7.1 Introduction

7.2 Pointer Variable Definitions and Initialization

7.1 Pointers *cannot* be used to

(a) find the address of a variable in memory.

(b) reference values directly.

(c) simulate call-by-reference.

(d) manipulate dynamic data structures.

ANS: (b)

7.2 Pointers may be assigned which of the following?

(a) all integer values

(b) an address

(c) NULL

(d) both (b) and (c)

ANS: (d)

7.3 Pointers are variables that contain \_\_\_\_\_\_\_\_\_\_ as their values.

a) strings

b) flowlines

c) memory addresses

d) directions

ANS: (c)

7.4 A non-pointer variable name \_\_\_\_\_\_\_\_\_\_ references a value and a pointer variable name \_\_\_\_\_\_\_\_\_\_ references a value.

a) directly, directly

b) directly, indirectly

c) indirectly, directly

d) indirectly, indirectly

ANS: (b)

7.5 The definition

int \*count;

a) is a syntax error because only pointers can be defined with \* notation.

b) is a compile-time error.

c) is a logic error.

d) is a correct definition of integer pointer count.

ANS: (d)

7.6 Referencing a value through a pointer is called \_\_\_\_\_\_\_\_\_\_.

a) interdiction

b) indirection

c) intermediation

d) redirection

ANS: (b)

7.7 Which statement about pointers is *false*?

a) They can be defined to point to objects of any data type.

b) The indirection operator \* distributes to all comma-separated variable names in a definition.

c) The letters Ptr in a pointer variable name are optional.

d) A pointer may be initialized to 0, NULL or an address.

ANS: (b)

7.8 Which statement about pointers is *false*?

a) A pointer with the value NULL points to nothing.

b) NULL is a symbolic constant defined in the <stdio.h> header file.

c) Initializing a pointer to 0 is equivalent to initializing a pointer to NULL, but NULL is preferred.

d) The values 0 and 1 are the only values that can be assigned directly to a pointer variable.

ANS: (d)

7.3 Pointer Operators

7.9 The & operator can be applied to

(a) constants

(b) variables defined with the storage class register

(c) variables defined with the storage class static

(d) *rvalues*

ANS: (c)

7.10 Three of the following expressions have the same value. Which of the following’s value is different from the others?

(a) \*&Ptr

(b) &\*Ptr

(c) \*Ptr

(d) Ptr

ANS: (c)

7.11 The \_\_\_\_\_\_\_\_\_\_, or address operator, is a unary operator that returns the address of its operand.

a) &

b) &&

c) \*

d) \*\*

ANS: (a)

7.12 The statement

y = &yPtr;

a) assigns the address of the variable y to pointer variable yPtr.

b) assigns the address of the variable yPtr to pointer variable y.

c) is a compilation error.

d) is a logic error.

ANS: (b)

7.13 Which statement is *false*?

a) The operand of the address operator must be a variable.

b) The address operator cannot be applied to constants or to expressions.

c) The address operator can be applied to variables defined with the storage class register.

d) The address operator can be applied to variables defined with the storage class static.

ANS: (c)

7.14 Which statement is *false*?

a) The unary \* operator is called the indirection operator or the dereferencing operator.

b) The operand of the unary \* operator must be a pointer.

c) The unary \* operator returns the value of its operand.

d) Placing a \* operator before a pointer dereferences the pointer.

ANS: (c)

7.15 Which statement is generally *false*?

a) Initializing pointers is optional.

b) Dereferencing an uninitialized pointer could lead to a fatal execution time error.

c) Deferencing an uninitialized pointer could accidentally modify important data.

d) Derefencing an uninitialized pointer causes a syntax error.

