项目说明文档

数据结构课程设计

——算数表达式求解

同济大学

Tongji University

目 录

[1 分析 1](#_Toc12003)

[1.1 背景分析 1](#_Toc25233)

[1.2 功能分析 1](#_Toc16483)

[2 设计 1](#_Toc19343)

[2.1 数据结构设计 1](#_Toc20115)

[2.2 类结构设计 2](#_Toc24729)

[2.3 成员与操作设计 2](#_Toc32582)

[2.4 系统设计 2](#_Toc31986)

[3 实现 3](#_Toc28359)

[3.1 翻译功能的实现 3](#_Toc27927)

[3.1.1 翻译功能流程图 3](#_Toc13295)

[3.1.2 输入功能核心代码 3](#_Toc8832)

[3.2 计算功能的实现（非括号） 6](#_Toc13036)

[3.2.1 计算功能流程图 6](#_Toc30602)

[3.2.2 计算功能核心代码 6](#_Toc17454)

[3.3 计算功能（括号）的实现 10](#_Toc8305)

[3.3.1 计算功能（括号）流程图 10](#_Toc30699)

[3.3.2 计算功能括号核心代码 10](#_Toc27041)

[3.4 错误判断功能的实现 15](#_Toc20094)

[3.4.1 错误判断功能流程图 15](#_Toc21913)

[3.4.2 判断功能核心代码 15](#_Toc12884)

[3.5 修改功能的实现 16](#_Toc26127)

[3.5.1 修改功能流程图 16](#_Toc16995)

[3.5.2 统计功能核心代码 16](#_Toc20158)

[3.7总体系统实现 18](#_Toc3421)

[3.7.1 总体系统流程图 18](#_Toc16498)

[3.7.2总体系统核心代码 18](#_Toc6635)

[4 测试 23](#_Toc2120)

[4.1 功能测试 23](#_Toc5129)

[4.1.1 计算功能（非括号）测试 23](#_Toc22189)

[4.1.2 计算功能（括号）测试 24](#_Toc23993)

[4.2 边界测试 24](#_Toc6560)

[4.2.1 输入不正确的字符 24](#_Toc30968)

[4.2.2 括号不匹配 25](#_Toc20281)

[4.2.3 除数为0 25](#_Toc10055)

[4.2.4 \*/%^前面字符非法 26](#_Toc10137)

[4.2.5 %后面字符非法 26](#_Toc609)

[4.2.6 输入的数字太少 26](#_Toc16192)

# 1 分析

## 1.1 背景分析

计算表达式是计算机的基础功能之一，计算机的研发给数值计算带来了巨大的突破，但是说到底计算机是基于电路的一种器械。它的基础是二进制，就算在高级的计算语言中也无法直接计算一个中缀的表达式，我们要将一段人们常见的表达式翻译成计算机语言，再进行计算，并检查是否输入正确，最后将答案输出。

## 1.2 功能分析

作为一个简易的计算中缀表达式的程序，它首先需要先把输入的一段字符串的中缀表达式转化成对应的数据，将符号和数字分开来，放入我们建立的一个符号栈和一个数字栈中，然后要设计一个运算逻辑，当什么时候什么符号进入栈中后要进行运算，什么时候要运算停止，同时判断该表达式是否有错误出现。综上所述，我们需要一个翻译功能，一个计算功能，一个计算功能（括号专有），一个错误判断功能，一个模块选择功能

# 

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该系统要求大量的增加、删除、修改操作，同时面对顺序有特殊的要求，而栈和链表进行增加、删除等操作十分简便，因此考虑使用链表数据结构。同时，为了实现简易，在第一个结点之前附加一个头结点，这样就使得增加或者删除头结点与处理其他结点方法相同，使得程序简洁。

## 2.2 类结构设计

经典的链表一般包括两个抽象数据类型（ADT）——链表结点类（date）与链表类（list），而两个类之间的耦合关系可以采用嵌套、继承等多种关系。为方便处理，本系统采用struct描述链表结点类(date)，这样使得链表结点类（list）可以访问链表结点。

## 2.3 成员与操作设计

**链表类（list）**

class numberstack //数字栈

{

public:

double number;

numberstack \* link;

numberstack() { number = 0; }

};

class charstack //字符栈

{

public:

char sign;

charstack \* link;

charstack() { sign = ' '; }

## }

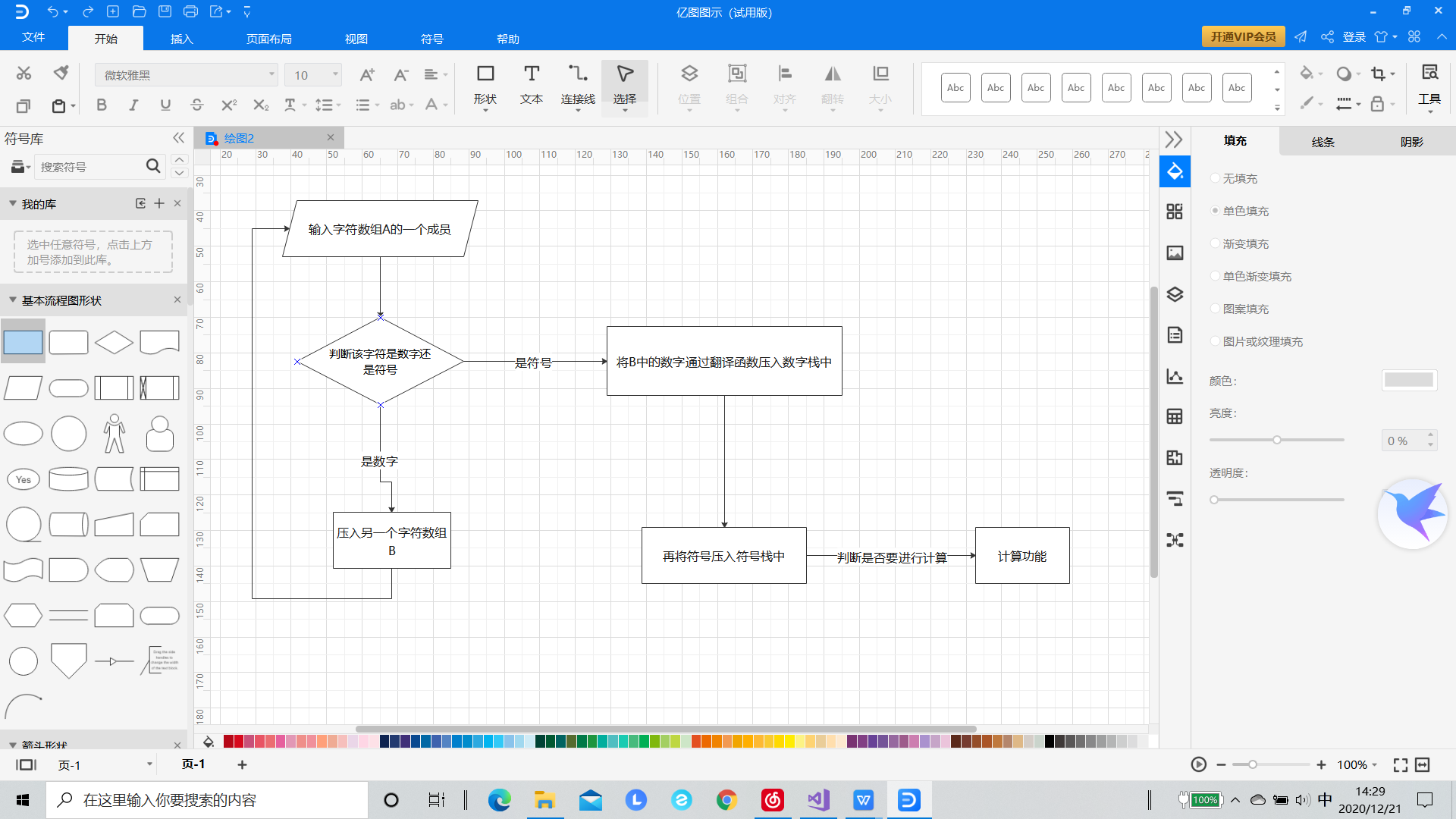
## ;2.4 系统设计

系统首先建立两个栈，操作者通过键盘输入一个中缀表达式进行计算，然后可以选择是否继续进行输入还是退出

# 3 实现

## 3.1 翻译功能的实现

### 3.1.1 翻译功能流程图



### 3.1.2 输入功能核心代码

翻译函数

double changechar(char a[5], int b,int \*c)//将字符变成double

{

double number = 0;

for (int n = 0; n <= b; n++)

