|  |  |  |
| --- | --- | --- |
| **American University of Sharjah**  **School of Engineering**  ­Computer Engineering Department  P. O. Box 26666 Sharjah, UAE |  | **Instructor:** Dr. Ghassan Z. Qadah  **Lab Instructor:** Ms.Praveena Kolli  **Office**: EB2-126  **Phone**: 971-6-515-2352  **e-mail**: pkolli@aus.edu  **Semester**: Spring 2018 |

**CMP305L Data Structures and Algorithms**

**Lab #2 – Stack implementation and applications**

***Note: Exercise 1* and *Exercise* 2 are on dynamic array based Stack ADT**

***Exercise 1:***

Add the following functions to the given *dynamic array* *Stack* class (provided in DynamicArrayStack folder):

* + void clear() Removes all items of the stack
  + Overload the assignment operator(=)
* Overload the insertion operator “<<”(to print all items as if they are popped out from the stack. Make sure that the original content of the stack is not disturbed. )

stack.h

/\*

Author:

Class Description:

\*/

#ifndef STACKTYPE\_H

#define STACKTYPE\_H

#include <iostream>

using namespace std;

typedef int ItemType;

enum Error\_Code {Success, Overflow, Underflow};

class StackType {

public:

StackType();

StackType (const StackType & );

StackType(int max);

bool IsFull () const;

bool IsEmpty() const;

Error\_Code Push( ItemType );

Error\_Code Pop();

Error\_Code Top(ItemType &) const;

//Functions that I created start here

void Clear();

StackType& operator=(const StackType &);

friend ostream& operator<<(ostream& , const StackType &);

#pragma once

typedef int ItemType;

class FullStack {};

class EmptyStack {};

struct NodeType

{

ItemType info;

NodeType\* next;

};

class StackType

{

public:

StackType();

~StackType();

void Push(ItemType newItem);

void Pop();

ItemType Top();

bool IsFull() const;

bool IsEmpty() const;

//Functions created by me start here

StackType(const StackType &);

bool hasDuplicate() const;

//and end here

private:

NodeType\* topPtr;

};

//and end here

~StackType();

private:

int top;

int maxStack;

ItemType\* items;

};

#endif

stack.cpp

***/\****

***Author: Sheikh Abdul Raheem Ali***

***Description: Implementation of Stack Class***

***\*/***

***#include "stack.h"***

***/\****

***Function Description: Constructor with passed size***

***Input: Integer***

***Output: A stack with maximum size max***

***\*/***

***StackType::StackType(int max)***

***{***

***maxStack = max;***

***top = -1;***

***items = new ItemType [maxStack];***

***}***

***/\****

***Function Description: Default Constructor***

***Input: No input parameters***

***Output: A stack with the default maximum size 500***

***\*/***

***StackType::StackType()***

***{***

***maxStack = 500;***

***top = -1;***

***items = new ItemType [maxStack];***

***}***

***/\****

***Function Description: Copy Constructor***

***Input: const reference to a StackType object***

***Output: A deep copy of the original***

***\*/***

***StackType::StackType(const StackType & original)***

***{***

***top = original.top;***

***maxStack = original.maxStack;***

***items = new ItemType[maxStack];***

***for (int i=0; i <= top; i++)***

***items[i] = original.items[i];***

***}***

***/\****

***Function Description: Destructor***

***Input:***

***Output:***

***\*/***

***StackType::~StackType()***

***{***

***delete [] items;***

***}***

***/\****

***Function Description: Checks whether or not the stack is empty***

***Input: No input parameters***

***Output: True if the stack is empty, False otherwise***

***\*/***

***bool StackType::IsEmpty() const***

***{***

***return (top == -1) ;***

***}***

***/\****

***Function Description: Checks if the stack is full***

***Input: No input parameters***

***Output: True if the stack is full, False otherwise***

***\*/***

***bool StackType::IsFull() const***

***{***

***return (top == maxStack-1);***

***}***

***/\****

***Function Description: Pushes (adds) an element newItem to the top of the stack***

***Input: A value of type ItemType***

***Output: Overflow if the stack is full, otherwise success.***

***\*/***

***Error\_Code StackType::Push(ItemType newItem)***

***{***

***if (IsFull()) return Overflow;***

***top++;***

***items[top] = newItem;***

***return Success;***

***}***

***/\****

***Function Description: Pops (removes) the top element from the stack***

***Input: No input parameters***

***Output: Underflow is the stack is empty, otherwise success.***

***\*/***

***Error\_Code StackType::Pop()***

***{***

***if (IsEmpty())***

***return Underflow;***

***top--;***

***return Success;***

***}***

***/\****

***Function Description: Mutates passed item so that its value is changed to the value of the item at the top of the stack***

