1.

（1）通过返回值将新数c传给主函数:

#include <iostream>

using namespace std;

int fun(int a, int b)

{

int c = (b / 10) + 10 \* (a / 10) + 100 \* (b % 10) + 1000 \* (a % 10);

return c;

}

int main()

{

int a;

int b;

cout << "输入一个两位数a："<< endl;

cin >> a;

cout << "输入一个两位数b："<< endl;

cin >> b;

int c = fun(a, b);

cout << "结果为：" << c << endl;

return 0;

}

（2）通过引用参数将新数传给主函数:

#include <iostream>

using namespace std;

int fun(int &a, int &b)

{

int c;

c = (b / 10) + 10 \* (a / 10) + 100 \* (b % 10) + 1000 \* (a % 10);

return c;

}

int main()

{

int x, y;

cout << "输入一个两位数x：" << endl;

cin >> x;

cout << "输入一个两位数y：" << endl;

cin >> y;

cout << "结果为：" << fun(x, y) << endl;

return 0;

}

（3）通过指针型参数将新数传给主函数:

#include <iostream>

#include <string>

using namespace std;

int fun(int \*a, int \*b)

{

int c = (\*b / 10) + 10 \* (\*a / 10) + 100 \* (\*b % 10) + 1000 \* (\*a % 10);

return c;

}

int main()

{

int a;

int b;

cout << "输入一个两位数a：" << endl;

cin >> a;

cout << "输入一个两位数b：" << endl;

cin >> b;

int c = fun(&a, &b);

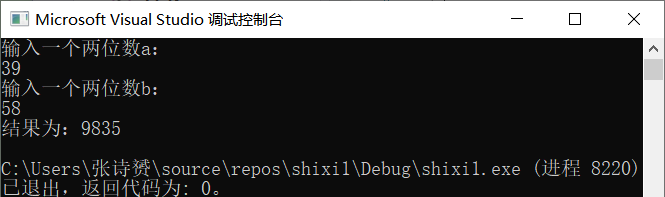
cout << "结果为：" << c << endl;

return 0;

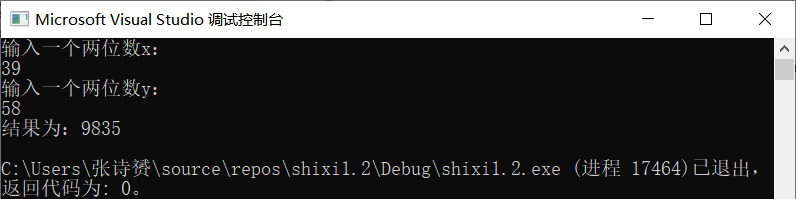
}

运行截图：

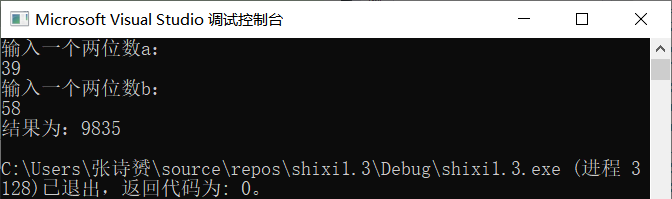
（1）通过返回值将新数c传给主函数:



（2）通过引用参数将新数传给主函数:



（3）通过指针型参数将新数传给主函数:



2.

（1）数组：

#include<iostream>

using namespace std;

int fun(int N, int M)

{

int i;

int monkey[100];

for (i = 0; i < N; i++)

monkey[i] = 1; //猴子状态初始化，为1表示可能被选上；

int sum = 0, //循环记数；

count = N; //累积记数初始化，大于1表明还有大王候选人；

while (count > 1)

{

count = 0;

for (i = 0; i < N; i++)

{

sum += monkey[i];

if (sum == M)

sum = monkey[i] = 0; //淘汰猴子；

count += monkey[i];

}

}

for (i = 0; i < N; i++)

if (monkey[i] != 0)

return i; //找到猴子大王编号（从0开始的）；

}

void main()

