APPENDIX Q

# Answers to Odd-Numbered Review Questions

## Chapter 1

1. programmed
3. arithmetic logic unit (ALU) and control unit
5. operating systems and application software
7. programming language
9. High-level
11. portability
13. programmer-defined symbols
15. Punctuation
17. variable
19. input, processing, output
21. Output
23. Main memory, or RAM, is volatile, which means its contents are erased when power is removed from the computer. Secondary memory, such as a disk or CD, does not lose its contents when power is removed from the computer.
25. A syntax error is the misuse of a key word, operator, punctuation, or other part of the programming language. A logical error is a mistake that tells the computer to carry out a task incorrectly or to carry out tasks in the wrong order. It causes the program to produce the wrong results.
27.
```
Account Balance High Level Pseudocode
    Have user input starting balance
    Have user input total deposits
    Have user input total withdrawals
    Calculate current balance
    Display current balance
```

```
Account Balance Detailed Pseudocode
   Input startBalance            // with prompt
   Input totalDeposits           // with prompt
   Input totalWithdrawals        // with prompt
   currentBalance = startBalance + totalDeposits - totalWithdrawals
   Display currentBalance
```

29. 45

31. 28

33. The error is that the program performs its math operation before the user has entered values for the variables `width` and `length`.

# Chapter 2

1. semicolon

3. `main`

5. braces `{}`

7. 9.7865E14

9. B

11. B (C is valid, but prints the contents of variable `Hello`, rather than the string "`Hello`".)

13. A) 11  B) 14  C) 3 (An integer divide takes place.)

15.
```
double temp,
       weight,
       height;
```

17. A) `d2 = d1 + 2;`
B) `d1 = d2 * 4;`
C) `c = 'K';`
D) `i = 'K';`
E) `i = i – 1;`

19.
```
cout << "Two mandolins like creatures in the\n\n\n";
cout << "dark\n\n\n";
cout << "Creating the agony of ecstasy.\n\n\n";
cout << "                   - George Barker\n\n\n";
```

21.
```
Input weeks            // with prompt
days = weeks * 7
Display days
```

23.
```
Input speed            // with prompt
Input time             // with prompt
distance = speed * time
Display distance
```

25. A)
```
0
100
```
B)
```
8
2
```
C)
```
I am the incrediblecomputing
machine
and I will
amaze
you.
```
27. The C-style comments symbols are backwards.
`iostream` should be enclosed in angle brackets.
There shouldn't be a semicolon after `int main()`.
The opening and closing braces of function main are reversed.
There should be a semicolon after `int a, b, c`.
The comment `\\ Three integers` should read `// Three integers`.
There should be a semicolon at the end of each of the following lines:
```
a = 3
b = 4
c = a + b
```
`cout` begins with a capital letter.
The stream insertion operator (that appears twice in the `cout` statement) should read << instead of <.
The `cout` statement uses the variable `C` instead of `c`.

# Chapter 3

1. A) `cin >> description;`
B) `getline(cin, description);`
3. A) `cin >> setw(25) >> name;`
B) `cin.getline(name, 25);`
5. `iostream` and `iomanip`
7. A) `price = 12 * unitCost;`
B) `cout << setw(12) << 98.7;`
C) `cout << 12;`
9. A) `a = 12 * x;`
B) `z = 5 * x + 14 * y + 6 * k;`
C) `y = pow(x, 4);`
D) `g = (h + 12) / (4 * k);`
E) `c = pow(a, 3) / (pow(b, 2) * pow(k, 4));`
11. 8
13. `const int RATE = 12;`
15. `east = west = north = south = 1;`
17. No, a named constant must be initialized at the time it is defined. It cannot be assigned a value at a later time.

19.
```
cout << fixed << showpoint << setprecision(4);
cout << setw(12) << profit;
```

**NOTE:** Now that you understand that inputs from the keyboard should *always* be preceded by prompts, the `// with prompt` comment can be omitted from the pseudocode. Beginning with Chapter 3, we have begun omitting it.

