智能代数运算系统：用户输入任意表达式，系统能正确计算出结果，表达式支持常用数学函数以及变量。 设计要求及提示如下：

（1）用户通过输入任意表达式，计算出结果，如：a*sin(b*x+c) + x^2 + sqrt(a);

（2）表达式中若出现未申明变量，则给该变量初始化一个随机值，如：a = random(0,10);

（3）支持常用数学函数，包括（不限于）如下函数：sin, cos, tan, floor, random, abs, sqrt, ^....

（4）可以修改变量的值，表达式根据变量的变化自动更新计算 结果；

（5）允许用户自定义函数，如定义 f(x) = (x+2)^2 后，用户可以在表达式中使用该函数：f(sin(a))+f(a)+sin(f(a))

（6）下图为程序效果参考：

https://www.netpad.net.cn/resource_web/course/#/575887

1. split 函数

分割表达式成一个个词。像是词法分析。要用到状态转换图。比如首先未输入前是在零状态，输入一个字符“3“就去到一状态，那么这个状态就应该是数字，然后接着想，接着输入其他又会去到什么状态。比如如果输入了字母，就会出错；输入了数字或者小数点，那么可能是数字。先列出每一种状态，然后列出每种状态中，再输

入后面一个字符会去到哪个状态。

比如输入的是 a*b+c 那么就要分词 a, *, b, +, c。然后要看这些词是常量还是变量还是自定义函数还是嵌入的函数。如果是没有申明过的变量，就在变量表中加入，并且给一个随机的值。

2. buildtree 函数

是写语法数的函数，是递归函数。我是怎么想的呢。这个相比于在数据结构中学的表达式求值，还加多了变量、自定义函数、嵌入函数。我应该怎么把它变成像一般的表达式求值呢。然后这个函数就是解决掉变量、自定义函数和嵌入函数的问题。首先加入头节点到树中。函数中有 3 个参数，双亲结点（用来记录这个结点下面会有什么儿子)、哪个词的下标（词列表是保存在一个表中。它记录词的内容和类别）和是否是函数的布尔（不重要，就是有没有逗号和括号的区别)。从第一个词开始，先判断输入的词是什么类型，是变量还是常量数字还是自定义函数还是嵌入函数。如果是常量或者变量或者运算符，那么把它保存到一个新的节点，然后把指针添加到双亲结点的儿子结点列表中。如果是自定义函数或者嵌入的函数或者括号里的表达式，那么首先把它的信息保存到一个新的结点，再连接双亲结点，接着开始递归，调用会这个 buildtree 函数，要传入参数，这个新结点、当前词的下一个词的下标和属于函数，那么接着会把这个自定义函数或者嵌入函数的参数连接到这个新的结点。然后就写好了。退出的条件是当不是函数时是当下标已经超过这个词

列表的长度，当是函数时是，括号匹配成功。

3. calc 函数

遍历上面的函数并且求得表达式的结果。把树变成两个栈，运算符栈 optrstack 和操作数栈 opndstack。首先再 opststack 推入一个记号“#”，用来标记这个是式子的开头。然后从树一个个加入操作数或者运算符，直到运算符栈顶部是“#”，加入的最后一个词也要是“#”，最后它们会抵消，表示已经完成。

1 如果是操作符栈，比较运算符栈栈顶的符号和新加入符号的优先级（优先级比如星号会大于加号）那么分下面 3 种情况。1 运算符栈栈顶的大于新加入的符号的优先级，直接推入到运算符栈；2 运算符栈栈顶的等于新加入的符号的优先级，不推入到运算符栈，反而把运算符栈栈顶的符号弹出；3 运算符栈栈顶的大于新加入的符号的优先级，弹出操作数栈栈顶的两个操作数，和弹出运算符栈栈顶的一个运算符，然后运算，把结果推入到操作数栈，然后这个优先级高的运算符，下一步也要继续加入。

2 如果加入的是操作数，那么这个操作数可能是括号里的表达式或者是变量、常数数字、自定义函数、嵌入函数。如果是括号里的表达式，那么调用回这个 calc 函数，计算这个括号里的表达式的结果；否则，这个结点会有 getval 成员，可以得到结果，(就像上面说的如果是变量，就去变量列表去找；如果是常数数字，就用 toint 函数)。然后把结果推入到操作数栈。

如果加入最后，操作数栈的长度是 1，操作数栈的这一个操作数就是答案；如果操作数栈的长度不是 1 就错误。

一、

4. 计算表达式

这里在框里填入数字或者函数或者就可以得到结果。如果输入错误会提示输入错误，然后就看不到结果了。

方法是当输入结束后就调用计算函数，这个函数具体怎么做后面再说。从这个计算的函数得到 m 和 3 个函数的值，如果它已经有定义。然后就把结果写上去。

5. 自定义函数

同样在框里填入数字或者函数或者就可以得到结果。如果输入错误会提示输入错误，然后就看不到结果了。这个是用来记录下一些东西，使在使用上面那个计算功能更加方便。其实跟上面是一样的。

6. 用滑块调整变量

可以用键盘填入数字或者用滑块调整。当滑动滑块或者数字有变动，上面的计算结果也会很智能地调整，计算出正确的结果。

7. Node 类

这个类用来写语法树。这个类有 3 个成员变量，分别是类型、表达

式内容、指向后继结点的指针。其中类型会有 5 类，分别是自定义函数、嵌入的函数、括号里的小表达式、常量数字、变量和运算符；表达式内容内容是用来放表达式内容的，比如如果一段表达式是“a*b”，那么它可能就放“a”。有 3 个成员函数，分别是构造函数、添加后继结点指针的函数和获取数据的函数。其中获取数据的函数是这么写的。首先看这个结点属于哪个类。如果是自定义函数，就返回这个函数的结果；如果是嵌入函数（一般指一些数学函数），那么就一个个枚举，比如用户输入“sin（a）”，那么在 C++就要写

if(this->type==EMBEDDEDFUNCTION&&this->content==”sin(a)”)return std::sin(a);如果这个类是变量，就返回一个记录变量的表中对应的结果，如果是常量数字，就把表达式内容转变成数字，Qt 中有 Qvariant(“string”).toFloat()函数，直接用。

