

Chapter 3. Data and C

Chapter Outline

- Keywords: *int, short, long, unsigned, char, float, double, _Bool, _Complex, _Imaginary*
- Operator: *sizeof*
- Function: *scanf()*
- The basic data types that C uses
- The distinctions between integer types and floating-point types
- Writing constants and declaring variables of those types
- How to use the *printf()* and *scanf()* functions to read and write values of different types

Why We Discuss Data?

Why We Discuss Data?

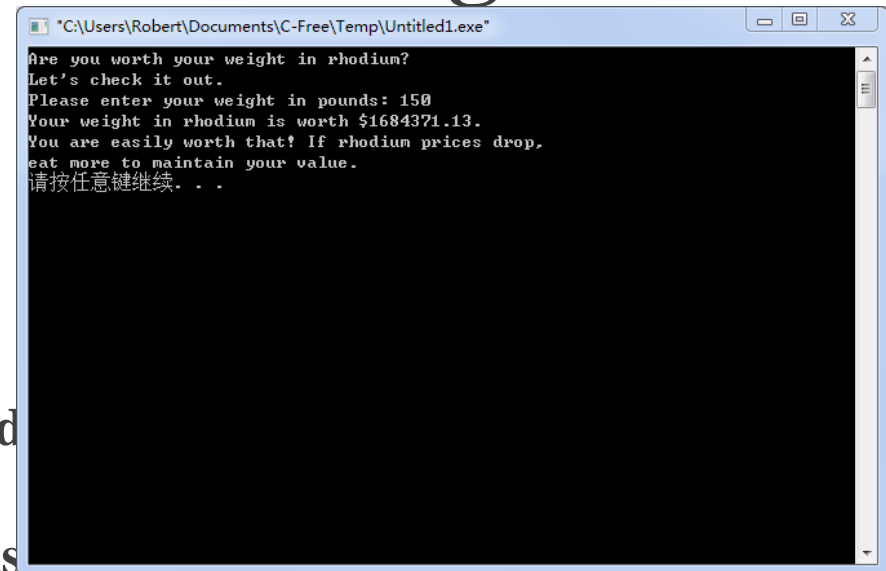
- Programs work with **data**
 - You feed numbers, letters, and words to the computer, and you expect it to do something with the data.
- You **do more than just read about data**
- You practice **manipulating data**, which is much more fun

What We Got In this Chapter

- Explore the two great families of data types: **integer** and **floating point**.
- C offers **several varieties of these types**.
- Tell you **what the types are, how to declare them, and how and when to use them**.
- Discover **the differences between constants and variables**
- Your first **interactive** program is coming up

Listing 3.1. The rhodium.c Program

```
/* rhodium.c -- your weight in rhodium */
#include <stdio.h>
int main(void)
{
    float weight; /* user weight */
    float value; /* rhodium equivalent */
    printf("Are you worth your weight in rhodium?\n");
    printf("Let's check it out.\n");
    printf("Please enter your weight in pounds: ");
    /* get input from the user */
    scanf("%f", &weight);
    /* assume rhodium is $770 per ounce */
    /* 14.5833 converts pounds avd. to ounces troy */
    value = 770.0 * weight * 14.5833;
    printf("Your weight in rhodium is worth $%.2f.\n", value);
    printf("You are easily worth that! If rhodium prices drop,\n");
    printf("eat more to maintain your value.\n");
    return 0;
}
```



```
"C:\Users\Robert\Documents\C-Free\Temp\Untitled1.exe"
Are you worth your weight in rhodium?
Let's check it out.
Please enter your weight in pounds: 150
Your weight in rhodium is worth $1684371.13.
You are easily worth that! If rhodium prices drop,
eat more to maintain your value.
请按任意键继续...
```

Errors and Warnings

- If you type this program incorrectly, the compiler gives you a syntax error message
- Even if you type it correctly, the compiler may give you a warning
- An **error message** means you did something wrong and prevents the program from being compiled.
- A **warning**, however, means you've done something that is valid code but possibly is not what you meant to do.
- **A warning does not stop compilation.**

Errors and Warnings

- Note that "entering" your weight means to type your weight and then **press the Enter or Return key**
- Pressing Enter informs the computer that you have finished typing your response.
- The program expects you to **enter a number**, such as *150*, **not words**, such as *too much*.
- Entering letters rather than digits causes problems that require an *if* statement to defeat, so please be polite and enter a number.

The Output of Listing 3.1

Are you worth your weight in rhodium?

Let's check it out.

Please enter your weight in pounds: 150

Your weight in rhodium is worth \$1684371.12.

**You are easily worth that! If rhodium prices drop,
eat more to maintain your value.**

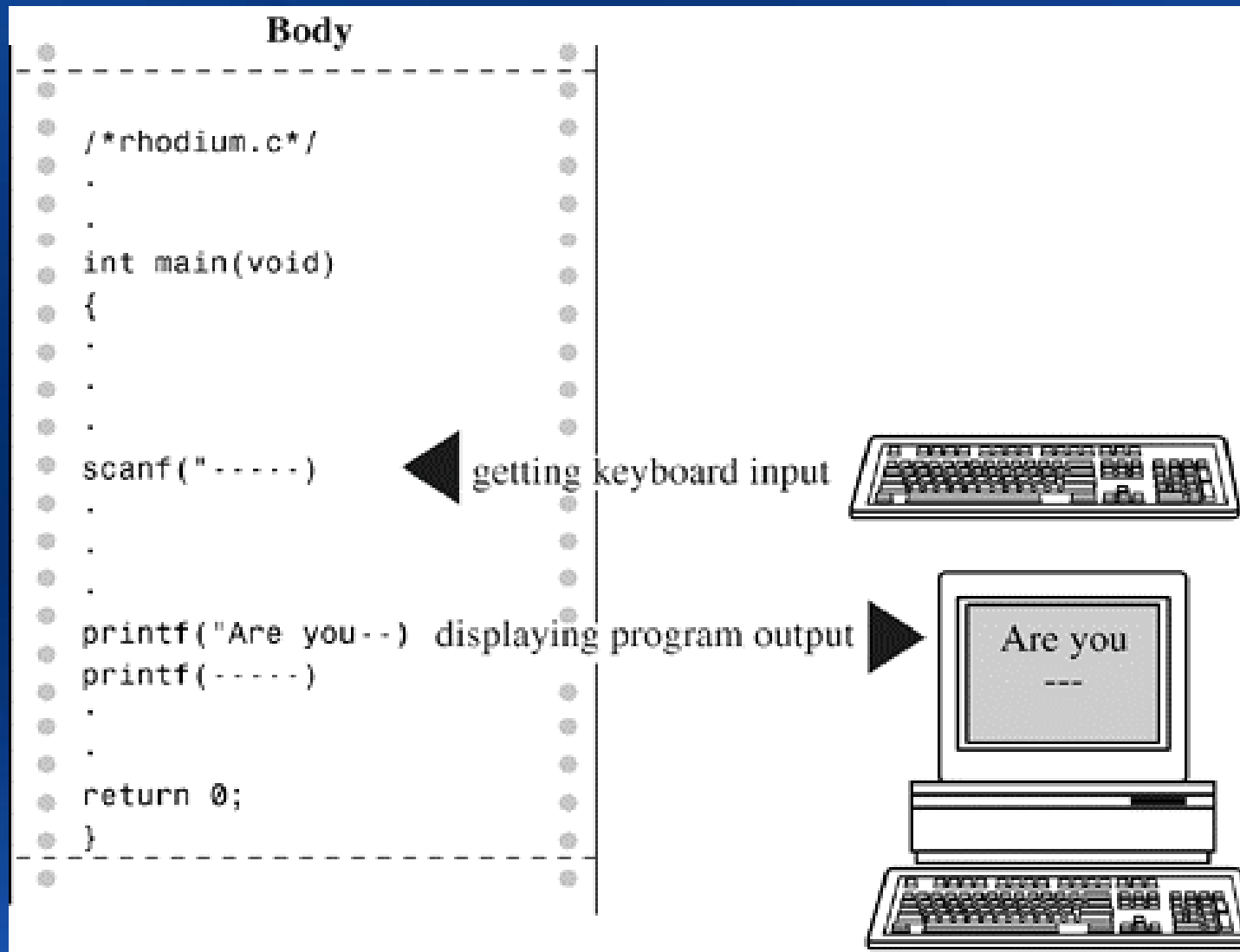
What's New in This Program?