{

number = number + (int(a[n]) - 48)\*pow(10, b - n);

}

if (\*c == 1 || \*c == 0)

{

\*c = 0; return number;

}

if (\*c == -1)

{

\*c = 0; return -number;

}

}

int checkchar(char a) //判断字符

{

switch (a)

{

case '#': {return -1; break; }

case '+': {return 1; break; }

case '-': {return 1; break; }

case '\*': {return 2; break; }

case '/': {return 2; break; }

case '%': {return 2; break; }

case '^': {return 3; break; }

case '(': {return -3; break; }

case ')': {return -4; break; }

case 10: {return-1; break; }

default: return 0;

break;

}

}

主函数中翻译片段

if (int(putin[n]) >= 48 && int(putin[n]) <= 57)

{

checkputinchar1 = 1;

putinchar[checkputinchar2] = putin[n];

checkputinchar2++;

} //数字处理

if (int(putin[n]) < 48 || int(putin[n]) > 57)

{

if (checkchar(putin[n]) == 0) { answerkind = 1; break; }//错误弹出

if (judge(putin, n) != 0) { answerkind = judge(putin, n); break; };

if (checkputinchar1 == 1) //进行数字栈的存储

{

numberstack1++;

head1->number = changechar(putinchar, checkputinchar2 - 1, &changenumber);//放入栈中

current1 = new numberstack;

current1->link = head1;

head1 = current1;

checkputinchar1 = 0;

checkputinchar2 = 0;

### }

## 3.2 计算功能的实现（非括号）

### 3.2.1 计算功能流程图

### 

### 3.2.2 计算功能核心代码

计算函数count：

int count(numberstack \*&head1, charstack \*&head2,char a)//对数字栈和字符栈操作

{

double temp;

numberstack \*p1;

charstack \*p2;

switch (head2->link->sign) //进行符号的计算

{

case '+':

{

temp = head1->link->number;

p1 = head1;

head1 = head1->link;

delete p1;

head1->link->number = head1->link->number + temp;

break;

}

case '-':

{

temp = head1->link->number;

p1 = head1;

head1 = head1->link;

delete p1;

head1->link->number = head1->link->number - temp;

break;

}

case '\*':

{

temp = head1->link->number;

p1 = head1;

head1 = head1->link;

delete p1;

head1->link->number = head1->link->number \* temp;

break;

}

case '/':

{

temp = head1->link->number;

p1 = head1;

head1 = head1->link;

delete p1;

head1->link->number = (head1->link->number/temp);

break;

}

case '%':

{

temp = head1->link->number;

p1 = head1;

head1 = head1->link;

delete p1;

head1->link->number = int(head1->link->number) %int(temp);

break;

}

case '^':

{

temp = head1->link->number;

p1 = head1;

head1 = head1->link;

delete p1;

head1->link->number = pow(head1->link->number ,temp);

break;

}

case'#':

{

return 0; break;

}

default: break;

}

p2 = head2;

head2 = head2->link;

delete p2;

//判断退了一个符号后可否将新的符号压入，不行则继续退栈

if (checkchar(a) > checkchar(head2->link->sign))

{

head2->sign = a;

p2 = new charstack;

p2->link = head2;

head2 = p2;

return 0;

}

else

{

if (checkchar(a) == checkchar(head2->link->sign) && checkchar(a) == -1)

{

return 0;

}

else

{

count(head1, head2, a);//递归

}

}

}

主函数中的计算（非括号）片段：

if (checkchar(putin[n]) > checkchar(head2->link->sign))//不需要退栈的符号

{ //

if ((putin[n] == '+' || putin[n] == '-') && (checkchar(putin[n - 1]) != 0 && checkchar(putin[n - 1]) != 1) && checkchar(putin[n - 1]) != -4) //单目运算判断

{

if (putin[n] == '+') { changenumber = 1; }

if (putin[n] == '-') { changenumber = -1; }

}

Else //直接压入栈

{

charstack1++;

head2->sign = putin[n];

current2 = new charstack;

current2->link = head2;

head2 = current2;

}

}

Else //需要退栈的符号

{

if ((putin[n] == '+' || putin[n] == '-') && (checkchar(putin[n - 1]) != 0 && checkchar(putin[n - 1]) != 1)) //单目运算判断

{

if (putin[n] == '+') { changenumber = 1; }

if (putin[n] == '-') { changenumber = -1; }

}

else

{ //退栈计算 退一个符号和两个数字

count(head1, head2, putin[n]);

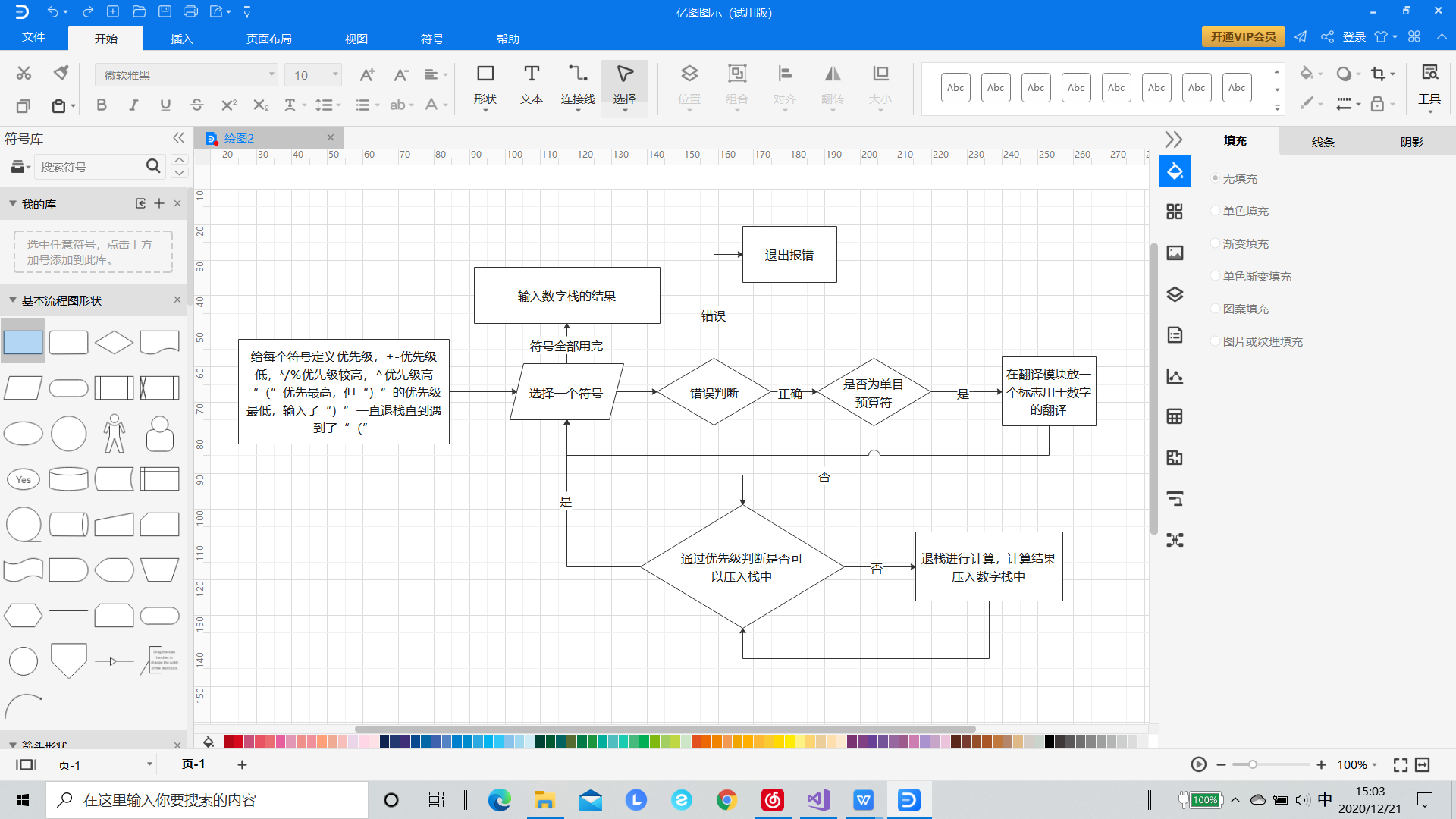
}

### }}

## 

## 3.3 计算功能（括号）的实现

### 3.3.1 计算功能（括号）流程图



### 3.3.2 计算功能括号核心代码

计算括号函数count2

int count2(numberstack \*&head1, charstack \*&head2, char a)//对数字栈和字符栈操作(括号)

