```
//Devin Hardy
//CS372
//Infix to Postfix

#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstdlib>
#include <ctype.h>

using namespace std;

template <typename v_t>
class List
{
private:

   static const int CAP = 50;

   v_t Array[CAP];
   int pos;
   int used;

   void toShift(int form, int to);

public:

   //Constructor
   List();
```

```
    //Work Methods
    bool empty();
    void first();
    void last();
    void prev();
    void next();
    int getPos();
    void setPos(int v);
    void insertBefore(v_t item);
    void insertAfter(v_t item);
    v_t getElement();
    int size();
    void replace(v_t val);
    void erase();
    void clear();

    //Overload
    bool operator==(List L1);
    bool operator!=(List L1);
    List operator+(List L1);
    void operator+=(List L1);
    void operator=(List L1);
    friend ostream& operator<<(ostream &out, List &L1);

};

 template <typename v_t>
class Stack
```

```
{
public:
   Stack(); // Constructor
   void push(v_t val); // Add to stack
   void pop(); // Remove from stack
   bool empty(); // Is empty?
   int size(); // Size of stack?
   v_t top(); // Return top element
   void clear(); // Clear Stack

private:
   List<v_t> Lstack;

};

 template <typename v_t>
class Queue
{
public:
   Queue(); // Constructor
   void inqueue(v_t val); // Add to queue
   v_t dequeue(); // Remove from queue
   int size(); // Size of queue
   bool empty(); // Is empty?
   void clear(); // Clear Queue

private:
   List<v_t> LQueue;
```

```
};

   ////////////////////
   // List Methods
   template <typename v_t>
   List<v_t>::List()
   {
      v_t zero = 0;
      pos = 0;
      used = 0;
      for(int i = 0; i < CAP; i++)
      {
         Array[i] = zero;
      }
   }

   template <typename v_t>
   bool List<v_t>::empty()
   {
      return !used;
   }

   template <typename v_t>
   void List<v_t>::first()
   {
      pos = 0;
   }

   template <typename v_t>
```

```
void List<v_t>::last()
{
   pos = used - 1;
   if(used == 0)
      pos = 0;
}

template <typename v_t>
void List<v_t>::prev()
{
   if(used == 0)
      pos = 0;
   else if(pos < 0)
      pos = 0;
   else pos = pos - 1;
}

template <typename v_t>
void List<v_t>::next()
{
   if(used == 0)
      pos = 0;
   else if(pos > used)
      pos = used - 1;
   else
      pos = pos + 1;
}

template <typename v_t>
```

```
int List<v_t>::getPos()
{
   return pos;
}

template <typename v_t>
void List<v_t>::setPos(int v)
{
   if(used == 0)
      pos = 0;
   else if(v > used)
      pos = used - 1;
   else
      pos = v;
}

template <typename v_t>
void List<v_t>::insertBefore(v_t item)
{
   if(used == 0)
   {
      used++;
      pos = 0;
      Array[pos] = item;
   }
   else
   {
      if(used == CAP)
         return;
```

```
        else
        {
            used++;

            for(int i = used-1; i > pos; i--)
            {
                Array[i] = Array[i-1];
            }

            Array[pos] = item;
        }
    }
}

template <typename v_t>
void List<v_t>::insertAfter(v_t item)
{
    if(used == 0)
    {
        used++;
        pos = 0;
        Array[pos] = item;
    }
    else
    {