- There are several **new elements of C** in this program:
 - Notice that the code uses a new kind of variable declaration: add **a floating-point variable type (float)** so that you can handle a wider variety of data. The float type can **hold numbers with decimal points**.
 - The program demonstrates some new ways of writing constants: **have numbers with decimal points**.
 - To print this new kind of variable, use the **%f** specifier in the printf() code to handle a floating-point value (Use the **.2 modifier** to the %f specifier to fine-tune the appearance of the output so that it **displays two places to the right of the decimal**.)

What's New in This Program?

- To **provide keyboard input to the program**, use the **scanf()** function: The **%f** instructs **scanf()** to **read a floating-point number from the keyboard**, and the **&weight** tells **scanf()** to **assign the input value to the variable named weight**. The **scanf()** function uses the **&** notation to **indicate where it can find the weight variable**.
- Perhaps the most outstanding new feature is that this program is **interactive: The interactive approach makes programs more flexible**. The **scanf()** and **printf()** functions make this interactivity possible. The **scanf()** function reads data from the keyboard and delivers that data to the program, and **printf()** reads data from a program and delivers that data to your screen. **Together, these two functions enable you to establish a two-way communication with your computer**

Figure 3.1. The scanf() and printf() functions at work.



Data Variables and Constants

- A computer, under the guidance of a program, can do many things.
- To do these tasks, the program needs to work with **data**, the numbers and characters that bear the information you use.
- Some types of data are preset before a program is used and keep their values unchanged throughout the life of the program. These are **constants**.
- Other types of data may change or be assigned values as the program runs; these are **variables**.

Listing 3.1. The rhodium.c Program

```
/* rhodium.c -- your weight in rhodium */
#include <stdio.h>
int main(void)
{
    float weight; /* user weight */
    float value; /* rhodium equivalent */
    printf("Are you worth your weight in rhodium?\n");
    printf("Let's check it out.\n");
    printf("Please enter your weight in pounds: ");
    /* get input from the user */
    scanf("%f", &weight);
    /* assume rhodium is $770 per ounce */
    /* 14.5833 converts pounds avd. to ounces troy */
    value = 770.0 * weight * 14.5833;
    printf("Your weight in rhodium is worth $%.2f.\n", value);
}
```

The price of rhodium isn't a constant in real life, but this program treats it as a constant.

weight is a variable

14.5833 is a constant

The difference between a variable and a constant is that a variable can have its value assigned or changed while the program is running, and a constant can't.

Data: Data-Type Keywords

- Beyond the distinction between variable and constant is **the distinction between different types of data.**
- Some types of data are numbers.
- Some are letters or, more generally, characters.
- The computer **needs a way to identify** and **use these different kinds.**
- C does this by **recognizing several fundamental data types.**

Data: Data-Type Keywords

- If a datum is a constant, **the compiler can usually tell its type just by the way it looks**: 42 is an integer, and 42.100 is floating point.
- A variable, however, needs to **have its type announced in a declaration statement**.

Data: Data-Type Keywords

Table 3.1. C Data Keywords

Original K&R Keywords	C90 Keywords	C99 Keywords
int	signed	_Bool
long	void	_Complex
short		_Imaginary
unsigned		
char		
float		
double		

Data: Data-Type Keywords

- The **int** keyword provides **the basic class of integers**.
- The next three keywords (**long**, **short**, and **unsigned**) and the ANSI addition **signed** are used to provide variations of the basic type.
- The **char** keyword designates the type used for **letters of the alphabet and for other characters, such as #, \$, %, and ***. Also can be used to **represent small integers**.
- **float**, **double**, and the **combination long double** are used **to represent numbers with decimal points**.

The types created with these keywords can be divided into two families on the basis of how they are stored in the computer: **integer types** and **floating-point types**.

Bits, Bytes, and Words – Bits

- The terms **bit**, **byte**, and **word** can be used to **describe units of computer data** or **to describe units of computer memory**.
- **The smallest unit of memory** is called **a bit**. It can hold one of two values: **0** or **1**. (Or you can say that the bit is set to "**off**" or "**on**.")
- You can't store much information in one bit, but a computer has a tremendous stock of them.
- The bit is the basic building block of computer memory.

Bits, Bytes, and Words – Bytes

- The **byte** is **the usual unit of computer memory**.
- For nearly all machines, **a byte is 8 bits**, and that is the standard definition, at least **when used to measure storage**.
- Because each bit can be either 0 or 1, there are **256** (that's 2 times itself 8 times) **possible bit patterns of 0s and 1s** that can fit in an 8-bit byte.
- These patterns can be used, for example, **to represent the integers from 0 to 255** or **to represent a set of characters**.
- **Representation can be accomplished with binary code**, which uses (conveniently enough) just 0s and 1s to represent numbers.

Bits, Bytes, and Words – Words

- A **word** is the natural unit of memory for a given computer design.
- For **8-bit microcomputers**, such as the original Apples, a word is just 8 bits.
- Early IBM compatibles using the 80286 processor are **16-bit machines**, meaning that they have a word size of 16 bits
- Machines such as the Pentium-based PCs and the Macintosh PowerPCs have **32-bit words**.
- More powerful computers can have **64-bit words or even larger**.

Integer Versus Floating-Point Types

- Integer types? Floating-point types?
- Do you have to learn all the details? Not really, not any more than you have to learn the principles of internal combustion engines to drive a car, but knowing a little about what goes on inside a computer or engine can help you occasionally.
- **For a human**, the difference between integers and floating-point numbers is reflected **in the way they can be written**.
- **For a computer**, the difference is reflected **in the way they are stored**.

Integer

- An integer is a number **with no fractional part**.
- In C, an integer **is never written with a decimal point**.
- Integers **are stored as binary numbers**.

