ASSIGNMENT: Data Structures and Algorithms

Asif Haider Khan; ID:1421341042

2018-12-08

CSE 225: Data Structures and Algorithms

Assignment 1

Deadline: 4th December 2018

Submission Procedure:

All materials should be compiled in a single pdf file and submitted to sifat.momen@northsouth.edu by the deadline. Your submission materials should include (1) justification of the data structure(s) to be used in solving the problem and (2) all codes related to this assignment.

Problem:

Given a list, and an integer value called **sum**, you are required to design and implement a code that will check whether there exists a pair of integer in the list whose sum is equal to **sum**. If it is, it should print out "Yes there exists a pair" followed by the two integers that make up the pair and if not, it should print "No such pair exists". More than one pair may exist but you need to come up with only 1 pair (if there exists a pair).

Example:

For instance if the list is {1,3, 3, 2, 4} and the **sum** = 4, then there exists a pair (1+3 = 4), but if **sum** = 8, then there is no pair of integers in the list whose sum is equal to 8.

1 Introduction

The Assignment is based on a problem statement where a list (array) of integers of a fixed length is given and an integer value named 'SUM' is given.
According to the problem stated, a program is to be written where a the list is traversed and there must exist a pair of integers in the list(array) where the sum of the pair of integers must be equal to the given integer value of 'SUM'.
If a pair of integers is found, the pair of integers along with a message string will be displayed "There exists such a pair".
If a pair of integers is not found, only the message string will be displayed "No such pair exists".

2 Justification for Data Structure used

A generic (in-class) implemented Stack is being used to solve the following problem. The data structure has the following functions : STACKTYPE() —- Constructor
STACKTYPE() — Destructor
IsFull(): Boolean — Checks whether stack is full
isEmpty(): Boolean — Checks whether stack is empty
makeEmpty(): Void — Makes the stack empty
PUSH(Item): Void — Pushes a template item into the stack
POP(): Void — Pops a template item from the stack
TOP(): Item — Returns the top item of the stack
This Data structure is being used so that the list can be stored in it and then each element in the stack will be compared to each element in the List/Array and then find out the pair of elements that add up together to give a value either equal to 'SUM' or not.

3 Implementation

The Data Structure is Instantiated inside a function named 'CheckSum(int *List,int num): void' that will check and print whether there exists a pair or not.

3.1 Header/Declaration File

File Name: Stacktype.h

```
#ifndef STACKTYPE_H_INCLUDED
#define STACKTYPE_H_INCLUDED

class fullStack{};
class emptyStack{};

template <class Itemtype>
class Stacktype{
    struct Nodetype{
        Itemtype data;
        Nodetype* next;
    };

    private:
        Nodetype *topPtr;
    public:
    Stacktype();
    ~Stacktype();
    bool isFull();
```

```
    bool isEmpty();
    void makeEmpty();
    void Push(Itemtype);
    void Pop();
    Itemtype Top();


};


#endif // STACKTYPE_H_INCLUDED
```

3.2 Source/Definition File

File Name: Stacktype.cpp

```
#include "stacktype.h"
#include <iostream>
#include <cstddef>
using namespace std;

template <class Itemtype>
Stacktype<Itemtype>::Stacktype(){
    topPtr = NULL;
}

template <class Itemtype>
Stacktype<Itemtype>::~Stacktype(){
    while(!isEmpty()){
        Pop();
    }
}

template <class Itemtype>
bool Stacktype<Itemtype>::isEmpty(){
    return (topPtr == NULL);
}

template <class Itemtype>
bool Stacktype<Itemtype>::isFull(){
    Nodetype *newNode;
    try{
        newNode = new Nodetype;
        delete newNode;
        return false;
```

```
    }
    catch(bad_alloc&exception){
        return true;
    }
}

template <class Itemtype>
void Stacktype<Itemtype>::Push(Itemtype item){
    if(isFull())
        throw fullStack();

    Nodetype *location;
    location = new Nodetype;
    location->data = item;
    location->next = topPtr;
    topPtr = location;

}

template <class Itemtype>
void Stacktype<Itemtype>::Pop(){
    if(isEmpty())
        throw emptyStack();

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

}

template <class Itemtype>
Itemtype Stacktype<Itemtype>::Top() {
    if(isEmpty())
        throw emptyStack();
    else
        return topPtr->data;
}

template <class Itemtype>
void Stacktype<Itemtype>::makeEmpty(){
    while(!isEmpty()){

    }
}
```

3.3 Main/Driver file (with CheckSum function)

File Name: main.cpp

```
#include <iostream>
#include "Stacktype.cpp"

using namespace std;

void CheckSum(int *List,int sum);
//void CheckSum2(int *List,int sum);


int main()
{
    int aList[5] = {1,3,9,4,2};
    int sum = 13;

    CheckSum(aList,sum);

}

void CheckSum(int *List,int sum){
    Stacktype<int>S1;
    bool found = false;
    int val1,val2;
    int x = sizeof(List);
    for(int i=0 ; i<x ; i++){
        S1.Push(List[i]);
    }
    while(!found && !S1.isEmpty()){
        for(int i=0 ; i<x ; i++){
            if(((List[i] + S1.Top()) == sum) && (List[i] != S1.Top())){
                val1 = List[i];
                val2 = S1.Top();
                found = true;
                break;
            }
            else{
                found = false;
            }
        }S1.Pop();
    }
```

```
    if(found){
        cout << "Exists: " << val1 << " & " << val2 << endl;
    }
    else{
        cout << "Not Exists" << endl;
    }

}
```

The following files were run and tested in Code::Blocks 16.01 with successful results.