ANS: (d)

7.16 The unary \* and \_\_\_\_\_\_\_\_\_\_ are complements of one another.

a) /

b) ^

c) &

d) |

ANS: (c)

7.4 Passing Arguments to Functions by Reference

7.17 When a compiler encounters a function parameter for a single-subscripted array of the form int a[]**,** it converts the parameter to

(a) int a

(b) int &a

(c) int \* a

(d) int \* const a

ANS: (c)

7.18 Which statement is *false*?

a) All function calls in C pass arguments call-by-value.

b) Call-by-reference enables a called function to modify variables in the calling function.

c) Call-by-value is always more efficient than call-by-reference.

d) In C, programmers use pointers and the indirection operator to simulate call-by-reference.

ANS: (c)

7.19 When calling a function with arguments that should be modified, the \_\_\_\_\_\_\_\_\_\_ of those arguments are passed.

a) memory

b) addresses

c) values

d) complements

ANS: (b)

7.20 If array name arrayName is passed to a function, C automatically passes \_\_\_\_\_\_\_\_\_\_.

a) &arrayName[0]

b) arrayName[1]

c) arrayName[0]

d) \*arrayName

ANS: (a)

7.21 When the \_\_\_\_\_\_\_\_\_\_ of a variable is passed to a function, the indirection operator (\*) may be used in the function to modify the \_\_\_\_\_\_\_\_\_\_ at that location in the caller’s memory.

a) address, address

b) value, address

c) value, value

d) address, value

ANS: (d)

7.22 Which statement is *false*?

a) It’s necessary to include names of pointer arguments in function prototypes.

b) A function receiving an address as an argument must define a pointer parameter to receive the address.

c) The compiler does not distinguish between a function that receives a pointer and a function that receives a single-subscripted array.

d) The function must “know” whether it is receiving a single-subscripted array or simply a single variable for which it is to perform simulated call by reference.

ANS: (a)

7.5 Using the const Qualifier with Pointers

7.23 A function that modifies an array by using pointer arithmetic to process every value should have a parameter that is

(a) a nonconstant pointer to nonconstant data

(b) a nonconstant pointer to constant data

(c) a constant pointer to nonconstant data

(d) a constant pointer to constant data

ANS: (a)

7.24 A function that prints a string should have a parameter that’s a

(a) nonconstant pointer to nonconstant data.

(b) nonconstant pointer to constant data.

(c) constant pointer to nonconstant data.

(d) constant pointer to constant data.

ANS: (b)

7.25 What method should be used to pass an array to a function that does *not* modify the array and only looks at it using array subscript notation?

(a) A constant pointer to constant data.

(b) A constant pointer to nonconstant data.

(c) A nonconstant pointer to constant data.

(d) A nonconstant pointer to constant data.

ANS: (a)

7.26. Which statement is *false*?

a) The const qualifier enables the programmer to inform the compiler that the value of a particular variable should not be modified.

b) The const qualifier did not exist in early versions of C; it was added to the language by the ANSI C committee.

c) Using the const qualifier typically violates the principle of least privilege

d) Adding const to C programs that do not use it presents significant opportunities for improvement in the software engineering of that code.

ANS: (c)

7.27 Which of the following most closely reflects the principle of least privilege.

a) Always grant a function at least enough access to the data in its parameters to accomplish its specified task.

b) Always grant a function enough access to the data in its parameters to accomplish its specified task, but no more.

c) Always grant a function the least amount of access to the data in its parameters.

d) Never grant a function access to the data in its parameters.

ANS: (b)

7.28 Which statement is *false*?

a) If a value should not change in the body of a function to which it is passed, the value should be defined const to ensure that it is not accidentally modified.

b) Attempts to modify the value of a variable defined const are caught at execution time.

c) One way to pass a pointer to a function is to use a non-constant pointer to non-constant data.

d) It is dangerous to pass a non-pointer into a pointer argument.

ANS: (b)

7.29 The highest level of data access is granted by a

a) non-constant pointer to non-constant data.

b) non-constant pointer to constant data.

c) constant pointer to non-constant data.

d) constant pointer to constant data.

ANS: (a)

7.30 Which statement is *false*?

a) Arrays and structures are aggregate data types.

b) Structures in C are sometimes called records in other languages.

c) A structure is capable of storing many related data items of different data types.

d) Structures are always passed call by reference.

