**编写DLL实现表达式中缀转后缀**

一、 DLL是什么？

  动态链接库（Dynamic Link Library）DLL文件与EXE文件一样也是可执行文件，但是DLL也被称之为库，因为里面封装了各种类、函数之类的东西，就像是一个库一样，存储着很多东西，主要是用来调用的。调用方式主要分为两种：隐式（通过lib文件与头文件） 与 显式（只通过DLL文件）。

二、为什么使用DLL?

微软方面一边开源了很多代码，比如MFC的，这些代码是直接暴露在程序员眼皮子底下的，和容易调用，当然也很容易与程序员的代码发生冲突，业内也称之为“白盒复用”，有一定的优势，但是也有弊端。在此基础上，我们再来聊DLL就比较好，DLL将要一系列的代码转化成二进制，直接供程序使用，称之为“黑盒复用”。

动态链接库DLL最大的好处就是节省内存,程序在没有运行到dll里的函数的时候，dll里的函数就没有运行，只有在运行到了dll，才通过dll的接口函数访问dll，这样比直接把函数代码写到主题函数，或静态链接库相比，运行的效率有了很大的提高．其次使用DLL文件编程，其更新也较为方便，不需要与程序链接。

三、C++编写DLL实现中缀转后缀

步骤：

1. 导出整个class

方法很简单，只需要在类的头文件中class和类名之间加上\_declspec(dllexport)，同时在另外一份提供给客户端调用程序使用的类的头文件中class和类名之间加上\_declspec(dllimport)。为了能让客户端程序和DLL程序公用该类的一份头文件，通常在类的头文件中使用宏和预编译指令来处理。

代码如下：

toPostfix.h:



toPostfix.cpp:

#include "stdafx.h"

#define DLL\_API \_declspec(dllexport) //导出类

#include<iostream>

#include<stdlib.h>

#include<string.h>

#include"toPostfix.h"

const int maxsize = 100;

using namespace std;

Stack::Stack()

{

s = new char[maxsize];

top = 0;

}

Stack::~Stack()

{

delete[]s;

}

void Stack:: push(char e)

{

if (top > maxsize - 1)

cout << "overflow!" << endl;

else

s[top++] = e;

}

int Stack:: pop()

{

return s[--top];

}

bool Stack:: isEmpty()

{

if (top == 0)

return true;

else

return false;

}

char Stack:: gettop()

{

return s[top - 1];

}

void Stack::Check(char \*s)

{

Stack bStack;//用于判断括号匹配

int m, len;

len = strlen(s);

for (m = 0; m < len; )

{

if (s[0] == ')' || s[0] == '+' || s[0] == '-' || s[0] == '\*' || s[0] == '/')//输入的第一个字符不能是操作符

{

cout << "输入错误，第一个字符不能为操作符和右括号" << endl;

break;

}

else

{

if ((s[m] >= 40 && s[m] <= 57) && s[m] != 44 && s[m] != 46)//判断输入的字符是否属于 +-\*/ 0123456789 ( )

{

if ((s[m] == '+' || s[m] == '-' || s[m] == '\*' || s[m] == '/') && (s[m + 1] == '+' || s[m + 1] == '-' || s[m + 1] == '\*' || s[m + 1] == '/'))//判断两个操作符是否挨着

{

cout << "输入错误,两个操作数不能挨着" << endl;

break;

}

else if ((s[m] >= '0' && s[m] <= '9') && (s[m + 1] >= '0' && s[m + 1] <= '9'))//判断两个数字是否挨着

{

cout << "输入错误，两个数字不能挨着" << endl;

break;

}

else //判断括号是否匹配

{

if (s[m] == '(')

{

bStack.push(s[m]);

}

if (s[m] == ')')

{

if (bStack.isEmpty())

{

printf("输入错误，括号不匹配");

}

bStack.pop();

}

if (m == len - 1)//遍历完毕，栈应为空

{

if (!bStack.isEmpty())

{

cout << "输入错误，括号不匹配" << endl;

break;

}

else

{

Transform(s);//调用函数将中缀表达式转换为后缀表达式

break;

}

}

else

m++;

}

}

else

{

cout << "输入错误，字符超出可选范围" << endl;

break;

}

}

}

}

void Stack::Transform(char \*s)

{

Stack mystack;

char index[20];

char ch;

int q = 0;

int len = strlen(s);

cout << "将其转化为后缀表达式后的结果为:";

for (int p = 0; p < len; p++)

{

if (s[p] >= '0' && s[p] <= '9') //如果遇到操作数，直接将其输出

{

cout << s[p];

}

if (s[p] == '(') //如果遇到左括号，直接将其入栈

{

mystack.push(s[p]);

}

if (s[p] == '+' || s[p] == '-' || s[p] == '\*' || s[p] == '/')//如果遇到操作符

{

if (mystack.isEmpty() || mystack.gettop() == '(')

{//如果栈为空或者栈顶为左括号，则入栈

mystack.push(s[p]);

}

else if ((s[p] == '\*' && (mystack.gettop() == '-' || mystack.gettop() == '+')) || (s[p] == '/' && (mystack.gettop() == '-' || mystack.gettop() == '+')))

{//如果进行判定的字符的优先级比栈顶元素优先级高，则入栈

mystack.push(s[p]);

}

else

{//如果优先级低，则栈顶元素出栈，再判断

do

{

char ch = mystack.pop();

cout << ch;

} while (!(mystack.isEmpty() || mystack.gettop() == '(' || (s[p] == '\*' && (mystack.gettop() == '-' || mystack.gettop() == '+')) || (s[p] == '/' && (mystack.gettop() == '-' || mystack.gettop() == '+'))));

mystack.push(s[p]);

}

}

if (s[p] == ')')//当遇到右括号时,弹出栈内字符，直到遇到 左括号为止（左括号不输出）

{

while (1)

{

char ch = mystack.pop();//取出栈顶元素

if (ch == '(')

{

q--;

break;

}

else

{

cout << ch;

q--;

}

}

}

}

if (!mystack.isEmpty())//当以上所有工作都完成以后，判断栈是否为空，如果不为空，则将栈中剩余的元素全部取出

{

while (!mystack.isEmpty())

{

ch = mystack.pop();

cout << ch;

}

}

cout << endl;

}

这样，在DLL编译时DLL\_API被定义为\_declspec(dllexport)，而且客户端程序编译时它被定义为\_declspec(dllimport)

DLL编写完成后，就只剩下客户端程序如何去调用该DLL了

1. DLL调用有两种方式静态调用和动态调用。这里使用静态调用

静态调用很简单，只需要将toPostfix.h头文件和toPostfix.lib，toPostfix.dll文件拷贝到客户端程序的当前目录下，在客户端程序中#include< toPostfix.h>，然后通过#pragma comment(lib,” toPostfix.lib”)的方式引入lib库，或者在客户端程序的工程属性里面增加对该lib文件的引入即可。

代码如下：



这样即可在客户端程序实现中缀转后缀。

测试结果如下：