***Input: An item that is passed by reference***

***Output: Underflow if stack is empty, success otherwise.***

***\*/***

***Error\_Code StackType::Top(ItemType& item) const***

***{***

***if (IsEmpty())***

***return Underflow;***

***item = items[top];***

***return Success;***

***}***

***/\****

***Function Description: Logically removes all items from the stack***

***Input: No input parameters***

***Output: A stack that is now empty***

***\*/***

***void StackType::Clear()***

***{***

***top = -1;***

***}***

***/\****

***Function Description: Assignment operator***

***Input: constant reference to stack object***

***Output: A deep copy of the original***

***\*/***

***StackType& StackType::operator=(const StackType& original)***

***{***

***top = original.top;***

***maxStack = original.maxStack;***

***items = new ItemType[maxStack];***

***for (int i=0; i <= top; i++)***

***items[i] = original.items[i];***

***}***

***/\****

***Function Description: Stream insertion operator***

***Input: Output stream and constant reference to a stack object***

***Output: All items in the stack are printed to standard output as if they were popped***

***\*/***

***ostream& operator<<(ostream& cout, const StackType& s)***

***{***

***for(int i=s.top; i>= 0; i--)***

***cout << std::hex << s.items[i];***

***}***

***Exercise 2:***

Write a program that reads in a positive integer and converts into one of the user chosen number system. Give user three options: binary, octal and Hex. You should use stack ADT to store the digits that are converted.

stackNumberConverter.cpp

#include"stack.cpp"

int main()

{

ItemType n;

cout << "Please enter a base 10 integer: ";

cin >> n;

int option = -1;

do{

cout << "Convert the number from decimal into: "

<< "\n 0 - Binary"

<< "\n 1 - Octal"

<< "\n 2 - Hexadecimal" << endl;

cin.clear();

cin >> option;

}while(option < 0 || option > 2);

StackType s;

switch(option)

{

case 0:

for(int i = n; i > 0; i/=2){

s.Push(i%2);

}

cout << "Binary: " << s << endl;

break;

case 1:

for(int i = n; i > 0; i/=8){

s.Push(i%8);

}

cout << "Octal: " << s << endl;

break;

case 2:

for(int i = n; i > 0; i/=16){

s.Push(i%16);

}

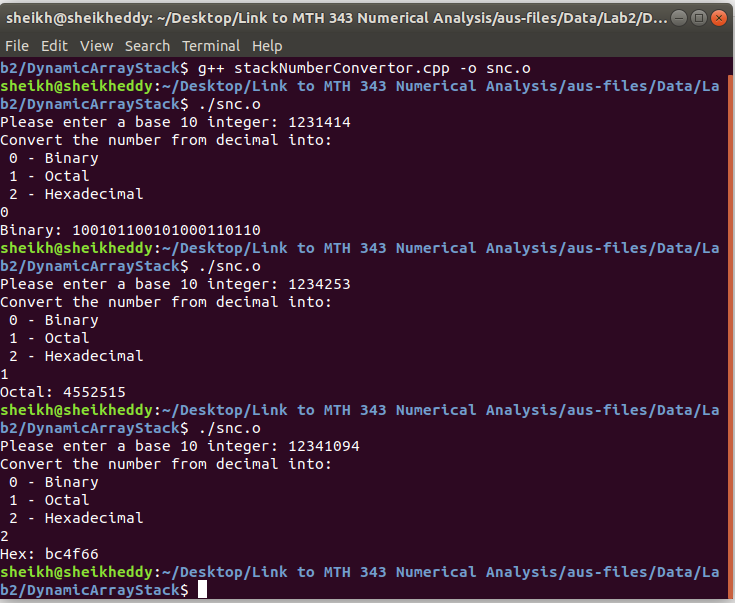
cout << "Hex: " << s << endl;

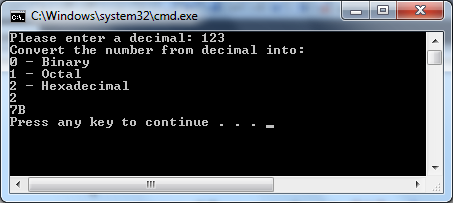
break;

}

return 0;

}





**Note: Exercise 3 and Exercise 4 are on linked structure based Stack ADT**

***Exercise 3:***

Add the following member function to the given *Linked Stack* class (provided in LinkedStack folder):

* bool hasDuplicate(), which returns true if the queue contains a duplicate value, false otherwise.

