|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Part** | **1** | **2** | **3** | **4** | **Total** |
| *maximum* | **25** points | **25** points | **25** points | **25** points | **100**G101010 pointsG |
| ***Your Score*** |  |  |  |  |  |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Stacks**

Reading Assignment: Thoroughly read Chapter 7 in the course textbook.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Part 1 Glossary Terms**

Define, in detail, each of these glossary terms from the realm of computer programming logic and design and computer topics, in general. If applicable, use examples to support your definitions. Consult your notes or course textbook(s) as references or by visiting Web sites such as: [**http://www.ask.com**](http://www.ask.com),[**http://www.bing.com**](http://www.bing.com), [**http://www.webopedia.com**](http://www.webopedia.com)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(a) Backtracking Routine**

|  |
| --- |
| A backtracking routine (or “backtracking algorithm”) traces back its steps whenever it reaches an unsuitable outcome, much like a hiker in the woods would backtrack their steps to the previous trail junction if they got lost. The backtracking routine picks a path, potentially at random, and if the end result of that path is undesirable, will backtrack to the last point where there was an unexplored alternative, and try that route instead.   The role of stacks in backtracking algorithms is to “remember alternative states at each juncture” (Lambert 169), popping items off the stack and examining each to see if the item state represents an “ending” or desirable state, and returning the successful conclusion if so. Else, the algorithm marks the state as visited and pushes onto the stack all unvisited adjacent states, until it reaches a successful conclusion or reaches the end of the stack (at which point it returns an “unsuccessful” conclusion).   Such backtracking routines are often used in games and puzzle progams. |

**(b) LIFO Protocol**

|  |
| --- |
| The “Last In, First Out” or LIFO protocol refers to a collection protocol that restricts users of a collection to accessing only the last item in the collection, and is typically accessed by using a “pop” operation. A good visual example would be a deck of cards in a card game where the players are only able to draw cards from the top of the deck. |

**(c) Peek Operation**

|  |
| --- |
| A “peek” operation allows clients to access the topmost (the last) item in a stack without removing it. It does not mutate the stack. |

**(d) Pop Operation**

|  |
| --- |
| A “pop” operations pops the last item off the top of the stack and returns it. It’s a mutative operation as it removes the item of the stack. |

**(e) Postfix Expression**

|  |
| --- |
| A “postfix” expression is what computers will translate an “infix” expression into when calculating arithmetic. While an infix expression has operators between operands (e.g. 2+2) a postfix expression has operators following the operands (e.g., 2 2 +). The ‘postfix’ format is easier for a computer to evaluate, and the postfix form of a infix expression is created using stack methods during compile time. |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Part 2 True / False Exercises**

For each of these exercises, enter True or False in the spaces provided.

**FALSE** **(1)** Access to a stack is restricted to the bottom end.

**TRUE** **(2)** Stacks adhere to a LIFO protocol.

**FALSE**  **(3)** The operator in an infix expression follows the operands.

**TRUE** **(4)** Using a stack is a good approach to determining if the use of brackets in an expression is balanced.

**FALSE** **(5)** When using a stack to determine if brackets are balanced in an expression, the stack should not be empty when you reach the end of the expression.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Part 3 Multiple Choice Exercises**

Select the correct response or responses.

**(1)** Which of the following is NOT an application of stacks in computer science?

**(a) implementation of a hash - based database search function**

(b) managing computer memory in support of function calls

(c) supporting the undo feature in a text editor

(d) maintaining a history of visited links by a web browser

**(2)** In what way does not the Python list data structure accurately emulate a stack?

**(a) the append method pushes elements onto the stack**

**(b) the pop method removes and returns the top element**

(c) a list uses an array as the underlying data structure

(d) list methods allow manipulation of items at any position

**(3)** If the current state of the stack is [ x , y , z ] where x is the bottom of the stack and z is the top of the stack, what is the state of the stack and the value returned after a pop operation?

**(a) the state is [ x , y ]; z is returned**

(b) the state is [ x , y , z ]; z is returned

(c) the state is [ y , z ]; x is returned

(d) the state is [ x , y , z ]; x is returned

**(4)** What is the function of the peek method in a stack implementation?

(a) to return the bottom item without removing it

**(b) to return the top item without removing it**

(c) to return the top item and remove it from the stack

(d) to return the bottom item and remove it from the stack

**(5)** If the portion of the postfix expression scanned so far is 7 6 2 + , what is the result when the operator is applied to the operands?

(a) 8 (b) 15 (c) 13 (d) 9

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Part 4 Programming Exercises**

**(1)** **( A Postfix Expression )**

If the entire postfix expression is 9 1 4 + - 2 \* , what is the final value?

**8**

**(2)** **( A Postfix Expression )**

What is the resulting postfix expression from the following infix expression?

( 12 + 5 ) \* 2 – 3  
  
 **12 5 + 2 \* 3 -**

**(3)** **( An Infix Expression )**

What is the resulting infix expression from the following postfix expression?

17 4 - 6 + 8 \*

**(17 - 4 + 6) \* 8**

**(4)** **( Backtracking Implemented by a Stack )**

Often much faster than a Brute Force Method, a Backtracking Process is implemented using a stack. With Backtracking, we require to return upon reaching a particular point or situation. For this reason, we will keep track of what we have processed in previous steps and hence the LIFO

( Last In First Out ) pattern helps us accomplishing this.

For the example code segment below, explain how backtracking is used.

|  |
| --- |
| **def permutation(list, start, end):**  **# function to display all permutations of a given list**  **if (start == end) :**  **print (list)**  **else :**  **for i in range(start, end + 1) :**  **list[start], list[i] = list[i], list[start]**  **permutation(list, start + 1, end)**  **list[start], list[i] = list[i], list[start]**  **testList = [4, 5, 6, 7]**  **permutation(testList, 0, 3)** |

**Backtracking is used in the program above to recursively look for a successful end condition (start == end) and return the state of the list (a possible permutation) once that end condition is met. By looping through different start/end values and recursively calling the program, this end condition can be met.  
  
(5) ( The Coin Changer Program )**

Examine the Coin Changer Program which appears below.

|  |
| --- |
| **coins = [50, 25, 10, 5, 1]**  **amount = 0; count = 0; i = 0**  **print ("please enter the amount you wish to change : ")**  **amount = int(input())**  **print ("----------------------------")**  **for i in range(len(coins)) :**  **count = int(amount / coins[i])**  **if (count != 0) :**  **print ("count of {0} cent(s) : {1}" . format(coins[i], count))**  **amount %= coins[i]** |

Does the above routine constitute a Stack type process? Explain your answer.

**The above routine doesn’t constitute a stack like process because it selectively returns (or, just prints) an items from the collection, which doesn’t follow the LIFO principle.**