{

int m, n;

cout << "输入猴子总数m: ";

cin >> m;

cout << "输入个数n: ";

cin >> n;

cout << "第" << fun(m, n) + 1 << "个猴子是猴子大王" << endl;

}

1. 链表：

#include <iostream>

using namespace std;

struct monkey

{

int num;

monkey\* next;

};

monkey\* head, \*tail;

void creat(int M)

{

int i;

monkey\* p, \*q;

p = new monkey;

p->num = 1;

p->next = NULL;

head = p;

q = p;

for (i = 2; i <= M; i = i + 1)

{

p = new monkey;

p->num = i;

q->next = p;

q = p;

p->next = NULL;

}

tail = q;

tail->next = head;

}

void fun(int N)

{

int x = 0;

monkey\* p, \*q;

q = tail;

do

{

p = q->next;

x = x + 1;

if (x % N == 0)

{

cout << p->num << " ";

q->next = p->next;

delete p;

p = NULL;

}

else

q = p;

} while (q != q->next);

head = q;;

}

int main()

{

int n, m;

head = NULL;

cout << "输入猴子总数m: ";

cin >> m;

cout << "输入个数n: ";

cin >> n;

creat(m);

cout << "离开的猴子是: ";

fun(n);

cout << " " << endl;

cout << "第" << head->num << "个猴子是猴子大王" << endl;

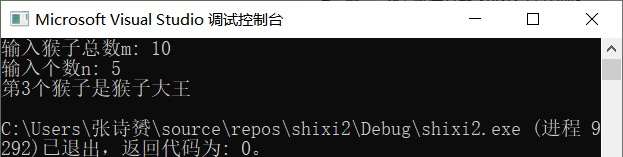
delete head;

return 0;

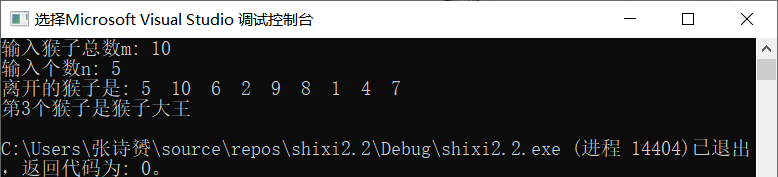
}

运行截图：

（1）数组：



（2）链表：



3.

#include<iostream>

#include <algorithm>

using namespace std;

struct student

{

int ch, math, eng, sum, num;

}stu[10];

bool cmp(student stu1, student stu2) //排序依据

{

if (stu1.sum != stu2.sum)

return stu1.sum > stu2.sum; //总成绩高则排在前

else if (stu1.ch != stu2.ch)

return stu1.ch > stu2.ch; //语文成绩高则排在前

else return stu1.num < stu2.num;

}

int main() {

int i, n;

bool first = true; //每个案例中间隔一个空行

while (cin >> n)

{

for (i = 0; i < n; i++) {

stu[i].num = i + 1;

cin >> stu[i].ch >> stu[i].math >> stu[i].eng;

stu[i].sum = stu[i].ch + stu[i].math + stu[i].eng;

}

if (first) first = false;

else cout << endl;

sort(stu, stu + n, cmp); //数组中从第一个到最后一个以定义的cmp函数要求排序

for (i = 0; i < 5; i++)

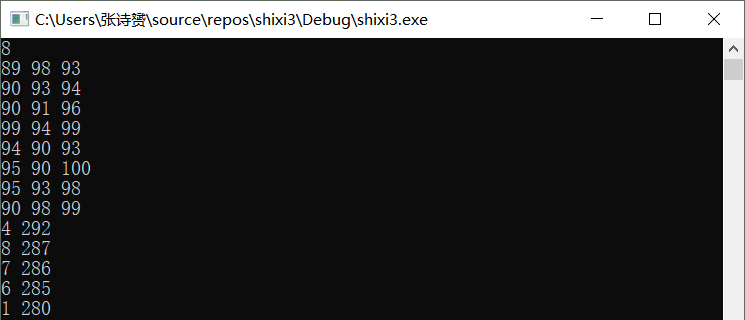
cout << stu[i].num << " " << stu[i].sum << endl;

}

return 0;

}

运行截图：



4.

