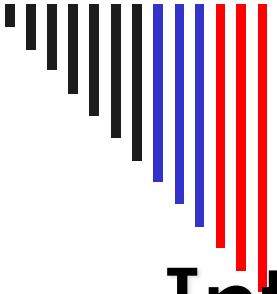

Chapter 2
Introduction to the C Language

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.

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. That is another reason why it is used by software developers whose applications have to run on many different hardware platforms.

2-2 C Programs

It's time to write your first C program.

Topics discussed in this section:

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

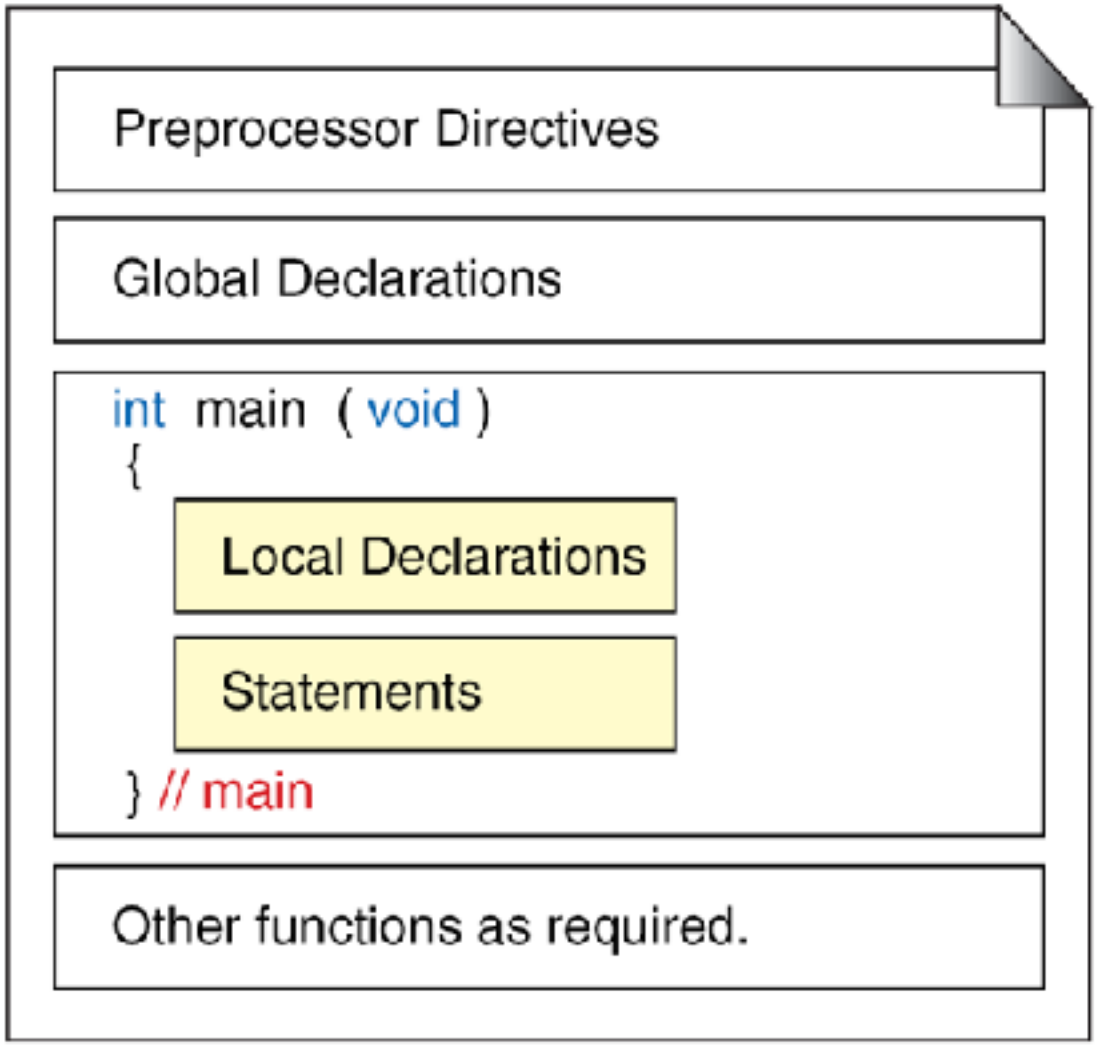

FIGURE 2-2 Structure of a C Program

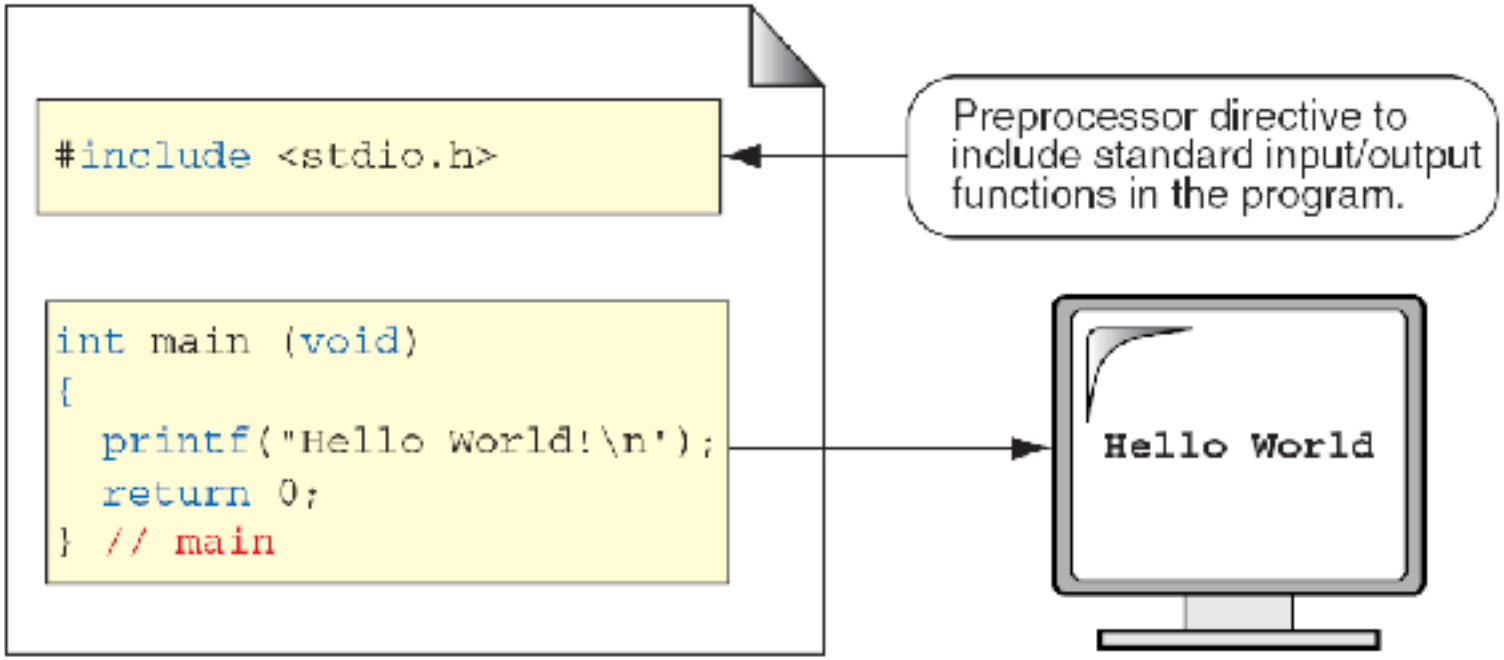

FIGURE 2-3 The Greeting Program

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
```

```
/* 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

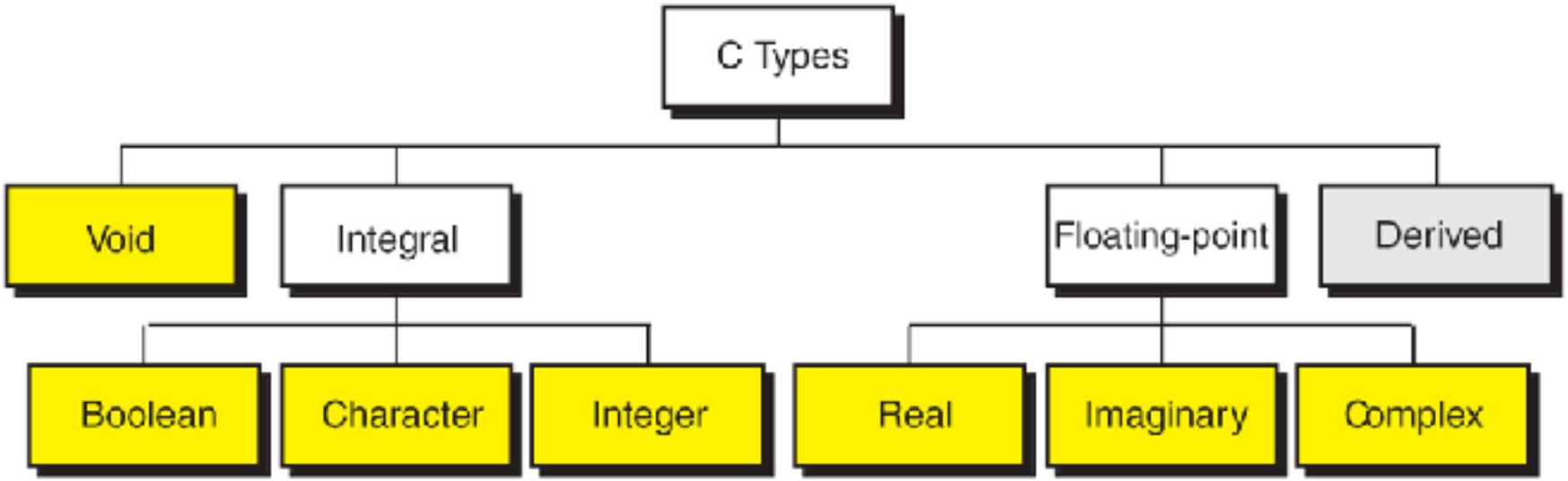

FIGURE 2-6 Nested Block Comments Are Invalid

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.

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.

Topics discussed in this section:

