in the same manner as structures. The definition of a union creates a memory overlay area, with each union member using the same memory storage locations. Thus, only one member of a union may be active at a time.

Chapter 13 Summary

1. An alternative to fixed memory allocation for variables at compile time is the dynamic allocation of memory at run time. This dynamic allocation, which is also known as a run-time allocation, allocates and deallocates memory storage under program control

and is extremely useful when dealing with a list of data that can expand and contract as items are added and deleted from the list. In C, the functions that are used for dynamic memory allocation are malloc(), calloc(), realloc(), and free().

2. The malloc() function reserves a requested number of bytes and returns a pointer to the first reserved byte. For example, the expression malloc(50 * sizeof(int)) reserves a sufficient number of bytes to store 50 integers. The malloc() function will either return a pointer to the first byte of reserved storage or return a NULL pointer if the request cannot be satisfied. Since the returned pointer always "points to" a void, it must always be cast into a pointer of the desired type. Thus, this specific request for storage would be made using the expression pointerToInt = (int *) malloc(50 * sizeof(int)), where pointerToInt has been declared as a pointer to an integer.

In using malloc(), you should always check its return value to ensure that a NULL pointer was not returned, which would indicate that the request for memory space was not satisfied. Continuing with our example, this check would take the form:

```
if (pointerToInt == (int *) NULL)
{
     // do an error procedure in here-typically an exit
}
```

3. The `realloc()` function operates in a similar fashion as the `malloc()` function except it is used to expand or contract an existing allocated space. If the new size is larger than the previously allocated space, only the additional space remains uninitialized and the previously allocated space retains its contents; otherwise, the new space retains its prior contents up to its new limits. As with `malloc()`, the return address provided by `realloc()` should always be checked.

4. The `free()` function is used to deallocate previously allocated memory space.

Chapter 14 Summary

1. A typedef statement creates synonym names, which are known as aliases, for any C data type name. For example, the statement
typedef int WHOLENUM;
makes WHOLENUM a synonym for int.

2. A conditional expression provides an alternate way of expressing a simple if-else statement. The general form of a conditional expression is
expression1 ? expression2 : expression3.
The equivalent if-else statement for this is

```
if (expression1)
   expression2;
```

```
else
   expression3;
```

3. C also provides a goto statement. In theory, this statement need never be used. In practice, it produces confusing and unstructured code and should be used only in a very limited and controlled manner, if at all.

4. Individual bits of character and integer variables and constants can be manipulated using C's bit operators. These are the AND, inclusive OR, exclusive OR, complement, left shift, and right shift operators.

5. The AND and inclusive OR operators are useful in creating masks. These masks can be used to pass or eliminate individual bits from the selected operand. The exclusive OR operator is useful in complementing an operand's bits.

6. When the AND and OR operators are used with operands of different sizes, the shorter operand is always increased in bit size to match the size of the larger operand.

7. The shift operators produce different results depending on whether the operand is a signed or an unsigned value.

8. Using the #define command, complete expressions can be equated to symbolic names. When these expressions include arguments, they are referred to as macros.

9. Arguments passed to main() are termed command-line arguments. C provides a standard argument-passing procedure in which main() can accept any number of arguments passed to it. Each argument passed to main() is considered a string and is stored using a pointer array named argv. The total number of arguments on the command line is stored in an integer variable named arg.

Chapter 15 Summary

1. C++ is an object-oriented language that also supports procedural programming.

2. Creating a C-like procedural program using C++ essentially involves changing syntax for the input and output statements. Instead of printf() and scanf() used in C, C++ uses cout and cin. Additionally, C++ requires a different set of header files than those used in C.

3. All object-oriented languages, including C++, must provide the ability to create classes and provide for inheritance and polymorphism.

4. A class is a programmer-defined data type that includes specification of data values

and operations that can be performed on these values.

5. Inheritance is the ability to create a new class by extending the definition of an existing class.

6. Polymorphism is the ability to have the same function perform different tasks depending on the class it is a member of.

7. Every variable in a C++ program must be declared as to the type of value it can store. Declarations within a function may be placed anywhere within a function, although a variable can only be used after it is declared. Variables may also be initialized when they are declared. Additionally, variables of the same type may be declared using a single declaration statement. Variable declaration statements have the general form:

dataType variableName(S);

8. A simple C++ program containing declaration statements typically has the form:

```
int main()
{
    declaration statements;
    other statements;
    return 0;
}
```

Although declaration statements may be placed anywhere within the function's body, a variable may only be used after it is declared.

9. The general form of a statement using `cout` to display output from a C++ program is:

```
cout << expression1 << expression2 << ... << expressionn;
```

where all expressions are either strings, variables, or specific values.

10. The general form of a statement using `cin` to accept data input from the keyboard is:

```
cin >> variable1 >> variable2 >> ... >> variablen;
```

where each variable must be preceded by the `>>` symbol.

11. It is good programming practice to display a message, prior to a `cin` statement, that alerts the user as to the type and number of data items to be entered. Such a message is called a prompt.

12. The use of `cout` and `cin` within a program requires that the header lines

```
#include <iostream>
using namespace std;
```

be placed at the top of the program.