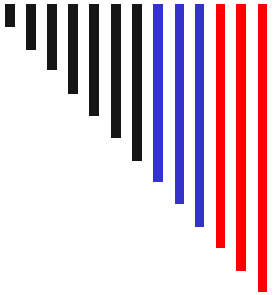

Chapter 2
Introduction to the C Language

Objectives

- To understand the structure of a C-language program.
- To write your first C program.
- To introduce the include preprocessor command.
- To be able to create good identifiers for objects in a program.
- To be able to list, describe, and use the C basic data types.
- To be able to create and use variables and constants.
- To understand input and output concepts.
- To be able to use simple input and output statements.
- To understand the software engineering role.

2-1 Background

C is a structured programming language. It is considered a high-level language because it allows the programmer to concentrate on the problem at hand and not worry about the machine that the program will be using. While many languages claim to be machine independent, C is one of the closest to achieving that goal. That is another reason why it is used by software developers whose applications have to run on many different hardware platforms.

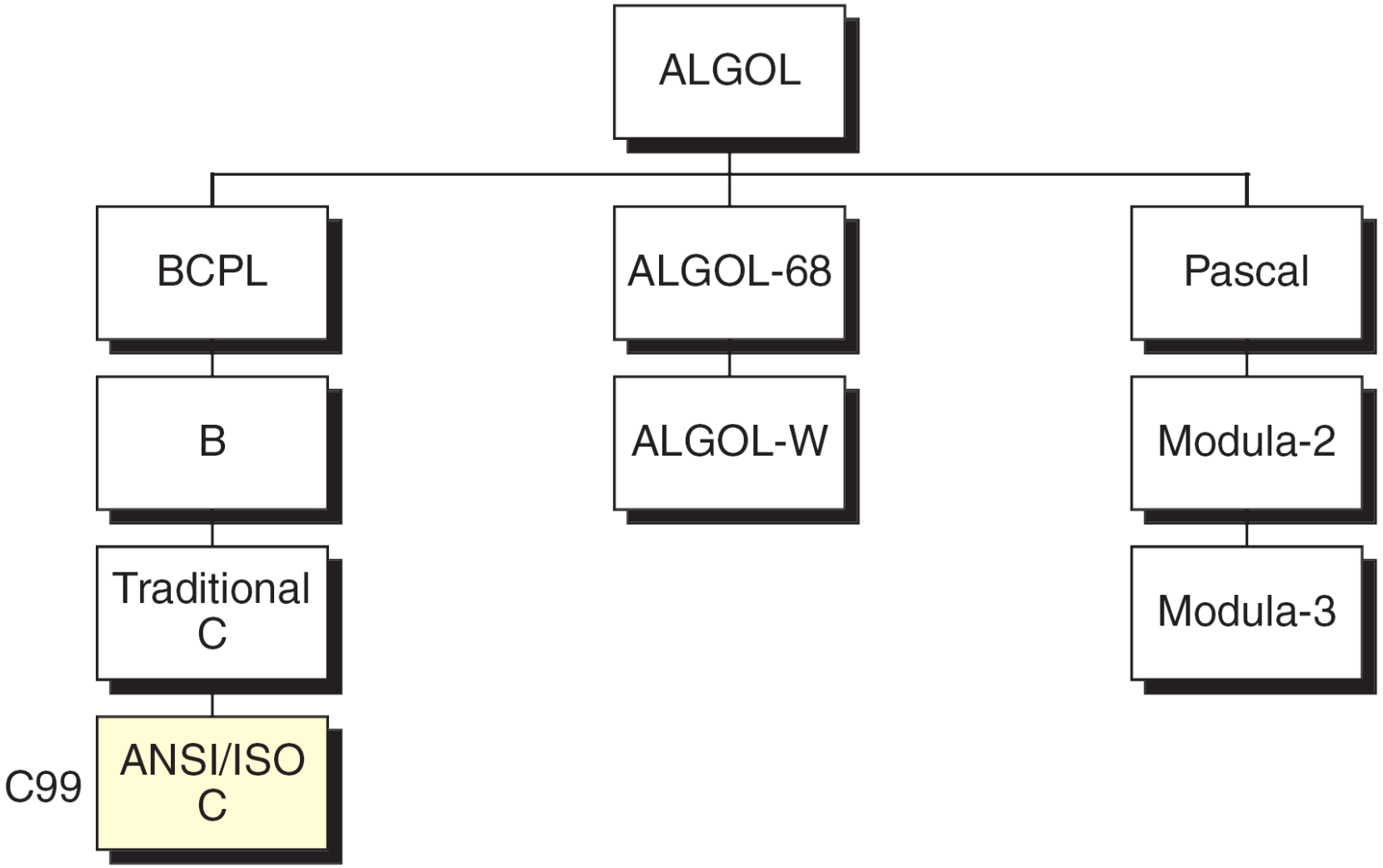

FIGURE 2-1 **Taxonomy of the C Language**

2-2 C Programs

It's time to write your first C program! This section will take you through all the basic parts of a C program so that you will be able to write it.

Topics discussed in this section:

Structure of a C Program
Your First C Program
Comments
The Greeting Program

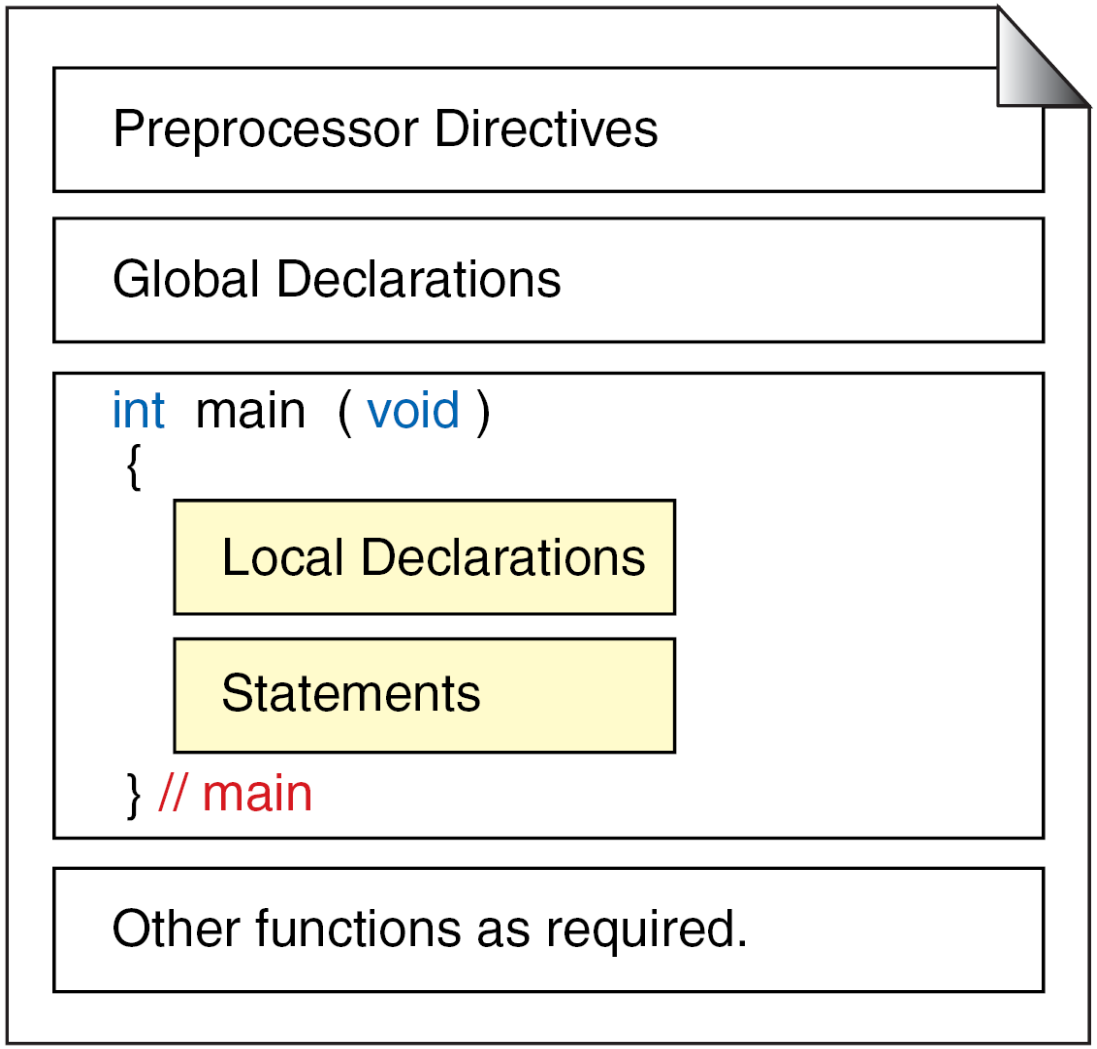

FIGURE 2-2 Structure of a C Program

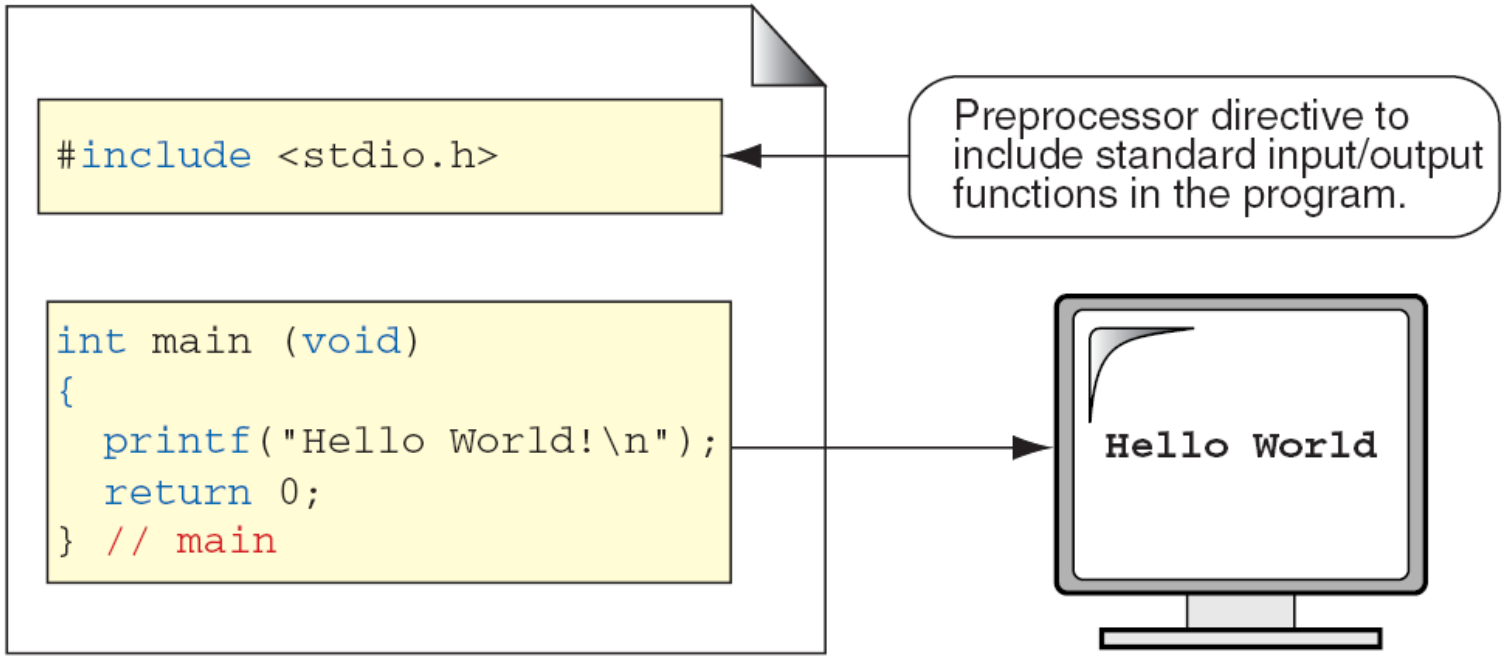

FIGURE 2-3 The Greeting Program

```
/* This is a block comment that
   covers two lines.                    */

/*
** It is a very common style to put the opening token
** on a line by itself, followed by the documentation
** and then the closing token on a separate line. Some
** programmers also like to put asterisks at the beginning
** of each line to clearly mark the comment.
*/
```

FIGURE 2-4 Examples of Block Comments

```
// This is a whole line comment

a = 5;              // This is a partial line comment
```

FIGURE 2-5 **Examples of Line Comments**

```
/* ====== /* ------ */ ====== */
```

FIGURE 2-6 **Nested Block Comments Are Invalid**

PROGRAM 2-1 The Greeting Program

```
1  /* The greeting program. This program demonstrates
2     some of the components of a simple C program.
3        Written by:  your name here
4        Date:        date program written
5  */
6  #include <stdio.h>
7
8  int main (void)
9  {
10 // Local Declarations
11
12 // Statements
13
14    printf("Hello World!\n");
15
16    return 0;
17 } // main
```

2-3 Identifiers

One feature present in all computer languages is the identifier. Identifiers allow us to name data and other objects in the program. Each identified object in the computer is stored at a unique address. If we didn't have identifiers that we could use to symbolically represent data locations, we would have to know and use object's addresses. Instead, we simply give data identifiers and let the compiler keep track of where they are physically located.

1. First character must be alphabetic character or underscore.
2. Must consist only of alphabetic characters, digits, or underscores.
3. First 63 characters of an identifier are significant.
4. Cannot duplicate a keyword.

Table 2-1 Rules for Identifiers

Note

An identifier must start with a letter or underscore: it may not have a space or a hyphen.

Note

C is a case-sensitive language.

Valid Names		Invalid Name	
a	// Valid but poor style	$sum	// $ is illegal
student_name		2names	// First char digit
_aSystemName		sum-salary	// Contains hyphen
_Bool	// Boolean System id	stdnt Nmbr	// Contains spaces
INT_MIN	// System Defined Value	int	// Keyword

Table 2-2 Examples of Valid and Invalid Names

2-4 Types

A type defines a set of values and a set of operations that can be applied on those values. For example, a light switch can be compared to a computer type. It has a set of two values, on and off. Only two operations can be applied to a light switch: turn-on and turn-off.

Topics discussed in this section:

Void Type
Integral Type
Floating-Point Types

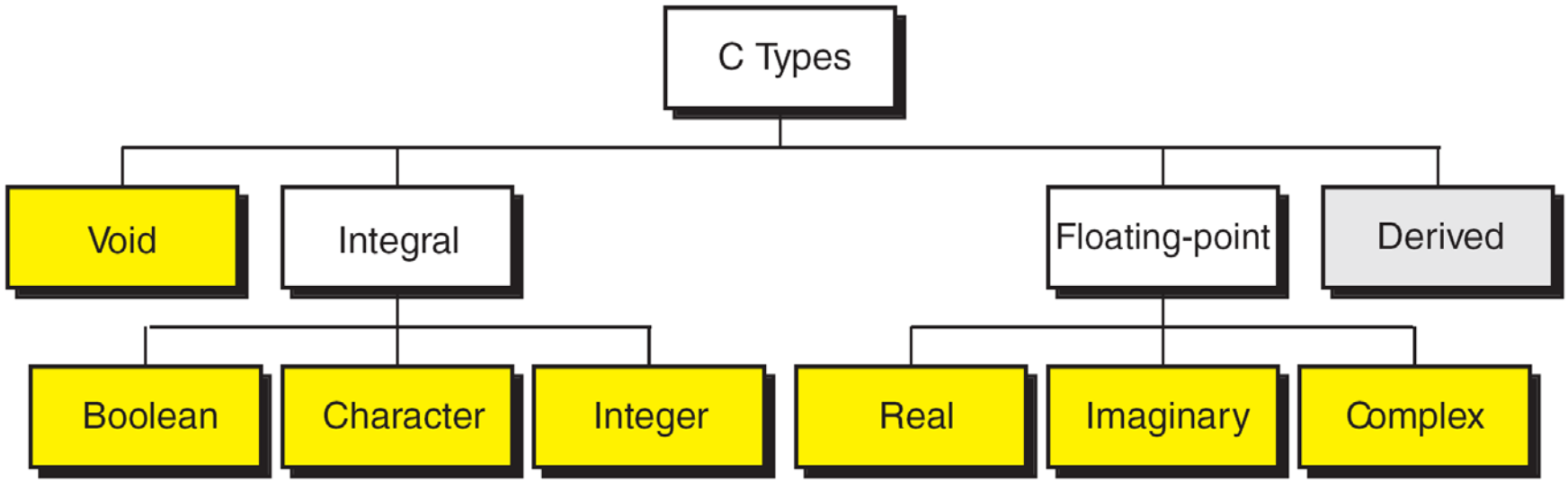

FIGURE 2-7 Data Types

- Void
  - Indicates no values and no operations
  - Useful for functions and pointers
  - Keyword: void
- Boolean
  - Either true (1) or false (0)
  - New addition in C99
  - Keyword: bool