Void Type
Integral Type
Floating-Point Types

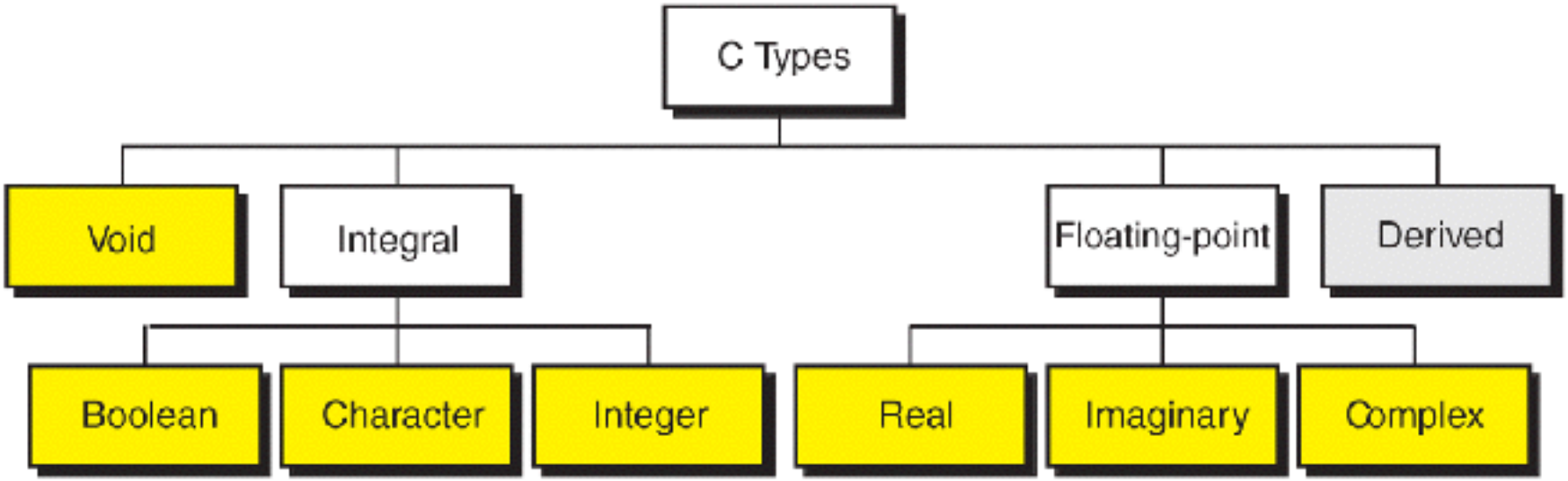

FIGURE 2-7 Data Types

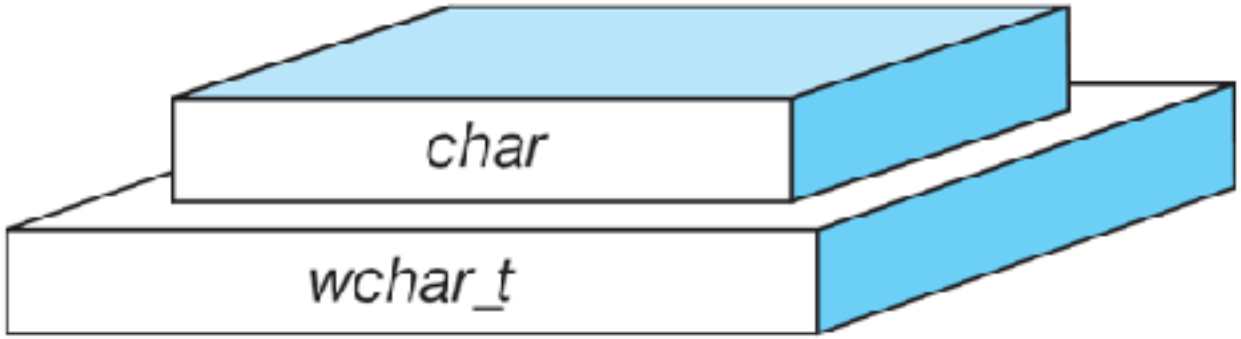

FIGURE 2-8 Character Types

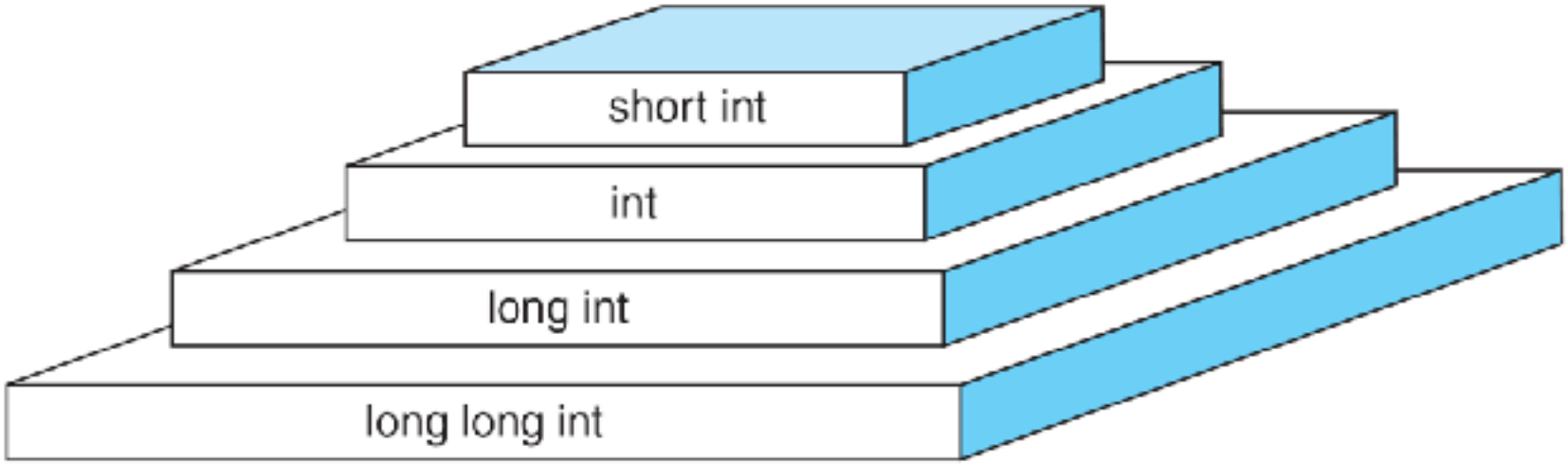

FIGURE 2-9 Integer Types

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

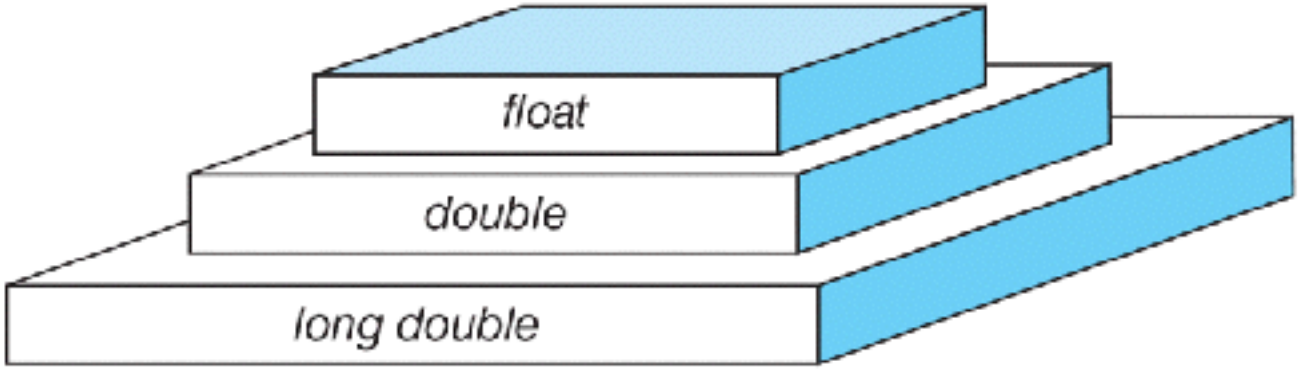

FIGURE 2-10 Floating-point Types

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

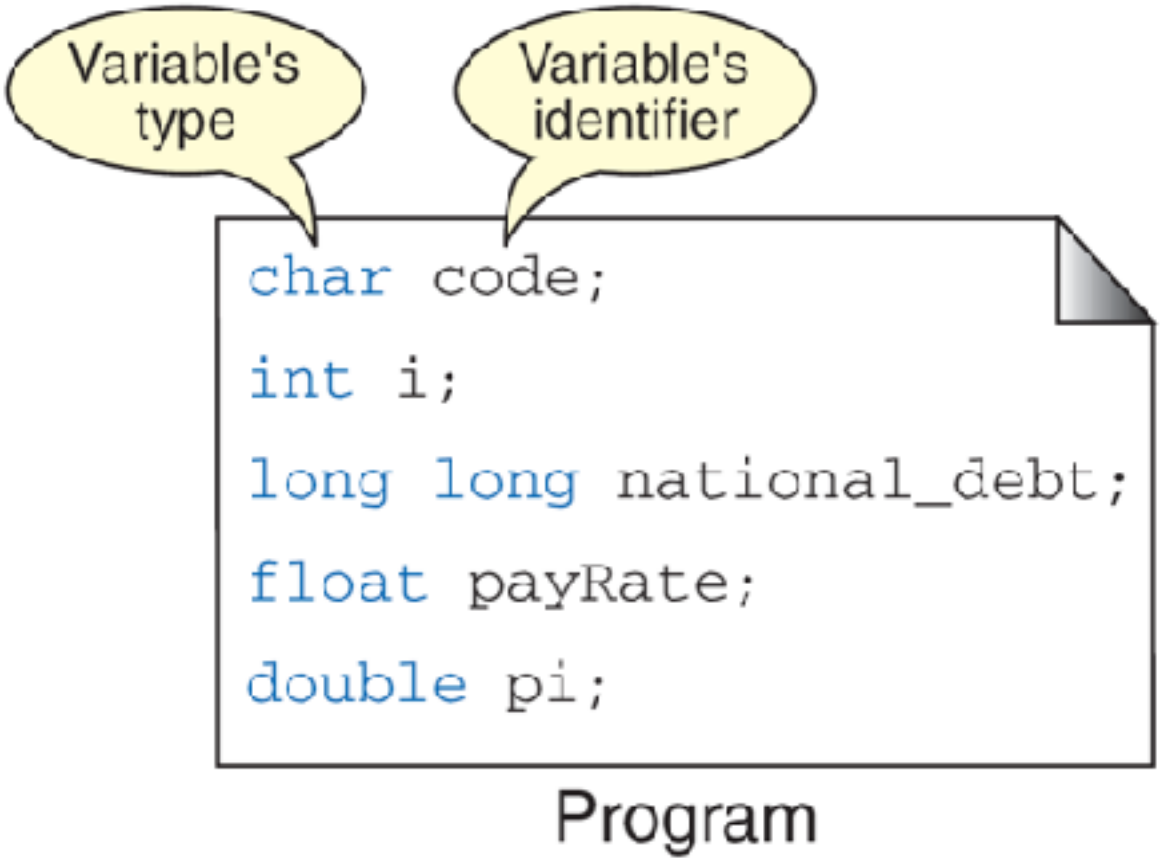

FIGURE 2-11 Variables

