# 头文件

#include <iostream>

using namespace std;

enum Error\_code {

success, fail, scope\_error, underflow, overflow, fatal,

not\_present, duplicate\_error, entry\_inserted, entry\_found,

internal\_error

};

#define maxstack 100

template <class Node\_entry>

struct Node {

// data members

Node\_entry entry;

Node<Node\_entry>\* next;

// constructors

Node();

Node(Node\_entry item, Node<Node\_entry>\* link = NULL);

};

template <class Node\_entry>

Node<Node\_entry>::Node()

{

next = NULL;

}

template <class Node\_entry>

Node<Node\_entry>::Node(Node\_entry item, Node<Node\_entry>\* link) {

entry = item;

next = link;

}

template <class stack\_entry>

class stack {

public:

// Specifications for the methods of the list ADT go here.

stack();

bool empty() const;

Error\_code push(const stack\_entry& x);

Error\_code top(stack\_entry& item) const;

Error\_code pop();

// The following methods replace compiler-generated defaults.

~stack();

stack(const stack<stack\_entry>& copy);

void operator = (const stack<stack\_entry>& copy);

protected:

// Data members for the linked list implementation now follow.

int count;

Node<stack\_entry>\* head;

// The following auxiliary function is used to locate list positions

Node<stack\_entry>\* set\_position(int position) const;

};

template <class stack\_entry>

stack<stack\_entry>::stack()

{

count = 0;

head = NULL;

}

template <class stack\_entry>

bool stack<stack\_entry>::empty() const

{

return (count == 0);

}

template <class stack\_entry>

Node<stack\_entry>\* stack<stack\_entry>::set\_position(int position) const

/\* Pre: position is a valid position in the List; 0 ≤ position < count.

Post: Returns a pointer to the Node in position. \*/ {

Node<stack\_entry>\* q = head;

for (int i = 0; i < position; i++)

q = q->next;

return q;

}

template <class stack\_entry>

Error\_code stack<stack\_entry>::top(stack\_entry& item) const

/\* Pre: None.

Post: If the Stack is not empty, the top of the Stack is returned in item. If the

Stack is empty an Error\_code of underflow is returned. \*/

{

Error\_code outcome = success;

Node<stack\_entry> \* previous;

if (count == 0)

outcome = underflow;

else {

previous = set\_position(count - 2);

item = previous->entry;

}

return outcome;

}

template <class stack\_entry>

Error\_code stack<stack\_entry>::pop()

{

Node<stack\_entry>\* p, \* previous;

Error\_code outcome = success;

if (count == 0)

outcome = underflow;

else {

double x;

if (count == 1) {

p = head;

head = head->next;

x = p->entry;

delete p;

count--;

return success;

}

else {

p = set\_position(count - 1);

previous = set\_position(count - 2);

previous->next = p->next;

x = p->entry;

delete p;

count--;

return success;

}

}

}

template <class stack\_entry>

Error\_code stack<stack\_entry>::push(const stack\_entry& x)

/\* Post: If the List is not full and 0 <= position < n, where n is the number of

entries in the List, the function succeeds: Any entry formerly at position

and all later entries have their position numbers increased by 1, and x is

inserted at position of the List.

Else: The function fails with a diagnostic error code. \*/

{

Node<stack\_entry>\* new\_node, \* previous, \* following;

Error\_code outcome = success;

if (count >= maxstack)

outcome = overflow;

else if (count == 0) {

following = head;

new\_node = new Node<stack\_entry>(x, following);

head = new\_node;

}

else {

previous = set\_position(count - 2);

following = previous->next;

new\_node = new Node<stack\_entry>(x, following);

previous->next = new\_node;

}

count++;

return outcome;

}

# 主函数

#include <iostream>

#include <cctype>

#include "l\_s.h"

using namespace std;

void introduction();

void instructions();

char get\_command();

bool do\_command(char command, stack<double>& numbers);

int main() {

stack<double> stored\_numbers;

introduction();

instructions();

while (do\_command(get\_command(), stored\_numbers));

}

void introduction() {

cout << "This program is a reverse Polish calculator." << endl << endl;

}

void instructions() {

cout << "Instructions are shown below:" << endl

<< " [?]push to stack" << endl

<< " [=]print top" << endl

<< " [+] [-] [\*] [/] are arithmetic operations" << endl

<< " [Q]uit." << endl << endl

<< "For example:" << endl

<< " ?a?b+= means a+b" << endl << endl;

}

char get\_command()

{

char command;

bool waiting = true;

cout << "Select command and press < Enter > :";

while (waiting) {

cin >> command;

command = tolower(command);

if (command == '?' || command == '=' || command == '+' ||

command == '-' || command == '\*' || command == '/' ||

command == 'q')

waiting = false;

else {

cout << "Please enter a valid command:" << endl

<< "[?]push to stack " << endl

<< "[=]print top" << endl

<< "[+] [-] [\*] [/] are arithmetic operations" << endl

<< "[Q]uit." << endl;

}

}

return command;

}

bool do\_command(char command, stack<double>& numbers)

/\* Pre: The first parameter specifies a valid calculator command.

Post: The command specified by the first parameter has been applied to the

Stack of numbers given by the second parameter. A result of true is returned unless command == 'q .

Uses: The class Stack. \*/ {

double p, q;

switch (command) {

case '?':

cout << "Enter a real number: " << flush;

cin >> p;

if (numbers.push(p) == overflow)

cout << "Warning: Stack full, lost number" << endl;

break;

case '=':

if (numbers.top(p) == underflow)

cout << "Stack empty" << endl;

else

cout << p << endl;

break;

case '+':

if (numbers.top(p) == underflow)

cout << "Stack empty" << endl;

else {

numbers.pop();

if (numbers.top(q) == underflow) {

cout << "Stack has just one entry" << endl;

numbers.push(p);

}

else {

numbers.pop();

if (numbers.push(q + p) == overflow)

cout << "Warning: Stack full, lost result" << endl;

}

}

break;

// Add options for further user commands.

case '-':

if (numbers.top(p) == underflow)

cout << "Stack empty" << endl;

else {

numbers.pop();

if (numbers.top(q) == underflow) {

cout << "Stack has just one entry" << endl;

numbers.push(p);

}

else {

numbers.pop();

if (numbers.push(q - p) == overflow)

cout << "Warning: Stack full, lost result" << endl;

}

}

break;

case '\*':

if (numbers.top(p) == underflow)

cout << "Stack empty" << endl;

else {

numbers.pop();

if (numbers.top(q) == underflow) {

cout << "Stack has just one entry" << endl;

numbers.push(p);

}

else {

numbers.pop();

if (numbers.push(q \* p) == overflow)

cout << "Warning: Stack full, lost result" << endl;

}

}

break;

case '/':

if (numbers.top(p) == underflow)

cout << "Stack empty" << endl;

else {

numbers.pop();

if (numbers.top(q) == underflow) {

cout << "Stack has just one entry" << endl;

numbers.push(p);

}

else {

if (p == 0) {

cout << " cannot divided by 0" << endl;

numbers.push(p);

break;

}

numbers.pop();

if (numbers.push(q / p) == overflow)

cout << "Warning: Stack full, lost result" << endl;

}

}

break;

case 'q':

cout << "Calculation finished.\n";

return false;

}

return true;

}