# 可编程计算器-实验报告

## 高瓴人工智能学院 人工智能1班 张宇尧 2020201710

**2021/11/4　指导老师：窦志成**



## 实验目的🎯

1. 掌握顺序表、链表、串和数组等结构的基本知识和使用技术。
2. 借助互联网资源，自主学习和运用数学知识。
3. 培养对问题建模和抽象的能力。
4. 培养设计和使用新工具的能力。
5. 培养自学能力。

# 实验摘要📚

设计一个交互式的计算器，用户可以提出不同的的数据处理或计算要求。

- 可编程表达式计算
- 可编程矩阵运算
- 表达式直接计算
- 用顺序表实现的向量计算
- 用顺序表实现一元多项式计算
- 用链表实现的一元多项式计算

**其中，每个模式均增加了一些比较方便实用的功能。**

如一元多项式和向量的相关计算基本上可**按照引导**进行，体现人机交互的简易；向量的计算中增加了两个向量之间的**余弦相似度**计算；针对表达式计算，则通过多个函数实现了**浮点数运算**，并且可以设置**变量参数**，通过赋值对表达式进行计算；可编程表达式计算中，支持一些**复合函数定义**和**函数复用**，方便用户使用；针对矩阵运算，则支持一系列➕，➖，✖，以及**矩阵行列式**和**特征值**的计算。😇

# 实验要求🙅

- 完成实验的实验报告，报告的格式采用《数据结构题集》的模板格式。
- 提供独立的计算器使用手册。
- 提供独立的功能测试报告。报告要附充分的测试用例。
- 提供完整的源代码、执行码以及生成执行码的项目工程文件。

# 实验步骤🐌

- 用C语言定义所需结构的 ADT 和基本操作。
- 设计测试用例。
- 实现各项功能和主函数
- 测试自己完成的练习，包括不断改进程序的输入、输出等。
- 撰写各种文档和实验报告。

# 使用手册✏️

用户在运行程序后，看到如下显示：

```
=========Welcome to my Counter(* ▽* ) / ========
选择计算模式
A:表达式计算;B:编程计算;C:编程矩阵运算;
D:向量的相关运算;E:用顺序表实现多项式的相关计算;F:用链表实现多项式的相关计算。
```

然后在终端输入**"A","B","C","D","E","F"**中的一个，注意需要为**【大写】**，然后直接按下回车键。

## 计算模式➕➖✖➗

**如上图可知，该计算器实现6种计算模式，实现了5种计算要求。**

- A:表达式计算
- B:编程计算
- C:编程矩阵运算
- D:向量的相关运算
- E:用顺序表实现多项式的相关计算
- F:用链表实现多项式的相关计算

其中，E和F属于同一计算要求，是《数据结构》课程中对两种结构：线性表和链表 的练习和实现。用户可以通过两种结构的实现互相印证，也可以帮我检查一下有没有什么隐藏的bug **XD**。

## 相关操作和使用🧨

**含参表达式计算**

**用户直接输入需要求取的表达式，然后按下回车即可计算答案。**

- 四则运算表达式求值，操作符包括加('+')、减('-')、乘('*')、除('/')、左右括号，而操作数则包括整数和浮点数等C/C++支持的不同类型的数值。
- 含单变量表达式的求值，用户设置形如$x, y, x_1, x_2, u, v$等变量。
- 含多变量表达式的求值，用户可以在一个表达式中输入不同变量的值，按照变量第一次出现的顺序输入。
- 相同的且重复出现的变量只需要输入一次。

**输入示例**：

```
请输入表达式：
(5+4)*3/2+1
结果为：14.5
```

```
请输入表达式：
x/2+x*3
请输入变量的值：
1
结果为：3.5
```

```
请输入表达式：
(5+x1)*x2/x3+1
请输入变量的值：
4.9
请输入变量的值：
2
请输入变量的值：
3
结果为：7.6
```

**注意**：

1. 表达式中，**【不需要】**也**【无法】**加空格，在输入时请注意，会报错（当然因此会简化算法的复杂醒，精力有限hhh）。
2. 表达式不易过长，会有长度限制，（但目前暂时未知最长是多少）。