```
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

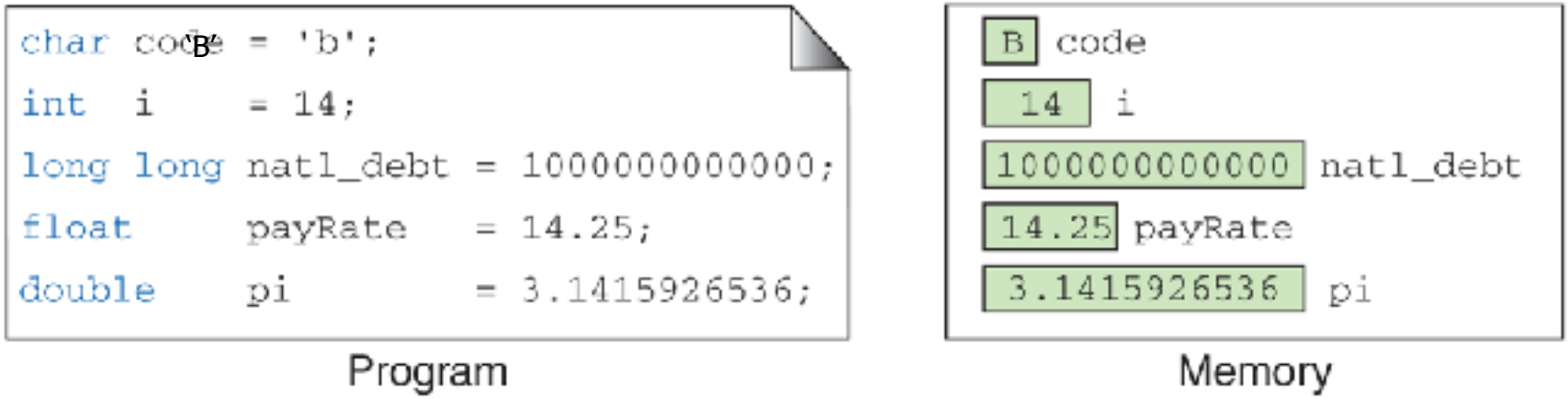

FIGURE 2-12 Variable Initialization

QUESTION

- Write a function in C that is used to display the values of identifiers:
  - Uninitialized values
  - After initialised values
    - Identifiers to check:
      - Int
      - float
      - Char
      - Long
      - Double

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

DISCUSSION ON THE INT DATA TYPE

- Being a signed data type, it can store positive values as well as negative values.
- Takes a size of 32 bits where 1 bit is used to store the sign of the integer.
- A maximum integer value that can be stored in an int data type is typically 2,147,483,647, around $2^{31} - 1$, but is compiler dependent.
- A minimum integer value that can be stored in an int data type is typically -2,147,483,648, around -2^{31}, but is compiler dependent.
- In case of overflow or underflow of data type, the value is wrapped around. For example, if -2,147,483,648 is stored in an int data type and 1 is subtracted from it, the value in that variable will become equal to 2,147,483,647. Similarly, in the case of overflow, the value will round back to -2,147,483,648.

```
""                                                    // 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 Memory 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", PI);
```

PROGRAM 2-3 Memory 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
```