{

double temp;

numberstack \*p1;

charstack \*p2;

switch (head2->link->sign)

{

case '+':

{

temp = head1->link->number;

p1 = head1;

head1 = head1->link;

delete p1;

head1->link->number = head1->link->number + temp;

break;

}

case '-':

{

temp = head1->link->number;

p1 = head1;

head1 = head1->link;

delete p1;

head1->link->number = head1->link->number - temp;

break;

}

case '\*':

{ //cout << "\*" << endl;

temp = head1->link->number;

p1 = head1;

head1 = head1->link;

delete p1;

head1->link->number = head1->link->number \* temp;

break;

}

case '/':

{ // cout << "/" << endl;

temp = head1->link->number;

p1 = head1;

head1 = head1->link;

delete p1;

head1->link->number = (head1->link->number / temp);

break;

}

case '%':

{

temp = head1->link->number;

p1 = head1;

head1 = head1->link;

delete p1;

head1->link->number = int(head1->link->number) % int(temp);

break;

}

case '^':

{

temp = head1->link->number;

p1 = head1;

head1 = head1->link;

delete p1;

head1->link->number = pow(head1->link->number, temp);

break;

}

default: break;

}

p2 = head2;

head2 = head2->link;

delete p2;

if (checkchar(a) > checkchar(head2->link->sign))

{

head2->sign = a;

p2 = new charstack;

p2->link = head2;

head2 = p2;

return 0;

}

else

{

if (checkchar(a)<checkchar(head2->link->sign)&&checkchar(head2->link->sign)==-3)

{

p2 = head2;

head2 = head2->link;

delete p2;

return -1;

}

else

{

if (checkchar(a) == -4 && checkchar(head2->link->sign) == -1)

{

return -2;

}

else

{

count2(head1, head2, a);

}

}

}

}

主函数中的计算功能（括号）片段

if (checkchar(putin[n]) == -3 || bracket == 1) //进行括号的储存

{

if (checkchar(putin[n]) == -3)

{

bracket++; //将（存储进入栈中

charstack1++;

head2->sign = putin[n];

current2 = new charstack;

current2->link = head2;

head2 = current2;

}

else

{

if (checkchar(putin[n]) > checkchar(head2->link->sign))//不需要退栈的符号

{

if ((putin[n] == '+' || putin[n] == '-') && (checkchar(putin[n - 1]) != 0 && checkchar(putin[n - 1]) != 1) && checkchar(putin[n - 1]) != -4)

{

if (putin[n] == '+') { changenumber = 1; }

if (putin[n] == '-') { changenumber = -1; }

}

else

{

charstack1++;

head2->sign = putin[n];

current2 = new charstack;

current2->link = head2;

head2 = current2;

}

}

else

{

if ((putin[n] == '+' || putin[n] == '-') && (checkchar(putin[n - 1]) != 0 && checkchar(putin[n - 1]) != 1) && checkchar(putin[n - 1]) != -4)

{

if (putin[n] == '+') { changenumber = 1; }

if (putin[n] == '-') { changenumber = -1; }

}

else

{

int k = count2(head1, head2, putin[n]);

if (k >= -1) { bracket = bracket + k; }

else { answerkind = -1; break; }//括号不匹配错误弹出

}

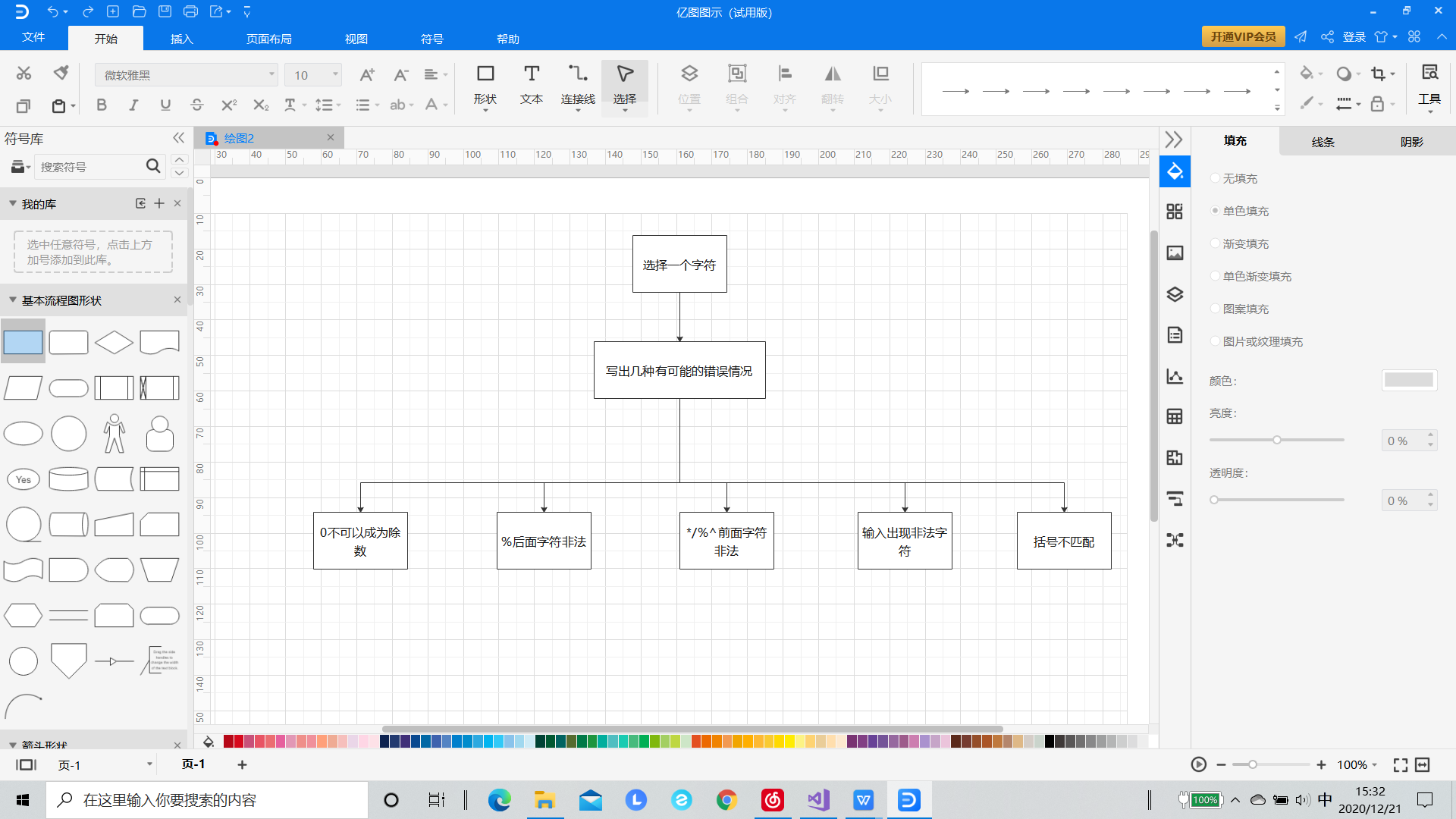
}

}

### }}

## 3.4 错误判断功能的实现

### 3.4.1 错误判断功能流程图



### 3.4.2 判断功能核心代码

Judge函数

int judge(char putin[100], int n) //判断是否表达式中符号数学错误

{

if (n >= 1)

{

if (checkchar(putin[n]) == 2 || checkchar(putin[n]) == 3)

{

if (checkchar(putin[n-1]) != -4 && checkchar(putin[n-1]) != 0)

{

return 2;

}

}

if (putin[n] == '%'&&putin[n + 1] == '-') { return 3; }

if (putin[n] == '/'&&putin[n + 1] == '0') { return 4; }

}

return 0;

}

错误提示函数

switch (answerkind)