8. getval

如果这个结点里的一个成员变量 type 是 variable，表示这个词是一个变量，那么从那个保存变量值的表 variables 中找到，就可以返回，应该是绝对可以找到，不会出错。如果是 constval，那么 toFloat()就行。如果是嵌入的数学函数，那么就枚举，同时要看参数的个数，如果不匹配就提示错误。如果是自定义函数 f 或者 g 或者 h。这个是最难的。

9. update 函数

这个函数用来重新计算，在窗口上更新计算的结果，它在窗口显示之前、变量有更新的时候、表达式输入完成的时候都要用。
有 6 个东西（表达式可以有 3 个，自定义函数是 3 个）要计算，3 个函数的结果和 3 个表达式的结果。每一个计算都有几个情况，1 如果表达式是空就跳过；2 如果表达式在 split 函数、buildtree 函数或者 calc 函数中出错，就显示一个信息盒子，提示哪个表达式在哪个函数中哪个过程出错。如果计算成功了，就把它显示到窗口上对应位置。

10. connect 函数

用来连接控件的信号和槽函数。信号是比如在行编辑输入完成、拖动滑块。槽函数是指接受这个信号并且做一些事的函数。
这里的我的 connect 有 3 种，因为控件也有 3 种。分别是 1 连接用来改变变量值的滑块、2 连接用来改变变量值的行编辑、3 连接输入函数或者表达式的行编辑。比如拖动滑块就有一个信号，然后一个槽函数就接收它，这个槽函数要做的东西有更新变量的值、计算自定义函数的值和表达式的值、把结果显示在窗口上。其他都一样，因为只要一个东西变了，其他东西都很可能要变。那么如果算的是函数，就要更新放函数结果的变量 3 个，这么做是因为在表达式中可能会重复用到，所以就不应该算很多次，不过这 3 个变量要及时更新，要在算表达式前更新。

11. 解决自定义函数的方法

替换。打个比方，表达式 m1 是 f(7)+9，函数 f 是函数是 f(x)=x*3，那么它需要用到自定义函数 f(x)看等号的位置把函数切成左右两份。那么左边是 f(x)，右边是 x*3。左边就是那么右边一份就是一个表达式。不过这个表达式是不可以直接用的，因为还没有换这个参数。接着要对左边那份语法分析，会得出它有几个参数。得到参数后，看下跟自己的参数数目是不是相同，相同就继续。然后通过左边的这个东西添加映射，那么映射就是 x 映射到 7。接着用映射把右边的表达式更换一些，那么右边就由 x*3 得到新的表达式 7*3，再计算这个新的表达式得到 21，所以调用 f(7)得到的结果就是 21。

二、 程序中使用的数据及主要符号说明

1. MainWindow 类

2. Splitresult 分割结果

它是 Qvector<Qvector<Qstring>>类型。首先它第二维的大小是输入的表达式的词的数量，第一维的大小是 2，第一个是词的内容，第二个放词的类别（常量、变量、自定义函数……）。在分割功能中把它填好。

3. Node 类

它用来写语法树。有 3 个成员变量分别是 1 词的类别（常量、变量、自定义函数……）、2 词的内容、3 后继结点的列表。有 3 个成员函数，分别是构造函数、添加后继结点的函数和获取计算结果的函数。

4. Headnode 结点

它是语法树根结点，是 Node*类的结点。

5. variablevalues 列表

它是映射变量名到它的值的映射。它在 split 函数中填好，包括了初始的 3 个 a,b,c；和表达式中未申明的。

6. funcset 集

用来放嵌入的函数的函数名，比如“sin”，“cos”，在 split 函数中要用到，用来区分变量和函数。

7. optrset 集合

用来放可以用的操作符。比如+-*/，在 split 函数中用到，如果输入的不是字母和数字，那么检查它是不是可用的符号，不是的话要提示错误。

8. surfacegraph 类

用来画函数图像，首先在第二页的的 widget 添加一个 graph，类型是 Q3Dsurface，然后调用 surfacegraph 类的构造函数，这个构造函数是用来给这个 graph 添加 QSurface3Dseries ，这个 series 是用来画图像，比如说在（x,y）坐标上的值是什么。这个 sereis 在第 2 页的表达式的行编辑输入结束，或者是下拉列表的索引改变时要更新。

