

```

1 //Bernard J. Gole Cruz, CS 202-2002, Assignment 6, problem 2
2 //This program implement stack class template with exception handling
3 #include <iostream>
4 #include <stdlib.h>
5
6 using namespace std;
7
8 //stack class template
9 template <class T>
10 class Stack{
11     public:
12         //stack operation
13         void push(T value);
14         T pop();
15         void disp();
16         bool isempty();
17         bool isfull();
18
19         //default constructor
20         Stack();
21         //parameterize constructor()
22         Stack(int);
23         //destructor
24         ~Stack();
25
26
27     private:
28         int stackPointer; //top of the stack
29         T *element = NULL; //pointer to stack array
30         int stacksize; //stack size
31
32
33
34
35 };
36 //function prototype and definitions
37
38 //default constructor set size to 0
39 template <class T>
40 Stack<T>::Stack(){
41
42     this->stacksize = -1;
43     stackPointer = 0;
44 }
45
46 //parameterized constructor
47 //deallocate size based on given parameter
48 template <class T>
49 Stack<T>::Stack(int stacksize){
50
51     element = new T [stacksize];
52     this->stacksize = stacksize;
53     stackPointer = 0;
54 }
55
56
57 //destructor, deallocate elements
58 template <class T>
59 Stack<T>::~~Stack(){
60
61     delete []element;
62 }
63
64 //add element in stack
65 template <class T>
66 void Stack<T>::push(T value){

```

```

67     //check if stack is full
68     if (isfull()){
69         cout <<"can't push in a full stack" <<endl;
70         cout <<"error operating the stack at position " << stackPointer << endl;
71     }
72     //add element into stack
73     else{
74         cout <<"Enter value: " ;
75         cin >> value;
76
77         //clear screen at every iteration
78         system("CLS");
79         //add item into stack and increment index
80         element[stackPointer] = value;
81         stackPointer++;
82     }
83 }
84
85 //remove element in stack
86 template <class T>
87 T Stack<T>::pop(){
88     //check if stack is empty
89     if (isempty()){
90         cout <<"can't pop from empty stack" << endl;
91         cout <<"error operating the stack at position " << stackPointer << endl;
92         return -1;
93     }
94     //remove item from stack
95     else{
96         stackPointer--;
97         return element[stackPointer];
98     }
99 }
100
101 //check if stack is empty
102 template <class T>
103 bool Stack<T>::isempty(){
104
105     return (stackPointer == 0);
106 }
107
108 //check if stack is full
109 template <class T>
110 bool Stack<T>::isfull(){
111
112     return (stackPointer == stacksize);
113 }
114
115 template <class T>
116 void Stack<T>::disp(){
117
118     if(isempty()){
119         cout <<"Stack: " ;
120     }
121     else{
122         cout <<"Stack: " ;
123         for (int i=0; i<stackPointer; i++){
124             cout << element[i];
125             if (i != (stackPointer - 1) ){
126                 cout << ",";
127             }
128         }
129     }
130 }
131
132 //menu

```

```

133 void menu(){
134     //display choices in menu
135     cout << "Menu: " << endl;
136     cout << "1. push Element" << endl;
137     cout << "2. pop element" << endl;
138     cout << "3. exit" << endl;
139
140 };
141
142 //prompt user
143 template <class T>
144 void prompt(Stack<T> &obj,int &choice)
145 {
146     //keep prompting if user choose number outside the menu
147     try{
148         cout <<endl;
149         menu();
150         cout << "Enter: ";
151         if(!(cin >>choice) ){
152             cin.clear();
153             cin.ignore(100, '\n');
154             throw choice;
155         }
156     }
157     catch(int choice){
158         throw;
159     }
160
161     int value;
162     //choices in menu
163     switch (choice){
164     case 1:
165         //add elements in stack
166         obj.push(value);
167         cout << endl;
168         //display stack contents
169         obj.disp();
170         break;
171
172     case 2:
173         //remove elements from stack
174         obj.pop();
175         cout << endl;
176         //display stack contents
177         obj.disp();
178         break;
179
180     case 3:
181         //exit menu
182         exit(0);
183
184     default:
185         //keep prompting until a correct choice is made
186         cout << "try again!!" << endl;
187     }
188 };
189
190
191 int main(){
192
193     //variables, declared object
194     int select;
195     Stack<int> number(4);
196     bool success = false;
197     cout <<endl;
198     cout <<"Stack: ";

```

```
199
200 //will keep prompting if choice 3 is not press
201 //exception handling
202 while(true){
203     try{
204         prompt(number, select);
205         success = true;
206     }
207     catch(...){
208     }
209 }
210 return 0;
211
212 }
```