```
#ifndef STACK_H
#define STACK_H

template <class t> class stack;

template <class u>
class node{
public:
    node():next(nullptr){}
    node(u m_var):next(nullptr, var(m_var)){}
private:
    u var;
    node* next;
    node& operator=(const u& n_var){
        return this->var = n_var;
    }
    friend class stack<u>;
   };

template <class t>
class stack{
private:
    int m_TOP;

    node<t>* m_current;
    node<t>* m_start;

public:

    // Overloaded Constructors Methods
    stack(int size = 0){
        m_TOP=-1;
        m_current=m_start=nullptr;
        for(int i(0); i<size; i++)
            push_back();
    }
    stack(int size, const t* array){
        m_TOP=-1;
        m_current=m_start=nullptr;
        for(int i(0); i<size; i++)
            push_back(t[i]);
    }
    stack(int size, const t var){
        m_TOP=-1;
        m_current=m_start=nullptr;
        for(int i(0); i<size; i++)
            push_back(t);
    }
    stack(const stack& obj){
        m_TOP=-1;
        m_current=m_start=nullptr;

        for(int i(0); i<=obj.size(); i++){
            push_back(obj.get(i));
        }
    }
    stack(const stack&& obj){
        m_TOP=-1;
        m_current=m_start=nullptr;
        m_start = obj[0];
        m_current = obj[obj.size()];
        obj.m_TOP = -1;
        obj.m_current = obj.m_start= nullptr;
        obj.clear();
    }

    // A Destructor Method
    ~stack(){
```

```
69          clear();
70      }
71
72      // Normal Methods:
73      void push_back(const t& var){
74          m_TOP++;
75          if(m_TOP==0){
76              m_current = new node<t>;
77              m_current->next = nullptr;
78              m_current->var = var;
79              m_start = m_current;
80          } else{
81              m_current->next = new node<t>;
82              m_current = m_current->next;
83              m_current->next = nullptr;
84              m_current->var = var;
85          }
86
87      }
88      void push_back(){
89          m_TOP++;
90          if(m_TOP==0){
91              m_current = new node<t>;
92              m_current->next = nullptr;
93              m_start = m_current;
94          } else{
95              m_current->next = new node<t>;
96              m_current = m_current->next;
97              m_current->next = nullptr;
98          }
99
100     }
101     t pop_out(){
102         t var;
103         if(m_TOP > -1){
104             if(m_TOP==0){
105                 var = m_current->var;
106                 delete m_current;
107                 m_current=m_start=nullptr;
108             }else if(m_TOP==1){
109                 var = m_start->next->var;
110                 delete m_start->next;
111                 m_start->next=nullptr;
112                 m_current=m_start;
113             } else{
114                 var = m_current->var;
115
116                 node<t>* m_del = m_start;
117                 for(int i(0); i<(m_TOP-1); i++)
118                       m_del = m_del->next;
119
120                 m_current = m_del;
121                 delete m_del->next;
122                 m_del->next=nullptr;
123             }
124
125             m_TOP--;
126
127         }
128         return var;
129     }
130
131     t get(unsigned int index){
132         t var;
133         node<t>* m_get = m_start;
134         if(index<=m_TOP)
135             for(int i(0); i<index; i++)
136                 m_get = m_get->next;
```

```
137         var = m_get->var;
138         return var;
139     }
140     t top(){
141         if(m_TOP > -1){
142         return m_current->var;
143         }
144     }
145     int size(){
146         return m_TOP;
147     }
148     void clear(){
149         if(m_TOP!=-1){
150             node<t>* m_del = m_start;
151             for(int i(0); i<=m_TOP; i++){
152                 m_current = m_del->next;
153                 m_del->next = nullptr;
154                 delete m_del;
155                 m_del = m_current;
156             }
157         }
158         m_TOP = -1;
159         m_current=m_start=nullptr;
160     }
161 
162     // Operator methods
163     node<t>* operator[](int index){
164         node<t>* m_get = m_start;
165         if(index<=m_TOP)
166             for(int i(0); i<index; i++)
167                 m_get = m_get->next;
168         return m_get;
169     }
170     stack& operator=(const stack& obj){
171         clear();
172 
173         for(int i(0); i<=obj.size(); i++){
174             push_back(obj.get(i));
175         }
176         return this;
177     }
178 
179 
180 
181 };
182 
183 
184 #endif
```