#include <stdio.h>

#include <iostream>

#include <algorithm>

using namespace std;

#define maxsize 100

int main()

{

int M, N, c;

int i, j, m, n, k = 0, count = 0;

char a[maxsize][maxsize] = { 0 };

char(\*p)[4] = 0;

cout << "输入地图的行数和列数：";

cin >> M >> N;

printf("输入感染信号\n");

for (i = 0; i < N; i++)

{

for (j = 0; j < M; j++)

{

cin >> a[i][j];

}

}

/\* for(i=0; i<N; i++)

{

for(j=0; j<M; j++)

{

cout<<a[i][j];

}

cout<<endl;

}

\*/

cout << "输入周期数：";

cin >> c;

while (k < c)

{

for (i = 0; i < N; i++)

{

for (j = 0; j < M; j++)

{

if (a[i][j] == 'X')

{

{ if (a[i - 1][j] = 'O')

a[i - 1][j] = 'm';

if (a[i + 1][j] = 'O')

a[i + 1][j] = 'm';

if (a[i][j + 1] = 'O')

a[i][j + 1] = 'm';

if (a[i][j - 1] = 'O')

a[i][j - 1] = 'm';

}

}

}

}

k++;

for (m = 0; m < N; m++)

{

for (n = 0; n < M; n++)

{

if (a[m][n] == 'm')

a[m][n] = 'X';

}

}

}

for (i = 0; i < N; i++)

{

for (j = 0; j < M; j++)

{

cout << a[i][j];

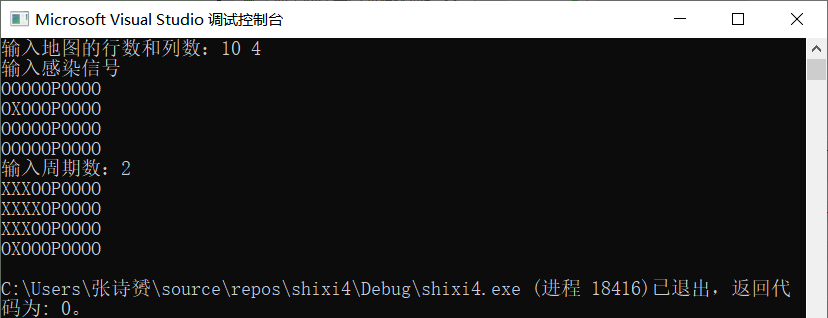
}

cout << endl;

}

}

运行截图：



5.

**Calculator.h：**

using namespace std;

class Calculator

{

private://私有成员（数据成员）

MyStack<char> optr;//操作符栈

MyStack<double> opnd;//操作数栈

MyStack<char> bracket;//用于存放左括号的栈

public://公有成员（供外界调用的成员函数，即方法）

Calculator()//构造函数

{

optr.setnull();

opnd.setnull();

bracket.setnull();

optr.push('#');//‘#’作为栈底哨兵

}

void evaluate();//利用栈结构将中缀表达式转为PRN来计算，并检查输入的合法性

void clear()//清空计算器

{

optr.setnull();

optr.setnull();

bracket.setnull();

optr.push('#');//与构造的实现是一致的

}

private://私有成员（对外隐藏的成员函数，只供内部调用）

int lp(char op);//返回左端操作符的优先级

int rp(char op);//返回右端操作符的优先级

double operate(char theta, double a, double b);//进行后缀表达式的运算，返回运算结果

};

**MyStack.h：**

const int maxstack = 10;//栈内元素个数的最大值

enum Error\_code { success, underflow, overflow };//函数运行成功与否的内部标签

template <class Stack\_entry>//类模板，数据类型叫Stack\_entry

class MyStack

{

public: //共有成员（类的所有成员函数）

MyStack();

bool empty() const;

Error\_code push(const Stack\_entry &item);

Error\_code pop();