{

case 4: {cout << "0不可以成为除数" << endl; break; }

case 3: {cout << "%后面字符非法" << endl; break; }

case 2: {cout << "\*/%^前面字符非法" << endl; break; }

case 1: {cout << "输入出现非法字符" << endl; break; }

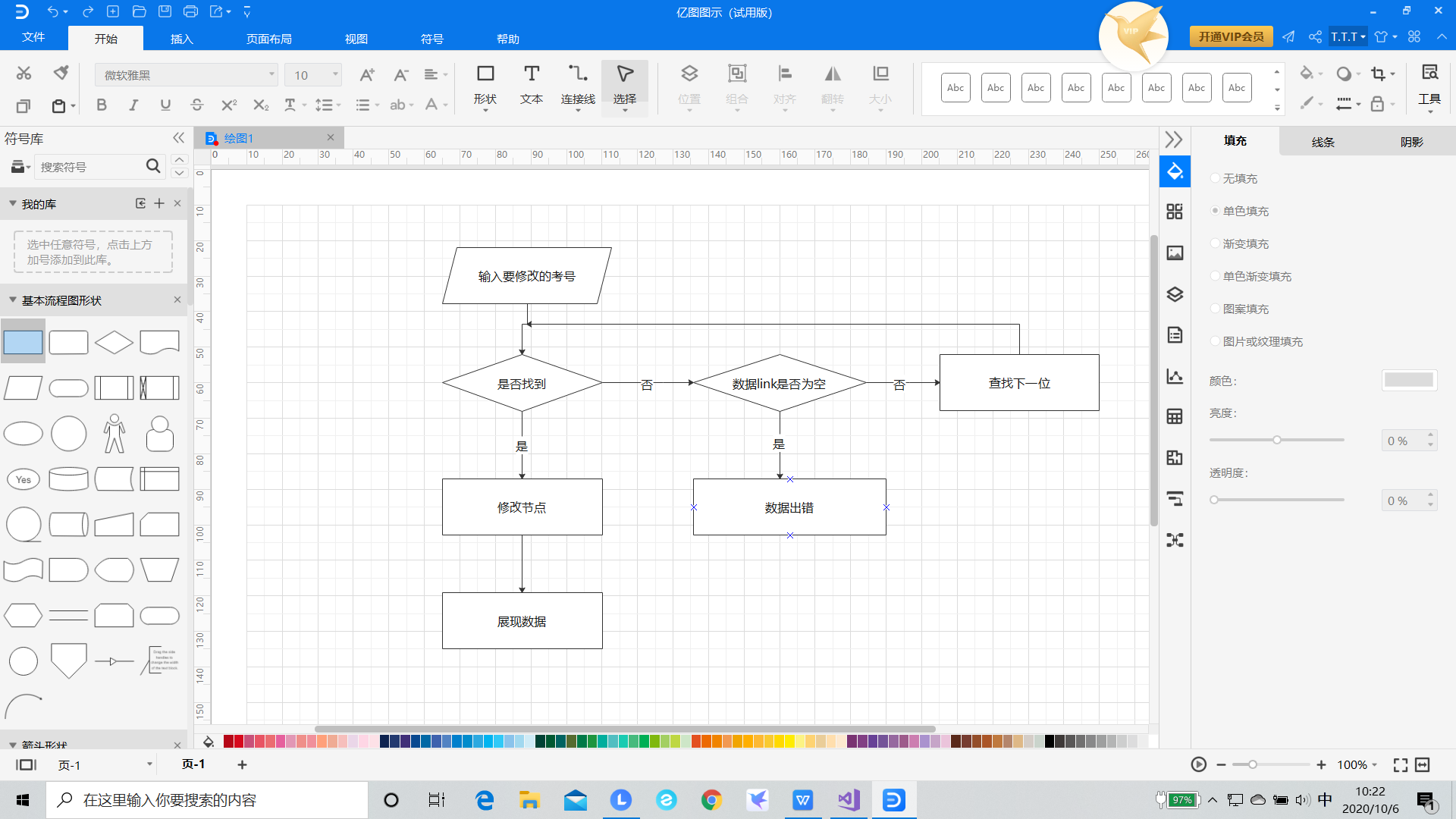
case -1: {cout << "括号不匹配" << endl; break; }

default:break;

### }

## 3.5 修改功能的实现

### 3.5.1 修改功能流程图



### 3.5.2 统计功能核心代码

bool list::change(int n)

{

current = head->link;

for (int m = 1; m <= nowsetnumber; m++)

{

if (current->number == n)

{

int a; cin >> a; current->number = a;

char b[20]; cin >> b; strcpy(current->name,b);

char c[20]; cin >> c; strcpy(current->gender, c);

int d; cin >> d; current->age = d;

char e[20]; cin >> e; strcpy(current->job, e);

return true;

}

else { current = current->link; }

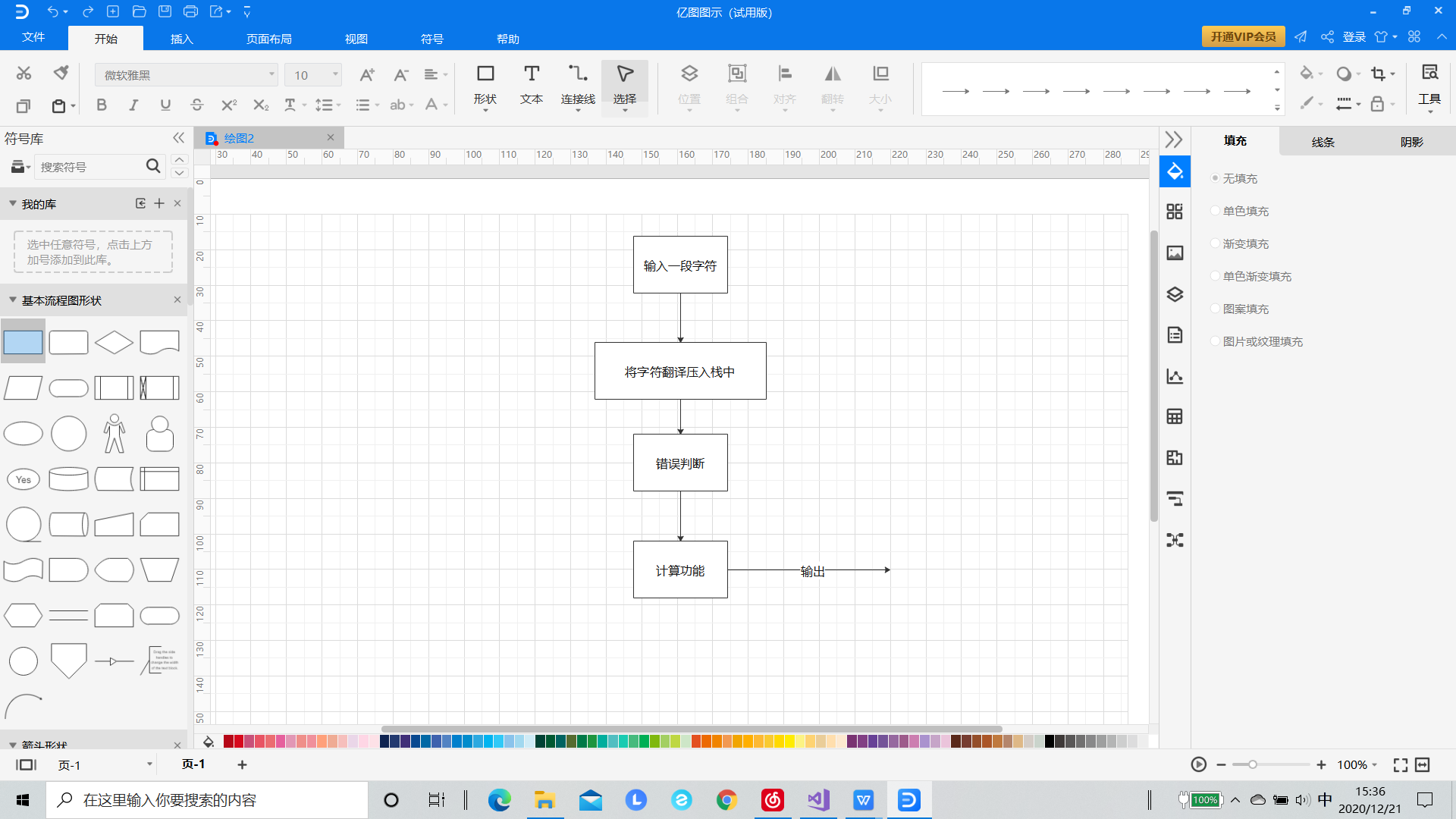
}

return false;

### }

# 3.7总体系统实现

### 3.7.1 总体系统流程图



### 3.7.2总体系统核心代码

int main()

