 Answers to review questions from Chapter 17

1. True or false: The elements of a set are unordered, so the set {3, 2, 1} and the set {1, 2, 3} represent the same set.

**True.**

2. True or false: A set can contain multiple copies of the same element.

**False.**

3. What sets are denoted by each of the following symbols: ∅, **Z**, **N**, and **R?**

∅ **The empty set**

**Z The set of integers**

**N The set of *natural numbers,* which consists of the nonnegative integers**

**R The set of real numbers**

4. What do the symbols ∈ and ∉ mean?

**The conditional tests *element of* and *not element of*.**

5. Use an enumeration to specify the elements of the following set:

{*x* | *x* ∈ **N** and *x* ≤ 100 and/Users/eroberts/Books/CS2-in-C++/Chapters/18-Sets/pictures/Math/rootx.png ∈ **N**}

**{0, 4, 9, 25, 36, 49, 64, 81, 100}**

6. Write a rule-based definition for the following set:

{0, 9, 18, 27, 36, 45, 54, 63, 72, 81}

**{*x* | *x* ∈ N and *x* ≤ 100 and *x* is divisible by 9}**

7. What are the mathematical symbols for the operations union, intersection, and set difference?

**Union: ∪**

**Intersection: ∩**

**Set difference:** –

8. Evaluate the following set expressions:

a. {**a**, **b**, **c**} ∪ {**a**, **c**, **e**} **{ a**, **b**, **c**, **e }**

b. {**a**, **b**, **c**} ∩ {**a**, **c**, **e**} **{ a**, **c }**

c. {**a**, **b**, **c**} – {**a**, **c**, **e**} **{ b }**

d. ({**a**, **b**, **c**} – {**a**, **c**, **e**}) ∪ ({**a**, **b**, **c**} – {**a**, **c**, **e**}) **{ b }**

9. What is the difference between a subset and a proper subset?

**A *proper subset* is a subset that is not equal to the original set.**

10. Give an example of an infinite set that is a proper subset of some other infinite set.

**{*x* | *x* ∈ N and *x* is odd} is a proper subset of N**

11. For each of the following set operations, draw Venn diagrams whose shaded regions illustrate the contents of the specified set expression:

|  |  |
| --- | --- |
| a. *A* ∪ (*B* ∩ *C*)  /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/VennDiagramSolution1.png | c. (A – B) ∪ (B – A)  /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/VennDiagramSolution3.png |
|  |  |
| b. (A – C) ∩ (B – C)  /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/VennDiagramSolution2.png | d. (A ∪ B) – (A ∩ B)  /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/VennDiagramSolution4.png |

12. Write set expressions that describe the shaded region in each of the following Venn diagrams:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| a. | /Users/eroberts/Books/CS2-in-C++/Chapters/18-Sets/pictures/VennDiagrams/VennDiagramExercise1.png | | b. | /Users/eroberts/Books/CS2-in-C++/Chapters/18-Sets/pictures/VennDiagrams/VennDiagramExercise2.png | |
| ((A ∩ B) – C) ∪ ((A ∩ C) – B) ∪ ((B ∩ C) – A) | | (A ∩ B) ∪ (C – (A ∪ B)) | | |

13. Draw Venn diagrams illustrating each of the identities in Table 17‑1.

|  |  |  |
| --- | --- | --- |
| ***Idempotence*** |  |  |
| S | S ∩ S | S ∪ S |
| /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/Idempotence.png | /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/Idempotence.png | /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/Idempotence.png |