        if(used == CAP)
            return;
        else
        {
```

```
          used++;

          pos++;

          Array[pos] = item;
       }
    }
}

template <typename v_t>
v_t List<v_t>::getElement()
{
   return(Array[pos]);
}

template <typename v_t>
int List<v_t>::size()
{
   return (used);
}

template <typename v_t>
void List<v_t>::replace(v_t val)
{
   Array[pos] = val;
}

template <typename v_t>
void List<v_t>::erase()
```

```
{
   // Erase / Shift / Done
   if(used == 0)
      return;
   else
   {
      for(int i = pos; i < used; i++)
      {
         Array[i] = Array[i+1];
      }
      used--;
   }
   if(pos >= used)
      pos = used - 1;
}

template <typename v_t>
void List<v_t>::clear()
{
   used = 0;
}

//////////
//Overload

template <typename v_t>
bool List<v_t>::operator==(List<v_t> L1)
{
   int temp;
```

```
    temp = L1.getPos();
    L1.first();
    for(int i = 0; i < used; i++)
    {
        if(Array[i] != L1.getElement())
            return 0;
        L1.next();
    }
    L1.setPos(temp);
    return 1;
}

template <typename v_t>
bool List<v_t>::operator!=(List<v_t> L1)
{
    int temp;
    temp = L1.getPos();
    L1.first();
    for(int i = 0; i < used; i++)
    {
        if(Array[i] == L1.getElement())
            return 0;
        L1.next();
    }
    L1.setPos(temp);
    return 1;
}

template <typename v_t>
```

```
List<v_t> List<v_t>::operator+(List<v_t> L1)
{
   int temp1, temp2;
   int length;
   List TempL;
   temp1 = pos;
   temp2 = L1.getPos();
   length = L1.size();
   L1.first();
   pos = used - 1;
   for(int i = 0; i < used ; i++)
   {
      TempL.insertAfter(Array[i]);
   }
   for(int i = 0; i < length ; i++)
   {
      TempL.insertAfter(L1.getElement());
      L1.next();
   }
   pos = temp1;
   L1.setPos(temp2);
   return TempL;
}

template <typename v_t>
void List<v_t>::operator+=(List<v_t> L1)
{
   int temp;
   int length;
```

```
    temp = L1.getPos();
    length = L1.size();
    L1.first();
    pos = used - 1;
    for(int i = 0; i < length ; i++)
    {
        this -> insertAfter(L1.getElement());
        L1.next();
    }
    L1.setPos(temp);
    return;
}

template <typename v_t>
void List<v_t>::operator=(List<v_t> L1)
{
    int length;
    L1.first();
    length = L1.size();
    for(int i = 0; i < length ; i++)
    {
        used++;
        Array[i] = L1.getElement();
        L1.next();
    }
}

template <typename v_t>
ostream& operator<<(ostream &out, List<v_t> &L1)
```

```
{
   int length;
   length = L1.size();
   L1.first();
   for(int i = 0; i < length ; i++)
   {
      out << L1.getElement() << " ";
      L1.next();
   }
   return out;
}

//////////////////
// Stack Methods

template <typename v_t>
Stack<v_t>::Stack()
{
   Lstack.clear();
}

template <typename v_t>
void Stack<v_t>::push(v_t val) // Add to stack
{
   Lstack.last();
   Lstack.insertAfter(val);
}

template <typename v_t>
```

```
void Stack<v_t>::pop() // Remove from stack
{
   Lstack.last();
   Lstack.erase();
}

template <typename v_t>
bool Stack<v_t>::empty() // Is empty?
{
   return Lstack.empty();
}

template <typename v_t>
int Stack<v_t>::size() // Size of stack?