ANS: (d)

7.31 When a structure must be passed to a function, we can use pointers to constant data to get the performance of a call by \_\_\_\_\_\_\_\_\_\_ and the protection of a call by \_\_\_\_\_\_\_\_\_\_.

a) value, value

b) value, reference

c) reference, value

d) reference, reference

ANS: (c)

7.32 The least access privilege is granted by a \_\_\_\_\_\_\_\_\_\_ pointer to \_\_\_\_\_\_\_\_\_\_ data.

a) non-constant, non-constant

b) non-constant, constant

c) constant, non-constant

d) constant, constant

ANS: (d)

7.6 Bubble Sort Using Pass-by-Reference

7.33 Which statement is *false*?

a) The notations int \*array and int array[] are interchangeable.

b) Function prototypes may not be placed inside functions.

c) Good advice: To encourage software reusability, when passing an array, also pass the size of the array.

d) Global variable violate the principle of least privilege.

ANS: (b)

7.7 sizeof Operator

7.34 sizeof

(a) is a binary operator

(b) returns the total number of elements in an array

(c) usually returns a double

(d) returns the total number of bytes in an array

ANS: (d)

7.35 Which of the following gives the number of elements in the array int r[]?

(a) sizeof (r)

(b) sizeof (\*r)

(c) sizeof (r) / sizeof (int)

(d) sizeof (\*r) / sizeof (int)

ANS: (c)

7.36 Which statement is *false*?

a) Using global variables, instead of passing arguments, can increase program performance, but reduce effective software engineering.

b) Hardcoding into a function the size of an array that will be passed as an argument rather than passing the size of the array as a parameter, reduces the software reusability of that function.

c) sizeof is an operator that calculates the size of its argument in bytes at compile time.

d) When applied to the name of an array, sizeof returns the number of elements in the array.

ANS: (d)

7.37 An expression such as

sizeof(arrayName) / sizeof(double)

might typically be used to determine

a) the size of an array

b) the number of elements in an array

c) the number of elements in half an array

d) the size of an element of an array

ANS: (b)

7.8 Pointer Expressions and Pointer Arithmetic

7.38 Which of the following can have a pointer as an operand?

(a) ++

(b) \*=

(c) %

(d) /

ANS: (a)

7.39 Given that k is an integer array starting at location 2000, kPtr is a pointer to k, and each integer is stored in 4 bytes of memory, what location does kPtr + 3 point to?

(a) 2003

(b) 2006

(c) 2012

(d) 2024

ANS: (c)

7.40 A pointer *cannot* be assigned to

(a) another pointer of the same type

(b) a pointer to void

(c) a pointer of a type other than its own type and void

(d) any other pointer by using the cast operator

ANS: (c)

7.41 Comparing pointers and performing arithmetic on them is meaningless unless

(a) they point to members of the same array

(b) they point to arrays of the same type

(c) they point to arrays of equal size

(d) they point to different locations

ANS: (a)

7.42 Which statement is *false*?

a) A pointer may be incremented or decremented.

b) An integer may be added to a pointer.

c) All operators normally used in arithmetic expressions, assignment expressions and comparison expressions are valid in conjunction with pointer variables.

d) A pointer may not be added to a pointer.

ANS: (c)

7.43 Which statement is *false*?

a) A pointer can always be assigned to another pointer of the same type.

b) A cast operator must always be used to convert the pointer on the right of an assignment to the pointer type on the left of the assignment.

c) Variables of all pointer types can be assigned a pointer to void.

d) A pointer to void can be assigned a pointer of any type.

ANS: (b)

7.9 Relationship Between Pointers and Arrays

7.44 Assuming that t is an array and tPtr is a pointer to that array, what expression refers to the address of element 3?