21.
```
Input score1
Input score2
Input score3
average = (score1 + score2 + score3) / 3.0
Display average
```

23.
```
Input maxCredit
Input creditUsed
availableCredit = maxCredit – creditUsed
Display availableCredit
```

25. A) `Your monthly wages are 3225`
    B) `6 3 12`
    C) `In 1492 Columbus sailed the ocean blue.`

27. A) `#include <iostream>` is missing.
    Each `cin` and `cout` statement starts with capital C.
    The << operator is mistakenly used with `cin`.
    The assignment statement should read:

    ```
    sum = number1 + number2;
    ```

    The last `cout` statement should have << after `cout`.
    The last `cout` statement is missing a semicolon.
    The body of the main function should be indented within the braces.

    B) The `cin` statement should read:

    ```
    cin >> number1 >> number2;
    ```

    The assignment statement should read:

    ```
    quotient = static_cast<double>(number1) / number2;
    ```

    The last `cout` statement is missing a semicolon.
    There is no `return 0;`

29. A) There shouldn't be a semicolon after the `#include` directive.
    The function header for main should read:

    ```
    int main()
    ```

    The variable `number` is defined, but it is called `number1` in the `cin` statement.
    The combined assignment operator is improperly used. The statement should read:

    ```
    half /= 2;
    ```

There is a logical error. The value divided by 2 should be `number`, not `half`.
The results are never output.
There is no `return 0;`

B) There shouldn't be a semicolon after the `#include` directive.
`name` should be declared as a `string` or a `char` array. If declared as `string`, a `#include <string>` directive is needed.
The statement `cin.getline >> name;` should read

```
cin >> name;
```

The statement `cin >> go;` should read

```
cin.get(go);
```

# Chapter 4

1. relational
3. false, true
5. true, false
7. false
9. `!`
11. `&&`
13. block (or local)
15. `break`
17.
```
if (y == 0)
    x = 100;
```
19.
```
if (score >= 90)
   cout << "Excellent";
else if (score >= 80)
   cout << "Good";
else
   cout << "Try Harder";
```
21.
```
if(x < y)
    q = a + b;
else
    q = x * 2;
```
23. T, F, T
25.
```
if (grade >= 0 && grade <= 100)
     cout << "The number is valid.";
```
27.
```
if (hours < 0 || hours > 80)
     cout << "The number is not valid.";
```

29.
```
if(sales < 10000)
   commission = .10;
else if (sales <= 15000)
   commission = .15;
else
   commission = .20;
```

31. It should read

```
    if (!(x > 20))
```

33. It should use `||` instead of `&&`.

35. A) The first `cout` statement is terminated by a semicolon too early.
    The definition of `score1`, `score2`, and `score3` should end with a semicolon.
    The following statement:

    ```
    if (average = 100)
    ```

    should read:

    ```
    if (average == 100)
    ```

    `perfectScore` is used before it is declared.
    The following `if` statement should not be terminated with a semicolon:

    ```
    if (perfectScore);
    ```

    The conditionally-executed block in the `if` statement shown above should end with a closing brace.

    B) The conditionally-executed blocks in the `if/else` construct should be enclosed in braces.
    The following statement:

    ```
    cout << "The quotient of " << num1 <<
    ```

    should end with a semi-colon, rather than with a <<.

    C) The trailing `else` statement should come at the end of the `if/else` construct.

    D) A `switch case` construct cannot be used to test relational expressions.
    An `if/else` if statement should be used instead.

