# Werkgroep Data Structures. September 14, 2009

## *Problem #1*

Question on the definition of ADT Stack

A) Our definition of ADT Stack has a.o. the following operations pop, push(newItem), and getStackTop (stackTop) .
B) Variations of our definition of the ADT Stack are common. There are authors which do not include the inspection/copy of the top of the stack as an operation. They will have then a push(newItem) and pop(topStack) where the top of the stack is copied into topStack.

Find ways to simulate the two operations in B) using the operations in A). And the other way around: find ways of simulating the three operations in A) using the two operations in B).

## *Problem #2*

Evaluating Postfix Expressions.

Some calculators use a postfix convention. For example, to compute the value of

**2*(3+4)**

By using a postfix calculator, you would enter the sequence 2, 3, 4, +, * which corresponds to the postfix expression

**2 3 4 + ***

Recall that an operator in a postfix expression applies to the two operands that immediately precede it. Thus, the calculator must be able to retrieve the operands entered most recently.

Formalize the action of the calculator using an appropriate ADT to obtain an algorithm that evaluates a postfix expression, which is entered as a string of characters. To avoid issues that cloud the algorithm with programming details assume that

- The string is a syntactically correct postfix expression
- No unary operators are present
- No exponentiation operators are present
- Operands are single uppercase letters that represent integer values.

Provide the algorithm in pseudocode.

## *Problem #3*

Do Exercises 1,2,3,4, and 8 on page 166 of Drozdek.

## *Problem #4*

Consider the following mathematical specification of an ADT

**Abstract Type**: T

**Uses Types**: Boolean, Element

**Operations**:

***Create***: → T

***z***: Element x T → T

***p***: T → Element

***r***: T → T

**Axioms**:

H is in T; v and w in Element

1. ***p***(***z***(v,***create***()) = v;
2. ***r***(***z***(v,***create***()) = ***create***();
3. ***p***(***z***(v,***z***(w,H))) = ***p***(***z***(w,H))
4. ***r***(***z***(v,***z***(w,H))) = ***z***(v,***r***(***z***(w,H))

Unravel the above mystery: is the above ADT specified one of the standard ADTs? If so which one?

## *Problem 5*

**Dictionary**

An *abstract data type* storing items, or values. A value is accessed by an associated *key*. Basic operations are create, insert, find and delete.

Can you describe this ADT axiomatically. Follow the same pattern as we have given for the above mystery ADT.

## *Problem 6*

# Compare Two ADTs

ADT ListDrozdek

```
//page 79 in Drozdek
//
#pragma once

class IntSLLNode {
public:
    int info;
    IntSLLNode * next;
    IntSLLNode (int el, IntSLLNode * ptr = 0) {
        info = el; next=ptr;
    }
};

// the following we can view as the programmed ADT ListDrozdek

class IntSLList {
public:
    IntSLList(); //{head =0; tail=0}
    ~IntSLList();
    bool isEmpty();
    void addToHead(int);
    void addToTail(int);
    int deleteFromHead(); // delete the head and return its info
    int deleteFromTail(); // delete the tail and return its info
    void deleteNode(int);
    bool isInList(int) const;
private:
    IntSLLNode * head, * tail;
};
```

ADT List

```
#pragma once

Class List {
public:

    List();
    ~List();
    int listLength() const;
    bool listIsEmpty() const;
    void listInsert(int, int, bool);
    void listDelete(int, bool);
    void listRetrieve(int, int &, bool)const;
//private:

/*************************
createList()
        Creates an empty list; taken care off by con
destroyList()
        Destroys a list; taken care of by destructor
listIsEmpty()
        Determines whether the list is empty.
listLength()
        Returns the number of items in list.
listInsert(newPosition, newItem, success)
        Inserts newItem at position newPosition of a list, if 1 =  newPosition = listLength() + 1
        if newPosition  =  listLength(), the items are shifted as follows
        the item at newPosition becomes the item at newPosition becomes the item at newPosition + 1, the item
        at newPosition + 2, and  so on. Success indicates whether the insertion was successful.
listDelete(pos, success)
        Deletes the item at position pos of a list, if  1  =  pos  =  listLength(). If  pos  <  lengthList(), the items
        shifted as follows: the item at pos+1 becomes the at pos, the item at pos+1 becomes the item at pos+1, and
        so on. Success indicates whether the deletion was successful.
        listRetrieve(pos, dataItem, success)
        sets dataItem to the item at position pos of a list, if 1 = position = listLength(). The list is left unchanged b
        this operation. Success indicates whether the retrieval was successful.
*************************/
};
```

a) How can you implement a stack using an implementation of ADT List?
b) How can you implement a stack using an implementation of ADT ListDrozdek?
c) What about a deque?
d) Given an implementation for ADT List how can you (quickly) find an implementation for ADT ListDrozdek?
e) What about the other way around?