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

Stack\_entry top()const; //重载top函数以适应不同的需要

void setnull();//清空栈

private: //私有成员（数据成员）

int count; //计数器

Stack\_entry entry[maxstack];//用数组实现存储

};

**Calculator.cpp:**

#include <iostream>

#include "MyStack.cpp"

#include "Calculator.h"

#include <ctype.h>//调用了isdigit和isalpha函数

using namespace std;

void Calculator::evaluate()//利用栈结构将键盘输入的中缀表达式转为RPN计算，并检查输入的正确性

{

char ch, op, theta;//当前读入字符、操作符、运算符

double val, a, b;//数值、左操作数、右操作数

cin >> ch;

op = '=';

bool is\_matched = true;//标记左右括号是否匹配

do

{

while (isalpha(ch)) { cin >> ch; }//过滤非法字符输入（英文字母）

is\_matched = true;

switch (ch)//左右括号匹配检查

{

case '(': bracket.push(ch); break;//左括号入栈

case ')': if (bracket.empty())

//右括号没有已入栈的左括号来匹配时，输入非法

is\_matched = false;

else

{

bracket.pop(); is\_matched = true;

}//匹配时，弹出一个左括号

break;

default: break;//非括号的字符一律过滤掉

}

while (is\_matched)//右括号匹配时，执行中缀表达式运算

{

if (isdigit(ch))//当前字符为数字时，将它放入操作数栈

{//isdigit遇到多位数会自动识别吗？

cin.putback(ch);//答：当然不能。因为isdigt的参数是char型的，它只能判断一位字符

cin >> val;//这也是为什么要cin.putback(ch)再cin>>out了。cin的功能很强大，自动匹配数据类型，而不用scanf的各种格式化输入符

opnd.push(val);

cin >> ch; break;

}

else if (lp(op) < rp(ch))

{//当前操作符优先级高于栈顶操作符时，入栈

optr.push(ch);

op = optr.top();

cin >> ch; break;

}

else if (lp(op) == rp(ch))

{//当前操作符优先级等于栈顶操作符时，当前栈顶操作符出栈

optr.pop();

op = optr.top();

cin >> ch; break;

}

else if (lp(op) > rp(ch))

{//当前操作符优先级低于栈顶，将栈顶取出进行运算，再将结果放回操作数栈

theta = op; optr.pop();

b = opnd.top(); opnd.pop();

a = opnd.top(); opnd.pop();//注意两个操作数的出栈顺序

opnd.push(operate(theta, a, b));

op = optr.top();

if ((ch == ')') && (op == '('))//解决左右括号匹配后的问题

{

optr.pop(); op = optr.top();//左括号出栈

cin >> ch;//刷新当前字符（避免对右括号二次判断）

if (ch == '(') is\_matched = false;//右括号紧邻左括号，报错

}

break;

}

}

if (is\_matched == false || (op == '(' && ch == '#'))

{//最后还剩没配上对的左括号，或者右括号没有左括号来匹配，则输入非法

cout << "Bad match." << endl;

ch = op = '#';

is\_matched = false;

}

} while (ch != '#' || op != '#');

if (is\_matched == true) cout << '=' << opnd.top();//循环结束后，结果在操作数栈顶

else {

cout << "sorry, cannot compute this expression." << endl;

}

}

int Calculator::lp(char op)//返回左端操作符的优先级

{

int prior;//优先级

switch (op)

{

case '+'://将优先级表的内容写入

case '-': prior = 3; break;

case '\*':

case '/': prior = 5; break;

case '(': prior = 1; break;

case ')': prior = 6; break;

case '#': prior = 0; break;

}

return prior;

}

int Calculator::rp(char op)//返回右端操作符的优先级

{

int prior;//优先级

switch (op)

{

case '+':

case '-': prior = 2; break;

case '\*':

case '/': prior = 4; break;

case '(': prior = 6; break;

case ')': prior = 1; break;

case '#': prior = 0; break;

}

return prior;

}

double Calculator::operate(char theta, double a, double b)

//进行后缀表达式的运算，返回运算结果

{

switch (theta)

{

case '+': return a + b;

case '-': return a - b;

case '\*': return a \* b;

case '/': return a / b;

default: return 0;

}

}

**MyStack.cpp:**

#include <iostream>

#include "MyStack.h"

using namespace std;

template <class Stack\_entry>

MyStack<Stack\_entry>::MyStack()

/\*Pre: None.