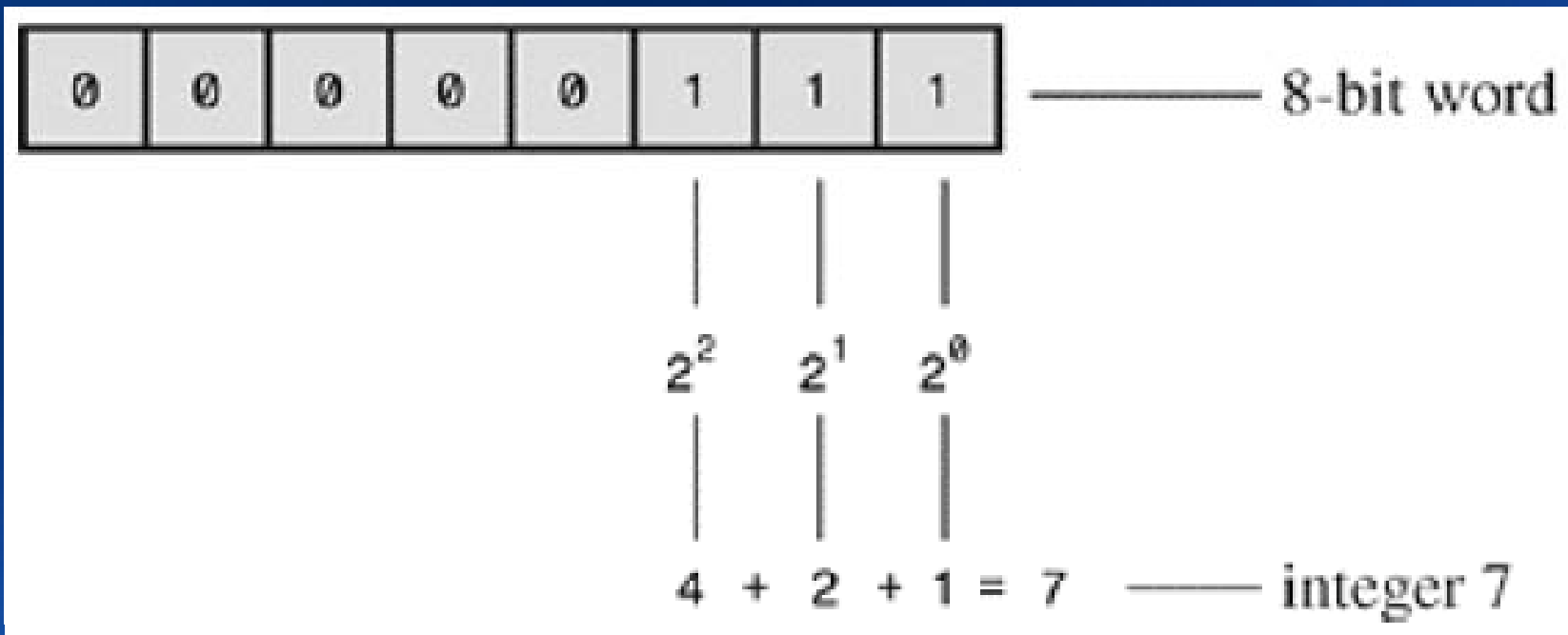


Figure 3.2. Storing the integer 7 using a binary code.

The Floating-Point Number

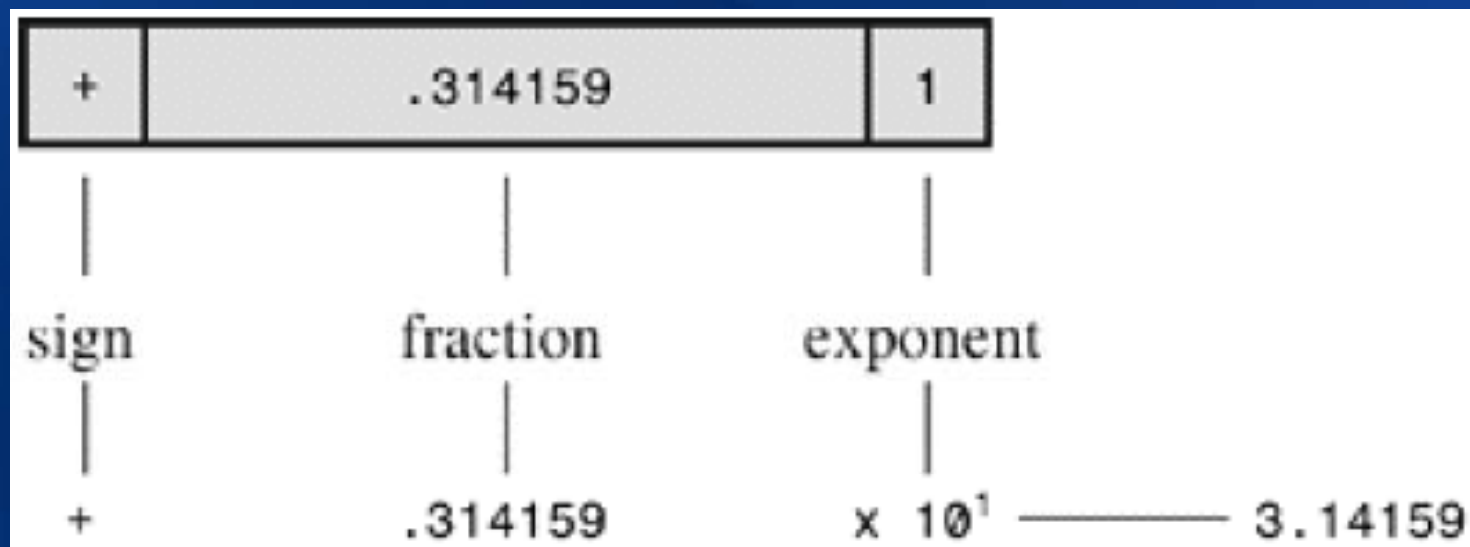
- A floating-point number more or less corresponds to what mathematicians call **a real number**.
- Real numbers include the numbers between the integers.
- Some floating-point numbers: 2.75, 3.16E7, 7.00, 2e-8.
- Notice that **adding a decimal point makes a value a floating-point value** (So 7 is an integer type but 7.00 is a floating-point type).
- Obviously, there is **more than one way** to write a floating-point number.
- The e-notation: the notation **3.16E7** means to **multiply 3.16 by 10 to the 7th power**; that is, by 1 followed by 7 zeros (The 7 would be termed the exponent of 10).

The Floating-Point Number

- The key point here is that **the scheme used to store a floating-point number is different from the one used to store an integer.**
- Floating-point representation involves **breaking up a number into a fractional part and an exponent part and storing the parts separately.**
- Therefore, the **7.00** in this list would not be stored in the same manner as the integer 7, even though both have the same value. The decimal analogy would be to write 7.0 as 0.7E1. Here, **0.7 is the fractional part**, and **the 1 is the exponent part.**

The Floating-Point Number

- Figure 3.3. Storing the number pi in floating-point format (decimal version)



A computer would use binary numbers and powers of two instead of powers of 10 for internal storage.

The Floating-Point Number

- Let's concentrate on the practical differences:
 - An integer has no fractional part; a floating-point number can have a fractional part.
 - Floating-point numbers can represent a much larger range of values than integers can.
 - For some arithmetic operations, such as subtracting one large number from another, **floating-point numbers are subject to greater loss of precision.**
 - Because there is an infinite number of real numbers in any range—for example, in the range between 1.0 and 2.0—**computer floating-point numbers can't represent all the values in the range.** Instead, **floating-point values are often approximations of a true value.** For example, 7.0 might be stored as a 6.99999 float value—more about precision later.

The Floating-Point Number

- Floating-point operations are **normally slower** than integer operations. However, microprocessors developed specifically to handle floating-point operations are now available, and they have closed the gap.

Basic C Data Types

- Now let's look at the specifics of the basic data types used by C
- For each type, we describe
 - **how to declare a variable**
 - **how to represent a constant**
 - **what a typical use would be.**

The int Type

- C offers many integer types, and you might **wonder why one type isn't enough.**
- The answer is that **C gives the programmer the option of matching a type to a particular use.**
- In particular, the C integer types **vary in the range of values offered** and **in whether negative numbers can be used.**
- The int type is the basic choice, but should you need other choices to meet the requirements of a particular task or machine, they are available.

The int Type

- The int type is **a signed integer**: means it must be an integer and it can be positive, negative, or zero.
- The range in possible values depends on the computer system: **Typically, an int uses one machine word for storage.**
- ISO/ANSI C specifies that **the minimum range** for type int should be from -32767 to 32767 .
- Typically, systems represent **signed integers by using the value of a particular bit to indicate the sign.**

Declaring an int Variable

- First comes **int**, and then the chosen name of the variable, and then a semicolon.
- To declare more than one variable, you can declare each variable separately, or you can follow the int with a list of names in which each name is separated from the next by a comma.

- The following are valid declarations:

int erns;

int hogs, cows, goats;

- The effect is the same: **Associate names** and **arrange storage space** for four int-sized variables.

Declaring an int Variable

- These declarations create variables but don't supply values for them.
- **How do variables get values?**
- You've seen two ways that they can pick up values in the program. First, there is **assignment**:
cows = 112;
- Second, a variable can pick up a value from a function—from **scanf()**, for example.