{

char putin[100];//一串输入的表达式

int putinnumber; //表达式的长度

char putinchar[5];//存储字符型的数字

int checkputinchar1 = 0; // 判断是否进行数字栈的存储

int checkputinchar2 = 0; // 记录写入putinchar数组的元素个数

double changechar(char a[5],int b,int \*c);//将字符变成double

int checkchar(char a);//符号辨别函数

int count(numberstack \*&head1, charstack \*&head2,char a);//对数字栈和字符栈操作

int count2(numberstack \*&head1, charstack \*&head2,char a);//对数字栈和字符栈操作(括号)

int judge(char putin[100], int n);//判断符号之间是否正确

int numberstack1 = 0; //中期检验数字串

int charstack1 = 0; //中期检验字符串

int changenumber = 0; //单目+-的判断

int bracket = 0;//判断是否有括号存在,同时可以判断是几级括号

int answerkind = 0;//判断答案的种类

for (;;)

{

checkputinchar1 = 0; // 判断是否进行数字栈的存储

checkputinchar2 = 0; // 记录写入putinchar数组的元素个数

numberstack1 = 0; //中期检验数字串

charstack1 = 0; //中期检验字符串

changenumber = 0; //单目+-的判断

bracket = 0;//判断是否有括号存在,同时可以判断是几级括号

putinnumber = 0;

cout << "请输入表达式" << endl;

putin[0] = '#';

for (int n = 1; n <= 99; n++)

{

putin[n]=getchar() ; putinnumber = n;

if (int(putin[n]) == 10) { break; }

}

//开始将数字和字符分开放入两个栈中

numberstack\* head1;

numberstack\* current1;

current1 = new numberstack;

head1 = current1;

charstack\* head2;

charstack\* current2;

current2 = new charstack;

head2 = current2;

for (int n = 0; n <= putinnumber; n++)

{

if (int(putin[n]) >= 48 && int(putin[n]) <= 57)

{

checkputinchar1 = 1;

putinchar[checkputinchar2] = putin[n];

checkputinchar2++;

} //数字处理

if (int(putin[n]) < 48 || int(putin[n]) > 57)

{

if (checkchar(putin[n]) == 0) { answerkind = 1; break; }//错误弹出

if (judge(putin, n) != 0) { answerkind = judge(putin, n); break; };

if (checkputinchar1 == 1) //进行数字栈的存储

{

numberstack1++;

head1->number = changechar(putinchar, checkputinchar2 - 1, &changenumber);//放入栈中

current1 = new numberstack;

current1->link = head1;

head1 = current1;

checkputinchar1 = 0;

checkputinchar2 = 0;

}

if (checkputinchar1 == 0) //进行符号栈的存储

{

if (charstack1 == 0) //先进行‘#’的存储

{

charstack1++;

head2->sign = putin[n];

current2 = new charstack;

current2->link = head2;

head2 = current2;

}

else //正常符号进行存储

{

if (checkchar(putin[n]) == -3 || bracket == 1) //进行括号的储存

{

if (checkchar(putin[n]) == -3)

{

bracket++; //将（存储进入栈中

charstack1++;

head2->sign = putin[n];

current2 = new charstack;

current2->link = head2;

head2 = current2;

}

else

{

if (checkchar(putin[n]) > checkchar(head2->link->sign))//不需要退栈的符号

{

if ((putin[n] == '+' || putin[n] == '-') && (checkchar(putin[n - 1]) != 0 && checkchar(putin[n - 1]) != 1) && checkchar(putin[n - 1]) != -4)

{

if (putin[n] == '+') { changenumber = 1; }

if (putin[n] == '-') { changenumber = -1; }

}

else

{

charstack1++;

head2->sign = putin[n];

current2 = new charstack;

current2->link = head2;

head2 = current2;

}

}

else

{

if ((putin[n] == '+' || putin[n] == '-') && (checkchar(putin[n - 1]) != 0 && checkchar(putin[n - 1]) != 1) && checkchar(putin[n - 1]) != -4)

{

if (putin[n] == '+') { changenumber = 1; }

if (putin[n] == '-') { changenumber = -1; }

}

else

{

int k = count2(head1, head2, putin[n]);

if (k >= -1) { bracket = bracket + k; }

else { answerkind = -1; break; }//括号不匹配错误弹出

}

}

}

}

else //非括号的存储

{

if (checkchar(putin[n]) > checkchar(head2->link->sign))//不需要退栈的符号

{ //

if ((putin[n] == '+' || putin[n] == '-') && (checkchar(putin[n - 1]) != 0 && checkchar(putin[n - 1]) != 1) && checkchar(putin[n - 1]) != -4)

{

if (putin[n] == '+') { changenumber = 1; }

if (putin[n] == '-') { changenumber = -1; }

}

else

{

charstack1++;

head2->sign = putin[n];

current2 = new charstack;

current2->link = head2;

head2 = current2;

}

}

else

{

if ((putin[n] == '+' || putin[n] == '-') && (checkchar(putin[n - 1]) != 0 && checkchar(putin[n - 1]) != 1))

{

if (putin[n] == '+') { changenumber = 1; }

if (putin[n] == '-') { changenumber = -1; }

}

else

{

count(head1, head2, putin[n]);

}}}}}}}

if (answerkind == 0)

{

cout << head1->link->number << endl;

}

else

{

switch (answerkind)

{

case 4: {cout << "0不可以成为除数" << endl; break; }

case 3: {cout << "%后面字符非法" << endl; break; }

case 2: {cout << "\*/%^前面字符非法" << endl; break; }

case 1: {cout << "输入出现非法字符" << endl; break; }

case -1: {cout << "括号不匹配" << endl; break; }

default:break;

}

}

cout << "是否要继续" << " " << "(Y,N)" << endl;

char going;

for (;;)

{

cin >> going;

if (going == 'Y' || going == 'N') { break; }

cout << "请输入正确的符号" << endl;

}

if (going == 'N') { break; }

getchar();//清除缓冲区的回车键

}

return 0;