**编程计算**

**这是一个由用户自定义函数和运行的模式，现将相关指令和注意事项约定如下：**

- **【DEF】**方法
  - 定义一个函数，格式为`DEF f(x)`；

    ```
    DEF f(x)=x+1
    ```

- **【RUN】**方法

- 运行一个函数，格式为`RUN f(1)`；

```
RUN f(1)
```

- **【END】**方法
  - 结束编程计算环境。直接输入`END`,按下回车键即可退出程序。
- **【^】方法**
  - 函数的次方。格式为`DEF g(x)=f(x)^2`;
- 保留了历史函数，可以运行历史函数。
- 函数的调用。已经定义了$f(x)$,$g(x)$中可以重复调用。

**输入示例**：

```
DEF f(x)=x+1
你定义了第1个多项式：f(x)=x+1
DEF g(x)=3+4*f(x)
你定义了第2个多项式：g(x)=3+4*(x+1)
RUN g(2.5)
结果为：17
```

```
DEF f(x)=x+1
你定义了第1个多项式：f(x)=x+1
DEF g(x)=f(x)^2
你定义了第2个多项式：g(x)=(x+1)*(x+1)
```

**编程矩阵计算**

**这是一个由用户自定义函数和运行的模式，现将相关指令和注意事项约定如下：**

- **【DEF】**方法
  - 定义一个矩阵，格式为`DEF m1`
  - 按下回车之后将输入【行数】和【列数】，格式为`row col`,row和col为输入的行和列
  - 按下回车后将输入矩阵的值，依次输入，输够后按下回车。
  - 然后将显示矩阵的名字和值。
- **【print 】**方法

    - 显示已经定义的矩阵，格式为`print m1`；
- **【DET】**方法
    - 求矩阵的行列式，格式为`DET m1`；
- **【EIG】**方法
    - 求矩阵的特征值，格式为`EIG m1`；
    - 该方法同时还调用了多个函数，如**【DET】**方法和判断是否为非奇异矩阵，并且用到了QR分解迭代方法求取矩阵特征值，精度为$1e-5$级别。
- 矩阵基本运算方法
    - 支持➕，➖，✖，数乘方法。
    - `m1+m2`
    - `m1*m2`
    - `1.5*m1`
- **【END】**方法
    - 使用方法同上。

**输入示例**：

```
m1
2  2  2
2  2  3
4  5  2
1.5*m1
数值:1.5
3  3  3
3  3  4.5
6  7.5  3
EIG m1
该矩阵的行列式为:-2
该矩阵时非奇异矩阵，可求取特征值
原矩阵化为上Hessenberg矩阵为:
2  -2.68328  -0.894427
-4.47214  5.2  3.4
4.44089e-16  1.4  -1.2
QR分解求取特征值:
在1e-5精度下，原矩阵的全部特征值为:

a1=7.87299      a2=-2   a3=0.127017
```

# 实验代码👩‍💻

贴几个关键函数。

### 字符串转double函数

```
double str2double(string str){
  bool demical = false ;
  double count = 0 ;
  double d = 0 ;
  double e = 0 ;
  if(str[0] == '-'){
    for(int index = 1 ; index < str.length() ; index++){
      int k = index ;
      for(int j = k ; str[j] != '.' && j < str.length() ; j++){
        count++ ;
      }
      for(;count > 0 ; count--){
        e += (str[index] - '0') * pow(10 , count - 1) ;
        index++ ;
      }
      count = 0 ;
      if(str[index] == '.'){
        demical = true ;
      }
      if(demical){
        for(int j = index + 1 ; j < str.length() ; j++){
          count += 1 ;
          e += double(str[j] - '0') / pow(10,count) ;
          k = j ;
        }
        break ;
      }
    }
    e = -e ;
  }
  else{
    for(int index = 0 ; index < str.length() ; index++){
```

```
      int k = index ;
      for(int j = k ; str[j] != '.' && j < str.length() ; j++){
        count++ ;
      }
      for(;count > 0 ; count--){
        e += (str[index] - '0') * pow(10 , count - 1) ;
        index++ ;
      }
      count = 0 ;
      if(str[index] == '.'){
        demical = true ;
      }
      if(demical){
        for(int j = index + 1 ; j < str.length() ; j++){
          count += 1 ;
          e += double(str[j] - '0') / pow(10,count) ;
          k = j ;
        }
        break ;
      }
    }
    e = e ;
  }
  return e ;
}
```