{
   return Lstack.size();
}

template <typename v_t>
v_t Stack<v_t>::top() // Return top element
{
   return Lstack.getElement();
}

template <typename v_t>
void Stack<v_t>::clear() // Clear Stack
{
   Lstack.clear();
}
```

```
//////////////////
// Queue Methods

template <typename v_t>
Queue<v_t>::Queue()
{
   LQueue.clear();
}

template <typename v_t>
void Queue<v_t>::inqueue(v_t val)
{
   LQueue.first();
   LQueue.insertBefore(val);
}

template <typename v_t>
v_t Queue<v_t>::dequeue()
{
   v_t val;
   LQueue.last();
   val = LQueue.getElement();
   LQueue.erase();
   return val;
}

template <typename v_t>
int Queue<v_t>::size()
```

```
{
   return LQueue.size();
}

template <typename v_t>
bool Queue<v_t>::empty()
{
   return LQueue.empty();
}

template <typename v_t>
void Queue<v_t>::clear()
{
   LQueue.clear();
}


//////////////////////////////////////////
////Fuctions

template <typename v_t>
int infixPriority(v_t item)
{
   int val;
   switch( item )
   {
   case '*':
      val = 2;
      break;
```

```
    case '/':
        val = 2;
        break;
    case '+':
        val = 1;
        break;
    case '-':
        val = 1;
        break;
    case '^':
        val = 3;
        break;
    case '(':
        val = 4;
        break;
    case ')':
        val = 0;
    case '&':
        val = 0;
        break;
    }
    return val;
}

template <typename v_t>
int stackPriority(v_t item)
{
    int val;
    switch( item )
```

```
    {
    case '*':
       val = 2;
       break;
    case '/':
       val = 2;
       break;
    case '+':
       val = 1;
       break;
    case '-':
       val = 1;
       break;
    case '^':
       val = 3;
       break;
    case '(':
       val = 0;
       break;
    case '&':
       val = 0;
       break;
    }
    return val;
  }



int main()
```

```
{
  ofstream outfile;
  ifstream infile;
  outfile.open("Outfile.out");
  infile.open("Math.txt");

  char read, test, fix;
  int infix, stk;
  float math1, math2, ans;
  char const endToken = '&';
  Stack<char> opStack;
  Queue<char> readIn;
  Queue<char> postFix;
  Stack<int> numStack;

  // Read and Work
  while(infile.peek() != EOF)
  {
     //Read
     while(infile.peek() != '\n')
     {
        infile >> read;
        readIn.inqueue(read);
     }
     infile.get(read);
     readIn.inqueue(endToken);

     //Post Fix it
     test = readIn.dequeue();
```

```
while(test != '&')
{
   outfile << test;
   if(isdigit(test))
   {
      postFix.inqueue(test);
   }
   else if(test == ')')
   {
      fix = opStack.top();
      while(fix != '(')
      {
         postFix.inqueue(fix);
         opStack.pop();
         fix = opStack.top();
      }
   }
   else
   {
      if(opStack.empty())
         opStack.push(test);
      else
      {
         infix = infixPriority<char>(test);
         fix = opStack.top();
         stk = stackPriority<char>(fix);
         if(infix <= stk)
         {
            postFix.inqueue(fix);
```

```
            opStack.pop();
         }
         opStack.push(test);
      }

   }
   test = readIn.dequeue();
}
// end token work
while(!opStack.empty())
{
   fix = opStack.top();
   postFix.inqueue(fix);
   opStack.pop();
}
postFix.inqueue(endToken);

outfile << "   ";
//Math
fix = postFix.dequeue();
while(fix != '&')
{
   if(fix != '(' || fix != ')')
      outfile << fix;
   if(isdigit(fix))
   {
      math1 = fix - '0';
      numStack.push(math1);
   }
```

```
else
{
   switch(fix)
   {
   case '+':
      math2 = numStack.top();
      numStack.pop();
      math1 = numStack.top();
      numStack.pop();
      ans = math1 + math2;
      numStack.push(ans);
      break;
   case '-':
      math2 = numStack.top();
      numStack.pop();
      math1 = numStack.top();
      numStack.pop();
      ans = math1 - math2;
      numStack.push(ans);
      break;
   case '/':
      math2 = numStack.top();
      numStack.pop();
      math1 = numStack.top();
      numStack.pop();
      ans = math1 / math2;
      numStack.push(ans);
      break;
   case '*':
```

```
            math2 = numStack.top();
            numStack.pop();
            math1 = numStack.top();
            numStack.pop();
            ans = math1 * math2;
            numStack.push(ans);
            break;
        case '^':
            math2 = numStack.top();
            numStack.pop();
            math1 = numStack.top();
            numStack.pop();
            ans = 1;
            for(int i = 0; i < math2; i++)
            {
               ans = ans * math1;
            }
            numStack.push(ans);
            break;
        }
    }
    fix = postFix.dequeue();
}
outfile << "   ";
ans = numStack.top();
outfile << ans << endl;

//Clear for next time
numStack.clear();
```

```
        readIn.clear();
        postFix.clear();
    }


    infile.close();
    outfile.close();
    return 0;
}
```