三、 部分关键程序的源代码

1.

```
1   //词法分析，分割表达式成词
2   //结果输出到 splitresult 列表，第一列是放词的内容，第二列是词的类型
3   QVector<QVector<QString>> MainWindow::split(int which,QString sourcecode="",bool
    updatevariable = true)
4   {
5       if(which>=0)sourcecode = codestortoglobal[which];
6
7       QVector<QVector<QString>>splitresult = {};
8
9       //如果是空就返回，会在上一个 update 函数，使窗口中对应显示表达式结果的标签不显示东西
10              if(sourcecode=="")return {{"-3"}};
11
12              //状态转换图，初始的状态是零
13              //做法是先列出有几种状态，比如从零开始，如果下一个字符是字母，就去到状态一
14              //如果下一个字符是数字，就去到状态二
15              //那么状态一的结果是一串数字，是一个常数变量
16              //状态二的结果是一串字母可能穿插着数字，那么可能是函数名或者变量名
17              double state = 0;
18
19              //
20              QString currentstring = "";
21              for(int i = 0;i<sourcecode.size();i++)
22              {
23
24                  QChar character = sourcecode[i];
25                  if(character != '*'&&character != '/'&&character != '+'&&character != '('&&character !=
    '-'
26                                  &&character != ')'&&character != ','&&character != '^'&&character != '
    '&&character != '.'&&character != '=')
27                  {
```

```
28
    if(!((character>='a'&&character<='z')||(character>='A'&&character<='Z')||(character>='0'&&character
    <='9')))
29                          {
30                              qDebug()<<"invalid character\n"<<i;
31                              for(double j = qMin(0,i-5);j<i+5&&j<sourcecode.size();j++)
32                              {
33                                  qDebug()<<QString(sourcecode[j]);
34                              }
35                              return {{"-1"}};
36                          }
37                      }
38                      if(character==' ')
39                      {
40                          if(currentstring=="")continue;
41                          splitresult.append({currentstring});
42                          currentstring="";
43                          state = 0;
44                          continue;
45                      }
46                      if(character == '*'||character == '/'||character == '+'||character == '('||
47                              character == ')'||character == ','||character == '^'||character == '=')
48                      {
49                          if(currentstring!="")splitresult.append({currentstring});
50                          currentstring="";
51                          splitresult.append({character});
52                          state = 0;
53                          continue;
54                      }
55
56                      if(state==0&&character=='-')
57                      {
58                          if(splitresult.size()==0||splitresult.back()[0]=="(")
59                          {
60                              currentstring+=character;
61                              state = 1;
62                              continue;
63                          }
64                          else
65                          {
66                              splitresult.push_back({"-"});
67                              continue;
68                          }
69                      }
70                      if(state==0&&(character>='0'&&character<='9'))
71                      {
72                          currentstring+=character;
73                          state = 1;
74                          continue;
75
76                      }
77
    if(state==0&&((character>='a'&&character<='z')||(character>='A'&&character<='Z')||character=='_'))
78                      {
79                          currentstring+=character;
80                          state = 2;
81                          continue;
82                      }
```

```
83                      if(state ==2)
84                      {
85
    if((character>='0'&&character<='9')||((character>='a'&&character<='z')||(character>='A'&&character
    <='Z')||character=='_'))
86                          {
87                              currentstring+=character;
88                          }
89                          else
90                          {
91                              splitresult.append({currentstring});
92                              currentstring="";
93                              state = 0;
94                              i--;
95                          }
96                          continue;
97
98                      }
99                      if(state ==3)
100                     {
101                         if(character>='0'&&character<='9')
102                         {
103                             currentstring+=character;
104                         }
105
106                         else
107                         {
108                             splitresult.append({currentstring});
109                             currentstring="";
110                             state = 0;
111                             i--;
112                         }
113                         continue;
114                     }
115                     if(state ==1)
116                     {
117                         if(character>='0'&&character<='9')
118                         {
119                             currentstring+=character;
120                         }
121                         else if(character=='.')
122                         {
123                             currentstring+=character;
124                             state = 3;
125                         }
126                         else
127                         {
128                             splitresult.append({currentstring});
129                             currentstring="";
130                             state = 0;
131                             i--;
132                         }
133                         continue;
134                     }
135                 }
136
137                 splitresult.append({currentstring});
138
```

```
139
140             //分类这个词，看他属于哪个类（自定义函数、嵌入函数、变量）
141             for(QVector<QString> &s:splitresult)
142             {
143                 QString symbol = s[0];
144                 bool in = isnumber(symbol);
145
146                 if(funcset.find(symbol)!=funcset.end())
147                 {
148                     s.append(EMBEDDEDFUNCTION);
149                     continue;
150                 }
151                 if(symbol=="f"||symbol=="g"||symbol=="h")
152                 {
153                     s.append(CUSTOMIZEFUNCTION);
154                     continue;
155                 }
156                 if(symbol=="a"||symbol=="b"||symbol=="c")
157                 {
158                     s.append(VARIABLE);
159                     continue;
160                 }
161
162                 if(optrset.find(symbol)!=optrset.end())
163                 {
164                     s.append(OPTR);
165                     continue;
166                 }
167                 if(in)
168                 {
169                     s.append(CONSTVALUE);
170                     continue;
171                 }
172                 if(updatevariable==true&& variablevalues.find(symbol)==variablevalues.end())
173                 {
174                     double randval = (rand()%200000-100000)/5000;
175                     variablevalues.insert(symbol,randval);
176
177                     s.append(VARIABLE);
178                     continue;
179                 }
180                 s.append(VARIABLE);
181             }
182
183
184
185             return splitresult;
186         }
```