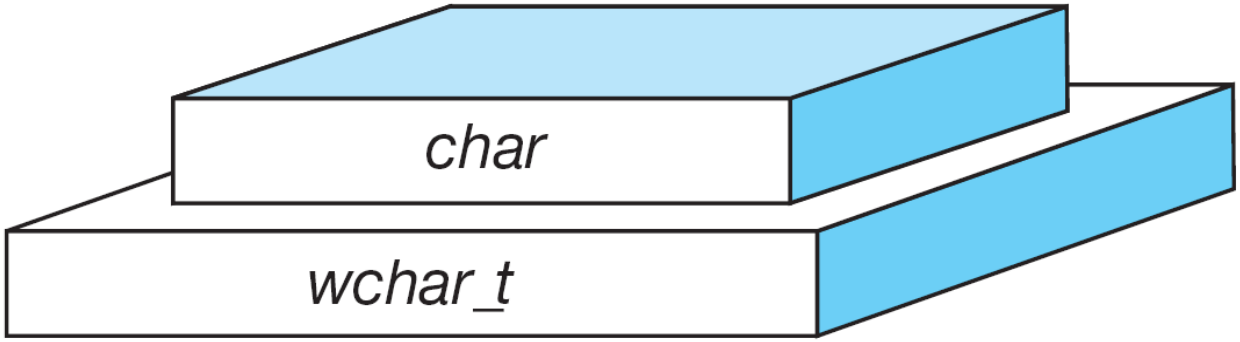

FIGURE 2-8 Character Types

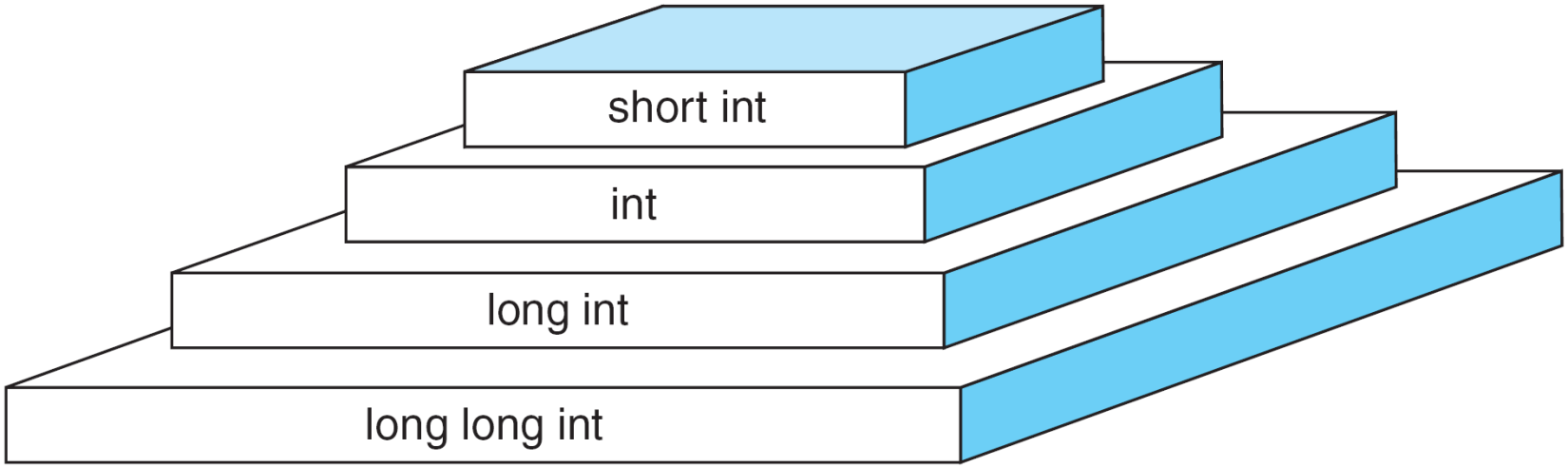

FIGURE 2-9 Integer Types

Note

sizeof (short) ≤ sizeof (int) ≤ sizeof (long) ≤ sizeof (long long)

Type	Byte Size	Minimum Value	Maximum Value
short int	2	–32,768	32,767
int	4	–2,147,483,648	2,147,483,647
long int	4	–2,147,483,648	2,147,483,647
long long int	8	–9,223,372,036,854,775,807	9,223,372,036,854,775,806

Table 2-3 Typical Integer Sizes and Values for Signed Integers

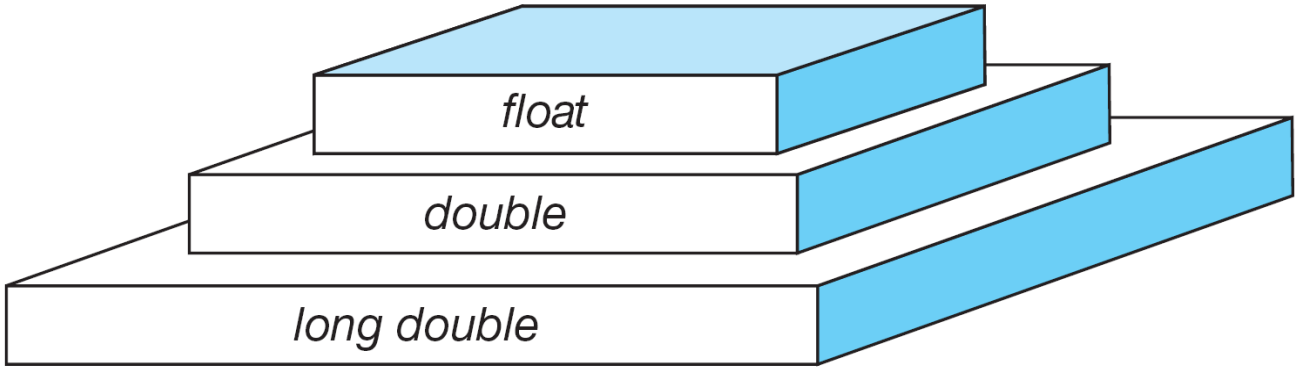

FIGURE 2-10 Floating-point Types

Note

sizeof (float) ≤ sizeof (double) ≤ sizeof (long double)

Category	Type	C Implementation
Void	Void	*void*
Integral	Boolean	*bool*
	Character	*char, wchar_t*
	Integer	*short int, int, long int, long long int*
Floating-Point	Real	*float, double, long double*
	Imaginary	*float imaginary, double imaginary, long double imaginary*
	Complex	*float complex, double complex, long double complex*

Table 2-4 Type Summary

2-5 Variables

Variables are named memory locations that have a type, such as integer or character, which is inherited from their type. The type determines the values that a variable may contain and the operations that may be used with its values.

Topics discussed in this section:

Variable Declaration

Variable Initialization

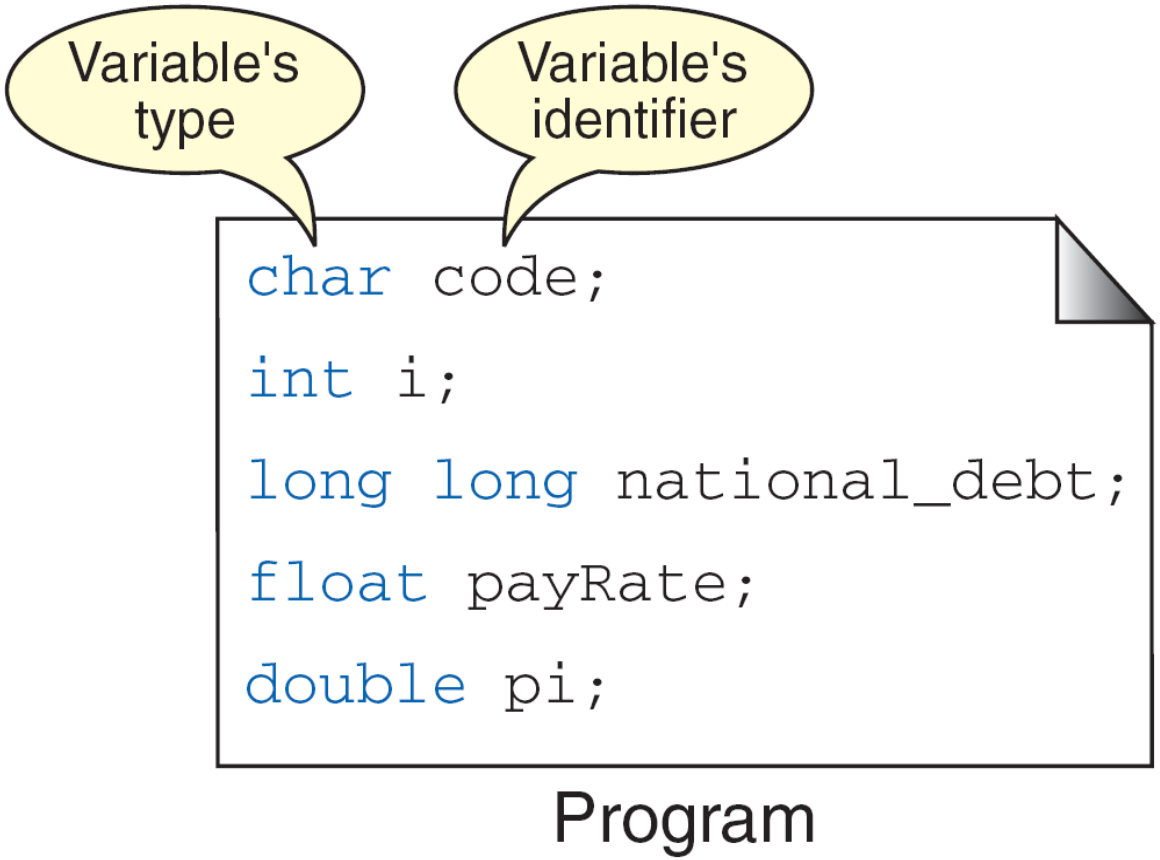

FIGURE 2-11 Variables

```
bool   fact;
short  maxItems;                 // Word separator: Capital
long   long national_debt;       // Word separator: underscore
float  payRate;                  // Word separator: Capital
double tax;
float  complex voltage;
char   code, kind;               // Poor style—see text
int    a, b;                     // Poor style—see text
```

Table 2-5 Examples of Variable Declarations and Definitions

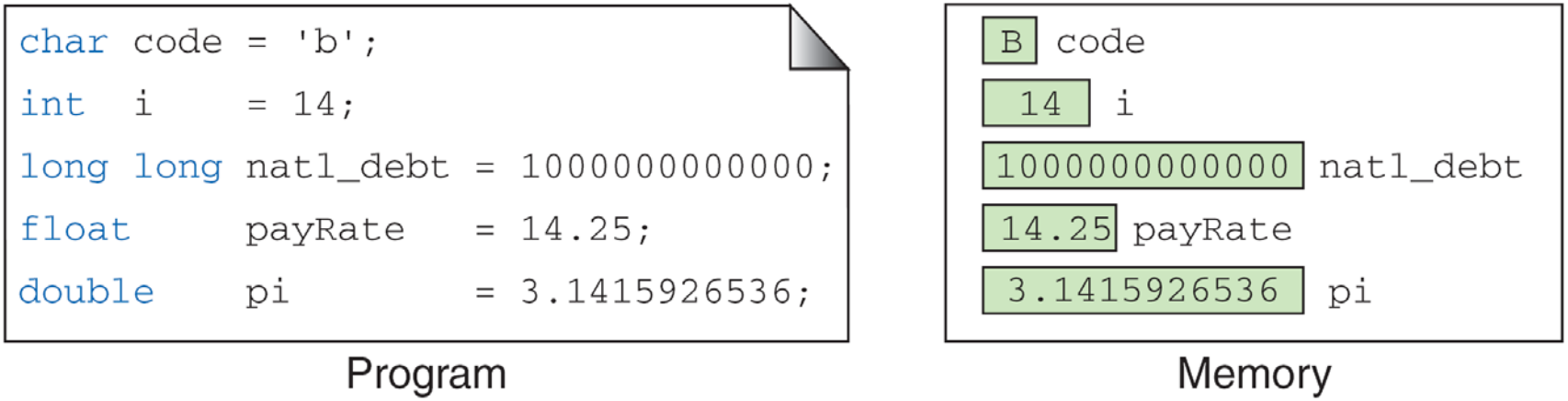

FIGURE 2-12 Variable Initialization

Note

When a variable is defined, it is not initialized. We must initialize any variable requiring prescribed data when the function starts.

PROGRAM 2-2 Print Sum of Three Numbers

```
 1  /* This program calculates and prints the sum of
 2     three numbers input by the user at the keyboard.
 3        Written by:
 4        Date:
 5  */
 6  #include <stdio.h>
 7
 8  int main (void)
 9  {
10  // Local Declarations
11     int a;
12     int b;
13     int c;
14     int sum;
15
```

PROGRAM 2-2 Print Sum of Three Numbers (continued)

```
16  // Statements
17     printf("\nWelcome. This program adds\n");
18     printf("three numbers. Enter three numbers\n");
19     printf("in the form: nnn nnn nnn <return>\n");
20     scanf("%d %d %d", &a, &b, &c);
21
22     // Numbers are now in a, b, and c. Add them.
23     sum = a + b + c;
24
25     printf("The total is: %d\n\n", sum);
26
27     printf("Thank you. Have a good day.\n");
28     return 0;
29  }  //  main
```

PROGRAM 2-2 Print Sum of Three Numbers (continued)

```
Results:
Welcome. This program adds
three numbers. Enter three numbers
in the form: nnn nnn nnn <return>
11 22 33

The total is: 66

Thank you. Have a good day.
```

2-6 Constants

Constants are data values that cannot be changed during the execution of a program. Like variables, constants have a type. In this section, we discuss Boolean, character, integer, real, complex, and string constants.

Topics discussed in this section:

Constant Representation

Coding Constants

Note

A character constant is enclosed in single quotes.

ASCII Character	Symbolic Name
null character	'\0'
alert (bell)	'\a'
backspace	'\b'
horizontal tab	'\t'
newline	'\n'
vertical tab	'\v'
form feed	'\f'
carriage return	'\r'
single quote	'\''
double quote	'\"'
backslash	'\\'

Table 2-6 Symbolic Names for Control Characters

Representation	Value	Type
+123	123	int
–378	–378	int
–32271L	–32,271	long int
76542LU	76,542	unsigned long int
12789845LL	12,789,845	long long int

Table 2-7 Examples of Integer Constants

Representation	Value	Type
0.	0.0	double
.0	0.0	double
2.0	2.0	double
3.1416	3.1416	double
–2.0f	–2.0	float
3.1415926536L	3.1415926536	long double

Table 2-8 Examples of Real Constants

Note

The two components of a complex constant must be of the same precision, that is, if the real part is type double, then the imaginary part must also be type double.

Representation	Value	Type
12.3 + 14.4 * I	$12.3 + 14.4 * (-1)^{1/2}$	double complex
14F + 16F * I	$14 + 16 * (-1)^{1/2}$	float complex
1.4736L+ 4.56756L * I	$1.4736 + 4.56756 * (-1)^{1/2}$	long double complex

Table 2-9 Examples of Complex Constants

```
""                                                    // A null string
"h"
"Hello World\n"
"HOW ARE YOU"
"Good Morning!"
L"This string contains wide characters."
```

FIGURE 2-13 **Some Strings**

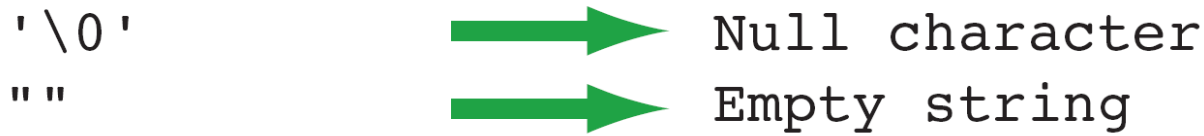

FIGURE 2-14 Null Characters and Null Strings

Note

Use single quotes for character constants.
Use double quotes for string constants.

