```cpp
#include <iostream>
using namespace std;


class Node{

    private:
        char data;
        Node* nextNodePtr;
        Node* prevNodePtr;

    public:
        Node(){}

        void setData(char d){
            data = d;
        }

        char getData(){
            return data;
        }

        void setNextNodePtr(Node* nodePtr){
                nextNodePtr = nodePtr;
        }

        Node* getNextNodePtr(){
            return nextNodePtr;
        }

        void setPrevNodePtr(Node* nodePtr){
                prevNodePtr = nodePtr;
        }

        Node* getPrevNodePtr(){
            return prevNodePtr;
        }

};

class Stack{

    private:
        Node* headPtr;
        Node* tailPtr;


    public:
        Stack(){
            headPtr = new Node();
            tailPtr = new Node();
            headPtr->setNextNodePtr(0);
            tailPtr->setPrevNodePtr(0);
        }

        Node* getHeadPtr(){
            return headPtr;
        }

        Node* getTailPtr(){
            return tailPtr;
        }

        bool isEmpty(){
```

```
65
66              if (headPtr->getNextNodePtr() == 0)
67                  return true;
68
69              return false;
70          }
71
72
73          void push(char data){
74
75              Node* newNodePtr = new Node();
76              newNodePtr->setData(data);
77              newNodePtr->setNextNodePtr(0);
78
79              Node* lastNodePtr = tailPtr->getPrevNodePtr();
80
81              if (lastNodePtr == 0){
82
83                  headPtr->setNextNodePtr(newNodePtr);
84                  newNodePtr->setPrevNodePtr(0);
85
86              }
87              else{
88
89                  lastNodePtr->setNextNodePtr(newNodePtr);
90                  newNodePtr->setPrevNodePtr(lastNodePtr);
91
92              }
93
94              tailPtr->setPrevNodePtr(newNodePtr);
95
96          }
97
98
99          char pop(){
100
101             Node* lastNodePtr = tailPtr->getPrevNodePtr();
102             Node* prevNodePtr = 0;
103
104             char poppedData = '$'; //empty stack
105
106             if (lastNodePtr != 0){
107                 prevNodePtr = lastNodePtr->getPrevNodePtr();
108                 poppedData = lastNodePtr->getData();
109             }
110             else
111                 return poppedData;
112
113             if (prevNodePtr != 0){
114                 prevNodePtr->setNextNodePtr(0);
115                 tailPtr->setPrevNodePtr(prevNodePtr);
116             }
117             else{
118                 headPtr->setNextNodePtr(0);
119                 tailPtr->setPrevNodePtr(0);
120             }
121
122             return poppedData;
123         }
124
125
126         char peek(){
127
128             Node* lastNodePtr = tailPtr->getPrevNodePtr();
```

```
129
130             if (lastNodePtr != 0)
131                 return lastNodePtr->getData();
132             else
133                 return '$'; //  empty stack
134
135
136         }
137
138
139
140 };
141
142 int main(){
143
144     Stack stack;
145
146     string expression;
147
148     cout << "Enter an expression: ";
149     cin >> expression;
150
151     int index = 0;
152
153     while (index < expression.size()){
154
155         char symbol = expression[index];
156
157         if (symbol == '{' || symbol == '(' || symbol == '['){
158             stack.push(symbol);
159             index++;
160             continue;
161         }
162         else if (symbol == '}' || symbol == ')' || symbol == ']'){
163
164             char topSymbol = stack.pop();
165             if ( (topSymbol == '{' && symbol == '}') ||
166                  (topSymbol == '(' && symbol == ')') ||
167                  (topSymbol == '[' && symbol == ']') ){
168
169                     index++;
170                     continue;
171
172             }
173             else{
174
175                 cout << "parenthesis not balanced!!" << endl;
176                 return 0;
177
178             }
179         }
180         else{
181
182             cout << "Invalid symbol "<< symbol << " in the expression!!" << endl;
183             return 0;
184         }
185
186
187     }
188
189     cout << expression << " is balanced!!" << endl;
190
191 return 0;
192 }
```

```
Enter an expression: {()[()]}
{()[()]} is balanced!!
```

```
Enter an expression: {()}
parenthesis not balanced!!
```

```
Enter an expression: {()p}
Invalid symbol p in the expression!!
```