**Chapter 6 Stacks**

**1. What operation in the ADT stack is used to add an entry to the top of the stack?**

the push operation

**2. What operation in the ADT stack is used to retrieve the value at the top of the stack?**

the peek operation

**3. What operation in the ADT stack is used to remove the entry at the top of the stack?**

the pop operation

**4. If you push the characters 'A', 'B' and 'C' in order onto a stack of characters, in what order will they be removed?**

After inserting the 3 characters, the stack will look like:

   top --> 'C'   'B'   'A'

Since you can only remove entries from the top, the order will be the reverse of the order in which they were inserted. They are removed in the order 'C', 'B' and finally 'A'.

**5. Assume that the ADT stack is implemented as a class template called Stack. Write C++ code that uses a stack to input 3 integers and then displays them in reverse order.**

   Stack <int> s;

int num;

for (int i=0; i<3; i++)

{

cout << "Enter an integer: ";

cin >> num;

s.push(num);

}

while(!s.empty())

{

num = s.peek();

s.pop();

cout << num << endl;

}

**6. If *stack1* starts as an empty stack, what does it look like after the following sequence of operations:**

**stack1.push(1)**

**stack1.push(2)**

**stackTop = stack1.peek()**

**stack1.pop()**

**stack1.push(stackTop+1)**

The resulting stack is:

   top --> 3   1

### Chapter 7 Stack Implementations

**1. Assume the array-based implementation of a stack (ArrayStack class template). For each operation on the stack, draw a diagram showing what array items looks like after the operation is executed. Include the value of top and indicate if the operation returns a value.**

ArrayStack<int> stack1;

+---------------------------------------+

items | ? | ? | ? | ? | ? | ? | ? | ? | top = -1

+---------------------------------------+

[0] [1] [2] [3] [4] [5] [6] [7]

int stackTop;

stack1.push(1);

+---------------------------------------+

items | 1 | ? | ? | ? | ? | ? | ? | ? | top = 0

+---------------------------------------+

[0] [1] [2] [3] [4] [5] [6] [7]

   stack1.push(2);

+---------------------------------------+

items | 1 | 2 | ? | ? | ? | ? | ? | ? | top = 1

+---------------------------------------+

[0] [1] [2] [3] [4] [5] [6] [7]

stackTop = stack1.peek();

+---------------------------------------+

items | 1 | 2 | ? | ? | ? | ? | ? | ? | top = 1 returns 2

+---------------------------------------+

[0] [1] [2] [3] [4] [5] [6] [7]

stack1.pop();

+---------------------------------------+

items | 1 | 2 | ? | ? | ? | ? | ? | ? | top = 0 Note that the element at index 1 still holds 2,

+---------------------------------------+ but it is not part of the stack

[0] [1] [2] [3] [4] [5] [6] [7]

stack1.push(stackTop+1);

+---------------------------------------+

items | 1 | 3 | ? | ? | ? | ? | ? | ? | top = 1

+---------------------------------------+

[0] [1] [2] [3] [4] [5] [6] [7]

**2. Repeat the previous exercise for the link-based implementation of the Stack (LinkedStack class template). Include the topPtr and indicate if the operation returns a value.**

ArrayStack<int> stack1;

+---+

topPtr | / | "/" represents nullptr

+---+

int stackTop;

stack1.push(1);

+---+ +---+---+

topPtr | --+---> | 1 | / |

+---+ +---+---+

stack1.push(2);

+---+ +---+---+ +---+---+

topPtr | --+---> | 2 | --+---> | 1 | / |

+---+ +---+---+ +---+---+

stackTop = stack1.peek();

+---+ +---+---+ +---+---+

topPtr | --+---> | 2 | --+---> | 1 | / | peek() returns 2

+---+ +---+---+ +---+---+

stack1.pop();

+---+ +---+---+

topPtr | --+---> | 1 | / |

+---+ +---+---+

stack1.push(stackTop+1);

+---+ +---+---+ +---+---+

topPtr | --+---> | 3 | --+---> | 1 | / |

+---+ +---+---+ +---+---+

**3. For debugging purposes, it would be nice to have a method that displays the entries in a stack from top to bottom. Add a method called display to the array-based implementation for a Stack (ArrayStack class template) that does this**

One implementation is:

template <class ItemType>

void ArrayStack<ItemType>::display() const

{

// Display items top to bottom

for (int i = top; i >= 0; i--)

cout << items[i] << " ";

cout << endl;

} // end display

4. Add a method called display to the linked-based implementation for a Stack (LinkedStack class template) that displays the entries in a stack from top to bottom.

One implementation is:

template <class ItemType>

void LinkedStack<ItemType>::display() const

{

// Display items top to bottom (top is in the first node of the chain)

Node<ItemType>\* curPtr = topPtr;

while (curPtr != nullptr)

{

cout << curPtr->getItem() << " ";

curPtr = curPtr->getNext();

} // end while

cout << endl;

} // end display

### Chapter 8 Lists

**1. Assume that the class template List implements the ADT list described in Chapter 8. Write a C++ function called swap that exchanges the values at positions i and jof a list. You can assume that the list has entries at positions i and j.**

// Swaps the i-th and j-th items in the list aList.

template <class ItemType>

void swap(List aList, int i, int j)

{

// Copy the i-th and j-th items.