}

# 4 测试

## 4.1 功能测试

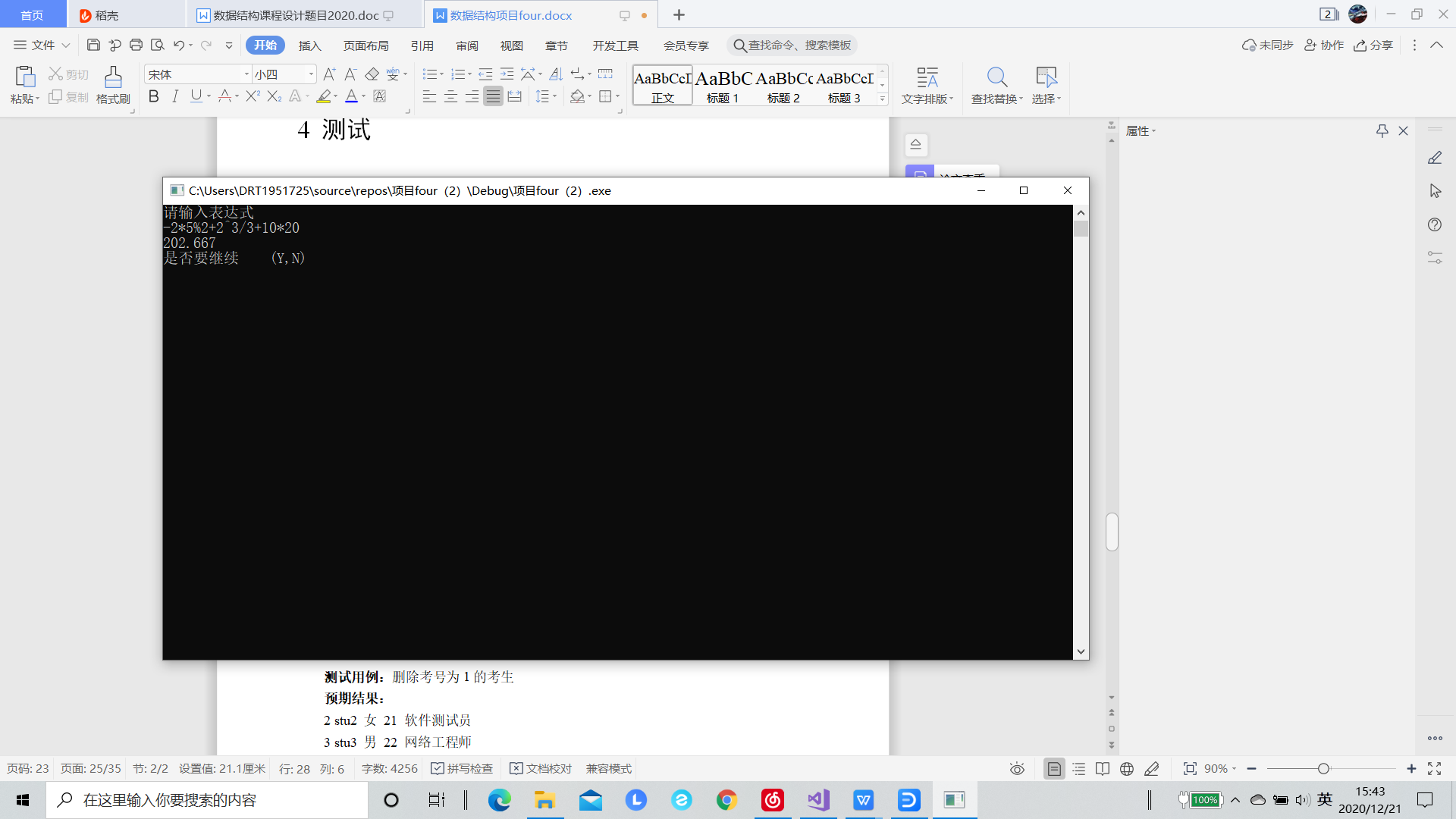
### 4.1.1 计算功能（非括号）测试

**测试用例**：-2\*5%2+10\*20+2^3/3=

**预期结果**：

**202.6667**

**实验结果**



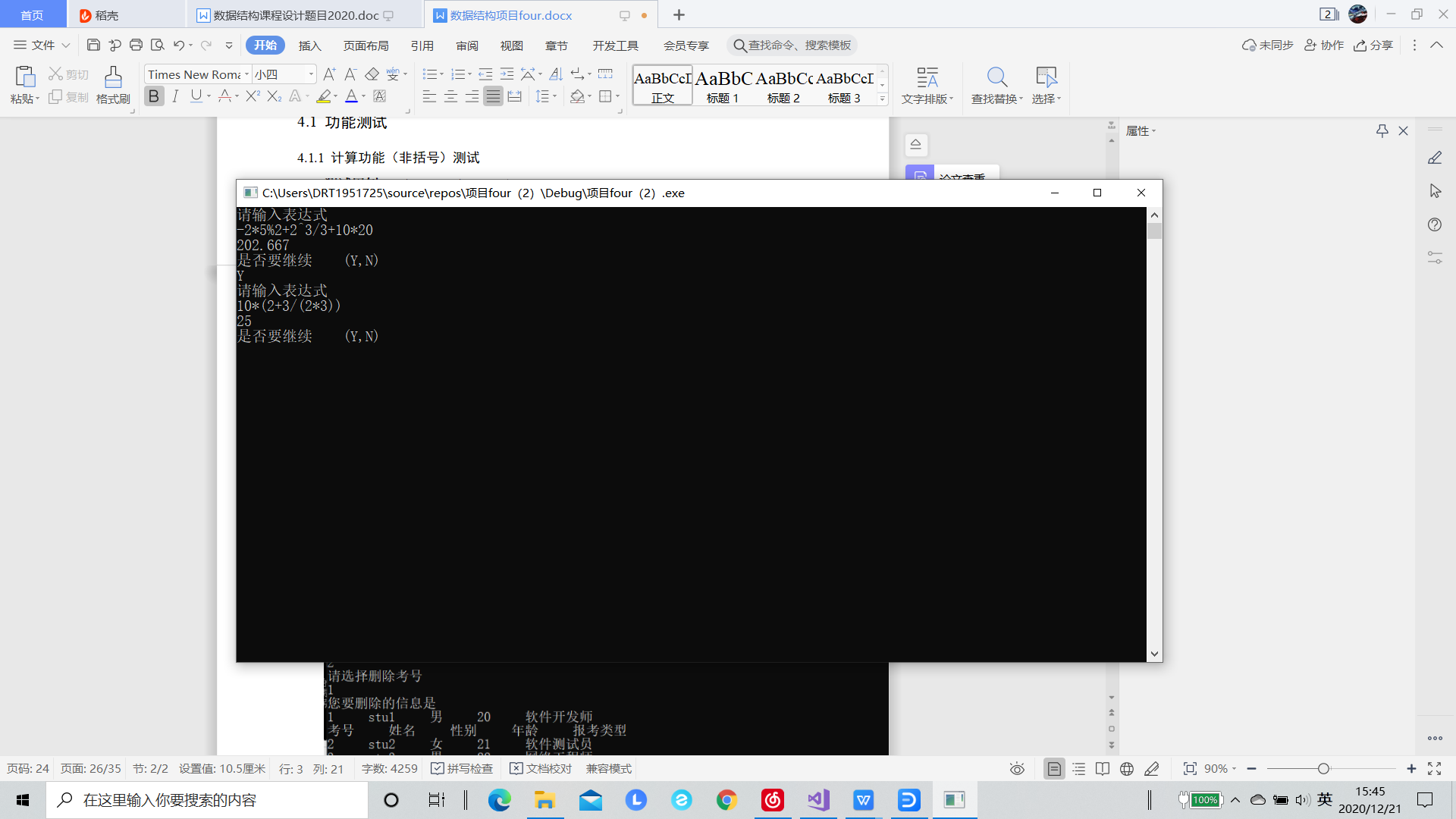
### 4.1.2 计算功能（括号）测试

**测试用例：10\*（2+3/（2\*3））=**

**预期结果：**

**25**

**实验结果：**



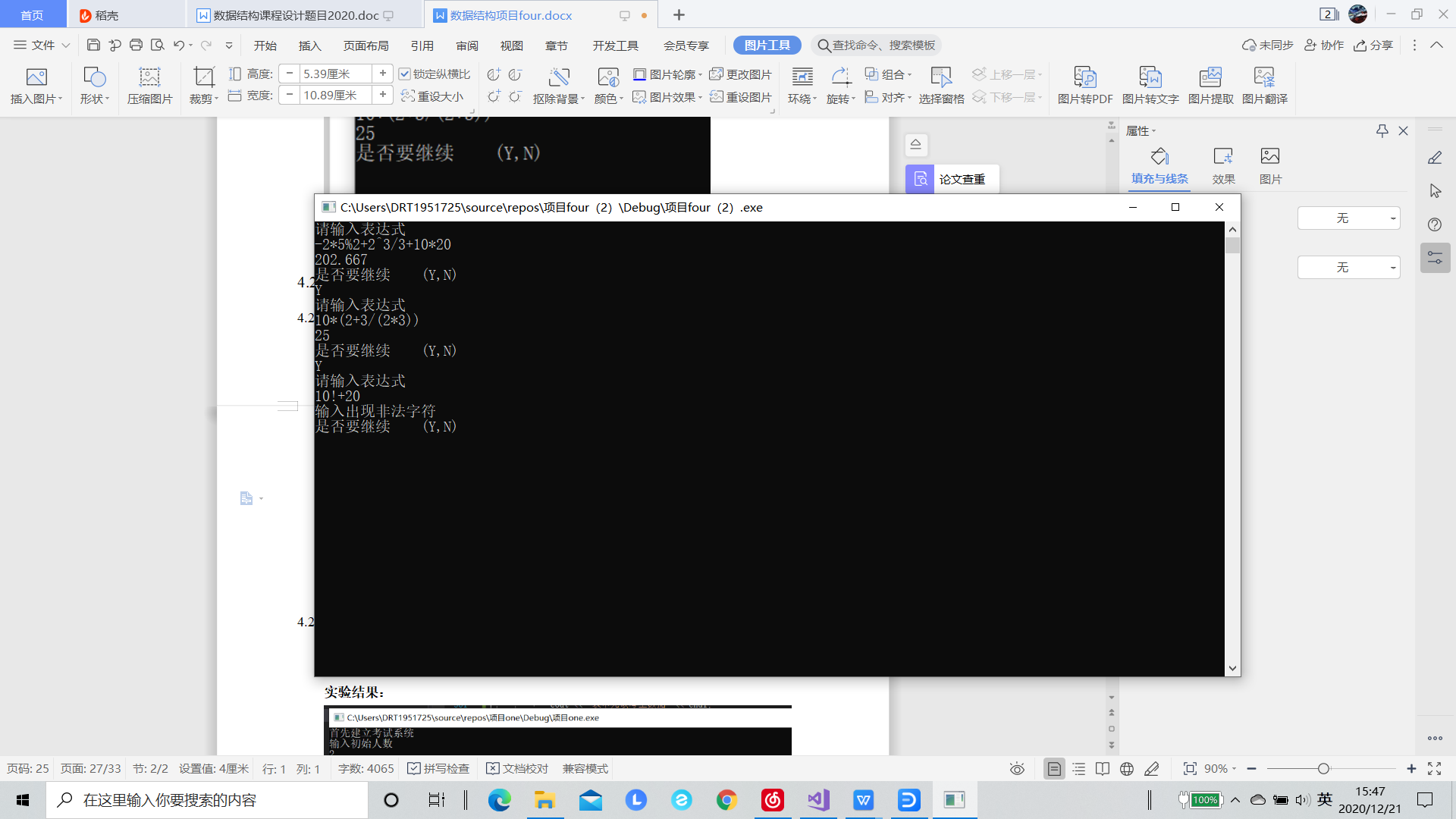