# Chapter 5

1. increment
3. prefix
5. body
7. pretest
9. infinite (or endless)
11. running total
13. sentinel
15. `do-while`

17. initialization, test, update

19. `break`

21.
```
int num;
cin >> num;
num *=2;
while (num < 50)
{   cout << num << endl;
    num *=2;
}
```

23.
```
for (int x = 0; x <= 1000; x += 10)
    cout << x;
```

25.
```
for (int row = 1; row <= 3; row++)
{   for (int star = 1; star <= 5; star++)
        cout << '*';
    cout << endl;
}
```

27.
```
char doAgain;
int sum = 0;

cout << "This code will increment sum 1 or more times.\n";
do
{   sum++;
       cout << "Sum has been incremented. "
            << "Increment it again(y/n)? ";
       cin  >> doAgain;
} while ((doAgain == 'y') || (doAgain == 'Y'));

cout << "Sum was incremented " << sum << " times.\n";
```

29.
```
for (int count = 0; count < 50; count++)
    cout << "count is " << count << endl;
```

31. Nothing will print. The erroneous semicolon after the `while` condition causes the `while` loop to end there. Because `x` will continue to remain `1`, `x < 10` will remain true and the infinite loop can never be exited.

33. 2 4 6 8 10

35. A) The statement `result = ++(num1 + num2);` is invalid.
    B) The `while` loop tests the variable `again` before any values are stored in it.
    The `while` loop is missing its opening and closing braces.

37. A) The expression tested by the `do-while` loop should be `choice == 1` instead of `choice = 1`.
    B) The variable `total` is not initialized to 0.
    The `while` loop does not change the value of `count`, so it iterates an infinite number of times.

# Chapter 6

1. header
3. `showValue(5);`
5. arguments
7. value
9. local
11. Global
13. local
15. `return`
17. last
19. reference
21. reference
23. parameter lists
25. Arguments appear in the parentheses of a function call. They are the actual values passed to a function. Parameters appear in the parentheses of a function heading. They are the variables that receive the arguments.
27. Function overloading means including more than one function in the same program that has the same name. C++ allows this providing the overloaded functions can be distinguished by having different parameter lists.
29. You want the function to change the value of a variable that is defined in the calling function.
31. Yes, but within that function only the local variable can be “seen” and accessed.
33.
```
double half(double value)
{
   return value / 2;
}
```
35.
```
void timesTen(int num)
{
   cout << num * 10;
}
```
37.
```
void getNumber(int &number)
{
   cout << "Enter an integer between 1 and 100): ";
   cin  >> number;
   while (number < 1 || number > 100)
   {
      cout << "This value is out of the allowed range.\n"
           << "Enter an integer between 1 and 100): ";
   }
}
```

39. A) The data type of `value2` and `value3` must be declared.
    The function is declared `void` but returns a value.
    B) The assignment statement should read:

    ```
    average = (value1 + value2 + value3) / 3.0;
    ```

    The function is declared as a `double` but returns no value.
    C) `width` should have a default argument value.
    The function is declared `void` but returns a value.
    D) The parameter should be declared as:

    ```
    int &value
    ```

    The `cin` statement should read:

    ```
    cin >> value;
    ```

    E) The functions must have different parameter lists.

# Chapter 7

1. Abstract Data Type
3. procedural and object-oriented
5. data and procedures (i.e., functions)
7. instantiation
9. member variables
11. encapsulation
13. member variables, member functions
15. mutator
17. class
19. return
21. destroyed
23. default
25. constructor, destructor
27. public
29. False. It can be both passed to a function and returned from a function.
31. separate (i.e., each in their *own* file)
33. `Canine.cpp`
35. public
37. initialization list, constructor
39. `Inventory trivet = {555, 110};`
41.
```
struct TempScale
{   double fahrenheit;
    double celsius;
};
```

```
struct Reading
{   int windSpeed;
    double humidity;
    TempScale temperature;
};
Reading today;
today.windSpeed = 37;
today.humidity = .32;
today.temperature.fahrenheit = 32;
today.temperature.celsius = 0;
```

43.
```
void inputReading(Reading &r)
{
    cout << "Enter the wind speed: ";
    cin  >> r.windSpeed;
    cout << "Enter the humidity: ";
    cin  >> r.humidity;
    cout << "Enter the fahrenheit temperature: ";
    cin  >> r.temperature.fahrenheit;
    cout << "Enter the celsius temperature: ";
    cin  >> r.temperature.celsius;
}
```

45.
```
union Items
{
    char  alpha;
    int   num;
    long  bigNum;
    double real;
};
Items anItem;
```

47.
```
Inventory(string id = 0, string descrip = “new”, int qty = 0)
{  prodID = id; prodDescription = descrip; qtyInStock = qty; }
```

49. A) The structure declaration has no tag.
    B) The semicolon is missing after the closing brace.

51. A) The `Names` structure needs a constructor that accepts 2 strings.
    B) Structure members cannot be initialized in the structure declaration.