Initializing a Variable

- To **initialize** a variable means to assign it a starting, or initial, value.
- In C, this can be done as part of the declaration.
- Just follow the variable name with the assignment operator (=) and the value you want the variable to have.

int hogs = 21;

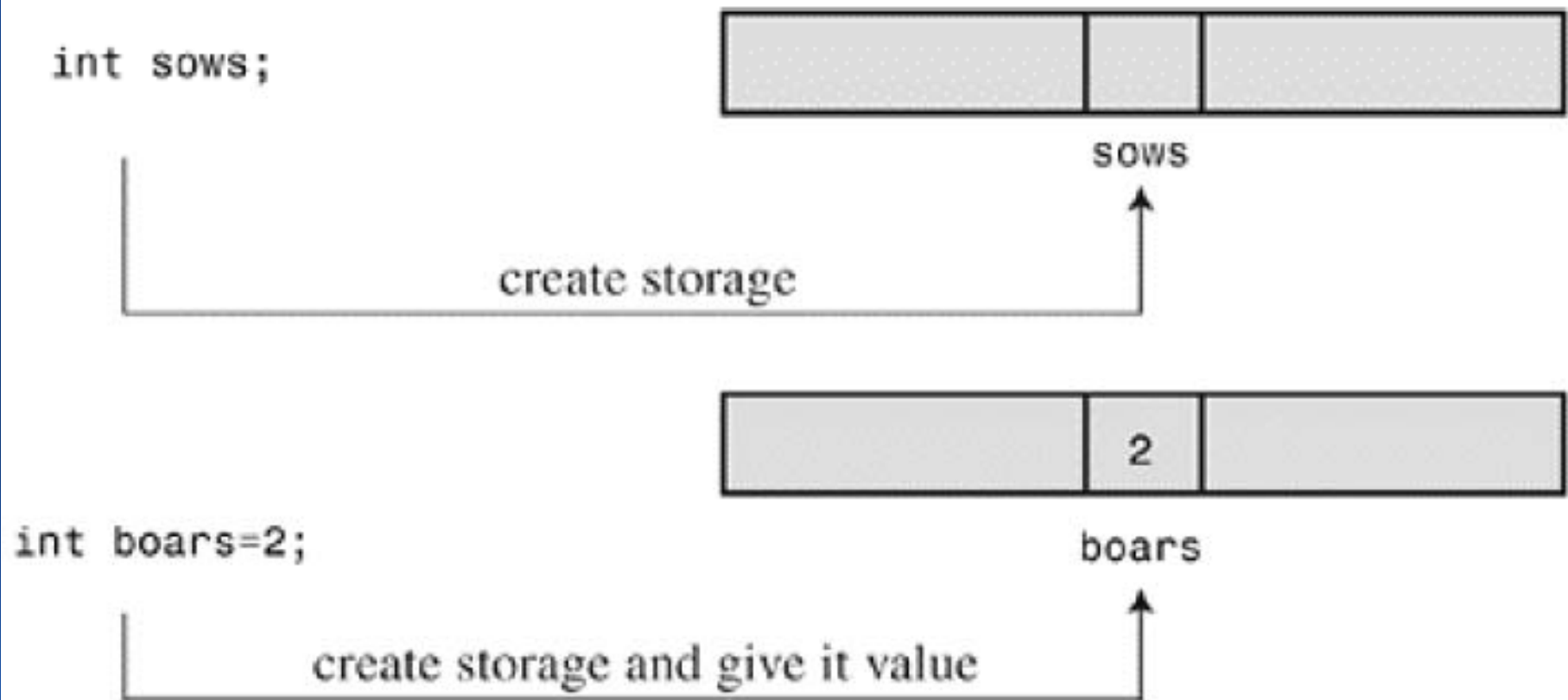
int cows = 32, goats = 14;

int dogs, cats = 94; / valid, but poor, form */*

In the last line, only `cats` is initialized.

Initializing a Variable

Figure 3.4. Defining and initializing a variable.



Type int Constants

- The various integers (21, 32, 14, and 94) in the last example are **integer constants**.
- When you write a number **without a decimal point and without an exponent**, C recognizes it as an integer.
- Therefore, 22 and -44 are integer constants, but 22.0 and 2.2E1 are not.
- C treats most integer constants as type **int**.
- **Very large integers** can be treated differently (see the long int type)

Printing int Values

- Use the printf() function to print **int** types
- The **%d** notation is used to indicate just where in a line the integer is to be printed.
- Each %d in the format string must be matched by a corresponding int value in the list of items to be printed.
- That value can be an int variable, an int constant, or any other expression having an int value.
- It's your job to make sure the number of format specifiers matches the number of values; the compiler won't catch mistakes of that kind.

Listing 3.2. The print1.c Program

```
/* print1.c-displays some properties of printf() */
#include <stdio.h>
int main(void)
{
    int ten = 10;
    int two = 2;
    printf("Doing it right: ");
    printf("%d minus %d is %d\n", ten, 2, ten - two );
    printf("Doing it wrong: ");
    printf("%d minus %d is %d\n", ten ); // forgot 2 arguments
    return 0;
}
```

Doing it right: 10 minus 2 is 8

Doing it wrong: 10 minus 10 is 2

Printing int Values

- Blame the unusual design of `printf()`.
- Most functions take a specific number of arguments, and the compiler can check to see whether you've used the correct number.
- However, `printf()` can have one, two, three, or more arguments, and that keeps the compiler from using its usual methods for error checking.
- Remember, check to see that the number of format specifiers you give to `printf()` matches the number of values to be displayed.

Octal and Hexadecimal

- Normally, C assumes that integer constants are decimal, or base 10, numbers.
- However, **octal (base 8)** and **hexadecimal (base 16)** numbers are popular with many programmers.
- Because **8 and 16 are powers of 2**, and 10 is not, these number systems occasionally offer a more convenient way for expressing computer-related values.
- For example, the number 65536, which often pops up in 16-bit machines, is just 10000 in hexadecimal.

Octal and Hexadecimal

- Each digit in a hexadecimal number **corresponds to exactly 4 bits**.
- For example, the hexadecimal digit 3 is 0011 and the hexadecimal digit 5 is 0101. So the hexadecimal value 35 is the bit pattern 0011 0101, and the hexadecimal value 53 is 0101 0011.
- This correspondence makes it easy to go back and forth between hexadecimal and binary (base 2) notation.

Octal and Hexadecimal

- But how can the computer tell whether 10000 is meant to be a decimal, hexadecimal, or octal value?
- In C, **special prefixes** indicate which number base you are using.
- A prefix of 0x or 0X (zero-ex) means that you are specifying a hexadecimal value, so 16 is written as 0x10, or 0X10, in hexadecimal.
- Similarly, a 0 (zero) prefix means that you are writing in octal.
- For example, the decimal value 16 is written as 020 in

Be aware that this option of using different number systems is provided as a service for your convenience. It doesn't affect how the number is stored.

Displaying Octal and Hexadecimal

- Just as C enables you write a number in any one of three number systems, it also enables you to display a number in any of these three systems.
- To display an integer in octal notation instead of decimal, use %o instead of %d.
- To display an integer in hexadecimal, use %x.
- If you want to display the C prefixes, you can use specifiers %#o, %#x, and %#X to generate the 0, 0x, and 0X prefixes, respectively.

Listing 3.3. The bases.c Program

```
/* bases.c--prints 100 in decimal, octal, and hex */
#include <stdio.h>
int main(void)
{
    int x = 100;
    printf("dec = %d; octal = %o; hex = %x\n", x, x, x);
    printf("dec = %d; octal = %#o; hex = %#x\n", x, x, x);
    return 0;
}
```

dec = 100; octal = 144; hex = 64
dec = 100; octal = 0144; hex = 0x64

Note that the 0 and the 0x prefixes are not displayed in the output unless you include the # as part of the specifier.

Integer Overflow

- What happens if an integer tries **to get too big** for its type?
- Let's set an integer to its largest possible value, add to it, and see what happens.
- Try both signed and unsigned types. (The printf() function uses the **%u** specifier to display unsigned int values.)

Listing 3.3. The bases.c Program

```
/* toobig.c-exceeds maximum int size on our system */
#include <stdio.h>
int main(void)
{
    int i = 2147483647;
    unsigned int j = 4294967295;
    printf("%d %d %d\n", i, i+1, i+2);
    printf("%u %u %u\n", j, j+1, j+2);
    return 0;
}
```

2147483647 -2147483648 -2147483647
4294967295 0 1

Integer Overflow

- The unsigned integer *j* is acting like a car's odometer.
- **When it reaches its maximum value, it starts over at the beginning.**
- The integer *i* acts similarly. The main difference is that the unsigned int variable *j*, like an odometer, begins at 0, but the int variable *i* begins at -2147483648 .
- Notice that you are not informed that *i* has exceeded (overflowed) its maximum value. You would have to include your own programming to keep tabs on that.