## 4.2 边界测试

### 4.2.1 输入不正确的字符

**测试用例：10！+20=**

**预期结果：**给出错误提示，程序运行正常不崩溃。

**实验结果：**

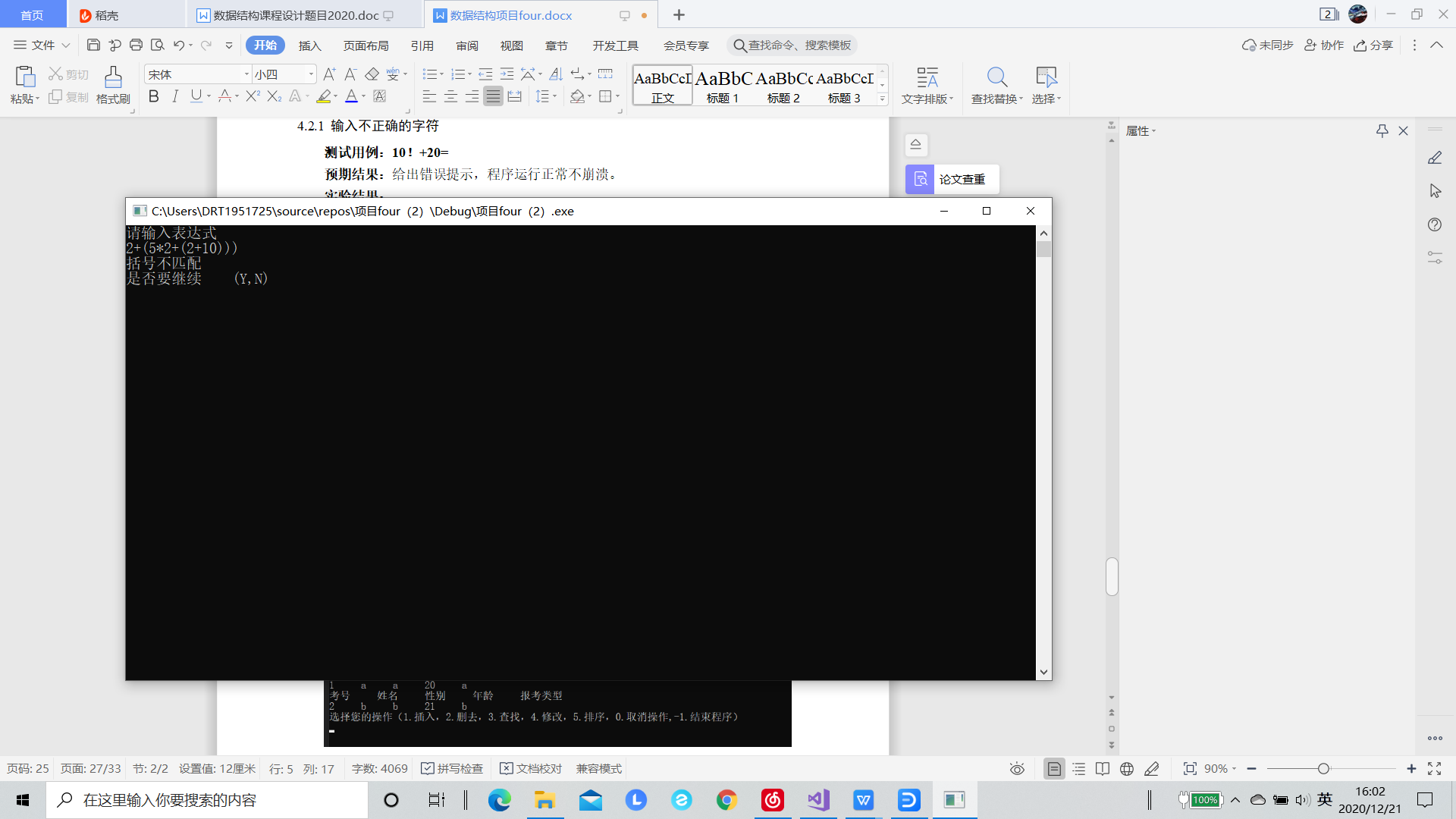


### 4.2.2 括号不匹配

**测试用例：2+（5\*2+（2+10）））=**

**预期结果：**程序正常运行，不崩溃。并正确报错

**实验结果：**

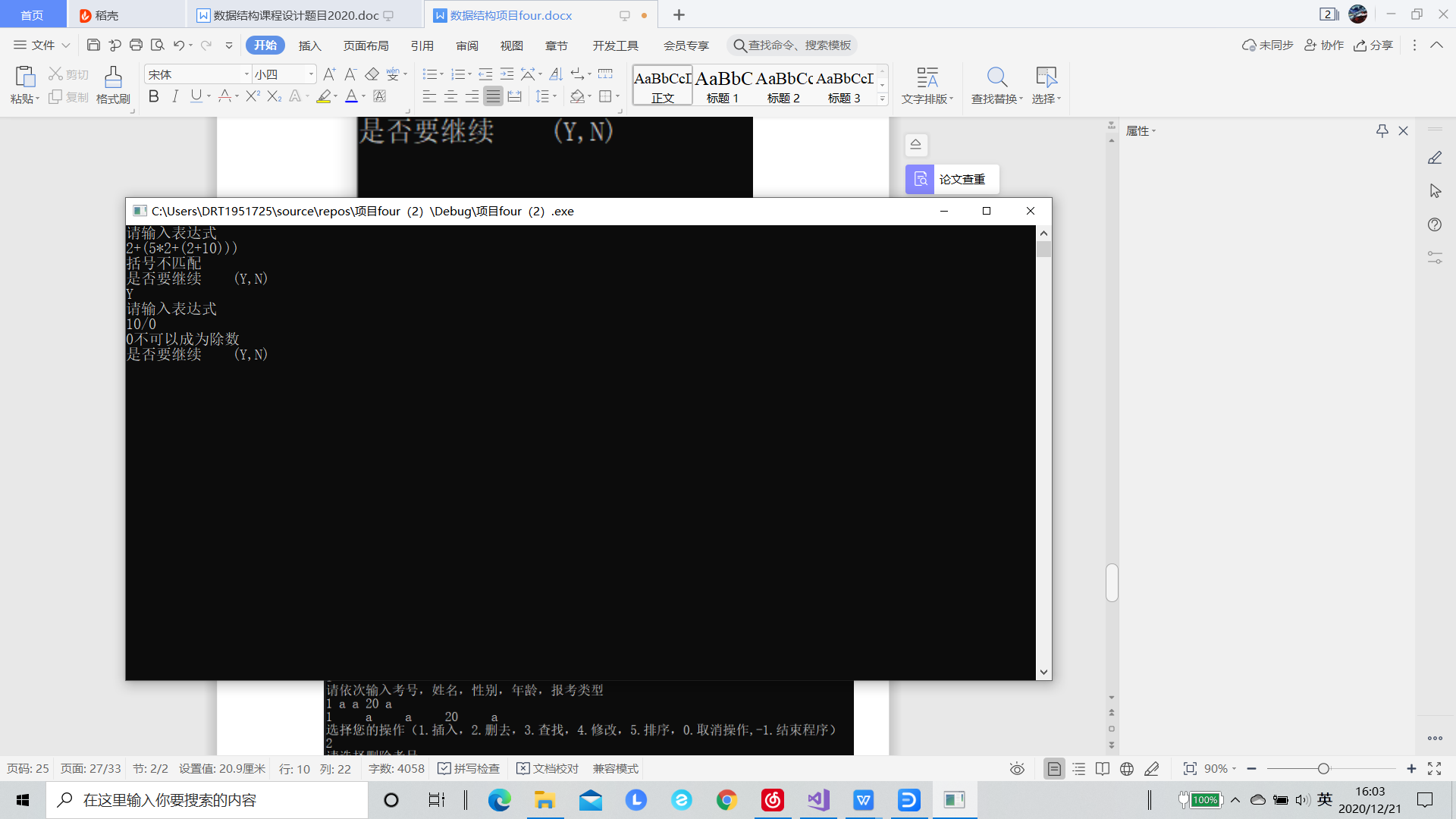


### 4.2.3 除数为0

**测试用例：10/0**

**预期结果：**程序正常运行，不崩溃。