StackType.h

***#pragma once***

***typedef int ItemType;***

***class FullStack {};***

***class EmptyStack {};***

***struct NodeType***

***{***

***ItemType info;***

***NodeType\* next;***

***};***

***class StackType***

***{***

***public:***

***StackType();***

***~StackType();***

***void Push(ItemType newItem);***

***void Pop();***

***ItemType Top();***

***bool IsFull() const;***

***bool IsEmpty() const;***

***//Functions created by me start here***

***StackType(const StackType &);***

***bool hasDuplicate() const;***

***//and end here***

***private:***

***NodeType\* topPtr;***

***};***

StackType.cpp

***// Implementation file for linked StackType***

***#include "StackType.h"***

***#include <new>***

***void StackType::Push(ItemType newItem)***

***// Adds newItem to the top of the stack.***

***// Stack is bounded by size of memory.***

***// Pre: Stack has been initialized.***

***// Post: If stack is full, FullStack exception is thrown;***

***// else newItem is at the top of the stack.***

***{***

***if (IsFull())***

***throw FullStack();***

***else***

***{***

***NodeType\* location;***

***location = new NodeType;***

***location->info = newItem;***

***location->next = topPtr;***

***topPtr = location;***

***}***

***}***

***void StackType::Pop()***

***// Removes top item from Stack and returns it in item.***

***// Pre: Stack has been initialized.***

***// Post: If stack is empty, EmptyStack exception is thrown;***

***// else top element has been removed.***

***{***

***if (IsEmpty())***

***throw EmptyStack();***

***else***

***{***

***NodeType\* tempPtr;***

***tempPtr = topPtr;***

***topPtr = topPtr->next;***

***delete tempPtr;***

***}***

***}***

***ItemType StackType::Top()***

***// Returns a copy of the top item in the stack.***

***// Pre: Stack has been initialized.***

***// Post: If stack is empty, EmptyStack exception is thrown;***

***// else a copy of the top element is returned.***

***{***

***if (IsEmpty())***

***throw EmptyStack();***

***else***

***return topPtr->info;***

***}***

***StackType::StackType() // Class constructor.***

***{***

***topPtr = NULL;***

***}***

***bool StackType::IsFull() const***

***// Returns true if there is no room for another ItemType***

***// on the free store; false otherwise.***

***{***

***NodeType\* location;***

***try***

***{***

***location = new NodeType;***

***delete location;***

***return false;***

***}***

***catch(std::bad\_alloc exception)***

***{***

***return true;***

***}***

***}***

***StackType::~StackType()***

***// Post: stack is empty; all items have been deallocated.***

***{***

***NodeType\* tempPtr;***

***while (topPtr != NULL)***

***{***

***tempPtr = topPtr;***

***topPtr = topPtr->next;***

***delete tempPtr;***

***}***

***}***

***bool StackType::IsEmpty() const***

***{***

***return (topPtr == NULL);***

***}***

***StackType::StackType(const StackType & original){***

***NodeType\* ptr = original.topPtr;***

***while(ptr != NULL){***

***NodeType\* location = new NodeType;***

***location->info = ptr->info;***

***location->next = topPtr->next;***

***topPtr = location;***

***ptr = ptr->next;***

***}***

***}***

***bool StackType::hasDuplicate() const***

***//naive O(N^2) approach***

***{***

***StackType temp1(\*this);***

***StackType temp2(temp1);***

***while(!temp1.IsEmpty()){***

***temp2.Pop();***

***while(!temp2.IsEmpty()){***

***if(temp2.Top() == temp1.Top()){***

***return true;***

***}***

***temp2.Pop();***

***}***

***temp1.Pop();***

***}***

***return false;***

***}***

***Exercise 4:***

One of the *Stack* data structure usages is to evaluate the values of postfix mathematical expressions. Postfix notation is a parenthesis-free way of writing arithmetic expressions, where one places the operator symbol *after* the operator's two operands. For example, the addition of 3 to 2 is written 3 2 +, and the multiplication of the result by 4 is written as 3 2 + 4 \*. Remarkably, *parentheses are never needed*. An example like