2.

```
1    //做语法树
2    //parent 是双亲结点的指针，index 指现在的词法分析结果列表的下标
3    void MainWindow::buildtree(QVector<QVector<QString>>&splitresult,Node*parent,int& index,bool
     isfunction)
4    {
```

```
5       //两分支
6       //一个是把函数的参数加入到函数结点的儿子结点列表
7       //另一个就像编译原理的做语法树
8       if(isfunction == false)
9       {
10
11                  for(int& i = index;i<splitresult.size();i++)
12                  {
13                      //
14                      QVector<QString>s = splitresult[i];
15                      if(s[1]!=OPTR&&parent->childs.size()>=1&&splitresult[i-1][1]!=OPTR)
16                      {
17                          parent->addchild(new Node(OPTR,"*"));
18                      }
19                      //如果词属于变量
20                      if(s[1]==VARIABLE)
21                      {
22                          //那么新建一个结点，把词的信息保存到结点，添加到双亲结点的儿子结点指针列表中
23                          Node*newnode = new Node(VARIABLE,s[0]);
24                          parent->addchild(newnode);
25                          parent->content+=s[0];
26                          continue;
27                      }
28                      //如果词是常量
29                      if(s[1]==CONSTVALUE)
30                      {
31                          //那么新建一个结点，把词的信息保存到结点，添加到双亲结点的儿子结点指针列表中
32                          QString hou = s[0];
33                          if(s[0].size()>0&&s[0][0]=='-'&&(parent->childs.size()>0&&splitresult[i-
    1][1]!=OPTR))
34                          {
35                              Node*newnode = new Node(OPTR,"-");
36                              if(s[0].size()>1)
37                                  hou = s[0].mid(1);
38                              parent->addchild(newnode);
39                          }
40                          Node*newnode = new Node(CONSTVALUE,hou);
41                          parent->addchild(newnode);
42                          parent->content+=s[0];
43                          continue;
44                      }
45                      //如果词是嵌入的函数
46                      if(s[1]==EMBEDDEDFUNCTION)
47                      {
48                          //那么新建一个结点，把词的信息保存到结点，添加到双亲结点的儿子结点指针列表中
49                          Node*newnode = new Node(EMBEDDEDFUNCTION,s[0]);
50                          parent->addchild(newnode);
51                          buildtree(splitresult,newnode,++i,true);
52                          parent->content+=s[0];
53                          continue;
54                      }
55
56                      //如果词是自定义的函数
57                      if(s[1]==CUSTOMIZEFUNCTION)
58                      {
59                          Node*newnode = new Node(CUSTOMIZEFUNCTION,s[0]);
60                          parent->addchild(newnode);
61                          buildtree(splitresult,newnode,++i,true);
```

```
62                          parent->content+=s[0];
63                          continue;
64                      }
65
66                      //如果词是左括号
67                      if(s[0]=="(")
68                      {
69                          //
70                          //那么新建一个结点，把词的信息保存到结点，添加到双亲结点的儿子结点指针列表中
71                          Node*newnode = new Node(INBACKET,"");
72                          parent->addchild(newnode);
73
74                          //然后递归，传入的参数是 1 这个新建的结点，下一个位置的下标，不是函数
75                          buildtree(splitresult,newnode,++i,false);
76                          parent->content+=s[0];
77                          continue;
78                      }
79                      if(s[0]==")"||s[0]==",")
80                      {
81                          return;
82                      }
83
84                      //如果词是运算符
85                      if(s[1]==OPTR)
86                      {
87                          //那么新建一个结点，把词的信息保存到结点，添加到双亲结点的儿子结点指针列表中
88                          Node*newnode = new Node(OPTR,s[0]);
89                          parent->addchild(newnode);
90                          parent->content+=s[0];
91
92                      }
93
94                  }
95              }
96              else
97              {
98                  if(index<splitresult.size()&&splitresult[index][0]!="(")
99                  {
100                     //函数名的下一个应该是左括号，否则错误。
101                     //QMessageBox::warning(nullptr,"输入错误","没有输入参数到函数
    "+parent->content);
102                     error = true;
103                     return;
104                 }
105
106                 for(;index<splitresult.size();)
107                 {
108                     if(splitresult[index][0]==")")return;
109                     Node*newnode = new Node(INBACKET,"");
110                     parent->addchild(newnode);
111                     buildtree(splitresult,newnode,++index,false);
112                 }
113             }
114         }
```

3.

```
double MainWindow::calc(Node *parentnode)
{

    parentnode->addchild(new Node(OPTR,"#"));

    //opndstack 是放操作数的栈
    QVector<float>opndstack;
    //放运算符的栈
    QVector<QString>optrstack={"#"};

        //先把儿子结点列表和它的大小保存起来，避免重复地调用
        QVector<Node*>childs = parentnode->childs;
        double size = childs.size();
        for(int i =0;i<size && optrstack.size()>0;i++)
        {

            Node*child = childs[i];
            //如果是空的就跳过
            if(child->content==""||child->content==" ")continue;

            //如果是运算符，
            if(child->type==OPTR)
            {
                //比较运算符栈顶端的运算符和输入的运算符的优先级
                QString compare = Precede(optrstack.last() ,child->content);
                //如果是大于
                if(compare==">")
                {
                    //如果操作树栈的长度小于 2，出错
                    if(opndstack.size()<2)
                    {
                        {error = true; return error;}
                    }

                    //弹出操作数栈顶端 2 个操作数和运算符栈顶端的一个运算符
                    QString optr = optrstack.last();
                    optrstack.pop_back();
                    double b = opndstack.last();
                    opndstack.pop_back();
                    double a = opndstack.last();
                    opndstack.pop_back();
                    //运算之后，把结果放回到操作数栈
                    opndstack.push_back(operate(a,optr,b));
                    //下一次继续要这个优先级高的运算符
                    i--;
                }
                //如果等于
                //就弹出运算符栈顶端的运算符
                else if(compare=="=")
                {
                    optrstack.pop_back();
                }
                //如果等于，就把运算符推入到运算符栈
                else if(compare=="<")
```

```
                    {
                        optrstack.push_back(child->content);
                    }
                }

                //如果是操作数
                else
                {

                    //如果是变量、常数、函数，就用它们的成员函数 getval
                    double result = child->getval();
                    //括号里的表达式不允许用 getval 函数
                    if (error)
                    {
                        return error;
                    }
                    opndstack.push_back(result);

                }
            }
            //最后如果操作数栈的长度是 1，就返回这个栈顶的操作数
            if(opndstack.size()==1)
            {
                return opndstack.back();

            }
            else
            {
                error = true;
                return error;
            }

        }
```

4. 结点的取值的函数

```
double getval()
    {
        //枚举分 5 类

        //如果词是常量数字
        if(type==CONSTVALUE)
        {
            //用 Qt 的 toFloat 函数
            return content.toDouble();
            }

            //如果词是变量
            if(type ==VARIABLE)
            {
                //去变量列表去找
                return variablevalues[content];
            }


            if(type ==VARIABLE)
```