(a) \*(tPtr + 3)

(b) tPtr[3]

(c) &t[3]

(d) \*(t + 3)

ANS: (c)

7.45 Consider the following function:

void reverse(char \* string1, const char \* string2)

{

int stringsize = sizeof(string1)/sizeof(char);

\*(string1 + stringsize – 1) = '\0';

string1 = string1 + stringsize – 2;

for (; \*string2 != '\0'; --string1, ++string2) {

\*string1 = \*string2;  
 }

}

What method does the function use to refer to array elements?

(a) array/subscript notation

(b) array/offset notation

(c) pointer/subscript notation

(d) pointer/offset notation

ANS: (d)

7.46 If bPtr is assigned b (the name of an array), then array element b[3] can alternatively be referenced with the pointer expression \_\_\_\_\_\_\_\_\_\_.

a) bPtr + 3

b) b[bPtr + 3]

c) \*b [bPtr + 3]

d) \*(bPtr + 3)

ANS: (d)

7.10 Arrays of Pointers

7.47 A string array

(a) stores an actual string in each of its elements

(b) can only provide access to strings of a certain length

(c) is actually an array of pointers

(d) is always less memory efficient than an equivalent double-subscripted array

ANS: (c)

7.48 Which statement is *false*?

a) In C, a string is essentially a pointer to its first character.

b) Arrays may contain pointers.

c) Each entry in an array of strings is actually a pointer to the first character of a string.

d) The size of an array of strings is the sum of the lengths of the strings.

ANS: (d)

7.11 Case Study: Card Shuffling and Dealing Simulation

7.49 An algorithm that could execute for an unknown amount of time because it depends on random numbers to exit a function may

(a) have a redundancy.

(b) get caught in an infinite loop.

(c) suffer from indefinite postponement.

(d) issue a compiler error.

ANS: (c)

7.50 Which statement about the algorithm we presented in Section 7.10 of the text for shuffling a deck of cards is *true*?

a) It’s guaranteed to terminate.

b) It could execute indefinitely.

c) It’s efficient.

d) It uses a one-dimensional array of 52 cards.

ANS: (b)

7.12 Pointers to Functions

7.51 Which of the following is false with regard to pointers to functions?

(a) They contain the starting address of the function code.

(b) They’re dereferenced in order to call the function.

(c) They can be stored in arrays.

(d) They cannot be assigned to other function pointers

ANS: (d)

7.52 (\*max)(num1, num2, num3);

(a) is the header for function max

(b) is a call to function max

(c) is the prototype for function max

(d) is part of a definition of a pointer to the function max

ANS: (b)

7.53 A pointer to a function contains \_\_\_\_\_\_\_\_\_\_.

a) the address of the function’s automatic variables

b) the address of the function on the stack

c) the address of the entry for that function in the System Function Table

d) the address of the function in memory

ANS: (d)

7.54 Which statement about the parameter definition

int (\*compare)(int, int)

is *false*?

a) It defines a parameter that is a pointer to a function that receives two integer arguments and returns a pointer to an integer as a result.

b) Parentheses are needed around \*compare because \* has a lower precedence than the parentheses enclosing the function parameters.

c) Without the parentheses it would have defined a function that receives two integers and returns a pointer to an integer.

d) The corresponding parameter in the function prototype would ordinarily be

int (\*)(int, int)

ANS: (a)

7.13 Secure C Programming

7.55 Which of the following statements is *false*?

(a) A key feature of functions like printf\_s and scanf\_s that makes them more secure is that they have runtime pointer constraints that are checked after attempting to use the pointers.

(b) In a scanf\_s, if any of the pointer arguments (including the format-control string) are NULL, the function returns EOF.

(c) In a printf\_s, if the format-control string or any argument that corresponds to a %s is NULL, the function stops outputting data and returns a negative number.

(d)None of the above.

ANS: (a)

7.56 Which of the following statements is *false*?

(a) Dereferencing NULL pointers typically causes programs to crash.

(b) If a function parameter points to a value that will not be changed by the function, const should be used to indicate that the data is constant.

(c) Attackers cannot execute code by dereferencing a NULL pointer.

(d)None of the above.

ANS: (c)