Using Characters: Type char

- The char type is used for storing characters such as letters and punctuation marks, but **technically it is an integer type**.
- Why? Because the char type **actually stores integers**, not characters.
- To handle characters, the computer **uses a numerical code** in which certain integers represent certain characters.
- The most commonly used code in the U.S. is **the ASCII code**

Using Characters: Type char

- For example, the integer value 65 represents an uppercase A. So to store the letter A, you actually need to store the integer 65.
- The standard ASCII code runs numerically from 0 to 127.
- This range is small enough that 7 bits can hold it.
- The char type is typically defined as an 8-bit unit of memory, so it is more than large enough to encompass the standard ASCII code.
- More generally, C guarantees that **the char type is large enough to store the basic character set for the system on which C is implemented.**

Using Characters: Type char

- Many character sets have many more than 127 or even 255 values.
- A platform that uses one of these sets as its basic character set could use a 16-bit or even a 32-bit char representation.
- The C language defines **a byte** to be the number of bits used by type char, so as far as C documentation goes, a byte would be 16 or 32 bits, rather than 8 bits on such systems.

Declaring Type char Variables

- As you might expect, char variables are declared in the same manner as other variables.

char response;

char itable, latan;

Character Constants and Initialization

- Suppose you want to initialize a character constant to the letter A.
- Computer languages are supposed to make things easy, so you shouldn't have to memorize the ASCII code, and you don't.

char grade = 'A';

Character Constants and Initialization

- A single letter contained between single quotes is a C character constant.
- When the compiler sees 'A', it converts the 'A' to the proper code value.
- The single quotes are essential.

char broiled; / declare a char variable */*

broiled = 'T'; / OK */*

broiled = T; / NO! Thinks T is a variable */*

broiled = "T"; / NO! Thinks "T" is a string */*

Character Constants and Initialization

- Because characters are really stored as numeric values, you can also use the numerical code to assign values:

char grade = 65; / ok for ASCII, but poor style */*

- Note, however, that this example assumes that the system is using ASCII code.
- Using 'A' instead of 65 produces code that works on any system.
- Therefore, it's much better to use character constants than numeric code values.

Nonprinting Characters

- The single-quote technique is fine for characters, digits, and punctuation marks
- But some of the ASCII characters are nonprinting
- How can these be represented?
- C offers three ways.
 - The first way we have already mentioned—just use the ASCII code.
char beep = 7;
 - The second way to represent certain awkward characters in C is to use special symbol sequences. These are called *escape sequences*.

Nonprinting Characters

Table 3.2. Escape Sequences

Sequence	Meaning
<code>\a</code>	Alert (ANSI C).
<code>\b</code>	Backspace.
<code>\f</code>	Form feed.
<code>\n</code>	Newline.
<code>\r</code>	Carriage return.
<code>\t</code>	Horizontal tab.
<code>\v</code>	Vertical tab.
<code>\\</code>	Backslash (\).
<code>\'</code>	Single quote (').
<code>\"</code>	Double quote (").
<code>\?</code>	Question mark (?).
<code>\0oo</code>	Octal value. (o represents an octal digit.)
<code>\xhh</code>	Hexadecimal value. (h represents a hexadecimal digit.)

Nonprinting Characters

- Escape sequences must be enclosed in single quotes when assigned to a character variable.

```
char nerf = '\n';
```

- Print the variable `nerf` to advance the printer or screen one line.
- The alert character (`\a`), added by C90, produces an audible or visible alert.
- The ANSI standard states that the alert character shall not change **the active position**.
- By active position, the standard means the location on the display device (screen, teletype, printer, and so on) at which the next character would otherwise appear.

Nonprinting Characters

- Next, the `\b`, `\f`, `\n`, `\r`, `\t`, and `\v` escape sequences are common output device control characters.
- They are best described in terms of how they affect the active position.
- These escape sequence characters do not necessarily work with all display devices.
- The next three escape sequences (`\\`, `\'`, and `\"`) enable you to use `\`, `'`, and `"` as character constants.

Gramps sez, "a \ is a backslash."

```
printf("Gramps sez, \"a \\ is a backslash.\\\"\\n");
```

Nonprinting Characters

- `\0oo` and `\xhh` are special representations of the ASCII code.
- To represent a character by its octal ASCII code, precede it with a backslash (`\`) and enclose the whole thing in single quotes.

beep = `'\007'`;

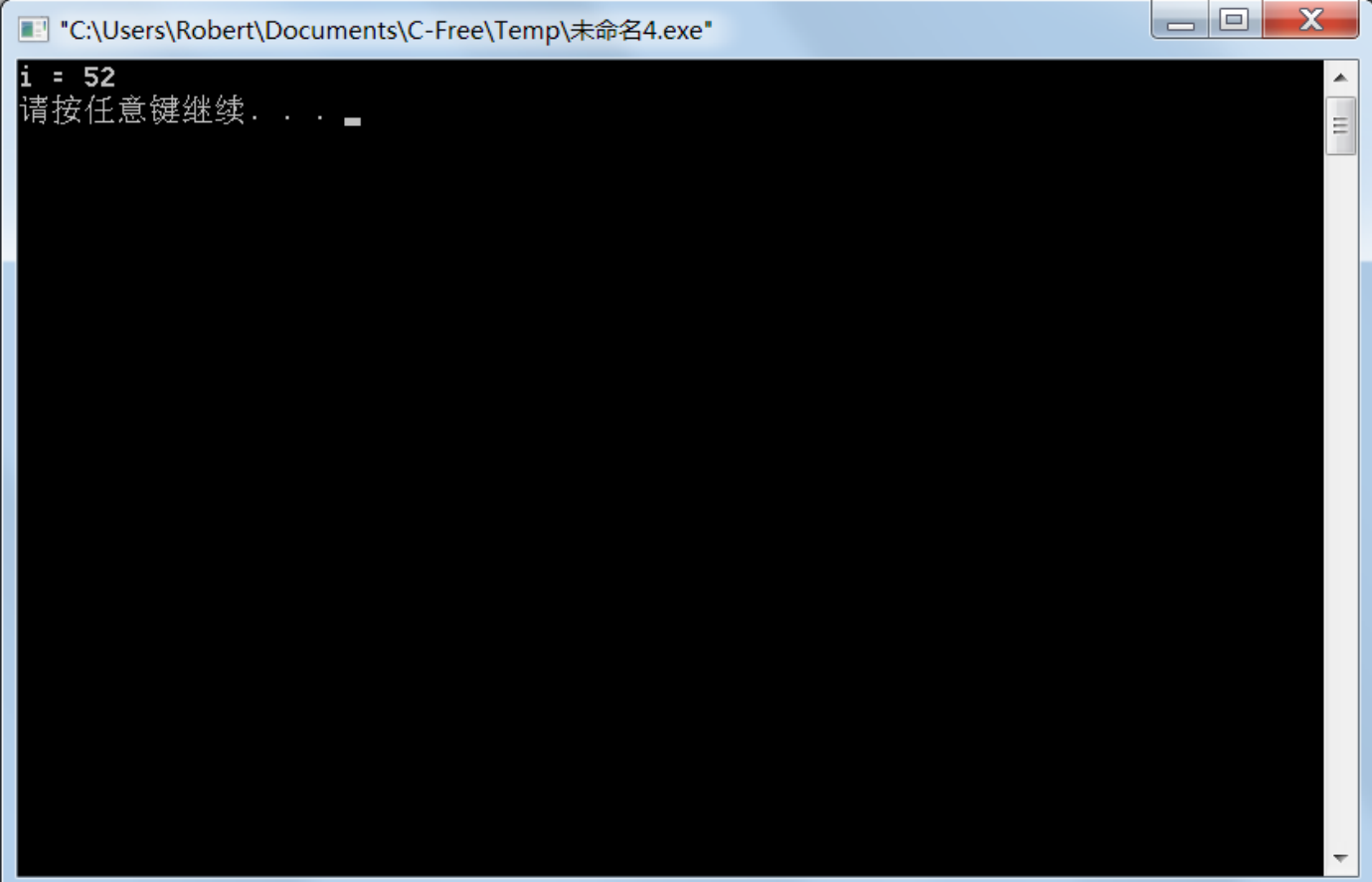
- You can omit the leading zeros, so `'\07'` or even `'\7'` will do. This notation causes numbers to be interpreted as octal, even if there is no initial 0.