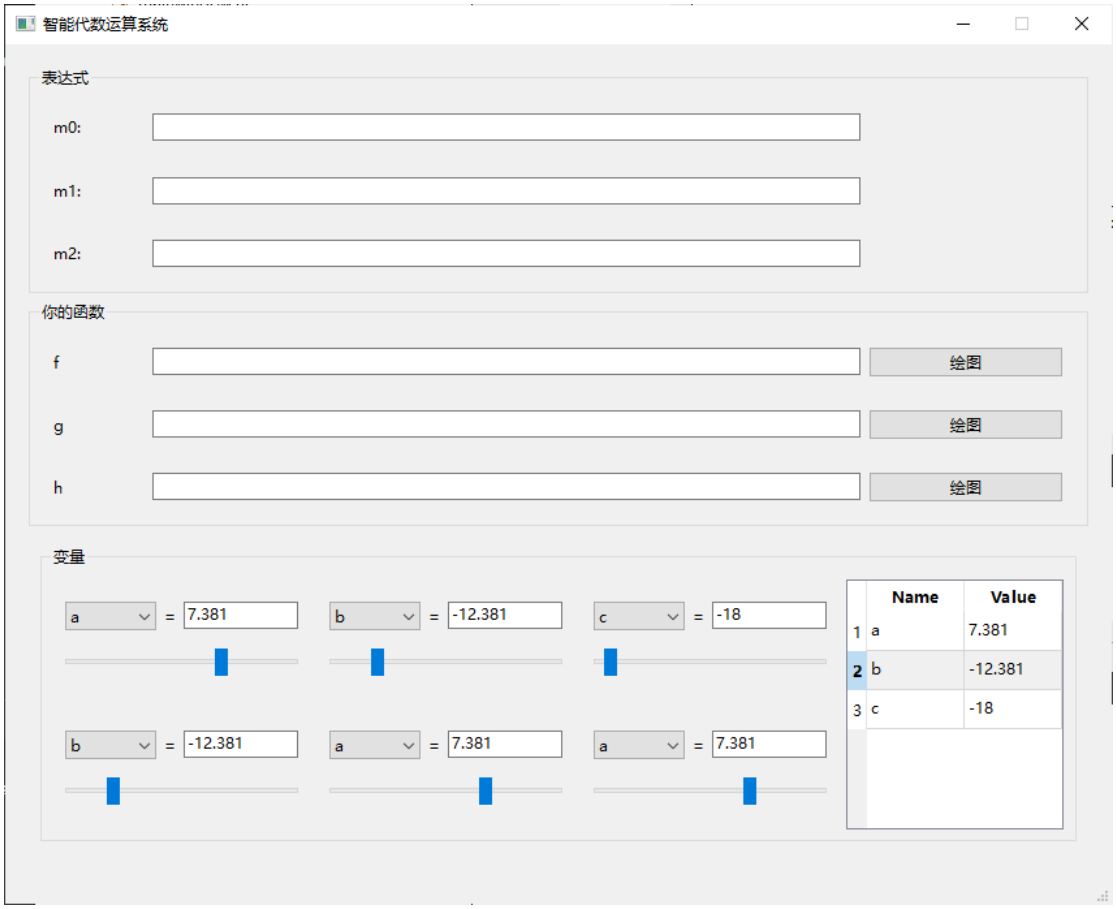
```
21                  {
22                      //去变量列表去找
23                      return calc(this);
24                  }
25                  //如果词是自定义函数的函数名
26                  //想法是切出函数中等号右边的表达式
27                  //然后把这个表达式的变量换成传入的参数
28                  //然后计算，返回计算的值
29                  if(type ==CUSTOMIZEFUNCTION)
30                  {
31
32                      double ans;
33                      //从代码内容中看它属于哪个函数 fgh
34                      double whichfunction = int(content[0].toLatin1())-'f'+3;
35                      //用来保存词法分析的结果
36                      QVector<QVector<QString>>ca = split(whichfunction,"",false);
37                      //等号分成左右两部分
38                      QVector<QVector<QString>>splitleft;
39                      QVector<QVector<QString>>splitright;
40                      //如果 ui 中那个行编辑内容是空，就什么都不做
41                      if(ca[0][0]=="-3"&&ca.size()==1)
42                      {
43                          error =true;
44                      }
45                      //用等号分割函数的表达式
46                      int i =0;
47                      for(;i<ca.size();i++)
48                      {
49                          if(ca[i][0]=="=")break;
50                          splitleft.push_back(ca[i]);
51                      }
52                      i++;
53                      if(i>=ca.size())
54                      {
55                          error = true;
56                          return error;
57                      }
58                      for(;i<ca.size();i++)
59                      {
60                          splitright.push_back(ca[i]);
61                      }
62
63                      //对左边的进行分析，左边就是 f(x)之类的
64                      Node*node0 = new Node(INBACKET,"");
65                      int index = 0;
66                      MainWindow::buildtree(splitleft,node0,index,false);
67                      if(this->childs.size()!=node0->childs[0]->childs.size())
68                      {
69                          error =true;
70                          return error;
71                      }
72                      //然后映射
73                      QMap<QString,QString>m;
74                      for(int i = 0;i<node0->childs[0]->childs.size();i++)
75                      {
76                          m[node0->childs[0]->childs[i]->content]=this->childs[i]->content;
77                      }
78                      //然后做出映射后的新的表达式，就是从原来函数右边的那个表达式，改变它的参数名
```

```
79                      QString newcode = " ";
80                      for(int i =0;i<splitright.size();i++)
81                      {
82                          QString p = splitright[i][0];
83                          if(m.find(p)!=m.end())
84                          {
85                              p = m[p];
86                          }
87                          newcode+=" "+p+" ";
88                      }
89                      newcode+=" ";
90                      //计算这个新表达式的值
91                      ca = split(-1,newcode,false);
92                      Node*node1 = new Node(INBACKET,"");
93                      int index1 = 0;
94                      buildtree(ca,node1,index1,false);
95                      ans = calc(node1);
96                      return ans;
97                  }
98
99                  //如果是词是嵌入函数的函数名
100                 if(type ==EMBEDDEDFUNCTION)
101                 {
102                     if(childs.size()<1)
103                     {
104                         error = true;
105                         return error;
106                     }
107                     //一个个写
108                     if(content=="sin")return sinf(calc(this));
109                     if(content=="cos")return cosf(calc(this));
110                     if(content=="tan")return tanf(calc(this));
111                     if(content=="asin")return asinf(calc(this));
112                     if(content=="acos")return acosf(calc(this));
113                     if(content=="atan")return atanf(calc(this));
114                     if(content=="acosh")return acoshf(calc(this));
115                     if(content=="asinh")return asinhf(calc(this));
116                     if(content=="atanh")return atanhf(calc(this));
117                     if(content=="sqrt")return sqrtf(calc(this));
118                     if(content=="cbrt")return cbrtf(calc(this));
119                     if(content=="ceil")return ceil(calc(this));
120                     if(content=="erf")return erf(calc(this));
121                     if(content=="erfc")return erfcl(calc(this));
122                     if(content=="exp")return exp2f(calc(this));
123                     if(content=="expm1")return expm1f(calc(this));
124                     if(content=="round")return roundf(calc(this));
125                     if(content=="ln")return qLn(calc(this));
126                     if(content=="abs")return qAbs(calc(this));
127                     if(content=="sec")return pow(cosf(calc(this)),-1);
128                     if(content=="floor")return floor(calc(this));
129                     if(content=="log")
130                     {
131                         if(childs.size()<2){error = true; return error;}
132                         return qLn(calc(childs[1]))/qLn(calc(childs[0]));
133                     }
134
135                     if(content=="max")
136                     {
```

