 Answers to review questions from Chapter 11

1. Define the following terms: *bit, byte,* and *word.*

**A *bit* is a data value with exactly two states, typically denoted by 0 and 1. A *byte* is a combined sequence of eight bits that can take on values between 0 and 255; a byte is also the smallest addressable unit in memory. A *word* is a longer collection of bits that represents the natural size for an integer in the hardware of the machine; typical word sizes for modern machines are 32 and 64 bits.**

2. What is the etymology of the word *bit?*

**The word *bit* was coined by Claude Shannon and is a contraction of *binary digit*.**

3. How many bytes of memory are there in a 2GB machine?

**2,147,483,648**

4. Convert each of the following decimal numbers to its hexadecimal equivalent:

a)17 **11** c)1729 **6C1**

b)256 **100** d)2766 **ACE**

5. Convert each of the following hexadecimal numbers to decimal:

a) **17** **23** c) **CC** **204**

b) **64** **100** d) **FADE** **64222**

6. How many bytes does C++ assign to a value of type **char**? How many bytes are typically required for a **double**?

**A char takes one byte in C++; on most machines, a double takes eight bytes.**

7. True or false: In C++, values of type **char** always require one byte of memory.

**True. C++ defines the size of char to be a single byte.**

8. True or false: In C++, values of type **int** always require four bytes of memory.

**False. C++ allows the size of type int to vary.**

9. If a machine uses two’s complement arithmetic to represent negative numbers, what is the internal representation of –7 in a 32‑bit integer format?

**11111111111111111111111111111001**

10. What are the three areas of memory in which values can be stored in a C++ program?

**The *static area,* the *heap,* and the *stack*.**

11. What is the purpose of the **sizeof** operator? How do you use it?

**The sizeof operator returns the number of bytes required for a data value. The sizeof keyword acts as a prefix operator that requires either an expression or a type name in parentheses.**

12. What is an *address?*

**An *address* is a numeric location that specifies where in memory a particular value is stored. Bytes in memory are assigned consecutive addresses.**

13. What is an *lvalue?*

**An *lvalue* is an expression that can appear on the left side of an assignment statement.**

14. What reasons for using pointers are cited in this chapter?

**• Pointers allow you to refer to a large data structure in a compact way.**

**• Pointers make it possible to reserve new memory during program execution.**

**• Pointers can be used to record relationships among data items.**

15. What are the types of the variables introduced by the following declaration:

int \* p1, p2;

**The variable p1 is declared to be a pointer to an int; the variable p2 is declared to be an int.**

16. What are the two fundamental pointer operations? Which one corresponds to the term *dereferencing?*

**The operators that apply to pointers are & (address of) and \* (dereference).**

17. Explain the difference between *pointer assignment* and *value assignment.*

**Pointer assignment copies the pointer value, making two pointers refer to the same address. In the statement**

p1 = p2;

**for example, copies the internal representation of the pointer p2 to p1. Value assignment requires dereferencing and copies the values to which the pointers refer. Thus, after executing the statement**

\*p1 = \*p2;

**the memory cell pointed to by p1 will have the same value as the cell pointed to by p2.**

18. Assuming that variables of type **int** and all pointers require four bytes of memory, draw a diagram showing a portion of the stack frame that contains the following declarations:

|  |  |
| --- | --- |
| int v1 = 10;  int v2 = 25;  int \*p1 = &v1;  int \*p2 = &v2; | /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/11-PointersAndArrays/pictures/ExerciseFigures/ExerciseStackFrame1.png |

In your diagram, trace through the operation of these statements:

|  |  |
| --- | --- |
| \*p1 += \*p2; | /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/11-PointersAndArrays/pictures/ExerciseFigures/ExerciseStackFrame2.png |

|  |  |
| --- | --- |
| p2 = p1; | /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/11-PointersAndArrays/pictures/ExerciseFigures/ExerciseStackFrame3.png |

|  |  |
| --- | --- |
| \*p2 = \*p1 + \*p2; | /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/11-PointersAndArrays/pictures/ExerciseFigures/ExerciseStackFrame4.png |