|  |  |  |
| --- | --- | --- |
| ***Absorption*** |  |  |
| A ∩ (A ∪ B) | A ∪ (A ∩ B) | A |
| /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/Absorption.png | /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/Absorption.png | /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/Absorption.png |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Commutative laws*** |  |  |  |
| A ∪ B | B ∪ A | A ∩ B | B ∩ A |
| /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/CommutativeUnion.png | /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/CommutativeUnion.png | /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/CommutativeIntersection.png | /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/CommutativeIntersection.png |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Associative laws*** |  |  |  |
| A ∪ (B ∪ C) | (A ∪ B) ∪ C | A ∩ (B ∩ C) | (A ∩ B) ∩ C |
| /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/AssociativeUnion.png | /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/AssociativeUnion.png | /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/AssociativeIntersection.png | /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/AssociativeIntersection.png |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Distributive laws*** |  |  |  |
| A ∪ (B ∩ C) | (A ∪ B) | (A ∪ C) | (A ∪ B) ∩ (A ∪ C) |
| /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/DistributiveUnion1.png | /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/DistributiveUnion2.png | /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/DistributiveUnion3.pdf | /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/DistributiveUnion1.png |
| A ∩ (B ∪ C) | (A ∩ B) | (A ∩ C) | (A ∩ B) ∪ (A ∩ C) |
| /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/DistributiveIntersection1.png | /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/DistributiveIntersection2.png | /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/DistributiveIntersection3.png | /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/DistributiveIntersection1.png |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Second De Morgan’s law* (the first is in the text)** | | | |
| A – (B ∪ C) | (A – B) | (A – C) | (A – B) ∩ (A – C) |
| /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/DeMorgan1.png | /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/DeMorgan2.png | /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/DeMorgan3.png | /Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/VennDiagrams/DeMorgan1.png |

14. What is the *cardinality* of a set?

**The *cardinality* of a set is the number of elements it contains.**

15. The general implementation of the **Set** class uses a data structure from an earlier chapter to represent the elements of a set. What is that structure? What properties make that structure useful for this purpose?

**The set implementation uses a map for its implementation because doing so provides efficient insertion, removal, and searching for keys.**

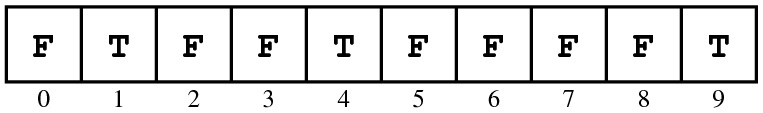
16. What is a characteristic vector?

**A *characteristic vector* is an array of bits in which each bit corresponds to the presence or absence of the corresponding index element in a set.**

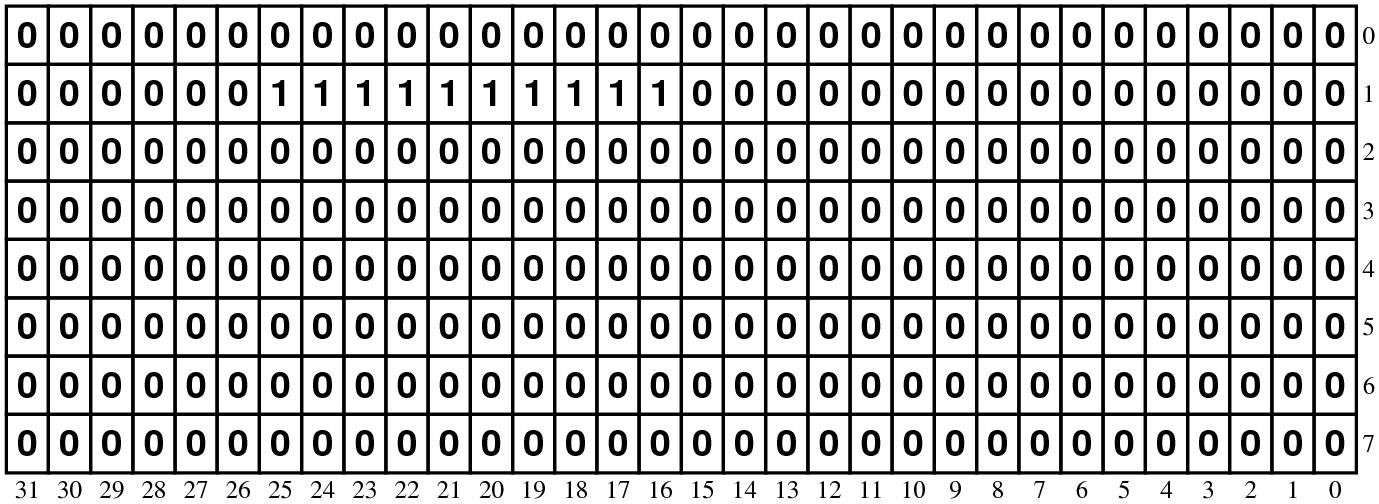
17. What restrictions must be placed on a set in order to use characteristic vectors as an implementation strategy?