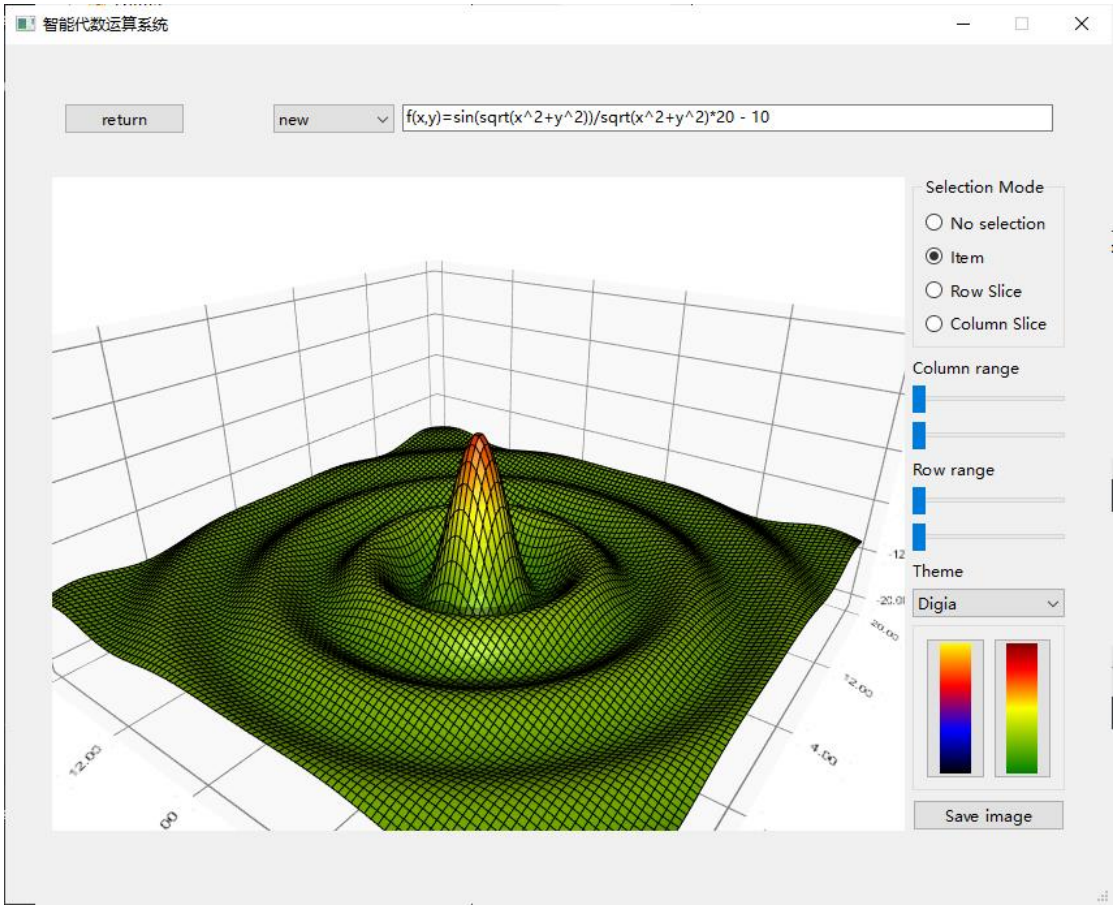
```
137                         double size = childs.size();
138                         if(size<1){error = true; return error;}
139                         double ret = calc(childs[0]);
140                         for(int i =1;i<size;i++)
141                         {
142                             ret = qMax(ret,calc(childs[i]));
143                         }
144                         return ret;
145                     }
146                     if(content=="min")
147                     {
148                         double size = childs.size();
149                         if(size<1){error = true; return error;}
150                         double ret = calc(childs[0]);
151                         for(int i =1;i<size;i++)
152                         {
153                             ret = qMin(ret,calc(childs[i]));
154                         }
155                         return ret;
156                     }
157                     if(content=="mod")
158                     {
159                         if(childs.size()<2){error = true; return error;}
160                         return int(calc(childs[0]))%int(calc(childs[1]));
161                     }
162                 }
163                 error = true;
164                 return error;
165             }
```