PROGRAM 2-3 Three Ways to Use Constants

```
1  /* This program demonstrates three ways to use con-
   stants.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6  #define PI 3.1415926536
7
8  int main (void)
9  {
10 // Local Declarations
11    const double cPi = PI;
12
13 // Statements
14    printf("Defined constant PI: %f\n", PI);
15    printf("Memory constant cPi: %f\n",cPI);
```

PROGRAM 2-3 Three Ways to Use Constants (continued)

```
16      printf("Literal constant:    %f\n", 3.1415926536);
17      return 0;
18  }   // main
```

```
Results:
Defined constant PI:  3.141593
Memory constant cPi:  3.141593
Literal constant:     3.141593
```

2-7 Input/Output

Although our programs have implicitly shown how to print messages, we have not formally discussed how we use C facilities to input and output data. We devote two chapters, Chapter 7 and 13, to fully explain the C input/output facilities and how to use them. In this section, we describe simple input and output formatting.

Topics discussed in this section:

Streams

Formatting Input/Output

Note

A terminal keyboard and monitor can be associated only with a text stream. A keyboard is a source for a text stream; a monitor is a destination for a text stream.

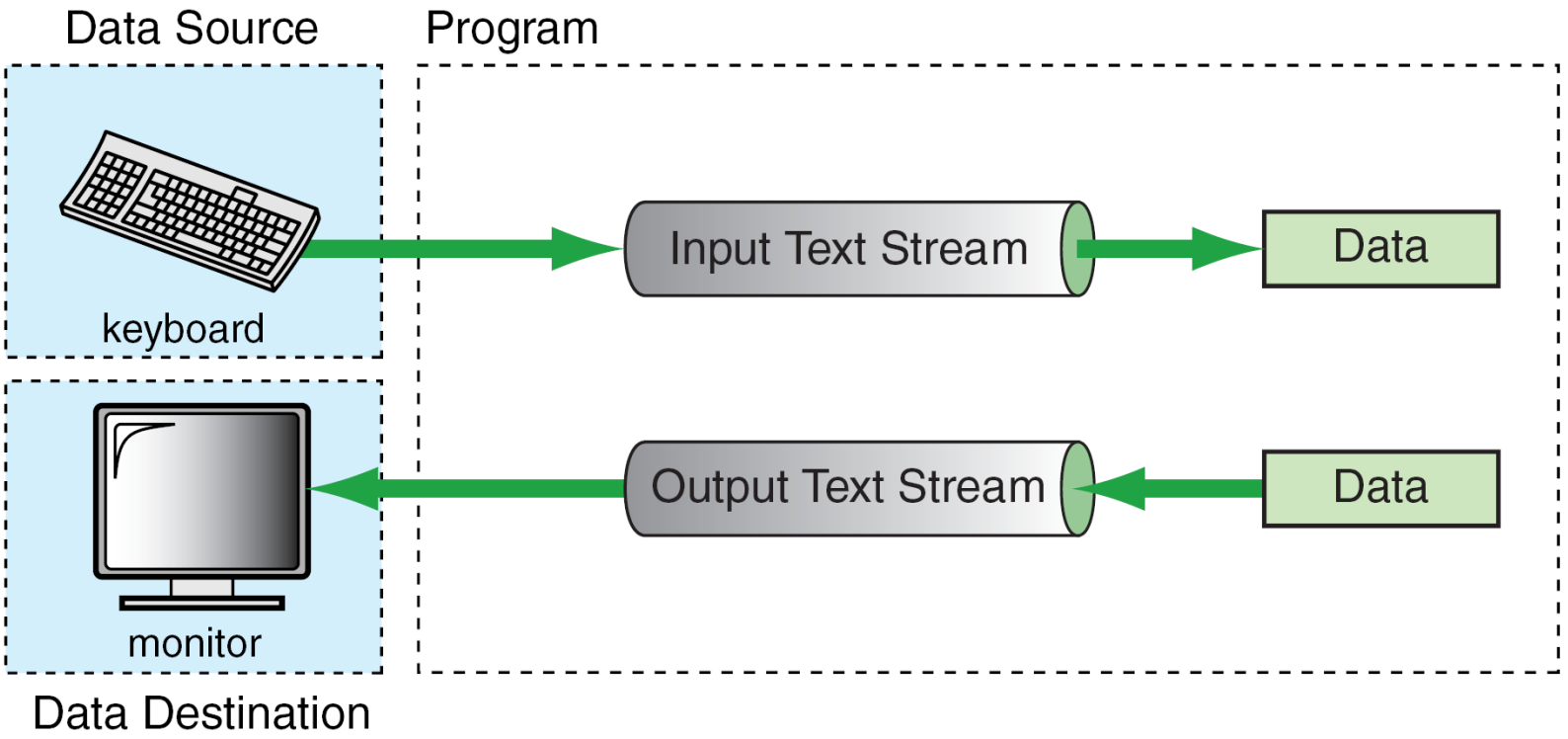

FIGURE 2-15 Stream Physical Devices

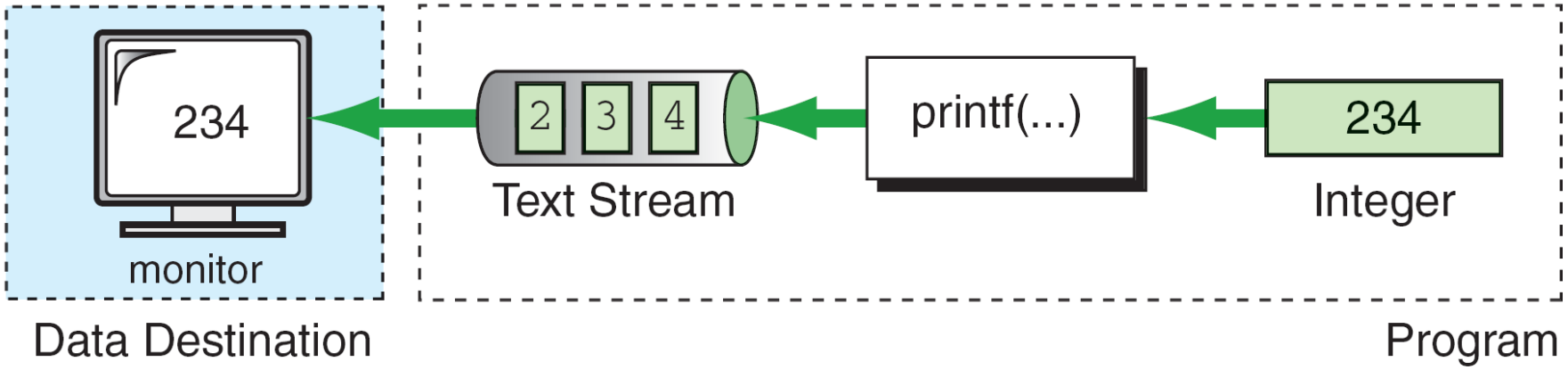

FIGURE 2-16 Output Formatting Concept

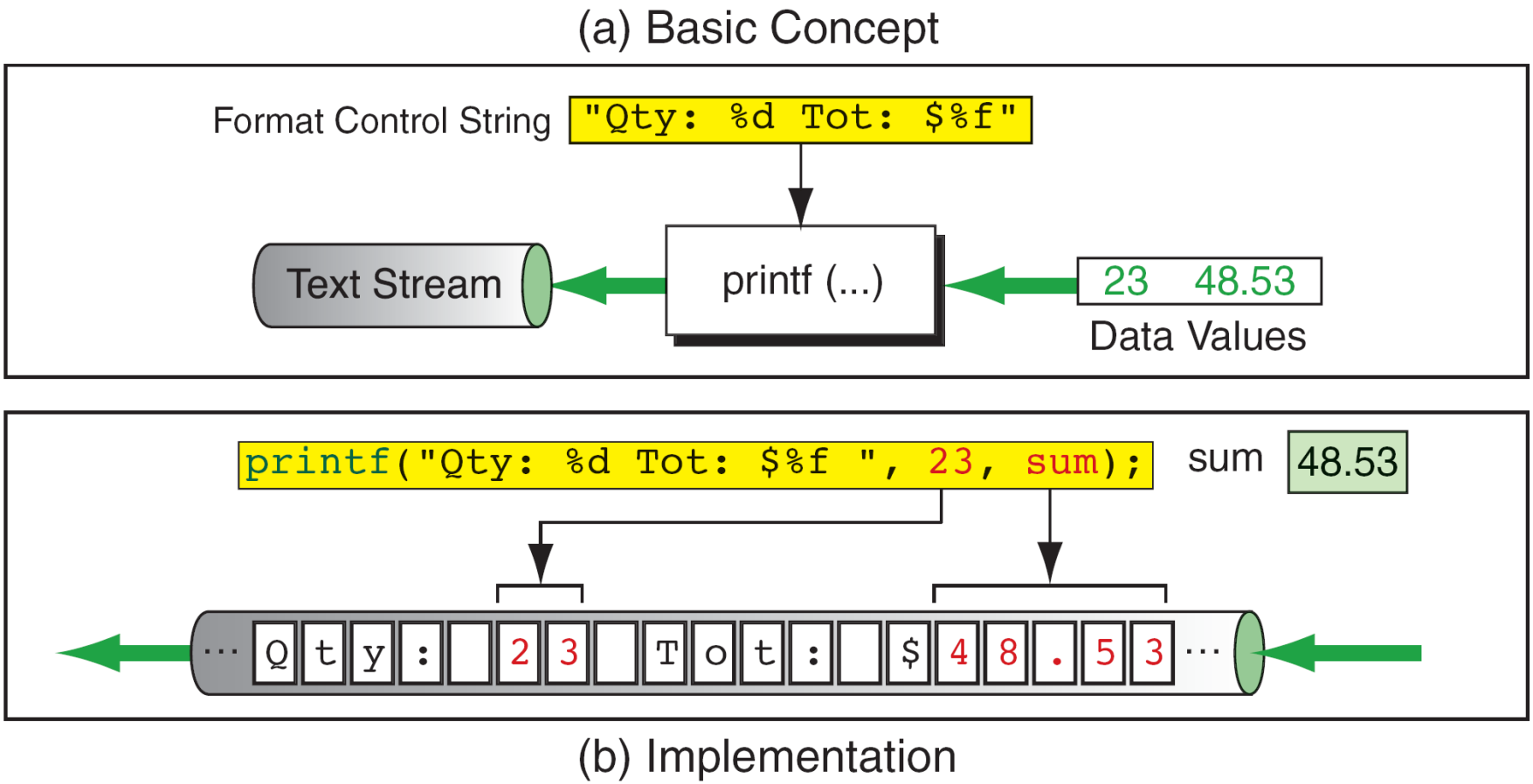

FIGURE 2-17 Output Stream Formatting Example

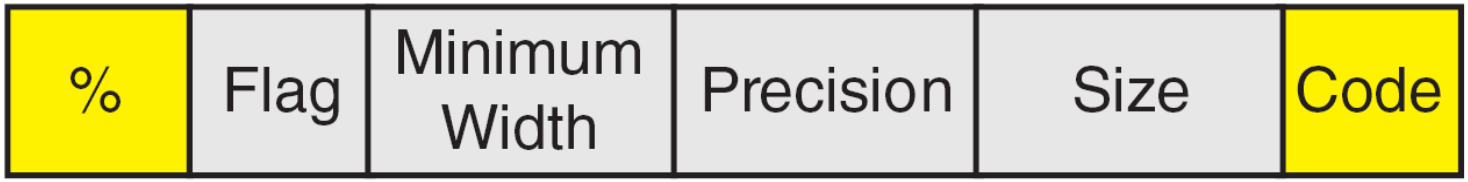

FIGURE 2-18 Conversion Specification

Type	Size[a]	Code	Example
char	None	c	%c
short int	h	d	%hd
int	None	d	%d
long int	None	d	%ld
long long int	ll	d	%lld
float	None	f	%f
double	None	f	%f
long double	L	f	%Lf

a. Size is discussed in the next section.

Table 2-10 Format Codes for Output

Flag Type	Flag Code	Formatting
Justification	None	right justified
	–	left justified
Padding	None	space padding
	0	zero padding
Sign	None	positive value: no sign negative value: –
	+	positive value: + negative value: –
	Space	positive value: space negative value: –

Table 2-11 Flag Formatting Options