53. A) The semicolon should not appear after the word `DumbBell` in the class declaration.

    Even though the `weight` member variable is private by default, it should be preceded with the `private` access specifier.

    Because the `setWeight` member function is defined outside the class declaration, its function header must appear as:

    ```
    void DumbBell::setWeight(int w)
    ```

    The line that reads: `DumbBell.setWeight(200);`
    should read: `bar.setWeight(200);`

    Because the `weight` member variable is private, it cannot be accessed outside the class, so the `cout` statement cannot legally output `bar.weight`. There needs to be a public `getWeight()` function that the main program can call.

B) Constructors must be public, not private.

Both constructors are considered the default constructor. This is illegal since there can be only one default constructor.

All the parameters in the `Change` function header should have a data type.

55. A) The nouns are

| | | | |
|---|---|---|---|
| Bank | Savings Account | Money | Interest rate |
| Account | Checking Account | Balance | |
| Customer | Money market account | Interest | |

After eliminating duplicates, objects, and simple values that can be stored in class variables, the potential classes are: `Bank`, `Account`, and `Customer`.

B) The only class needed for this particular problem is `Account`.

C) The `Account` class must know its balance and interest rate.

The `Account` class must be able to handle deposits and withdrawals and calculate interest earned. It is this last capability, calculating interest earned, that this application will use.

## Chapter 8

1. size declarator
3. subscript
5. size declarator, subscript
7. initialization
9. initialization list
11. subscript
13. value
15. multidimensional
17. two
19. columns
21. A) 10 B) 0 C) 9 D) 40
23. A) 3 B) 0
25. the starting address of the array
27. A) 8 B) 10 C) 80 D) `sales[7][9] = 3.52;`

29.
```
Car forSale[35] = { Car("Ford", "Taurus",  2006, 21000),
                    Car("Honda","Accord",  2004, 11000),
                    Car("Jeep", "Wrangler",2007, 24000) };
```

31.
```
for (int index = 0; index < 25; index++)
     array2[index] = array1[index]
```

33.
```
int id[10];
double grossPay[10];
for (int emp = 0; emp < 10; emp++)
    cout << id[emp] << "   " << grossPay[emp] << endl;
```

35.
```
struct PopStruct
{  string name;
   long   population;
};
PopStruct country[12];
ifstream dataIn;
dataIn.open("pop.dat");
for (int index = 0; index < 12; index++)
{  getline(dataIn, country[index].name);
   dataIn >> country[index].population;
   dataIn.ignore();
}
dataIn.close();
```

37. A) The size declarator cannot be a variable.
    B) The size declarator cannot be negative.
    C) The initialization list must be enclosed in braces.

39. A) The parameter should be declared as `int nums[]`.
    B) The parameter must specify the number of columns, not the number of rows. Also, a second parameter is needed to specify the number of rows.

# Chapter 9

1. linear
3. linear
5. N/2
7. first
9. 1/8
11. ascending
13. one
15. there were no number exchanges on the previous pass
17. Bubble sort normally has to make many data exchanges to place a value in its correct position. Selection sort determines which value belongs in the position currently being filled with the correctly ordered next value and then places that value directly there.
19.

| Array Size → | 50 Elements | 500 Elements | 10,000 Elements | 100,000 Elements | 10,000,000 Elements |
|---|---|---|---|---|---|
| Linear Search (Average Comparisons) | 25 | 250 | 5,000 | 50,000 | 5,000,000 |
| Linear Search (Maximum Comparisons) | 50 | 500 | 10,000 | 100,000 | 10,000,000 |
| Binary Search (Maximum Comparisons) | 6 | 9 | 14 | 17 | 24 |

