# Worksheet 3 Stacks

**Task 1 Crushing cars**

1. (a) Complete the following to show the operations implemented on a collection of burnt-out cars. The stack can hold a maximum of 6 items.

**Cars:** Mondeo, Golf, Fiesta, Punto, Civic, Porsche

Representation of the stack drawn both horizontally and vertically are shown. Show the state of the stack after each push and pop operation in both representations, and in the first table, show any results returned.

|  |  |  |
| --- | --- | --- |
|  | **Stack** | **Result returned** |
| carStack 🡨 Stack() | [] |  |
| carStack.push(Mondeo) |  |  |
| carStack.push(Golf) |  |  |
| carStack.isEmpty() |  |  |
| carStack.push(Fiesta) |  |  |
| carStack.push(Punto) |  |  |
| carStack.pop() |  |  |
| carStack.push(Civic) |  |  |
| carStack.push(Porsche) |  |  |
| carStack.isFull() |  |  |
| carStack.pop() |  |  |
| carStack.pop() |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

2. Complete the pseudocode below for a program which uses a stack to test an input string to determine whether it is a palindrome (the same backwards and forwards, like “peep”)

Assume that a class **Stack** implements the operations in the table in question 1.

OUTPUT “Please enter a word or phrase to be tested”)

INPUT myString

list1 🡨 list(myString) #convert myString to a list of characters

numChars 🡨 len(list1)

s 🡨 Stack()

**Task 2 Subroutine snake**

3. (a) Fill in the return addresses to show the state of the stack created during the execution of the progam beginning at **main()**. Each {stack frame} is indicated with curly brackets and contains parameters, local variables and return addresses.

|  |  |
| --- | --- |
| 78 | SUB subA(p1) |
| 79 | subB (p1, 4) |
| 80 | … |
| 97 | ENDSUB |
|  |  |
| 99 | SUB subB(p10, y) |
| 100 | x 🡨 12 |
| 101 | subC (p10, x) |
| 102 | … |
| 110 | ENDSUB |
|  |  |
| 144 | SUB subC(p3, p4) |
| 145 | … |
| 146 | … |
| 151 | ENDSUB |
|  |  |
| 222 | main() |
| 223 | subA(7) |
| 224 | … |

|  |  |
| --- | --- |
| **Line** | **Stack** |
| 223 | {call subA: parameter = 7, return address = } |
| 78 |  |
| 79 | {call subA: p1 =7, return address = } { call subB: p10 = 7, y =4, return address = } |
| 99 |  |
| 101 | {call subA: p1 =7, return address = } {call subB: p10 = 7, y =4, x = 12, return address = }{call subC: p3 = , p4 = , return address = } |
| 144 |  |
| 145 |  |
| 151 | (pop the stack, go to the return address) |
| 102 | {call subA: 7, return address = } {call subB: 7, 4, , return address = } |
| 110 | (pop the stack, go to the return address) |
| 80 | {call subA: p1 =7, return address = } |
| 97 | (pop the stack, go to the return address) |
| 224 | … |