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

template <class T>
class Stack
{
private:
    T* stackArray;
    int stackSize;
    int numElements;
public:
    Stack(int);
    ~Stack();

    void push(T);
    void pop(T&);
    bool isFull() const;
    bool isEmpty() const;
    void display() const;
};
template <class T>
Stack<T>::Stack(int Size)
{
    stackArray = new T[Size];
    stackSize = Size;
    numElements = 0;
}

template <class T>
Stack<T>::~Stack()
{
    delete[] stackArray;
}

template <class T>
void Stack<T>::push(T num)
{
    if (isFull())
        cout << "The stack is full.\n";
    else
    {
        stackArray[numElements] = num;
        numElements++;
    }
}
template <class T>
void Stack<T>::pop(T& num)
{
    if (isEmpty())
```

```
50          cout << "The stack is empty.\n";
51      else
52      {
53          numElements--;
54          num = stackArray[numElements];
55      }
56  }
57
58  template <class T>
59  bool Stack<T>::isFull() const
60  {
61      return numElements == stackSize;
62  }
63
64  template <class T>
65  bool Stack<T>::isEmpty() const
66  {
67      return numElements == 0;
68  }
69
70  template <class T>
71  void Stack<T>::display() const
72  {
73      if (isEmpty())
74          cout << "The stack is empty.\n";
75      else
76          for (int i = 0; i < numElements; i++)
77              cout << stackArray[i] << "\t";
78      cout << endl;
79  }
80
81  class Rational
82  {
83  private:
84      int numer;
85      int denom;
86  public:
87      int getNumer() const;
88      int getDenom() const;
89      void setNumer(int);
90      void setDenom(int);
91      void input();
92      void output() const;
93      Rational();
94      Rational(int, int = 1);
95      void reduce();
96      friend istream& operator>>(istream& strm, Rational& obj);
97  };
98  void Rational::reduce()
```

```
 99 {
100     int x = abs(numer);
101     int y = abs(denom);
102     // find minimum of x and y
103     int min = x;
104     if (y < x)
105         min = y;
106 
107     // finding a common factor greater than 1
108     int gcf = 1;
109     for (int i = 2; i <= min; i++) {
110         if (x % i == 0 && y % i == 0) {
111             gcf = i;
112         }
113     }
114     numer = numer / gcf;
115     denom = denom / gcf;
116     if (denom < 0)
117     {
118         numer = -numer;
119         denom = -denom;
120     }
121 }
122 Rational::Rational()
123 {
124     numer = 0;
125     denom = 1;
126 }
127 Rational::Rational(int x, int y)
128 {
129     numer = x;
130     if (y != 0)
131         denom = y;
132     else
133         denom = 1;
134     reduce();
135 }
136 int Rational::getNumer() const
137 {
138     return numer;
139 }
140 int Rational::getDenom() const
141 {
142     return denom;
143 }
144 void Rational::setNumer(int x)
145 {
146     numer = x;
147     reduce();
```

```
148 }
149 void Rational::setDenom(int x)
150 {
151     denom = x;
152     if (denom == 0)
153         denom = 1;
154     reduce();
155 }
156 void Rational::input()
157 {
158     cout << "Numerator? ";
159     cin >> numer;
160     cout << "Denominator? ";
161     cin >> denom;
162     while (denom == 0)
163     {
164         cout << "Denominator can't be zero!\n";
165         cout << "Denominator? ";
166         cin >> denom;
167     }
168     reduce();
169 }
170 void Rational::output() const
171 {
172     if (denom != 1)
173         cout << numer << "/" << denom << endl;
174     else
175         cout << numer << endl;
176 }
177 ostream& operator<<(ostream& strm, const Rational& obj)
178 {
179     if (obj.getDenom() != 1)
180         strm << obj.getNumer() << "/" << obj.getDenom();
181     else
182         strm << obj.getNumer();
183     return strm;
184 }
185 istream& operator>>(istream& strm, Rational& obj)
186 {
187     cout << "Numerator? ";
188     strm >> obj.numer;
189     cout << "Denominator? ";
190     strm >> obj.denom;
191     while (obj.denom == 0)
192     {
193         cout << "Denominator can't be zero!\n";
194         cout << "Denominator? ";
195         strm >> obj.denom;
196     }
```

```
197     obj.reduce();
198     return strm;
199 }
200 int main()
201 {
202    int catchVar;
203    string strCatchVar;
204    Rational rationalCatchVar;
205 
206    Stack<int> stack(5);
207    stack.push(5);
208    stack.push(10);
209    stack.push(15);
210    stack.push(20);
211    stack.push(25);
212    stack.display();
213 
214    cout << "Popping...\n";
215    stack.pop(catchVar);
216    cout << catchVar << endl;
217    stack.pop(catchVar);
218    cout << catchVar << endl;
219    stack.pop(catchVar);
220    cout << catchVar << endl;
221    stack.display();
222 
223    Stack<string> strStack(5);
224    strStack.push("Jack");
225    strStack.push("Joe");
226    strStack.push("John");
227    strStack.push("Jim");
228    strStack.push("Jeff");
229    strStack.display();
230 
231    cout << "Popping...\n";
232    strStack.pop(strCatchVar);
233    cout << strCatchVar << endl;
234    strStack.pop(strCatchVar);
235    cout << strCatchVar << endl;
236    strStack.display();
237 
238    Stack<Rational> rationalStack(5);
239    rationalStack.push(Rational(3, 4));
240    rationalStack.push(Rational(-2, 7));
241    rationalStack.push(Rational(5, 6));
242    rationalStack.push(Rational(-1, 2));
243    rationalStack.push(Rational(3, 8));
244    rationalStack.display();
245 
```

```
246     cout << "Popping...\n";
247     rationalStack.pop(rationalCatchVar);
248     cout << rationalCatchVar << endl;
249     rationalStack.pop(rationalCatchVar);
250     cout << rationalCatchVar << endl;
251     rationalStack.pop(rationalCatchVar);
252     cout << rationalCatchVar << endl;
253     rationalStack.display();
254 
255     return 0;
256 }
257 
258 
```