**The number of possible elements must be small enough to allow the entire characteristic vector to fit in a reasonable amount of memory.**

18. Assuming that **RANGE\_SIZE** has the value 10, diagram the characteristic vector for the set {1, 4, 9}.



19. What set is represented by the following characteristic vector:



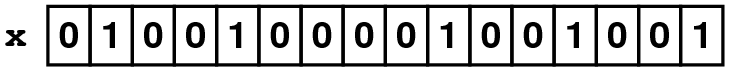
By consulting the ASCII chart in Table 1-2, identify the function in **<cctype>** to which this set corresponds.

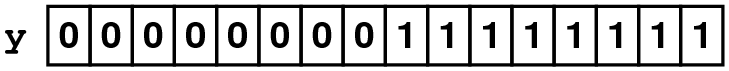
**This set contains the ASCII codes for the digit characters and therefore implements isdigit.**

20. In the diagrams used to represent characteristic vectors, an **unsigned** **long** is shown as taking 32 bits. Suppose that you have a machine on which a **long** takes 64 bits instead. Does the code given in the chapter continue to work?

**Yes. The code uses the sizeof operator to determine the size of an unsigned** **long and then adjusts the array sizes accordingly.**

21. Suppose that the variables **x** and **y** are of type **unsigned short** and contain the following bit patterns:





Expressing your answer as a sequence of bits, compute the value of each of the following expressions:

|  |  |
| --- | --- |
| a. **x & y**  **/Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/BitVectors/BitsExercise-xANDy.png** | f. **x & ~y**  **/Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/BitVectors/BitsExercise-xANDNOTy.png** |
| b. **x | y**  **/Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/BitVectors/BitsExercise-xORy.png** | g. **~x & ~y**  **/Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/BitVectors/BitsExercise-NOTxANDNOTy.png** |
| c. **x ^ y**  **/Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/BitVectors/BitsExercise-xXORy.png** | h. **y >> 4**  **/Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/BitVectors/BitsExercise-yRSH4.png** |
| d. **x ^ x**  **/Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/BitVectors/BitsExercise-xXORx.png** | i. **x << 3**  **/Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/BitVectors/BitsExercise-xLSH3.png** |
| e. **~x**  **/Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/BitVectors/BitsExercise-NOTx.png** | j. **(x >> 8) & y**  **/Users/eroberts/Books/ProgrammingAbstractionsInC++/chapters/17-Sets/pictures/BitVectors/BitsExercise-xRSH8ANDy.png** |

22. Express the values of **x** and **y** from the preceding exercise as constants using both octal and hexadecimal notation.

**x = 441118 = 484916**

**y = 3778 = FF16**

23. Suppose that the variables **x** and **mask** are both declared to be of type **unsigned**, and that the value of **mask** contains a single **1** bit in some position. What expressions would you use to implement the following operations:

a. Test the bit in **x** corresponding to the bit in **mask** to see if it is nonzero.

if ((x & mask) != 0)

b. Set the bit in **x** corresponding to the bit in **mask**.

x |= mask;

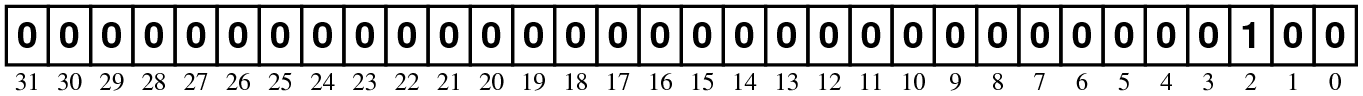
c. Clear the bit in **x** corresponding to the bit in **mask**.

x &= ~mask;

d. Complement the bit in **x** corresponding to the bit in **mask**.

x ^= mask;

24. Write an expression that constructs a mask of type **unsigned** in which there is a single **1** bit in bit position **k**, where bits are numbered from 0 starting at the right end of the word. For example, if **k** is 2, the expression should generate the following mask:



((unsigned) 1) << k