**说明**:这个函数用来解析【RUN】方法中的字符串，作为double类型变量来进行后续输入和计算。刚开始，没能考虑到负数的情况，导致负数输入存在bug。如上为修改后的算法。

### 高斯列主元方法求矩阵行列式

```
//高斯列主元方法求行列式
double Det(Matrix m){
  Matrix A = m ;
  int n = m.row ;
  Matrix tmp = m ; //保存原数组，结束后要复原用
  if(m.row != m.col){
    cout << "非合法矩阵，无法求行列式" << endl ;
```

```
    return 0 ;
  }
  double det = 1 ;
  double t ;
  for(int k = 0 ; k < n - 1 ; k++){
    double max = m.matrix[k][k] ;
    int Ik = k ;
    for(int j = k + 1 ; j < n ; j++){
      if(max <= m.matrix[j][k]){
        max = m.matrix[j][k] ;
        Ik = j ;
      }
    }
    if(max == 0){
      return 0 ;
    }
    if(Ik != k){
      for(int j = k ; j < n ; j++){
        t = m.matrix[Ik][j] ;
        m.matrix[Ik][j] = m.matrix[k][j] ;
        m.matrix[k][j] = t ;
      }
      det *= -1 ;
    }
    for(int i = k + 1 ; i < n ; i++){
      t = m.matrix[i][k] / m.matrix[k][k] ;
      m.matrix[i][k] = t ;
      for(int j = k + 1 ; j < n ; j++){
        m.matrix[i][j] = m.matrix[i][j] - t * m.matrix[k][j] ;
      }
    }
    det *= m.matrix[k][k] ;
  }
  if(m.matrix[n - 1][n - 1] == 0){
    cout << "该矩阵的行列式为:0" << endl ;
    return 0 ;
  }
  else{
```

```
    cout << "该矩阵的行列式为:" << det * m.matrix[n - 1][n - 1] << endl ;
    double ans = det * m.matrix[n - 1][n - 1] ;
    m = tmp ;
    return ans ;
  }
}
```

**说明**：这个算法用来求行列式。

### 将原矩阵化为Hessenberg矩阵

```
//将原矩阵化为Hessenberg矩阵
Matrix Hessenberg(Matrix m){
  int n = m.row ;
  double T[n][n] ;
  double B[n][n] ;
  double C[n][n] ;
  double R[n - 1][n - 1] ;
  double I[n - 1][n - 1] ;
  double c[n] ;
  double v[n] ;
  double u[n] ;
  double t , w , s ;
  int i , j , k , l ;
  Matrix tmp = m ;
  for(k = 0 ; k < n - 2 ; k++){
    for(i = 0 ; i < n - k - 1 ; i++){
      for(j = 0 ; j < n - k - 1 ; j++){
        if(i == j) I[i][j] = 1 ;
        else I[i][j] = 0 ; //定义单位矩阵I
      }
    }
    double max = fabs(m.matrix[k + 1][k]) ;
    for(i = 0 ; i < n - k - 1 ; i++){
      if(max < fabs(m.matrix[i + k + 1][k])) max = fabs(m.matrix[i + k + 1][k]) ;
    //求最大值
    }
```

```
    for(i = 0 ; i < n - k - 1 ; i++){
      c[i] = m.matrix[i + k + 1][k] / max ;
    //标准化数组
    }

    if(IsZero(c , n - k - 1)) continue ;
    //若数组为0，则这一步不需要约化

    for(i = 0 , t = 0.0 ; i < n - k - 1 ; i++){
      t += c[i] * c[i] ;
    }

    v[k] = sgn(m.matrix[k+1][k]) * sqrt(t) ;
    u[0] = c[0] + v[k] ;

    for(j = 1 ; j < n - k - 1 ; j++) u[j] = c[j] ;
    w = v[k] * (c[0] + v[k]) ;

    for(i = 0 ; i < n - k - 1 ; i++){
      for(j = 0 ; j < n - k - 1 ; j++) R[i][j] = I[i][j] - u[i] *
u[j] / w ;
    }

    for(i = 0 ; i < n ; i++){
      for(j = 0 ; j < n ; j++){
        if(i == j) T[i][j] = 1 ;
        else T[i][j] = 0 ;
      }
    }

    for(i = 0 ; i < n - k - 1 ; i++){
      for(j = 0 ; j < n - k - 1 ; j++){
        T[i + k + 1][j + k + 1] = R[i][j] ;
      }
    }

    for(i = 0 ; i < n ; i++){
      for(j = 0 ; j < n ; j++){
        for(l = 0 , s = 0.0 ; l < n ; l++){
```

```
            s += T[i][l] * m.matrix[l][j] ;
          }
        B[i][j] = s ;
        }
      }
      for(i = 0 ; i < n ; i++){
        for(j = 0 ; j < n ; j++){
          for(l = 0 , s = 0.0 ; l < n ; l++){
            s += B[i][l] * T[l][j] ;
          }
        C[i][j] = s ;
        }
      }
      for(i = 0 ; i < n ; i++){
        for(j = 0 ; j < n ; j++){
          m.matrix[i][j] = C[i][j] ;
        }
      }
    }
    cout << "原矩阵化为上Hessenberg矩阵为：" << endl ;
    printMatrix(m) ;
    return tmp ;
}
```