Output Examples

1.
```
printf("%d%c%f", 23, 'z', 4.1);
```
```
23z4.100000
```

2.
```
printf("%d %c %f", 23, 'z', 4.1);
```
```
23 z 4.100000
```

3.
```
int num1 = 23;
char zee = 'z';
float num2 = 4.1;
printf("%d %c %f", num1, zee, num2);
```
```
23 z 4.100000
```

Output Examples

4.
```
printf("%d\t%c\t%5.1f\n", 23, 'z', 14.2);
printf("%d\t%c\t%5.1f\n", 107, 'A', 53.6);
printf("%d\t%c\t%5.1f\n", 1754, 'F', 122.0);
printf("%d\t%c\t%5.1f\n", 3, 'P', 0.1);
```

```
23      z        14.2
107     A        53.6
1754    F       122.0
3       P         0.1
```

5.
```
printf("The number%dis my favorite.", 23);
```

```
The number23is my favorite.
```

6.
```
printf("The number is %6d", 23);
```

```
The number is     23
^^^^^^^^^^^^^^^^^^^^
```

Output Examples

7. `printf("The tax is %6.2f this year.", 233.12);`

```
The tax is 233.12 this year.
```

8. `printf("The tax is %8.2f this year.", 233.12);`

```
The tax is   233.12 this year.
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

9. `printf("The tax is %08.2f this year.", 233.12);`

```
The tax is 00233.12 this year.
```

10. `printf("\"%8c   %d\"", 'h', 23);`

```
"       h   23“
^^^^^^^^^^^^^^^
```

Output Examples

11.
```
printf("This line disappears.\r...A new
line\n");
printf("This is the bell character \a\n");
printf("A null character\0kills the rest of the
line\n");
printf("\nThis is \'it\' in single quotes\n");
printf("This is \"it\" in double quotes\n");
printf("This is \\ the escape character it
self\n");
```

```
...A new line
This is the bell character
A null character
This is 'it' in single quotes
This is "it" in double quotes
This is \ the escape character it self
```

Output Examples

12. `printf("|%-+8.2f| |%0+8.2f| | %-0+8.2f|", 1.2, 2.3, 3.4);`

```
|+1.20   | |+0002.30| | +3.40   |
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

Common Output Errors

1.
```
printf("%d %d %d\n", 44, 55);
```
```
44 55 1638280
```

2.
```
printf("%d %d\n", 44, 55, 66);
```
```
44 55
```

3.
```
float x = 123.45;
printf("The data are: %d\n", x);
```
```
The data are: -1073741824
```