19. True or false: For any variable **x**, the expression **\*&x** is essentially a synonym for **x**.

**True. Those expressions can be used interchangeably.**

20. True or false: For any variable **p**, the expression **&\*p** is essentially a synonym for **p**.

**False. If p is not a pointer variable, the expression &\*p is illegal.**

21. How are pointers used in the implementation of call by reference?

**When a parameter is passed by reference, the stack frame stores only the address of the calling value. C++ automatically dereferences the pointer on any access to the parameter variable.**

22. Write array declarations for the following array variables:

a) An array **realArray** consisting of 100 floating-point values

const int N\_VALUES = 100;

double realArray[N\_VALUES];

b) An array **inUse** consisting of 16 Boolean values

const int N\_INUSE\_FLAGS = 16;

bool inUse[N\_INUSE\_FLAGS];

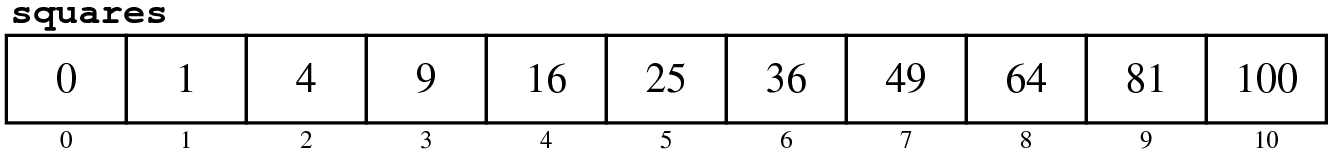
c) An array **lines** that can hold up to 1000 strings

const int MAX\_LINES = 1000;

string lines[MAX\_LINES];

Remember to declare constants to specify the allocated size for these arrays.

23. Write the variable declaration and **for** loop necessary to create and initialize the following integer array:



const int MAX\_SQUARE = 10;

int squares[MAX\_SQUARE + 1];

for (int i = 0; i <= MAX\_SQUARE; i++) {

squares[i] = i \* i;

}

24. What is the difference between the *allocated size* and the *effective size* of an array?

**The *allocated size* is the number of elements assigned to the array in memory and serves as a maximum element count for the array. The *effective size* is stored in a separate variable and indicates how many of the elements are actively in use.**

25. Assuming that **intArray** is declared as

int intArray[10];

and that **j** is an integer variable, describe the steps the computer would take to determine the value of the following expression:

&intArray[j + 3];

**The expression asks for the computer to calculate the address of the element in intArray that occurs at position j + 3. The first step in the process is to add j and 3 as integers to compute the actual index. This value is then multiplied by the size of an integer and added to the base address of the array to generate the desired address.**

26. If **array** is declared to be an array, describe the distinction between the expressions

array[2]

and

array + 2

**The first expression refers to the value in array at index position 2. The second calculates the address of that element.**

27. Assume that variables of type **double** take up eight bytes on the computer system you are using. If the base address of the array **doubleArray** is **FF00**, what is the address value of **doubleArray + 5**?

**FF28** **The offset 28 is the hexadecimal equivalent of 40 (8 × 5)**

28. True or false: If **p** is a pointer variable, the expression **p++** adds 1 to the internal representation of **p**.

**False. The expression p++ adds the size of the target variable to the internal representation of p.**

29. Describe the effect of the idiomatic C++ expression

\*p++

**This expression increments the pointer p but returns the value in the cell that p pointed to *before* the increment operation. The effect is to return the current element of an array and move on to the next.**

30. In the expression **\*p++**, which operator (**\*** or **++**) is applied first? In general, what rule does C use to determine the precedence of unary operators?

**The ++ is evaluated first. In C++, unary operators are evaluated from right to left.**