**说明**：QR分解的关键一步。高等代数会学到。

**QR分解求矩阵特征值**

```
//QR算法求特征值
void QRAlgorithm(Matrix m){
  int count = 1 ;
  int n = m.row ;
  int i , j , k , l ;
  double p[n][n] ;
  double Q[n][n] ;
  double R[n][n] ;
  double F[n][n] ;
  double V[n][n] ;
  double c , s , t , y , max ;
```

```
  for(int i = 0 ; i < n ; i++){
    for(int j = 0 ; j < n ; j++){
      if(i != j) Q[i][j] = 0 ;
      else Q[i][j] = 1 ;
    }
  }//Q为单位矩阵

  for(l = 0 ; l < n - 1 ; l++){
    for(i = 0 ; i < n ; i++){
      for(j = 0 ; j < n ; j++){
        if(i != j) p[i][j] = 0 ;
        else p[i][j] = 1 ;
      }
    }
    c = m.matrix[l][l] / sqrt(m.matrix[l][l] * m.matrix[l][l] + m.matrix[l + 1][l] * m.matrix[l + 1][l]) ;
    s = m.matrix[l + 1][l] / sqrt(m.matrix[l][l] * m.matrix[l][l] + m.matrix[l + 1][l] * m.matrix[l + 1][l]) ;
    p[l][l] = c ;
    p[l][l + 1] = s ;
    p[l + 1][l] = -s ;
    p[l + 1][l + 1] = c ;
    for(i = 0 ; i < n ; i++){
      for(j = 0 ; j < n ; j++){
        t = 0 ;
        y = 0 ;
        for(k = 0 ; k < n ; k++){
          t += p[i][k] * m.matrix[k][j] ;
          y += p[i][k] * Q[k][j] ;
        }
        R[i][j] = t ;
        F[i][j] = y ;
      }
    }
    for(i = 0 ; i < n ; i++){
      for(j = 0 ; j < n ; j++){
        m.matrix[i][j] = R[i][j] ;
        Q[i][j] = F[i][j] ;
      }
```

```
    }
  }
  for(i = 0 ; i < n ; i++){
    for(j = 0 ; j < n ; j++) V[i][j] = Q[j][i] ;
  }
  for(i = 0 ; i < n ; i++){
    for(j = 0 ; j < n ; j++) Q[i][j] = V[i][j] ;
  }
  for(i = 0 ; i < n ; i++){
    for(j = 0 ; j < n ; j++){
      t = 0 ;
      for(k = 0 ; k < n ; k++) t += R[i][k] * Q[k][j] ;
      m.matrix[i][j] = t ;
    }
  }
  count ++ ;
  max = fabs(m.matrix[1][0]) ;
  for(i = 1 ; i < n ; i++){
    for(j = 0 ; j < i ; j++){
      if(fabs(m.matrix[i][j]) > max) max = fabs(m.matrix[i][j]) ;
    }
  }
  if(max < eps){
    cout << "在1e-5精度下，原矩阵的全部特征值为：" << endl ;
    for(i = 0 ; i < n ; i++){
      if(i % 3 == 0) cout << '\n' ;
      cout << "a" << i + 1 << '=' << m.matrix[i][i] << '\t' ;
    }
    cout << '\n' ;
    return ;
  }
  QRAlgorithm(m) ;
}
```