Nonprinting Characters

- Beginning with C90, C provides **a third option**—using a hexadecimal form for character constants.
- In this case, the backslash is followed by an x or X and one to three hexadecimal digits.
- For example, the Ctrl+P character has an ASCII hex code of 10 (16, in decimal), so it can be expressed as `'\x10'` or `'\X010'`.
- When you use ASCII code, note **the difference between numbers and number characters**.
- For example, the character 4 is represented by ASCII code value 52. The notation `'4'` represents the symbol 4, not the numerical value 4.

Listing 3.3. The bases.c Program

```
#include <stdio.h>
int main(void)
{
    char i = '4';
    printf("i = %d\n", i);
    return 0;
}
```



A screenshot of a Windows command prompt window. The title bar shows the file path: "C:\Users\Robert\Documents\C-Free\Temp\未命名4.exe". The window has a black background. The output of the program is displayed in white text: "i = 52" followed by a newline and the Chinese prompt "请按任意键继续. . .".

Nonprinting Characters

- At this point, you may have three questions:
 - Why aren't the escape sequences enclosed in single quotes in the last example (`printf("Gramps sez, \"a \\ is a backslash\\\"\\n");`)? When a character, be it an escape sequence or not, is part of a string of characters enclosed in double quotes, don't enclose it in single quotes.
 - When should I use the ASCII code, and when should I use the escape sequences? If you have a choice between using **one of the special escape sequences**, say `'\f'`, or **an equivalent ASCII code**, say `'\014'`, use the `'\f'`. First, the representation is more mnemonic. Second, it is more portable. If you have a system that doesn't use ASCII code, the `'\f'` will still work.

Nonprinting Characters

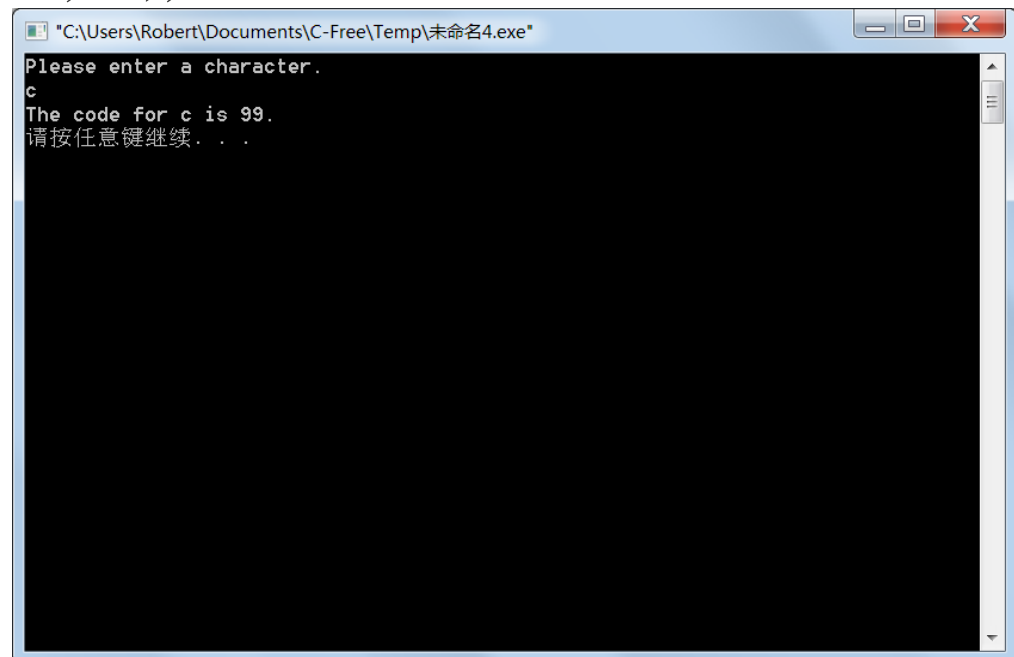
- At this point, you may have three questions:
 - If I need to use numeric code, why use, say, `'\032'` instead of `032`? First, using `'\032'` instead of `032` makes it clear to someone reading the code that you intend to represent a character code. Second, an escape sequence such as `\032` can be embedded in part of a C string, the way `\007` was in the first point.

Printing Characters

- The printf() function uses **%c** to indicate that a character should be printed.
- Recall that a character variable is stored as a **1-byte integer value**.
- Therefore, if you print the value of a char variable with the usual %d specifier, you get an integer.
- The %c format specifier tells printf() to display the character that has that integer as its code value.

Listing 3.5. The charcode.c Program

```
/* charcode.c-displays code number for a character */
#include <stdio.h>
int main(void)
{
    char ch;
    printf("Please enter a character.\n");
    scanf("%c", &ch); /* user inputs character */
    printf("The code for %c is %d.\n", ch, ch);
    return 0;
}
```



The screenshot shows a Windows command prompt window titled "C:\Users\Robert\Documents\C-Free\Temp\未命名4.exe". The window has a black background with white text. The text displayed is as follows:

```
Please enter a character.
c
The code for c is 99.
请按任意键继续. . .
```

The first line is the prompt "Please enter a character.". The second line shows the user input 'c'. The third line shows the output "The code for c is 99.". The fourth line is a Chinese prompt "请按任意键继续. . ." (Press any key to continue. . .).

Printing Characters

- When you use the program, remember to **press the Enter or Return key after typing the character.**
- The **scanf()** function then fetches the character you typed, and **the ampersand (&)** causes the character to be assigned to the variable **ch**.
- The **printf()** function then prints the value of **ch** twice, **first as a character** (prompted by the **%c** code) and then as **a decimal integer** (prompted by the **%d** code).
- Note that the **printf()** specifiers **determine how data is displayed, not how it is stored**