ItemType ithItem = aList.getEntry(i);

ItemTypejthItem = aList.getEntry(j);

// Replace the i-th item with the j-th item.

aList.remove(i);

aList.insert(i, jthItem);

// Replace the j-th item with the i-th item.

aList.remove(j);

aList.insert(j, ithItem);

}

**2. Assume that list1 is a list that contains this sequence of 5 objects: a b c d e. What does list1 contain after executing the following insert operation?**

**list1.insert(5, w)**

   a b c d w d

**3. Assume that list2 is a list that contains this sequence of 5 objects: a b c d e. What does list2 contain after executing the following insert operation?**

**list1.insert(6, w)**

   a b c d e w

### Chapter 9 List Implementations

**1. When we load data into a C++ array, we usually start at index zero. Why does the array-based implementation for ADT List ignore index 0 in the array items and insert the first entry at index 1?**

Answer:

In the specification for the ADT List, we chose to use a position number of 1 for the first entry, 2 for the second number, etc. If we used index 0 to store the first entry (the entry at position 1), that would make coding for the class methods a little more complicated. To keep things simple and help avoid errors, we chose to store the entry at position 1 at index 1, etc.

**2. In the array-based implementation of a List (class template ArrayList), if a list currently holds 6 entries, what are the legal positions where you could insert the new entry?**

Answer:

Positions 1 through 7.

**3. Assume the array-based implementation of a List (class template ArrayList). If myList is a list of strings that currently holds 8 entries and we execute the following operations in sequence:**

**myList.insert(2,** "Hello");  
**myList.insert(8, "World");**

**a. Is the first insert valid? Is the second insert valid?**

The first insert is valid because the valid insertion positions are 1 through 9. The second insert is valid because the valid insert positions are 1 through 10.

**b. Which insert takes less time to execute, or do they both take the same amount of time?**

The first insert will require the entries in positions (indexes) 2 through 8 to be moved one position to the right. The second insert will require the entries at positions 8 and 9 to be moved to the right. The first insert will take more time to execute than the second insert.

**4. Assume the array-based implementation of a List (class template ArrayList). If myList is a list of strings that currently holds 8 entries and we execute the following operations in sequence:**

**string first = myList.getEntry(2);**

**string next =myList.getEntry(8);**

**a. Is the first operation valid? Is the second operation valid?**

Both operations are valid because the legal positions are 1 through 8.

**b. Which operation takes less time to execute, or do they both take the same amount of time?**

For the array-based implementation, retrieving an entry from the array takes a constant amount of time. The time does not depend on the position.

**5. Assume the link-based implementation of a List (class template LinkedList). If myList is a list of strings that currently holds 8 entries and we execute the following operations in sequence:**

**myList.insert(2, "Hello");**

**myList.insert(8, "World");**

**a. Is the first insert valid? Is the second insert valid?**

The position numbers are valid, so both inserts are valid.

**b. Which insert takes less time to execute, or do they both take the same amount of time?**

For the link-based implementation, you do not have to move any data to perform an insertion. However, you do have to traverse the list until you find the insertion point. The second insert operation will take more time because you haev to traverse more of the list.

**6. Assume the link-based implementation of a List (class template LinkedList). If myList is a list of strings that currently holds 8 entries and we execute the following operations in sequence:**

**string first = myList.getEntry(2);**

**string next =myList.getEntry(8);**

**a. Is the first operation valid? Is the second operation valid?**

Both are valid.

**b. Which operation takes less time to execute, or do they both take the same amount of time?**

The second operation takes more time because you have to traverse more of the list to find the entry at position 8.

**7. The method clear from the link-based implementation of a List (class template LinkedList) repeatedly calls remove(1). To avoid the overhead of repeated method calls, rewrite this method to deallocate the nodes of the list directly without calling remove.**

**template <class ItemType>**

**void LinkedList<ItemType>::clear()**

**{**

**while (!isEmpty())**

**remove(1);**

**} // end clear**

Answer:

template <class ItemType>

void LinkedList<ItemType>::clear()

{

Node<ItemType>\* ptr = headPtr;

while (ptr != nullptr)

{

headPtr = headPtr->getNext();

delete ptr;

ptr = headPtr;

}

} // end clear

### Chapter 10 Algorithm Efficiency

**1. The growth-rate function of an algorithm is a formula that expresses the number of major operations in an algorithm as a function of the size n of the problem. Big-O notation is a simplified version of the growth rate function. The textbook does not use this term, but the growth-rate of an algorithm expressed in Big-O notation is often referred to as the time complexity of the algorithm. I expect you to know which functions grow faster and which functions grow slower For example, see:**

[Big-O Algorithm Complexity](http://bigocheatsheet.com/)

**Express each of the following growth-rate functions in Big-O notation:**

**a. 8 x n2 - 3 x n**       Answer: O(n2)

**b. 4 x log2 n + 20**    Answer: O(log n)

**c. 7 x log2 n + n**Answer: O(n)

**2. Which type of analysis is most commonly used - best-case analysis, average-case analysis or worst-case analysis?**

Answer:

Generally the worst-case analysis is used. Occasionally the average-case analysis is used. The best-case analysis is rarely useful.

**3. What is the Big-O time complexity of the sequential search? What is the Big-O time complexity of the binary search?**

The sequential search is O(n). The binary search is O(log n). For large arrays, binary search is much faster.

**4. For the following code segment, what is the worst-case computing time as a function of n (the problem size): O(1), O(log n), O(n), O(n2) or O(n3)?**

**// Matrix addition**

**for (int i = 0; i < n; i++)**

**for (int j = 0; j < n; j++)**

**c[i][j] = a[i][j] + b[i][j];**

When a loop that depends on n is nested inside another loop that depends on n, the code inside the inner-most loop is executed n 2 times. This code is O(n2).

**5. For the following code segment, what is the worst-case computing time as a function of n (the problem size): O(1), O(log n), O(n), O(n2) or O(n3)?**

**for (int i=0; i < n; i++)**

**for (int j = 0; j < 12; j++)**

**a[i] \*= (1 + 0.01);**

The execution time for the inner-most loop does not depend on n. It executes in a constant amount of time. The outer loop does depend on n, so this code is O(n).