四、 程序运行时的效果图

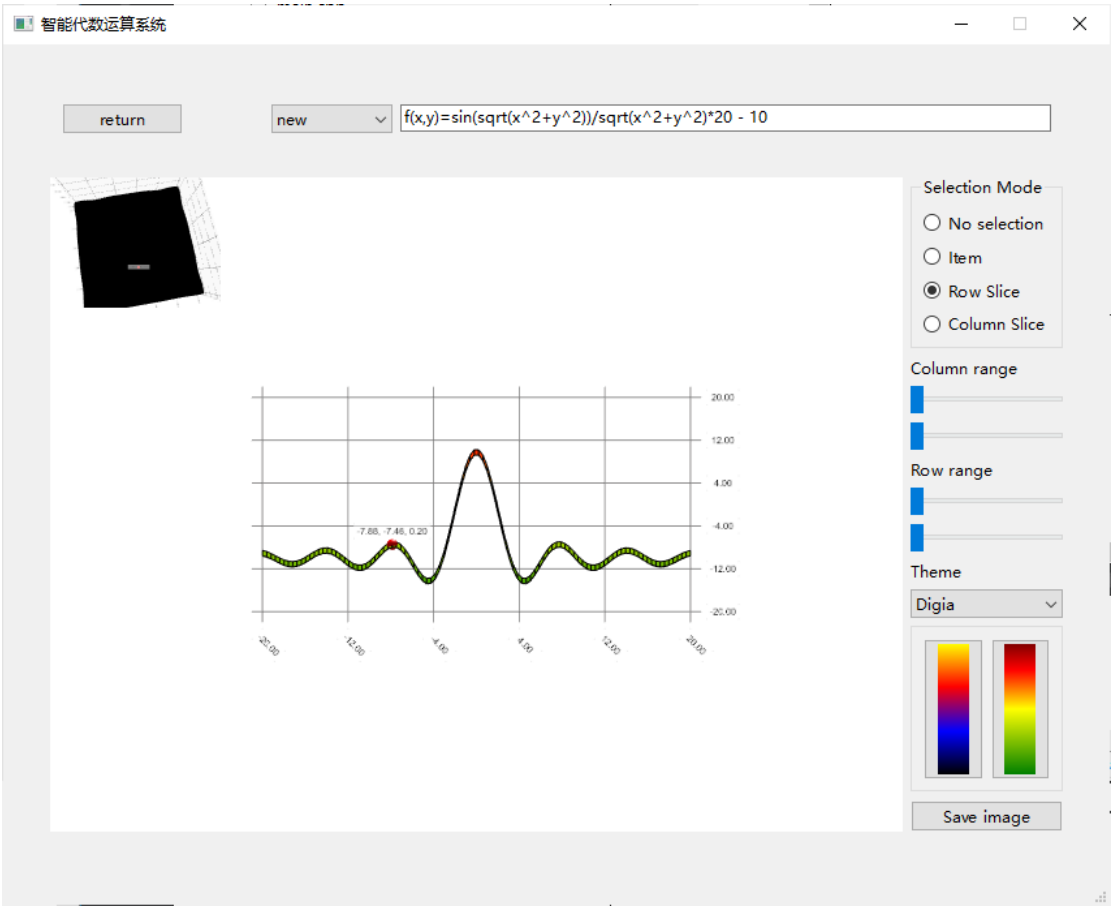
第 1 页


智能代数运算系统
表达式
m0:
m1:
m2:
你的函数
f
绘图
g
绘图
h
绘图
变量
a = 7.381
b = -12.381
c = -18
b = -12.381
a = 7.381
a = 7.381
Name
Value
1 a 7.381
2 b -12.381
3 c -18

图表 1 第 1 页

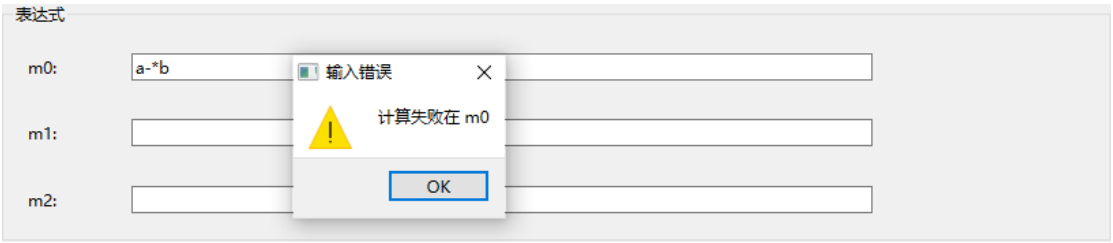

智能代数运算系统
return
new
f(x,y)=sin(sqrt(x^2+y^2))/sqrt(x^2+y^2)*20 - 10
Selection Mode
No selection
Item
Row Slice
Column Slice
Column range
Row range
Theme
Digia
Save image

