//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;

}