Post: The stack is initialized to be empty.\*/

{

count = 0;

}

template <class Stack\_entry>

bool MyStack<Stack\_entry>::empty() const

/\*Pre: None.

Post: if the stack is empty, true is returned. Ohterwise

false is returned.\*/

{

bool outcome = true;

if (count > 0) outcome = false;

return outcome;

}

template <class Stack\_entry>

Error\_code MyStack<Stack\_entry>::push(const Stack\_entry &item)

/\*Pre: None.

Post: if the stack is not full, item is added to the top

of the stack. Ohterwise an error code of overflow is returned

and the stack is left unchanged.\*/

{

Error\_code outcome = success;

if (count >= maxstack) outcome = overflow;

else entry[count++] = item;

return outcome;

}

template <class Stack\_entry>

Error\_code MyStack<Stack\_entry>::pop()

/\*Pre: None.

Post: If the stack is not empty, the top of the stack is

removed. If the stack is empty, an Error code of underflow

is returned.\*/

{

Error\_code outcome = success;

if (count == 0) outcome = underflow;

else --count;

return outcome;

}

template <class Stack\_entry>

Error\_code MyStack<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;

if (count == 0) outcome = underflow;

else item = entry[count - 1];

return outcome;

}

template <class Stack\_entry>

Stack\_entry MyStack<Stack\_entry>::top()const//重载top函数以适应不同的需要

{

if (count == 0) underflow;

return entry[count - 1];

}

template <class Stack\_entry>

void MyStack<Stack\_entry>::setnull()

{

count = 0;//与构造实现是一致的

}

**Shixi5.cpp:**

#include <iostream>

#include "MyStack.cpp"

#include "Calculator.h"

using namespace std;

int main()

{

Calculator calc1;//创建计算器对象

do

{

calc1.clear();

cout << "Please input an expression, ended with a '#'" << endl;

calc1.evaluate();//计算表达式并显示结果

cout << endl << "Would you like to continue? [y/n]" << endl;

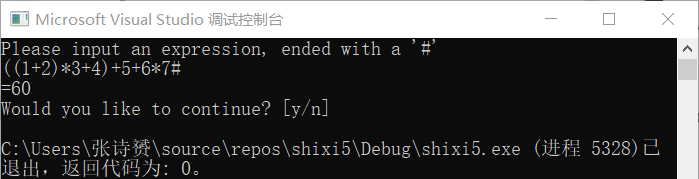
fflush(stdin);

} while (cin.get() == 'y');

return 0;

}

运行截图：



6.

#include<iostream>

#include <math.h>

using namespace std;

class vehicle

{

protected:

int wheels;

int weight;

public:

vehicle(int wh, int we)

{

wheels = wh;

weight = we;

}

void disp()

{

cout << "wheels=" << wheels << " weight=" << weight << endl;

}

};

class car :private vehicle

{

int passenger\_load;

public:

car(int wh, int we, int pa) :

vehicle(wh, we)

{

passenger\_load = pa;

}

void disp()

{

vehicle::disp();

cout << "passenger\_load=" << passenger\_load << endl;

}

};

class truck :private vehicle

{

int passenger\_load;

int payload;

public:

truck(int wh, int we, int pa, int pay) :vehicle(wh, we)

{

passenger\_load = pa;

payload = pay;

}

void disp()

{

vehicle::disp();

cout << "passenger\_load=" << passenger\_load << " payload=" << payload << endl;

}

};

int main()

{

vehicle v(4,1000);

v.disp();

car c(4,1500,5);

c.disp();

truck t(6,14000,3,16000);

t.disp();

system("pause");

}

运行截图：

