UNIVERSIDADE DA CORUÑA

FACULTY OF COMPUTER SCIENCE
Department of Computer Science and Information Technologies
Programming II

Exercises: Applications of Stacks and Queues

The present handout contains a couple of exercises about code development using the Stack and Queue ADTs. Students can assume that those operations described in the corresponding Informal Specifications documents, which have been used in class and are available in CampusVirtual, are already available.

Exercise 1. Applications of stacks: Symbol balancing. When the code of our programs includes an algebraic expression such as $[(a+b)*(c+d)]*f$, it is necessary for the compiler to check that the balance of opening/closing symbols such as (), {}, [] is correct.

For example, the following sequence is valid

$$\ldots[\ldots(\ldots)\ldots]\ldots$$

but not this

$$\ldots[\ldots(\ldots]\ldots)\ldots$$

The presence of a misplaced symbol in the program source code could produce hundreds of totally meaningless compiler error messages.

A stack is a useful tool for checking the balance of symbols. Let us assume an algebraic expression for which we want to check its symbol balance. The pseudo-code of the program to do it is the following:

- Create an **empty stack**.
- Read characters until reaching the end of the expression.
  - If the character read is an opening symbol, it is pushed into the stack
  - If the character read is a closing symbol,
    - If the stack is empty, an error is generated
    - If the stack is not empty, an element is popped out from the stack
      - If the opening token just popped out does not correspond to the closing token, an error is generated
      - Otherwise, we continue
- If the stack is not empty, an error is generated

As an exercise, students are asked to:

1. Analyze the behavior of this algorithm by applying it to the following expressions:

   a) $[(a+b)*(c+d))*f$
   b) $[(a+b)*(c+d)]*f$
   c) $[(a+b)*(c+d)]*f]$

d) $[((a+b)*(c+d)*f]$

2. Write the C code of a function that implements this algorithm. The function will receive as input a string with the expression to be checked, and it will return a bool indicating whether the balance of symbols (), {} and [] is correct or not.

Exercise 2. Write a function `void invert_stack(tStack* S)` that, using a Queue ADT as an auxiliary structure, modifies a stack by inverting the order of its content. For example, given a stack containing the elements `ABCD`, the function would return the same stack now containing `DCBA`.

Solutions

```
#include "static_stack.h"
#include <stdio.h>
#include <stdbool.h>
#include <string.h>

bool balanceChecking (char exp[]) {
        /* Objetivo: Dada una expresión aritmética determinar si está correctamente parentizada
        Entrada: expresión artimetica. Salida: true si la expresión está equilibrada, false en caso contrario
        PreCD: suponemos memoria suficiente para insertar en la pila */

        tStack stack;
        bool goOn;
        int i, l;
        char p;

        createEmptyStack(&stack);
        goOn = true;
        i = 0;
        l = strlen(exp);
        while (goOn && (i <= l)) {
               switch(exp[i]) {
                      case '(' :
                      case '[' :
                      case '{' :
                             push(exp[i], &stack);
                             break;
                             case ')':
                             case ']':
                             case '}':
                                    if (isEmptyStack(stack))
                                           goOn = false;
                                    else{
                                           p=peek(stack);
                                           pop(&stack);
                                           switch(exp[i]) {
                                                  case ')':
                                                         if (p != '(')
                                                                goOn = false;
                                                         break;
                                                  case ']':
                                                         if (p != '[')
                                                                goOn = false;
                                                         break;
                                                  case '{':
                                                         if (p != '}')
                                                                goOn = false;
                                                         break;
                                           }
                                    }
                             break;
               } // switch
               i++;
        } //while
        return(goOn && isEmptyStack(stack));
}
```

```
void invert_stack(tStack* S) {
	tQueue Q;

	// an auxiliar queue is initialized
	createEmptyQueue(&Q);

	// dumps the stack onto the queue
	while (!isEmptyStack(*S)) {
		enqueue(peek(*S), &Q);
		pop(S);
	}

	// dumps the queue onto the original stack
	while (!isEmptyQueue(Q)) {
		push(front(Q), S);
		dequeue(&Q);
	}
}
```