21. A) Map directly from the desired ID to the array location as follows:
    *index = desiredID –101*

    B) Do a linear search starting from the last array element and working backwards until the item is found or until a smaller ID is encountered, which means the desired ID is not in the array. Here is the pseudocode:

```
index = 299        // start at the last element
position = -1
found = false
While index >= 0 and array[index].customerID >= desiredID
                 and not found
   If array[index].customerID = desiredID
      found = true
      position = index
   End If
   Decrement index
End While
Return position
```

# Chapter 10

1. address
3. pointer
5. pointers
7. new
9. null
11. new
13. Sending `*iptr` to `cout` will display 7. Sending `iptr` to `cout` will display the address of `x`.
15. You can increment or decrement a pointer using `++` and `--`, you can add an integer to a pointer, and you can subtract an integer from a pointer.
17. 8
19. If `new` fails to allocate the requested amount of memory, it throws the `bad_alloc` exception. In programs compiled with older compilers, new returns the value 0.
21. `delete` is used to deallocate memory allocated by `new`.
23. `const int *p;`
25. `change(&i);`
27.
```
void exchange(int *p, int *q)
{
    int temp = *p;
    *p = *q;
    *q = temp;
}
```

29. A) 30
    B) 30
    C) 0
    D) 0
    E) 20
    F) 10
    G) 10
    H) 20

# Chapter 11

1. static
3. static
5. friend
7. Memberwise assignment
9. this
11. postfix increment (or decrement)
13. has-a
15. copy constructor
    overloaded = operator
    overloaded = operator
    copy constructor
17. Place the static keyword in the function's prototype. Calls to the function are performed by connecting the function name to the *class* name with the scope resolution operator.
19. In object composition, one object is a nested inside another object, which creates a has-a relationship. When a class is a friend of another class, there is no nesting. If a class A is a friend of a class B, member functions of A have access to all of B's members, including the private ones.
21. If a pointer member is used to reference dynamically allocated memory, a memberwise assignment operation will only copy the contents of the pointer, not the section of memory referenced by the pointer. This means that two objects will exist with pointers to the same address in memory. If either object manipulates this area of memory, the changes will show up for both objects. Also, if either object frees the memory, it will no longer contain valid information for either object.
23. If an object were passed to the copy constructor by value, a copy of the argument would have to be created before it can be passed to the copy constructor. But then the creation of the copy would require a call to the copy constructor with the original argument being passed by value. This process will continue indefinitely.
25.
```
Dollars Dollars::operator++();     // Prefix
Dollars Dollars::operator++(int);  // Postfix
```
27.
```
ostream &operator<<(ostream &strm, Length obj);
```

29. The overloaded operators offer a more intuitive way of manipulating objects, similar to the way primitive data types are manipulated.
31. members
33. `Pet`
35. private
37. inaccessible, private, private
39. inaccessible, protected, public
41. last
43. A) The first line of the class declaration should read

    ```
    class Car : public Vehicle
    ```

    Also, the class declaration should end in a semicolon.

    B) The first line of the class declaration should read

    ```
    class Truck : public Vehicle
    ```

# Chapter 12

1. `isupper`
3. `isdigit`
5. `toupper`
7. `cctype`
9. concatenate
11. `strcpy`
13. `strcmp`
15. `atoi`
17. `atof`
19. 
    ```
    char lastChar(char *str)
    {  //go to null terminator at end
       while (*str != 0)
          str++;
       //back up to last character
       str--;
       return *str;
    }
    ```
21. h
23. 9
25. Most compilers will print “not equal”. Some compilers store only one copy of each literal string: such compilers will print “equal” because all copies of “a” will be stored at the same address.
27. abrasion

29. A) This is probably a logic error because C-strings should not be compared with the == operator
    B) `atoi` converts a string to an integer, not an integer to a string.
    C) The compiler will not allocate enough space in `string1` to accommodate both strings.
    D) `strcmp` compares C-strings, not characters.