Printing Characters

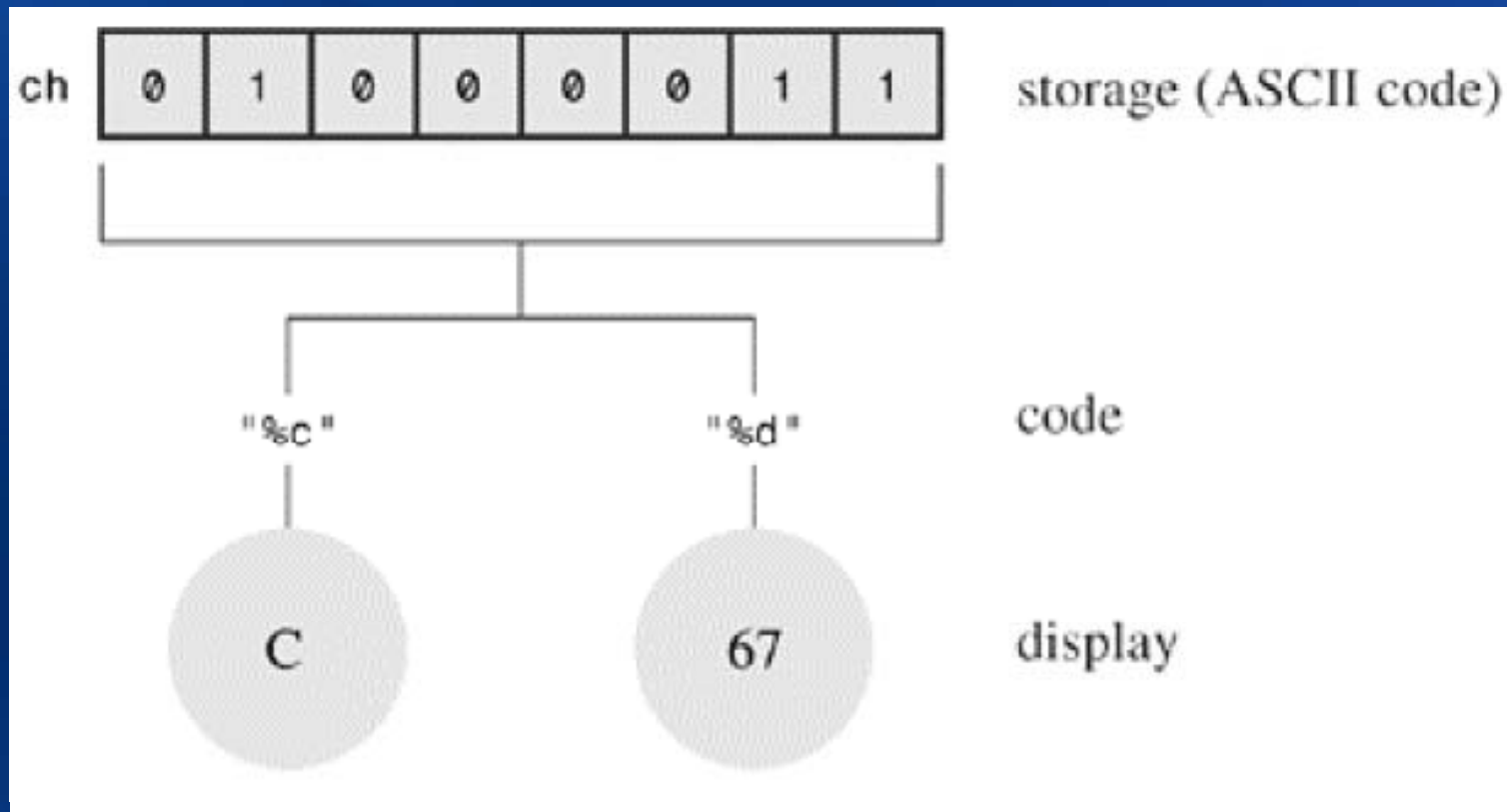


Figure 3.6. Data display versus data storage.

The `_Bool` Type

- The `_Bool` type is a C99 addition that's used to represent Boolean values—that is, the logical values **true** and **false**.
- Because C uses the value 1 for true and 0 for false, the `_Bool` type really is just an integer type, but one that, in principle, only requires 1 bit of memory, because that is enough to cover the full range from 0 to 1.
- Programs use Boolean values to choose which code to execute next.
- Code execution is covered more fully later. Let's defer further discussion until then.

Types float, double, and long double

- The various integer types serve well for most software development projects.
- However, financial and mathematically oriented programs often make use of floating-point numbers.
- In C, such numbers are called type **float**, **double**, or **long double**.
- The floating-point approach enables you to represent a much greater range of numbers, including decimal fractions.
- Floating-point number representation is similar to **scientific notation**, a system used by scientists to express very large and very small numbers.

Types float, double, and long double

- In scientific notation, numbers are represented as decimal numbers times powers of 10.

Number	Scientific Notation	Exponential Notation
1,000,000,000	$= 1.0 \times 10^9$	$= 1.0e9$
123,000	$= 1.23 \times 10^5$	$= 1.23e5$
322.56	$= 3.2256 \times 10^2$	$= 3.2256e2$
0.000056	$= 5.6 \times 10^{-5}$	$= 5.6e-5$

- The first column shows the usual notation
- The second column scientific notation
- The third column exponential notation, or e-notation

Types float, double, and long double



Figure 3.7. Some floating-point numbers.

Types float, double, and long double

- The **C standard** provides that a float has to be able to represent **at least six significant figures** and allow a range of at least 10^{-37} to 10^{+37} .
- The first requirement means, for example, that a float has to represent accurately **at least the first six digits** in a number such as 33.333333.
- The second requirement is handy if you like to use numbers such as the mass of the sun (2.0e30 kilograms), the charge of a proton (1.6e-19 coulombs), or the national debt.

Types float, double, and long double

- Often, systems use **32 bits** to store a floating-point number.
- **Eight bits** are used to give the exponent its value and sign, and **24 bits** are used to represent the nonexponent part, called the mantissa or significand, and its sign.
- C also has a **double** (for double precision) floating-point type
- The double type **has the same minimum range requirements as float**, but it extends **the minimum number of significant figures that can be represented to 10**.
- Typical double representations use **64 bits** instead of 32.

Types float, double, and long double

- Some systems use all **32 additional bits** for the nonexponent part. This **increases the number of significant figures** and **reduces round-off errors**.
- Other systems use some of the bits to **accommodate a larger exponent**; this increases the range of numbers that can be accommodated.
- Either approach leads to **at least 13 significant figures**, more than meeting the minimum standard.

Declaring Floating-Point Variables

- Floating-point variables are declared and initialized in the same manner as their integer cousins.

```
float noah, jonah;
```

```
double trouble;
```

```
float planck = 6.63e-34;
```

```
long double gnp;
```

Floating-Point Constants

- There are many choices open to you when you write a floating-point constant.
- The basic form of a floating-point constant is **a signed series of digits**, including a decimal point, followed by **an e or E**, followed by **a signed exponent** indicating the power of 10 used.

-1.56E+12

2.87e-3

Floating-Point Constants

- You can leave out positive signs.
- You can do without a decimal point (2E5) or an exponential part (19.28), but not both simultaneously.
- You can omit a fractional part (3.E16) or an integer part (.45E-6), but not both (that wouldn't leave much!).

3.14159

.2

4e16

.8E-5

100.

Floating-Point Constants

- Don't use spaces in a floating-point constant

Wrong: 1.56 E+12

- **By default**, the compiler assumes floating-point constants are **double precision**.

Suppose, for example, that **some** is a float variable and that you have the following statement:

```
some = 4.0 * 2.0;
```

Then 4.0 and 2.0 are stored as double, using (typically) **64 bits** for each.

- The product is calculated using double precision arithmetic, and only then is the answer trimmed to regular float size.
- This ensures greater precision for your calculations, but it can slow down a program.

Floating-Point Constants

- C enables you to override this default by **using an f or F suffix** to make the compiler treat a floating-point constant as type float
- Examples are 2.3f and 9.11E9F.
- **An l or L suffix makes a number type long double**
- Examples are 54.3l and 4.32e4L.
- Note that L is less likely to be mistaken for 1 (one) than is l. **If the floating-point number has no suffix, it is type double.**

Printing Floating-Point Values

- The printf() function uses the **%f** format specifier to print type **float** and **double** numbers using **decimal** notation
- Use **%e** to print them in **exponential** notation.
- If your system supports the C99 **hexadecimal** format for **floating-point numbers**, you can use **a** or **A** instead of **e** or **E**.
- The **long double** type requires the **%Lf**, **%Le**, and **%La** specifiers to print that type.