**说明**：QR分解算法，递归的思想。

## 编程计算

```
//编程计算
void ProgrammingEvaluate(){
  int flag ;
  int i = 0 ;
  int j = 0 ;
  string str ;
  string Expression ;
  SString tmp ;
  string Expression_array[20] ;
  Func Func[20] ;
  Expression = '\n' ;
  getline(cin , Expression) ;
  getline(cin , Expression) ;
  while(Expression != "END"){
    int index1 = Expression.find("DEF") ;
    int index2 = Expression.find("RUN") ;
    if(index1 != -1){
      for(j = index1 + 4 ; Expression[j] != '=' ; j++){
        Func[i].ch += Expression[j] ;
      }
      string exp_tmp = Expression.substr(j+1) ;
      for(int k = 0 ; k <= i ; k++){
        int index_sub = exp_tmp.find(Func[k].ch) ;
        if(index_sub != -1){
          string tmp1 = Func[k].expression ;
          string tmp_before = exp_tmp.substr(0 , index_sub) ;
          string tmp_after = exp_tmp.substr(index_sub +
Func[k].ch.length()) ;
          string tmp3 = exp_tmp.substr(0,exp_tmp.find(Func[k].ch))
+ '(' + tmp1 + ')' + exp_tmp.substr(index_sub +
Func[k].ch.length()) ;
          exp_tmp = tmp3 ;
        }
      }
      if(exp_tmp.find('^') != -1){
        int expm = exp_tmp[exp_tmp.find('^') + 1] - '0' ;
        int index ;
```

```
        string exp_tmp0 = '(' + exp_tmp.substr(0 , exp_tmp.find('^')) + ')' ;
        string exp_tmp1 = exp_tmp0 ;
        for(int i = 0 ; i < expm - 1 ; i++){
          exp_tmp1 += '*' + exp_tmp0 ;
        }
        exp_tmp = exp_tmp1 ;
      }
      Func[i].expression = exp_tmp ;
      cout << "你定义了第" << i + 1 << "个多项式: " << Func[i].ch << '=' << Func[i].expression << endl ;
      i++ ;
    }
    if(index2 != -1){
      for(int k = 0 ; k <= i ; k++){
        if(Expression[4] == Func[k].ch[0]){
          string tmp ;
          int tmp0 = 6 ;
          while(Expression[tmp0] != ')'){
            tmp += Expression[tmp0] ;
            tmp0++ ;
          }
          double ans = EvaluateExpression2(Func[k].expression , tmp) ;
          cout << "结果为: " << ans << endl ;
          break ;
        }
      }
    }
    getline(cin , Expression) ;
  }
}
```