# Chapter 13

1. file name
3. close
5. `ifstream`, `ofstream`, `fstream`
7. `ifstream`
9. `ofstream people("people.dat");`
11. `fstream places("places.dat");`
13. `pets.open("pets.dat", ios::in);`
    `fstream pets("pets.dat", ios::in);`
15. null or 0
17. `cout`
19. `getline`
21. `put`
23. text, ASCII text
25. structures
27. `read`
29. sequential
31. `seekg`
33. `tellg`
35. `ios::beg`
37. `ios::cur`
39. Open the file in binary mode, seek to the end, and then call tellg to determine the position of the last byte:

```
ifstream inFile(fileName, ios::binary);
inFile.seekg(0L, ios::end);
long len = inFile.tellg();
```

41. Open the two files in binary mode, the first file for input and the second file for output. Seek to the end of the first file, and then keep backing up in the first file while writing to the second.

```
fstream inFile(file1name, ios::in | ios::binary);
fstream outFile(file2name, ios::out | ios::binary);
char ch;
// seek to end of source file
// and then position just before that last
// character
inFile.seekg(0L, ios::end);
inFile.seekg(-1, ios::cur);
while (true)
{
   // we are positioned before a character we need to read
   inFile.get(ch);
   outFile.put(ch);
   // back up two characters  to skip the character just read
   // and go to the character before it.
   inFile.seekg(-2, ios::cur);
   if (inFile.fail())
      break;
}
```

43. A) File should be opened as

```
fstream file("info.dat", ios::in | ios::out);
```

or

```
fstream file;
file.open("info.dat", ios::in | ios::out);
```

B) Should not specify `ios::in` with an `ofstream` object. Also, the `if` statement should read

```
if (!File)
```

C) File access flags must be specified with `fstream` objects.

D) Should not write to a file opened for input. Also, the << operator should not be used on binary files.

E) The while statement should read

```
while(!dataFile.eof())
```

F) The last line should be

```
dataFile.getline(line, 81, '\n');
```

G) The `get` member function that takes a single parameter cannot be used to read a string: it can only read single characters.

H) The file access flag should be `ios::in`. Also, the `put` member function cannot be used to write a string.

I) The file access flag should be `ios::out`. Also, the last line should read

```
dataFile.write(&dt, sizeof(date));
```

J) The `seekp` member function should not be used since the file is opened for input.

# Chapter 14

1. Indirect recursion. There are more function calls to keep up with.
3. When the problem is more easily solved with recursion, and the recursive calls do not repeatedly solve the same subproblems.
5. direct
7. A) 55

   B)
   ```
   **********
   *********
   ********
   *******
   ******
   *****
   ****
   ***
   **
   *
   ```

   C) evE dna madA

# Chapter 15

1. abstract class
3. abstract
5. compile
7. polymorphism
9. virtual
11. multiple
13. yes
15. yes
17. `pAnimal = new Dog; pDog = static_cast<Dog *>(pAnimal);`
19. A pure virtual function cannot have a body, and the function `myFun` has no return type.

# Chapter 16

1. throw point
3. catch
5. template prefix
7. vector, list, or any sequence container
9. iterators

11. This solution uses recursion to perform the reversal. It needs the inclusion of the STL algorithm header file to allow use of swap.

```
template<class T>
void reverse(T arr[ ], int size)
{  if (size >= 2)
   {  swap(arr[0], arr[size-1]);
      reverse(arr+1, size-2);
   }
}
```

13. The stiring of characters stored in the array will be reversed.

15. A) The try block must appear before the catch block.
 B) The cout statement should not appear between the try and catch blocks.
 C) The return statement should read `return number * number;`
 D) The type parameter, T, is not used.
 E) The type parameter, T2 is not used.
 F) The declaration should read `SimpleVector<int> array(25);`
 G) The statement should read `cout << valueSet[2] << endl;`