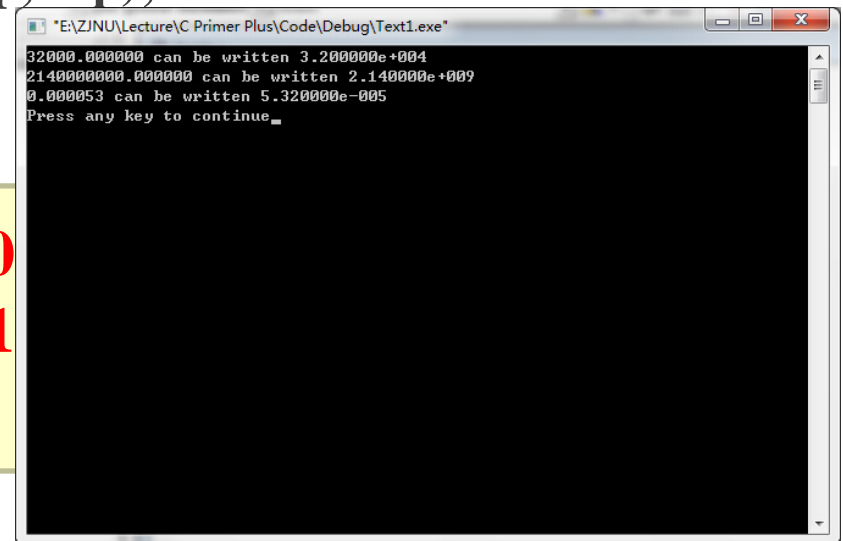
Printing Floating-Point Values

- Note that **both float and double use the %f, %e, or %a specifier for output.**
- That's because C automatically expands type float values to type double when they are passed as arguments to any function that doesn't explicitly prototype the argument type.

Listing 3.7. The showf_pt.c Program

```
/* showf_pt.c -- displays float value in two ways */
#include <stdio.h>
int main(void)
{
    float aboat = 32000.0;
    double abet = 2.14e9;
    long double dip = 5.32e-5;
    printf("%f can be written %e\n", aboat, aboat);
    printf("%f can be written %e\n", abet, abet);
    printf("%f can be written %e\n", dip, dip);
    return 0;
}
```

32000.000000 can be written 3.200000e+04
2140000000.000000 can be written 2.140000e+09
0.000053 can be written 5.320000e-05



The screenshot shows a Windows command prompt window titled "E:\Z\NU\Lecture\C Primer Plus\Code\Debug\Text1.exe". The output of the program is displayed as follows:

```
32000.000000 can be written 3.200000e+04
2140000000.000000 can be written 2.140000e+09
0.000053 can be written 5.320000e-05
Press any key to continue_
```

Floating-Point Overflow and Underflow

Suppose the biggest possible float value on your system is about 3.4E38

```
float toobig = 3.4E38 * 100.0f;  
printf("%e\n", toobig);
```

- What happens?
- This is an example of **overflow**—when a calculation leads to a number too large to be expressed.
- The behavior for this case used to be undefined, but now C specifies that toobig gets assigned a special value that stands for infinity and that printf() displays either **inf** or **infinity** (or some variation on that theme) for the value.

Floating-Point Overflow and Underflow

- What about dividing very small numbers?
- Here the situation is more involved. Recall that a float number is stored as an exponent and as a value part, or mantissa.
- This will be **the smallest number** that still is represented to the full precision available to a float value (There will be a number that has the smallest possible exponent and also the smallest value that still uses all the bits available to represent the mantissa.)
- Now divide it by 2.

Floating-Point Overflow and Underflow

- Normally, this reduces the exponent, but the exponent already is as small as it can get.
- So, instead, the computer **moves the bits in the mantissa over, vacating the first position and losing the last binary digit**.
- An analogy would be taking a base 10 value with four significant digits, such as $0.1234\text{E}-10$, dividing by 10, and getting $0.0123\text{E}-10$.
- You get an answer, but you've lost a digit in the process. This situation is called **underflow**, and C refers to floating-point values that have lost the full precision of the type as **subnormal**.

Floating-Point Round-off Errors

- Take a number, add 1 to it, and subtract the original number. What do you get? You get 1. A floating-point calculation, such as the following, may give another answer:

Listing 3.7. The showf_pt.c Program

```
/* floaterr.c--demonstrates round-off error */
#include <stdio.h>
int main(void)
{
    float a,b;
    b = 2.0e20 + 1.0;
    a = b - 2.0e20;
    printf("%f\n", a);
    return 0;
}
```

0.000000 ← older gcc on Linux

-13584010575872.000000 ← Turbo C 1.5

4008175468544.000000 ← CodeWarrior 9.0, MSVC++ 7.1

Floating-Point Round-off Errors

- The reason for these odd results is that the computer doesn't keep track of enough decimal places to do the operation correctly.
- The number $2.0e20$ is 2 followed by 20 zeros and, by adding 1, you are trying to change the 21st digit. To do this correctly, the program would need to be able to store a 21-digit number.
- A float number is typically just **six or seven digits** scaled to bigger or smaller numbers with an exponent. The attempt is doomed.
- On the other hand, if you used $2.0e4$ instead of $2.0e20$, you would get the correct answer

Beyond the Basic Types

- C does have other types derived from the basic types.
- These types include arrays, pointers, structures, and unions.
- Although they are subject matter for later chapters, we have already smuggled some pointers into this chapter's examples. (A pointer points to the location of a variable or other data object. The & prefix used with the scanf() function creates a pointer telling scanf() where to place information.)

Beyond the Basic Types

- C does have other types derived from the basic types.
- These types include arrays, pointers, structures, and unions.
- Although they are subject matter for later chapters, we have already smuggled some pointers into this chapter's examples. (A pointer points to the location of a variable or other data object. The & prefix used with the scanf() function creates a pointer telling scanf() where to place information.)

Summary: The Basic Data Types

- Keywords - The basic data types are set up using 11 keywords:

int, long, short, unsigned, char, float, double, signed, _Bool, _Complex, and _Imaginary.

The Basic Data Types - Signed Integers

Have positive or negative values:

- `int`— The basic integer type for a given system. C guarantees **at least 16 bits for `int`**.
- `short` or `short int`— The largest short integer is no larger than the largest `int` and may be smaller. C guarantees **at least 16 bits for `short`**.
- `long` or `long int`— Can hold an integer at least as large as the largest `int` and possibly larger. C guarantees **at least**
 - Typically, `long` will be bigger than `short`, and `int` will be the same as one of the two.
 - You can, if you like, use the keyword `signed` with any of the signed types, making the fact that they are signed explicit.

The Basic Data Types - Unsigned Integers

These have zero or positive values only:

- This extends the range of the largest possible positive number.
- Use the keyword **unsigned** before the desired type: unsigned int, unsigned long, unsigned short.
- A lone unsigned is the same as unsigned int

The Basic Data Types - Characters

These are typographic symbols such as A, &, and +:

- By definition, the char type uses **1 byte of memory** to represent a character.
- Historically, this character byte has **most often been 8 bits**, but it can be **16 bits or larger**, if needed to represent the base character set.
- **char**— The keyword for this type.
- Some implementations use a **signed** char, but others use an **unsigned** char.
- C enables you to use the keywords signed and unsigned to specify which form you want

The Basic Data Types - Boolean

Boolean values represent true and false; C uses 1 for true and 0 for false:

- **_Bool**— The keyword for this type.
- It is an unsigned int and need only be large enough to accommodate the range 0 through 1.

The Basic Data Types - Real Floating Point

These can have positive or negative values:

- **float**— The basic floating-point type for the system; it can represent **at least six significant figures** accurately.
- **double**— A (possibly) larger unit for holding floating-point numbers. It may **allow more significant figures** (at least 10, typically more) and **perhaps larger exponents than float**.
- **long double**— A (possibly) even larger unit for holding floating-point numbers. It may **allow more significant figures** and **perhaps larger exponents than double**.

Summary: How to Declare a Simple Variable

1. Choose the type you need.
2. Choose a name for the variable using the allowed characters.
3. Use the following format for a declaration statement:

type-specifier variable-name;

The **type-specifier** is formed from one or more of the type keywords;

here are examples of declarations:

int ertest;

unsigned short cash;

Summary: How to Declare a Simple Variable

4. You can declare more than one variable of the same type by separating the variable names with commas.

Here's an example:

```
char ch, init, ans;
```

5. You can initialize a variable in a declaration statement:

```
float mass = 6.0E24;
```

Summary

- C has a variety of data types.
- The basic types fall into two categories: integer types and floating-point types.
- The two distinguishing features for integer types are the amount of storage allotted to a type and whether it is signed or unsigned.
- The smallest integer type is char. The other integer types: short, int, long, and long long type. C guarantees that each of these types is at least as large as the preceding type.



Questions & Homeworks