图表 2 默认的模式



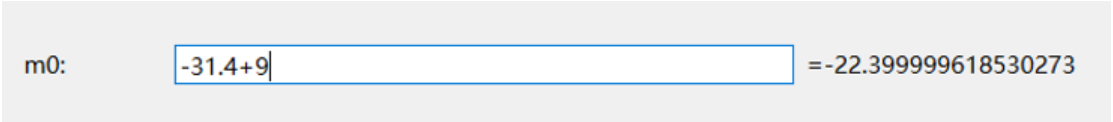
图表 3 切片的模式

错误提示

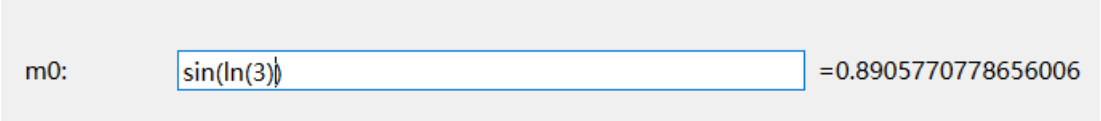


图表 4 错误提示

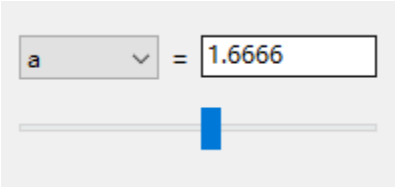
好功能



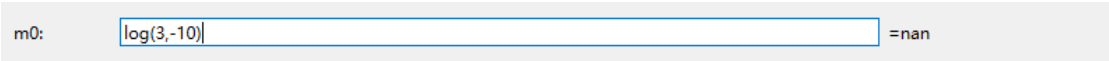
图表 5 计算小数和负数和大于 10 的数



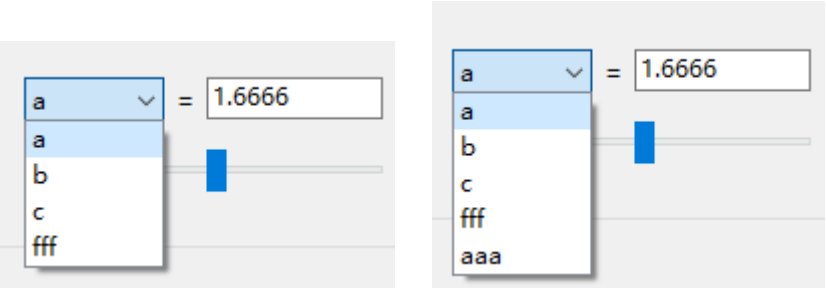
图表 6 嵌套着用函数



图表 7 拖动滑块



图表 8 当变量不在定义域中，比如负数不可用开方，会显示 nan



图表 9 通过滑块和行编辑改变未申明变量的值

五、 实验结果分析，实验收获和体会

结果

实验结果是我写出一个智能代数运算系统。功能有这么几点，列在下面。

1. 计算加减乘除，和幂
2. 可以用一些数学函数
3. 可以省略变量或者数字之间的乘号
4. 可以用自定义的函数
5. 默认给未申明的变量随机的初始的值

6. 可以用滑块调整变量的值
7. 可以画出函数图像

分析

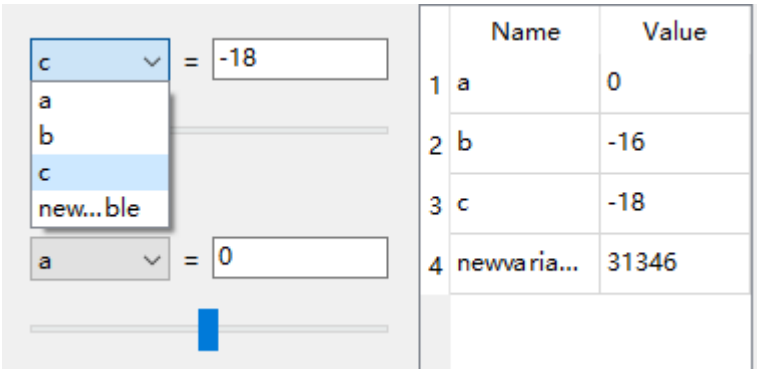
表达式求值的步骤是 1 词法分析 2 做语法树 3 算。

1. 词法分析是把表达式分割，输出一个放词的列表。接着给列表中每个词分类，分类成常量数字、变量名、嵌入的函数的函数名、自定义的函数的函数名和运算符。如果是常量，就把它放到变量的一个表，如果这个变量没有出现过，就给一个随机的值。
2. 做语法树是把词法分析输出的一个词的列表转变一个树。它的结点可以是常量数字、变量名、嵌入的函数的函数名、自定义的函数的函数名和运算符。
3. 算是根据语法树算，像数据结构中的表达式求值。跟它不同的是操作数是各个结点的 getval 的返回值。

收获

1. 熟悉了把表达式分割，和树，和遍历树，和栈，和表达式求值。
2. 调整未申明的变量的值实现方法是，在分割表达式时，当得到一个词，如果它不是符号或者数字常量时，那么它就是以字母开头的词，它可能是变量名或者函数名。当在变量名表（映射变量名到变量值的表，类型是 map<string,int>m）和函数名表找不到它的时候，那么它就是未申明的变量。首先把它添加到变量表，

m[“newvar”]=rand();，接着更新界面中的下拉列表和变量表，就完成。



体会

六、 自评成绩

100。

首先因为我参考老师的建议。

1. 改进了滑块我没有改滑块的范围，我改的是滑块和变量值的映射。滑块的范围是（-100000，100000）变量的值是滑块的值除以 5000，也就是说变量的范围是（-20，20）。这样做虽然牺牲了变量的范围，不过带来了几点好处，1 滑块更加灵敏，2 变量可以是小数，它可以是最小每步改 0.0002。
2. 添加了取模的函数取模。要注意的问题是，被除数不应该是 0；输入参数可能是小数，要强制类型转换，把它变成 int 型再继续做。

因为相比于老师参考的程序，我的新功能还有以下几点。

1. 智能的嵌入的数学函数，比如在 max 函数中，参数可以多于 2 个，可以用 max(a,b,c,d,e,f,g)。
2. 绘制三维的函数图像。用到的工具是 Qt DataVisualization, Q3dSurface。这个东西没有多少参考。
3. 可以调整未申明的变量的值，作业要求上写的是，如果是未申明的变量，就给一个随机的初始的值。那么如果用户想要改这个值呢，我的可以通过下拉选项更改未申明的变量的值。
4. https://drive.google.com/drive/folders/1rO3hRRiTh6RswvlVRTU9Y-Abged-Jz8i?usp=sharing
5. 界面设计优美。

综上所述，我的自评成绩是 100 分。