# Chapter 17

1. head pointer
3. NULL or 0
5. Inserting
7. circular
9.

```
void printFirst(ListNode *ptr)
{
   if (!ptr) { cout << "Error"; exit(1);}
   cout << ptr->value;
}
```

11.

```
double lastValue(ListNode *ptr)
{
   if (!ptr) { cout << "Error"; exit(1);}
   if (ptr->next == NULL)
      return ptr->value;
   else
      return lastValue(ptr->next);
}
```

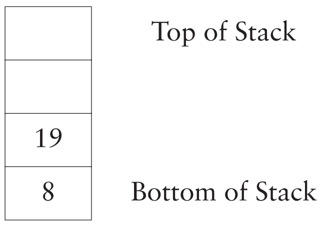
13.

```
ListNode *ListConcat(ListNode *list1, ListNode *list2)
{
   if (list1 == NULL)
      return list2;
   // Concatenate list2 to end of list1
   ListNode *ptr = list1;
   while (ptr->next != NULL)
      ptr = ptr->next;
   ptr->next = list2;
   return list1;
}
```

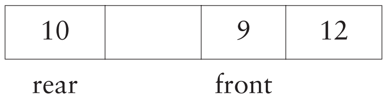
15. 56.4
17. A) The `printList` function should have a return type of `void`. Also, the use of the head pointer to walk down the list destroys the list: use an auxiliary pointer initialized to head instead.
    B) Eventually the pointer `p` becomes `NULL`, at which time the attempt to access `p->next` will result in an error. Replace the test `p->next` in the `while` loop with `p`. Also, the function fails to declare a return type of `void`.
    C) The function should declare a return type of `void`. Also, the function uses `p++` erroneously in place of `p = p->next` when attempting to move to the next node in the list.
    D) Replace `nodeptr->next = NULL;` with `delete nodeptr;`

# Chapter 18

1. Last In First Out
3. A static stack has all its storage allocated at once, when the stack is created. A dynamic stack allocates storage for each element as it is added. Normally, static stacks use array-based implementations, whereas dynamic stacks use linked lists.
5. It takes an existing container and implements a new interface on top of it to adapt it to a different use.
7. First In First Out
9. the front of the queue
11. lists and deques
13.



15. Assuming a circular array buffer:



17. Use two stacks, a main stack and an auxiliary stack. The main stack will store all items that are currently enqueued.
    - To enqueue a new item, push it onto the main stack.
    - To dequeue an item, keep popping items from the main stack and pushing them onto the auxiliary stack until the main stack is empty, then pop and store the top element from the auxiliary stack into some variable X. Now keep popping items from the auxiliary stack and pushing them back onto the main stack till the auxiliary stack is empty. Return the stored item X.
    - To check if the queue is empty, see if the main stack is empty.

# Chapter 19

1. root node
3. leaf node
5. inorder, preorder, and postorder
7.
```
struct TreeNode
{
   int value;
   TreeNode *left, *middle, *right;
};
```
9. To traverse a ternary tree in preorder, visit the root, then traverse the left, middle, and right subtrees.
```
preorder(ternarytree)
   If (ternarytree != NULL)
      visit the root
      preorder left subtree of ternarytree
      preorder middle subtree of ternarytree
      preorder right subtree of ternarytree
   End If
End preorder
```
11. We must decide whether to visit the root right after the traversal of the left subtree or right after the traversal of the middle subtree.
13.
```
int largest(TreeNode *tree)
   Set a pointer p to the root node of tree
   While node at p has a right child Do
      Set p to the right child of the node at p
   End While
   return value in the node at p
End largest
```
15.
```
int smallest(TreeNode *tree)
   Set a pointer p to the root node of tree
   While node at p has a left child Do
      Set p to the left child of the node at p
   End While
   return value in the node at p
End smallest
```
17. 3 7 9 10 12 14 18 20 22 24 30
19. 3 10 9 7 14 20 18 30 24 22 12