并正确报错

**实验结果：**

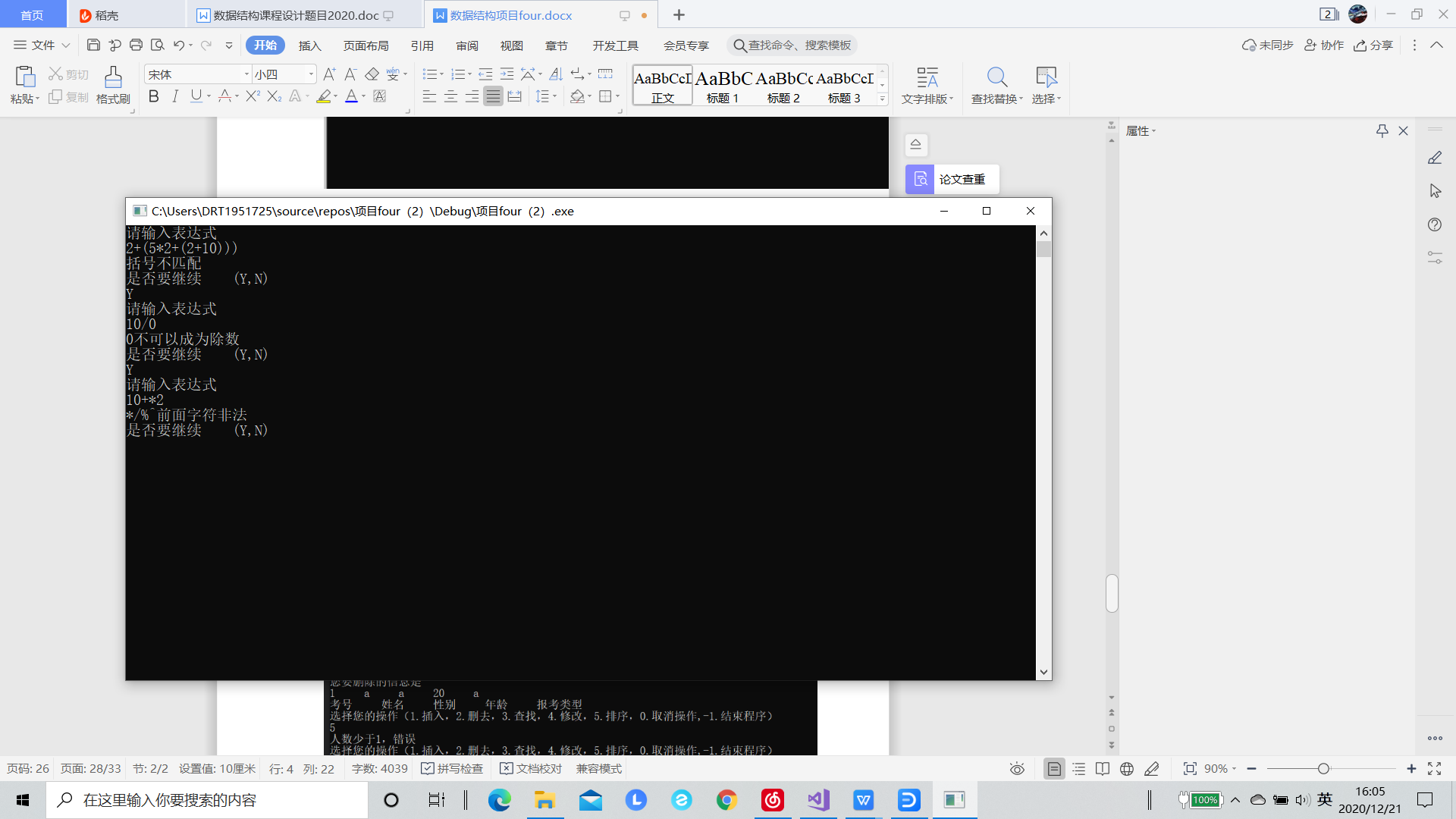


### 4.2.4 \*/%^前面字符非法

**测试用例：10+\*2**

**预期结果：**程序正常运行，不崩溃。并正确报错

**实验结果：**

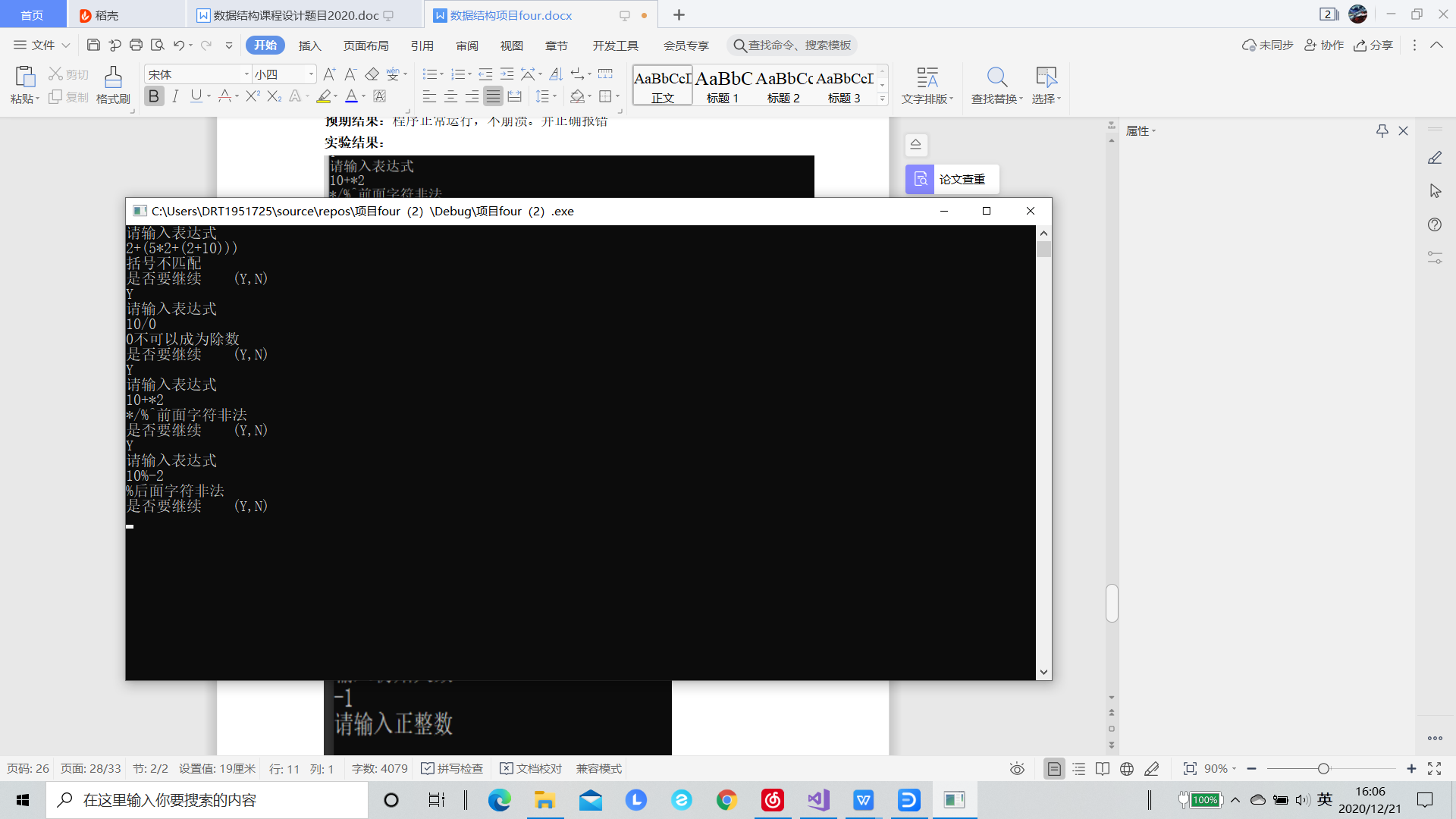


### 4.2.5 %后面字符非法

**测试用例：10%-2**

**预期结果：**程序正常运行，不崩溃。并正确报错

**实验结果：**



### 4.2.6 输入的数字太少

**测试用例：）**

**预期结果：**程序正常运行，不崩溃。并正确报错

**实验结果：**