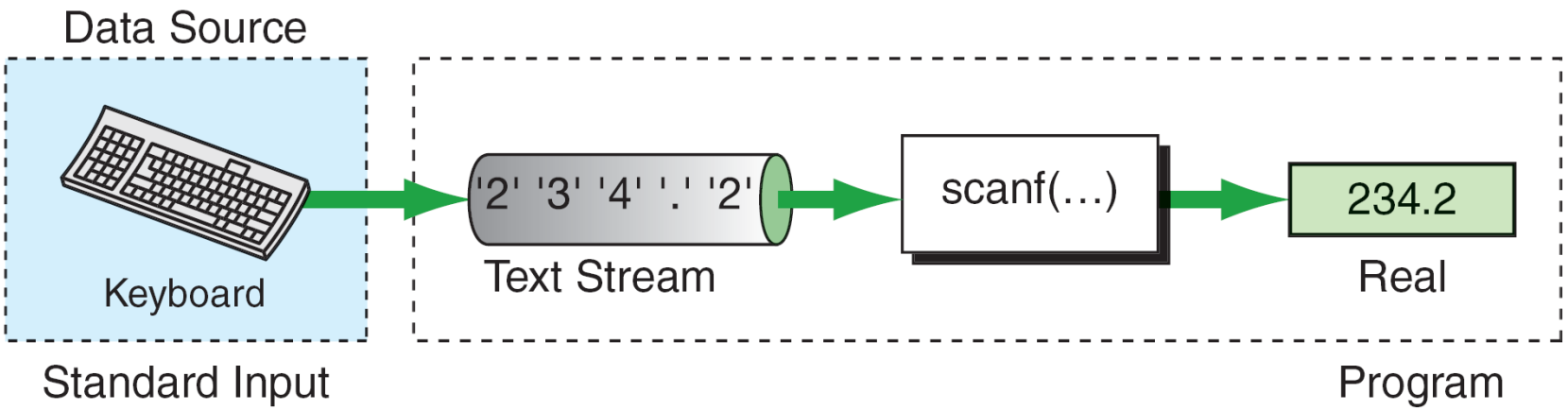

FIGURE 2-19 Formatting Text from an Input Stream

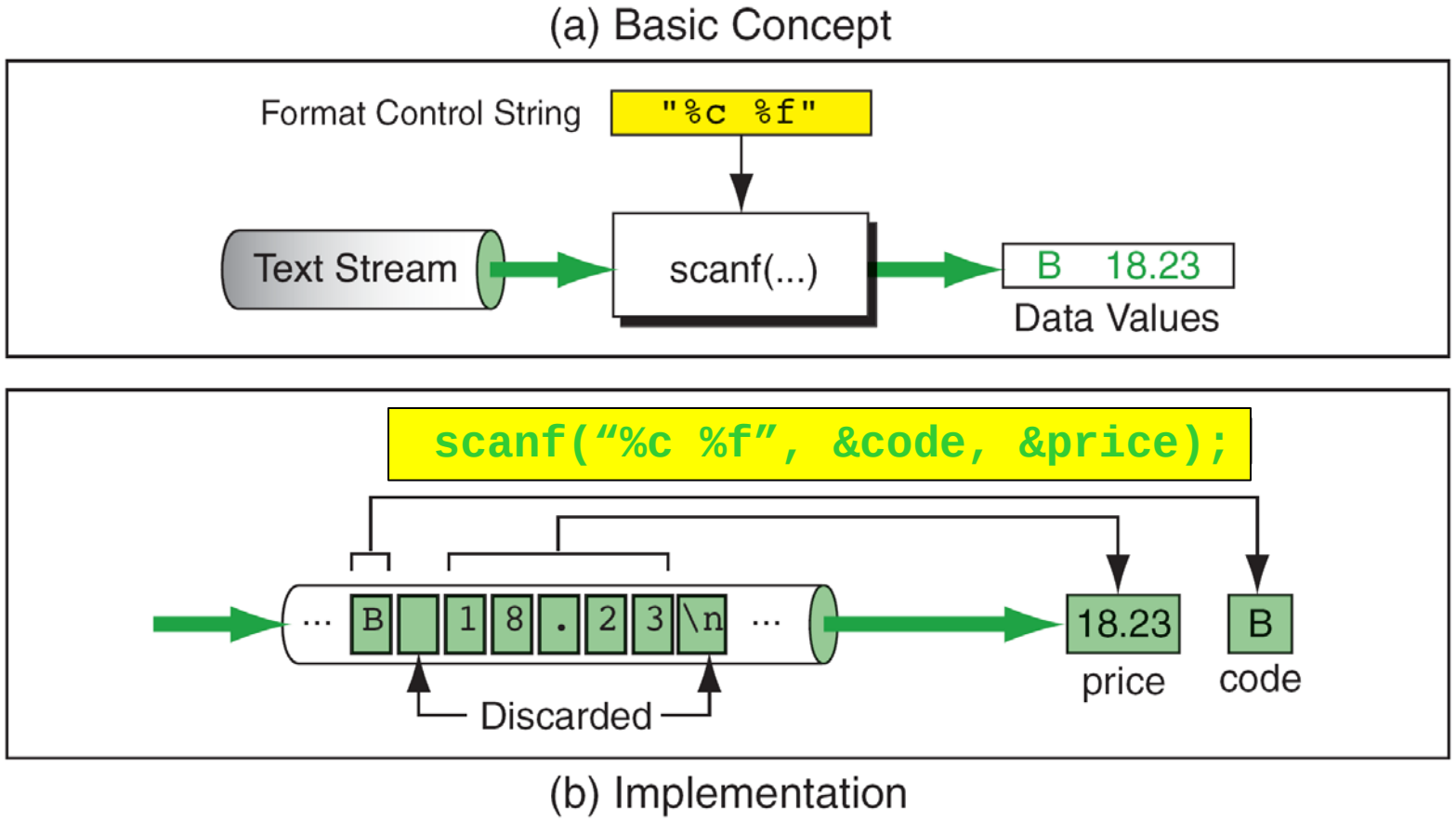

FIGURE 2-20 Input Stream Formatting Example

FIGURE 2-21 Conversion Specification

Note

***scanf* requires variable addresses in the address list.**

1. The conversion operation processes until:

   a. End of file is reached.
   b. The maximum number of characters has been processed.
   c. A whitespace character is found after a digit in a numeric specification.
   d. An error is detected.

2. There must be a conversion specification for each variable to be read.

3. There must be a variable address of the proper type for each conversion specification.

4. Any character in the format string other than whitespace or a conversion specification must be exactly matched by the user during input. If the input stream does not match the character specified, an error is signaled and *scanf* stops.

5. It is a fatal error to end the format string with a whitespace character. Your program will not run correctly if you do.

Table 2-12 *scanf* Rules

Input Examples

1-1. 214 156 14z

```
scanf("%d%d%d%c", &a, &b, &c, &d);
```

1-2. 214 156 14 z

```
scanf("%d%d%d %c", &a, &b, &c, &d);
```

2. 2314 15 2.14

```
scanf("%d %d %f", &a, &b, &c);
```

Input Examples

3. 14/26 25/66

```
scanf("%2d/%2d %2d/%2d", &num1, &den1, &num2,
&den2);
```

4. 11-25-56

```
scanf("%d-%d-%d", &a, &b, &c);
```

Common Input Errors

1.
```
int a=0;
scanf ("%d",      a);
printf("%d\n",    a);
```
```
234                 (Input)
0                   (Output)
```

2.
```
float a = 2.1;
scanf ("%5.2f", &a);
printf("%5.2f",  a);
```
```
74.35                     (Input)
2.10                (Output)
```

Common Input Errors

3.
```
int a;
int b;
scanf ("%d%d%d", &a, &b);
printf("%d %d\n", a,  b);
```
```
5 10                (Input)
5 10                (Output)
```

4.
```
int a = 1;
int b = 2;
int c = 3;
scanf ("%d%d",       &a, &b, &c);
printf("%d %d\n", a,  b,  c);
```
```
5 10 15                   (Input)
5 10 3                    (Output)
```

2-8 Programming Examples

In this section, we show some programming example to emphasize the ideas and concepts we have discussed about input/output.

PROGRAM 2-4 A Program That Prints "Nothing!"

```
 1  /* Prints the message "Nothing!".
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6
 7  int main (void)
 8  {
 9  // Statements
10     printf("This program prints\n\n\t\"Nothing!\"");
11     return 0;
12  }  // main
```

```
Results:
This program prints

    "Nothing!"
```

PROGRAM 2-5 Demonstrate Printing Boolean Constants

```
1  /* Demonstrate printing Boolean constants.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6  #include <stdbool.h>
7
8  int main (void)
9  {
10 // Local Declarations
11    bool x = true;
12    bool y = false;
13
```

PROGRAM 2-5 Demonstrate Printing Boolean Constants (continued)

```
14  // Statements
15     printf ("The Boolean values are: %d %d\n", x, y);
16     return 0;
17  }  // main
```

```
Results:
The Boolean values are: 1 0
```

PROGRAM 2-6 Print Value of Selected Characters

```
 1  /* Display the decimal value of selected characters,
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6
 7  int main (void)
 8  {
 9  // Local Declarations
10     char A          = 'A';
11     char a          = 'a';
12     char B          = 'B';
13     char b          = 'b';
14     char Zed        = 'Z';
15     char zed        = 'z';
```

PROGRAM 2-6 Print Value of Selected Characters (continued)

```
16      char zero      = '0';
17      char eight     = '8';
18      char NL        = '\n';       // newline
19      char HT        = '\t';       // horizontal tab
20      char VT        = '\v';       // vertical tab
21      char SP        = ' ';        // blank or space
22      char BEL       = '\a';       // alert (bell)
23      char dblQuote  = '"';        // double quote
24      char backSlash = '\\';       // backslash itself
25      char oneQuote  = '\'';       // single quote itself
26
27  // Statements
28      printf("ASCII for char 'A'  is: %d\n",  A);
29      printf("ASCII for char 'a'  is: %d\n",  a);
30      printf("ASCII for char 'B'  is: %d\n",  B);
```

