CptS 122

Stacks

Objectives

Explain and describe the ADT Stack

Implement a Stack ADT with a dynamically linked structure

Implement a Stack ADT with a static array.

The Stack ADT

A stack is a collection of elements, which can be stored and retrieved one at a time. Elements are retrieved in reverse order of their time of storage, i.e. the latest element stored is the next element to be retrieved. A stack is sometimes referred to as a Last-In-First-Out (LIFO) or First-In-Last-Out (FILO) structure. Elements previously stored cannot be retrieved until the latest element (usually referred to as the 'top' element) has been retrieved.

The Stack ADT is very similar to the SimpleList ADT. In fact its operations are basically the same. The essential difference in practice is that the SimpleList is usually implemented as an immutable type. The Stack ADT is usually implemented as a mutable type.

Specification of a Stack ADT

Operations:

create: -> Stack

push: Element x Stack -> Stack + 1

top: Stack -> Element

pop: Stack -> Stack - 1

isEmpty: Stack -> Boolean

Specification of a Stack ADT

Requirements:

isEmpty(create) = true

For every *x,s isEmpty(push(x,s)) = false*

For every *x,s peek(push(x,s)) = x*

For every *x,s pop(push(x,s)) = (x,s)*

Specification of a Stack ADT

Errors:

peek(create) is illegal

pop(create) is illegal

It is also a problem to read (top) or pop from an empty stack.

Dynamically Linked stack

```
//  STRUCT DEFINITIONS
struct stackNode
{
   int data;
   struct stack* next;
};

//  TYPEDEFS
typedef struct stackNode StackNode;
typedef StackNode* StackNodePtr;
```

createNode

```
StackNodePtr createNode(int item)
{
   StackNodePtr tmp = NULL;
   tmp = (StackNodePtr)
           malloc( sizeof(StackNode) );
   tmp->data = item;
   tmp->next = NULL;
   return tmp;
}
```

push

```
void push(StackNodePtr* stack,
          StackNodePtr node)
{
   node->next = *stack;
   *stack = node;
}
```

pop

```
void pop(StackNodePtr* stack)
{
   StackNodePtr tmp = *stack;
   *stack = tmp->next;
   free( tmp );
}
```

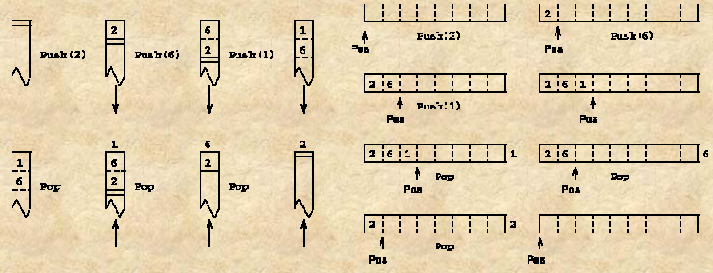
top

```
int  top(StackNodePtr stackTopNode)
{
   return stackTopNode->data;
}
```

isEmpty

```
int  isEmpty(StackNodePtr stackTopNode)
{
   return (NULL == stackTopNode);
}
```

Stack with a static array.



Array stack

```
//  CONSTANTS
#define SIZE 100
//  STRUCT DEFINITIONS
struct arrayStack
{
   unsigned stackTop;
   unsigned stackSize; // set to SIZE when created
   int      stackItems[SIZE];
};

//  TYPEDEFS
typedef struct arrayStack ArrayStack;
typedef ArrayStack*       ArrayStackPtr;
```


Array stack

```
ArrayStackPtr createStack(void)
{
  ArrayStackPtr tmp = NULL;
  tmp = (ArrayStackPtr) malloc ( sizeof(ArrayStack) );
  tmp->stackSize = SIZE;
  tmp->stackTop  = -1;   // the stack is empty, top points to an
                         //invalid value for the array.
}
```


Array stack

```
void push(ArrayStackPtr stack, int item)
{
   stack->stackTop++;
   stack->stackItems[stack->stackTop] = item;
}
```

Array stack

```
void pop(ArrayStackPtr stack)
{
   stack->stackTop--;
}
```

Array stack

```
int  top(ArrayStackPtr stack)
{
   return stack->stackItems[stack->stackTop];
}
```

Array stack

```
int  isEmpty(ArrayStackPtr stack)
{
   return (0 > stack->stackTop);
}

int  isFull(ArrayStackPtr stack)
{
   return
     (stack->stackSize <= stack->stackTop + 1);
     // could just check ==
}
```