((3 + 2) \* 4) / (5 - 1)

is written

3 2 + 4 \* 5 1 - /

and gets manually computed as follows:

3 2 + 4 \* 5 1 - /

=> 5 4 \* 5 1 - /

=> 20 5 1 - /

=> 20 4 /

=> 5

We see that an operator evaluates with the two operands that immediately precede it. This explains why the division operator is written last in the original expression, because the division is performed only after all the other sub expressions are evaluated. To automate the computation of a postfix expression, one may use a stack data structure as follows:

Stack Expression

| |

--- 3 2 + 4 \* 5 1 - /

(empty)

| 3 | 2 + 4 \* 5 1 - /

---

| 2 |

| 3 | + 4 \* 5 1 - /

---

| 5 | 4 \* 5 1 - /

---

| 4 |

| 5 | \* 5 1 - /

---

| 20| 5 1 - /

---

| 5 |

| 20| 1 - /

---

| 1 |

| 5 |

| 20| - /

---

| 4 |

| 20| /

---

| 5 | (finished)

---

Implement an application program that reads a postfix expression from the user and uses a stack to compute its value and returns it to the user. The expression will be assumed to contain integers and the following binary arithmetic operators, namely, + (addition), - (subtraction), \* (multiplication) and / (division).

**Note:** Your program must also handle*integers* that have *mor****e*** than *one digit.*

StackDr.cpp

**#define \_CRT\_SECURE\_NO\_WARNINGS**

**#include <iostream>**

**#include <cstdlib>**

**#include <cstring>**

**#include "StackType.cpp"**

**using namespace std;**

**bool isInteger(char \* s)**

**{**

**if ((s[0] - '0') >= 0 && (s[0] - '0') <= 9)**

**return true;**

**else**

**return false;**

**}**

**int main()**

**{**

**StackType s;**

**//This is for EX4**

**char pstfix[80];**

**cout << "Enter post fix notation : ";**

**cin.getline(pstfix, 80);**

**char \* pch;**

**cout << "Splitting string " << pstfix << " into tokens: " << endl;**

**pch = strtok(pstfix, " ");**

**while (pch != NULL)**

**{**

**if (isInteger(pch))**

**{**

**int i = atoi(pch);**

**cout << "integer = " << i << endl;**

**s.Push(i);**

**}**

**else{**

**cout << "Operator = " << pch[0] << endl;**

**ItemType b = s.Top(); s.Pop();**

**ItemType a = s.Top(); s.Pop();**

**switch(pch[0]){**

**case '+':**

**s.Push(a+b);**

**break;**

**case '-':**

**s.Push(a-b);**

**break;**

**case '/':**

**s.Push(a/b);**

**break;**

**case '\*':**

**s.Push(a\*b);**

**break;**

**}**

**cout << "Result = " << s.Top() << endl;**

**}**

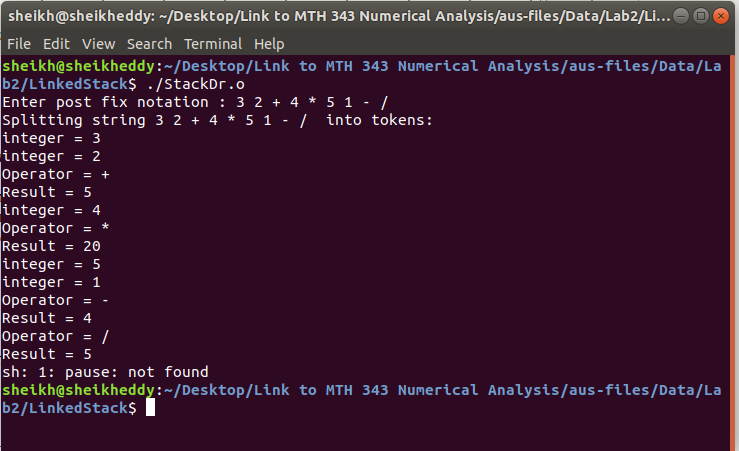
**pch = strtok(NULL, " ");**

**}**

**system("pause");**

**return 0;**

**}**

****