PROGRAM 2-6 Print Value of Selected Characters (continued)

```
31      printf("ASCII for char 'b'  is: %d\n",  b);
32      printf("ASCII for char 'Z'  is: %d\n",  Zed);
33      printf("ASCII for char 'z'  is: %d\n",  zed);
34      printf("ASCII for char '0'  is: %d\n",  zero);
35      printf("ASCII for char '8'  is: %d\n",  eight);
36      printf("ASCII for char '\\n' is: %d\n", NL);
37      printf("ASCII for char '\\t' is: %d\n", HT);
38      printf("ASCII for char '\\v' is: %d\n", VT);
39      printf("ASCII for char ' '  is: %d\n",  SP);
40      printf("ASCII for char '\\a' is: %d\n", BEL);
41      printf("ASCII for char '\"'  is: %d\n", dblQuote);
42      printf("ASCII for char '\\'  is: %d\n", backSlash);
43      printf("ASCII for char '\''  is: %d\n", oneQuote);
44
45      return 0;
46  }   // main
```

PROGRAM 2-6 Print Value of Selected Characters (continued)

```
Results:
ASCII for character 'A'  is: 65
ASCII for character 'a'  is: 97
ASCII for character 'B'  is: 66
ASCII for character 'b'  is: 98
ASCII for character 'Z'  is: 90
ASCII for character 'z'  is: 122
ASCII for character '0'  is: 48
ASCII for character '8'  is: 56
ASCII for character '\n' is: 10
ASCII for character '\t' is: 9
ASCII for character '\v' is: 11
ASCII for character ' '  is: 32
ASCII for character '\a' is: 7
ASCII for character '"'  is: 34
ASCII for character '\'  is: 92
ASCII for character '''  is: 39
```

PROGRAM 2-7 Calculate a Circle's Area and Circumference

```
 1  /* This program calculates the area and circumference
 2     of a circle using PI as a defined constant.
 3        Written by:
 4        Date:
 5  */
 6  #include <stdio.h>
 7  #define PI  3.1416
 8
 9  int main (void)
10  {
11  // Local Declarations
12     float circ;
13     float area;
14     float radius;
15
```

PROGRAM 2-7 Calculate a Circle's Area and Circumference (continued)

```
16  // Statements
17     printf("\nPlease enter the value of the radius: ");
18     scanf("%f", &radius);
19
20     circ   = 2   * PI      * radius;
21     area   = PI * radius * radius;
22
23     printf("\nRadius is :         %10.2f", radius);
24     printf("\nCircumference is : %10.2f", circ);
25     printf("\nArea is :           %10.2f", area);
26
27     return 0;
28  }  // main
```

```
Results:
Please enter the value of the radius: 23

Radius is :              23.00
Circumference is :      144.51
Area is :              1661.91
```

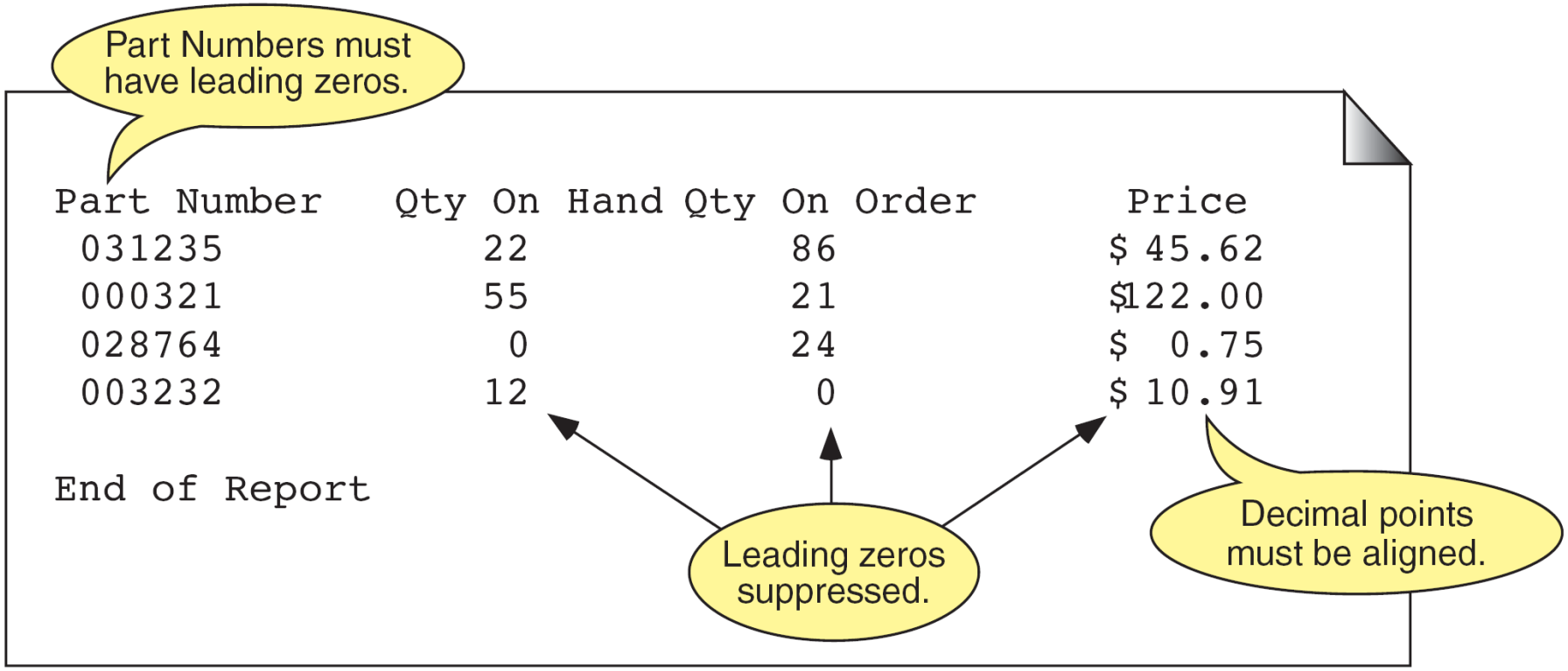

FIGURE 2-22 Output Specifications for Inventory Report

PROGRAM 2-8 A Sample Inventory Report

```
1  /* This program will print four lines of inventory data
2     on an inventory report to give the user an idea of
3     what a new report will look like. Since this is not a
4     real report, no input is required. The data are all
5     specified as constants
6        Written by:
7        Date:
8  */
9  #include <stdio.h>
10
11 int main (void)
12 {
13 // Statements
14    // Print captions
15    printf("\tPart Number\tQty On Hand");
```

PROGRAM 2-8 A Sample Inventory Report (continued)

```
16      printf("\tQty On Order\tPrice\n");
17
18      // Print data
19      printf("\t %06d\t\t%7d\t\t%7d\t\t $%7.2f\n",
20              31235, 22, 86, 45.62);
21      printf("\t %06d\t\t%7d\t\t%7d\t\t $%7.2f\n",
22              321, 55, 21, 122.);
23      printf("\t %06d\t\t%7d\t\t%7d\t\t $%7.2f\n",
24              28764, 0, 24, .75);
25      printf("\t %06d\t\t%7d\t\t%7d\t\t $%7.2f\n",
26              3232, 12, 0, 10.91);
27
28      // Print end message
29      printf("\n\tEnd of Report\n");
30      return 0;
31  }   // main
```

2-9 Software Engineering

Although this chapter introduces only a few programming concepts, there is still much to be said from a software engineering point of view. We will discuss the concepts of program documentation, data naming, and data hiding.

Topics discussed in this section:

Program Documentation
Data Names
Data Hiding

PROGRAM 2-11 Sample of General Program Documentation

```
1  /* A sample of program documentation. Each program
2     starts with a general description of the program.
3     Often, this description can be taken from the
4     requirements specification given to the programmer.
5       Written by: original author
6       Date:       Date first released to production
7     Change History:
8       <date> Included in this documentation is a short
9              description of each change.
10 */
```

Good Names	Poor Names
ficaTaxRate	rate ftr frate fica
fica_tax_rate	
ficaWitholding	fwh ficaw wh
fica_witholding	
ficaWthldng	fcwthldng wthldng
fica_wthldng	
ficaMax ficaDlrMax	max fmax

Table 2-13 Examples of Good and Poor Data Names

#define SPACE ' '	#define BANG '!'
#define DBL_QTE '"'	#define QUOTE '\"'
#define COMMA ','	#define COLON ':'

Table 2-14 Examples of Defined Constants

Note

Programming Standard
No variables are to be placed in the global area of a program.