**说明**：类似matlab输入格式的编程计算实现的函数。

### 编程矩阵计算

```
//编程矩阵计算
void ProgrammingMatrixEvaluate(){
  string Expression ;
  Matrix matrix[20] ;
  int ind[2] = {0};
  Expression = '\n' ;
  int i = 0 ;
  cout << "进入矩阵编程模式:" << endl ;
  getline(cin , Expression) ;
  getline(cin , Expression) ;
  while(Expression != "END"){
    int index1 = Expression.find("DEF") ;
    if(index1 != -1){
      matrix[i] = CreatMatrix() ;
      matrix[i].var_name = Expression.substr(4) ;
      cout << matrix[i].var_name << endl ;
      printMatrix(matrix[i]) ;
      i++ ;
    }
    else if(Expression.find("DET") != -1){
      string tmp = Expression.substr(4) ;
      for(int k = 0 ; k <= i ; k++){
        if(tmp == matrix[k].var_name){
          int n = matrix[k].row ;
          double a[n][n] ;
          for(int i = 0 ; i < n ; i++){
            for(int j = 0 ; j < n ; j++){
              a[i][j] = matrix[k].matrix[i][j] ;
            }
          }
          Matrix tmp1 = matrix[k] ;
          Det(matrix[k]) ;
          for(int i = 0 ; i < n ; i++){
            for(int j = 0 ; j < n ; j++){
              matrix[k].matrix[i][j] = a[i][j] ;
            }
          }
```

```
38            break ;
39          }
40          }
41      }
42      else if(Expression.find("EIG") != -1){
43        string tmp = Expression.substr(4) ;
44        for(int k = 0 ; k <= i ; k++){
45          if(tmp == matrix[k].var_name){
46            int n = matrix[k].row ;
47            double a[n][n] ;
48            for(int i = 0 ; i < n ; i++){
49              for(int j = 0 ; j < n ; j++){
50                a[i][j] = matrix[k].matrix[i][j] ;
51              }
52            }
53            SeekEigenvalue(matrix[k]) ;
54            for(int i = 0 ; i < n ; i++){
55              for(int j = 0 ; j < n ; j++){
56                matrix[k].matrix[i][j] = a[i][j] ;
57              }
58            }
59            break ;
60          }
61          }
62      }
63      else if(Expression.find("print") != -1){
64        string tmp = Expression.substr(6) ;
65        for(int k = 0 ; k <= i ; k++){
66          if(tmp == matrix[k].var_name){
67            printMatrix(matrix[k]) ;
68            break ;
69          }
70          }
71      }
72      else if(Expression.find('+') != -1){
73        string tmp = Expression.substr(0 , Expression.find('+')) ;
74        string tmp2 = Expression.substr(Expression.find('+') + 1) ;
75        for(int k = 0 ; k <= i ; k++){
76          if(tmp == matrix[k].var_name){
```

```
          ind[0] = k ;
        }
        if(tmp2 == matrix[k].var_name){
          ind[1] = k ;
        }
      }
      printMatrix(add(matrix[ind[0]] , matrix[ind[1]])) ;
    }
    else if(Expression.find('-') != -1){
      string tmp = Expression.substr(0 , Expression.find('-')) ;
      string tmp2 = Expression.substr(Expression.find('-') + 1) ;
      for(int k = 0 ; k <= i ; k++){
        if(tmp == matrix[k].var_name){
          ind[0] = k ;
        }
        if(tmp2 == matrix[k].var_name){
          ind[1] = k ;
        }
      }
      printMatrix(sub(matrix[ind[0]] , matrix[ind[1]])) ;
    }
    else if(Expression.find('*') != -1){
      string tmp = Expression.substr(0 , Expression.find('*')) ;
      string tmp2 = Expression.substr(Expression.find('*') + 1) ;
      if(tmp[0] >= '0' && tmp[0] <= '9'){
        double dbl = str2double(tmp) ;
        for(int k = 0 ; k <= i ; k++){
          if(tmp2 == matrix[k].var_name){
            printMatrix(numMul(matrix[k] , dbl)) ;
          }
        }
      }
      else{
        for(int k = 0 ; k <= i ; k++){
          if(tmp == matrix[k].var_name){
            ind[0] = k ;
          }
          if(tmp2 == matrix[k].var_name){
            ind[1] = k ;
```

```
116             }
117           }
118           printMatrix(mul(matrix[ind[0]] , matrix[ind[1]])) ;
119         }
120       }
121       getline(cin , Expression) ;
122     }
123   }
```

**说明**：类似matlab编程矩阵计算的算法的实现代码。

## 实验反思🤔🧐

在本次实验中，我深入理解了数据结构前五章中学到的各种结构的知识，并且做到了：

- 能够理解进而掌握对线性表、栈、串、数组等结构的定义和使用。
- 能够设计简单的软件。
- 能够查找资料，如何解析输入的表达式字符串。

实验过程中，我认为重要的事情就是如何存储各种各样的数据，以及设计相应的算法和不同函数间的借口，接口是十分重要的，接口反应了程序的嵌套和复杂功能的普适性。各种复杂的过程，均可以被封装成一个函数，并提供接口供其他算法函数复用，能大量提高代码的简洁性。

这个设计的简易计算器，还有许多不足，如暂且未发现的bug（hhh），输入的格式存在一些限制，不是那么的鲁棒以及人性化，只支持一些简单的命令语句，以及有的负数处理似乎过于复杂，所以我推测负数定义和运算一定也存在着格式的bug，只能运行一些不那么复杂的算式。以及在本次实验中，我意识到了C++的强大性与简洁性，在未来的学习中，需要一边学习数据结构，一边自学部分C++的内容。希望未来呢，能够再接再厉，完结撒花🥳🎉