

**CSE-225  
 Assignment-2**

Name: Md. Najmus Sakib.

ID: 1813345642.

Section: 14.

# Ans. To The Question: Stack implementation using linked list

This program is the implementation of stack using linked list. The Stack operations “**IsEmpty”**, “**Push”**, and “**Pop”** is implemented. This program allow user to add item in the stack and remove item from the stack and print the removed item. This program have a function “**PrintStack**” to print all the items in the Stack.

**Here is the code of “main.cpp” driver file**

#include <iostream>

#include "stacktype.cpp"

using namespace std;

int main()

{

StackType<int> link; // object creation

int number;

int item;

int i=0;

cout<<"Enter the number of items you want to add: ";

cin>>number; // the number of items we want to add will store in this variable

cout<<"\nEnter the items: ";

while(i< number) // loop to take user inputs and add them into the list of stack until the number

{

cin>>item;

link.Push(item); // every item will be pushed into the list of stack

i++;

}

int pop\_item=link.Pop(); // it will remove the last item of the list of stack

cout<<"\nRemoved item is: "<<pop\_item<<endl;

if(link.IsEmpty()==true) // finding if the list is empty or not

{

cout<<"\nThe list is empty"<<endl; // if the list is empty then show this as message.

}

Else{

cout<<"\nThe list is not empty"<<endl; // if the list is not empty then show this as message.

}

cout<<"\nThe list of values are: "<<endl;

link.PrintStack(); // print the items in the list of stack

return 0;

}

**Here is the “stack.h” header file-**

#ifndef STACKTYPE\_H\_INCLUDED

#define STACKTYPE\_H\_INCLUDED

class FullStack

{};

class EmptyStack

{};

template <class ItemType> // standard template library

class StackType

{

struct NodeType //structure

{

ItemType info; // holds the item

NodeType\* next; // next pointer

};

public: // public variables and methods

StackType(); // constructor

~StackType(); // destructor

void Push(ItemType); // method of add item

int Pop(); // method of removing an item

bool IsEmpty(); // method of finding if the list of stack is empty

bool IsFull(); // method of finding if the list of stack is full

void PrintStack(); // print method to print the list of stack

private: // private variables and methods

NodeType\* topPtr; // private pointer

};

#endif // STACKTYPE\_H\_INCLUDED

**Here is the code of “stack.cpp” file where all the functions are implemented-**

#include <iostream>

#include "stacktype.h"

using namespace std;

template <class ItemType>

StackType<ItemType>::StackType()

{

topPtr = NULL; // after object creation the list top fill with null.

}

template <class ItemType>

bool StackType<ItemType>::IsEmpty()

{

return (topPtr == NULL); // top pointer null returns true when the list of stack is empty otherwise false

}

template <class ItemType>

bool StackType<ItemType>::IsFull()

{

NodeType\* location; // location pointer

try // try catch block to face any exception

{

location = new NodeType;

delete location;

return false;

}

catch(bad\_alloc& exception)

{

return true;

}

}

template <class ItemType>

void StackType<ItemType>::Push(ItemType newItem)

{

if (IsFull())

throw FullStack(); // throw exception if the list of stack is full

else

{

NodeType\* location; // location pointer

location = new NodeType;

location->info = newItem; // new item added

location->next = topPtr; // location is updated

topPtr = location; // top pointer updated

}

}

template <class ItemType>

int StackType<ItemType>::Pop()

{

int item=0;

if (IsEmpty())

throw EmptyStack(); // throw exception if the the list of stack doesn't have item

else

{

NodeType\* tempPtr; // temporary pointer

tempPtr = topPtr;

topPtr = topPtr->next; // top pointer will be in the next node

item= tempPtr->info; // it contains the item which is being removed from the the list of stack.

delete tempPtr; // deleting temporary pointer

}

return item; // return the value which is being removed from the list of stack

}

template <class ItemType>

void StackType<ItemType>::PrintStack()

{

if (IsEmpty())

throw EmptyStack(); // throw exception if the the list of stack doesn't have item

else{

while(!IsEmpty()) // iterate till the list is empty.

{

NodeType\* tempPtr; // new pointer created

tempPtr = topPtr; // assigning the top pointer into the temporary pointer.

topPtr = topPtr->next; // top pointer updated by its next item

cout<<tempPtr->info<<endl; // it prints the item whose are in the list of stack.

delete tempPtr;

}

}

}

template <class ItemType>

StackType<ItemType>::~StackType()

{

NodeType\* tempPtr; // temporary pointer

while (topPtr != NULL) // loop till the top pointer is not having null value

{

tempPtr = topPtr; // temporary pointer indicating top pointer

topPtr = topPtr->next; //top pointer updated by the next node

delete tempPtr; // delete temporary pointer

}

}

**Sample input output:-**

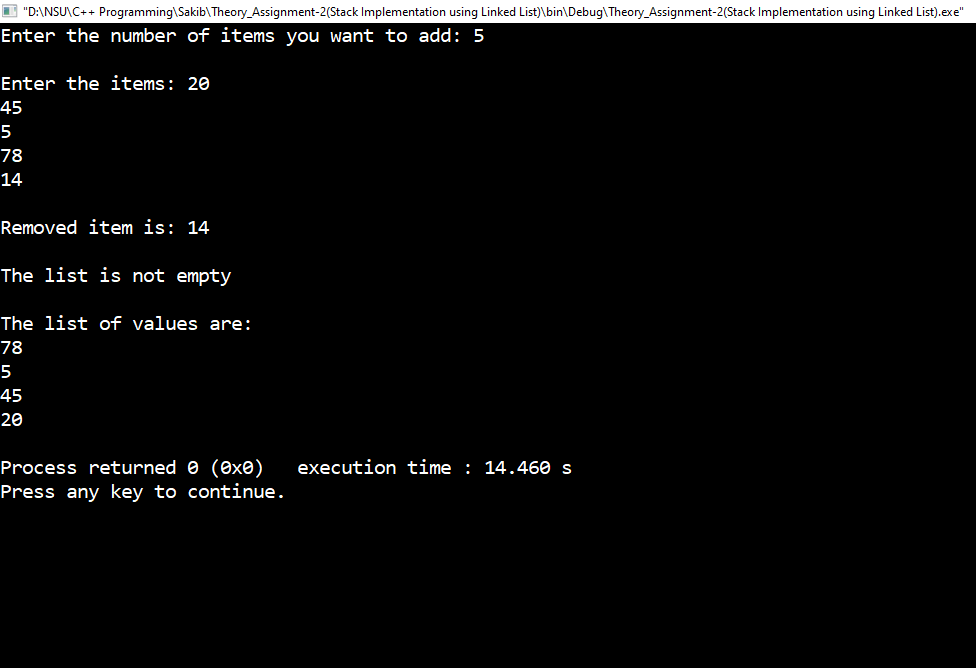


Figure 1: Sample input and output of the program.