**编译原理实验2-常数处理机**

**一、实验目的与任务**

编译原理实验 “常数处理机” 的核心目的在于强化学生对编译系统中常数处理流程的理论认知与实践能力。一方面，通过实验，学生能够深入理解词法分析如何识别常数、语法分析怎样处理常数表达式、语义分析如何对常数进行类型检查与计算等知识，弥补课堂理论学习的抽象性；另一方面，培养学生运用编程语言实现常数处理算法的能力，提升其解决实际工程问题的技术水平，为后续开发完整编译系统奠定基础。

**二、实验涉及的相关知识点**

**1. DFA的相关概念**

有限自动机（DFA）是编译原理中词法分析的重要工具，以下是其相关概念：

****（1）定义****：DFA 是一个五元组*M*=(*S*,∑,*δ*,*s*0​,*F*)，其中：

* + *S*是一个有限状态集合。例如，在识别整数常数的 DFA 中，可能有起始状态、数字状态、结束状态等。
  + ∑是一个有限输入字母表，它定义了输入符号的集合。对于处理常数的 DFA，∑可能包含数字字符0−9、小数点 .、正负号 +、- 等。
  + *δ*是状态转移函数，它描述了在当前状态下，接收一个输入符号后应转移到的下一个状态。例如，在起始状态下接收一个数字字符，就转移到数字状态。
  + *s*0​是初始状态，是自动机开始工作时的状态。
  + *F*是终止状态集合，当 DFA 处于这些状态时，表示识别到了合法的词法单元。比如，识别整数常数的 DFA，当扫描完一个整数序列后到达的状态就是终止状态。

****（2）工作原理****：DFA 从初始状态*s*0​开始，依次读取输入字符串中的每个字符，根据当前状态和输入字符，通过状态转移函数*δ*确定下一个状态。当输入字符串结束时，如果 DFA 处于终止状态集合*F*中的某个状态，则表示输入字符串被 DFA 接受，即识别出了一个符合规则的词法单元，如常数。

****（3）特点****

* + 确定性：对于每一个状态和输入符号，DFA 都有唯一的下一个状态，不存在不确定性。
  + 有限性：状态集合和输入字母表都是有限的，这使得 DFA 可以用有限的资源实现。

****（4）应用****：在编译中，DFA 主要用于词法分析，将输入的源程序字符流按照词法规则识别为一个个单词，如识别各种常数、标识符、关键字等。通过构建不同的 DFA，可以实现对不同类型词法单元的识别，为后续的语法分析提供基础。

1. **计算机系统中常数的表示方式**

**（1）整数常数**

无符号整数：通常用于表示非负整数，其所有位都用于表示数值大小。例如，在 8 位无符号整数中，可表示的范围是 0 到 255（\(2^8 - 1\)）。

有符号整数：常用的表示方法是补码。最高位为符号位，0 表示正数，1 表示负数。正数的补码与原码相同，负数的补码是其原码除符号位外各位取反，末位加 1。以 8 位有符号整数为例，能表示的范围是 -128 到 127。

**（2）浮点数常数**

一般采用 IEEE 754 标准来表示。该标准规定了单精度（32 位）和双精度（64 位）浮点数的格式。以单精度浮点数为例，32 位中，最高位为符号位，接下来的 8 位为指数位，剩余的 23 位为尾数位。通过这种方式，可以表示绝对值很大或很小的数以及带有小数部分的数。

**三、实验内容**

根据下图DFA，设计并实现一个常数自动机。



1. **实验步骤**
2. 通过生成式AI提示词工程，生成面向C语言的常数识别程序，并尝试逐步规范成本实验要求的实验内容。验证代码，分析AI生成的各函数模块的功能，并根据函数模块调用的顺序和过程，绘制词常数识别程序流程图。
3. 根据该自动机的状态转换矩阵和程序流程图，结合参考实验代码，验证该常数处理机，记录10条以上不同输入的实验结果。

自动机的转换矩阵



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | d | . | e | +|- | 其他（-1） |
| 1 | 2 | 0 | 0 | 0 | 8 |
| 2 | 2 | 3 | 5 | 0 | 0 |
| 3 | 4 | 0 | 0 | 0 | 8 |
| 4 | 4 | 0 | 5 | 0 | 0 |
| 5 | 7 | 0 | 0 | 6 | 0 |
| 6 | 7 | 0 | 0 | 0 | 0 |
| 7 | 7 | 0 | 0 | 0 | 8 |
| 8 | 0 | 0 | 0 | 0 | 0 |

0---err

1. **实验思考**
2. 参考代码中act（）中为什么要减去常数48操作？
3. 尝试进行能力范围内的修改及验证，比如，如何将其修改为识别二进制常数的处理机。
4. **参考代码**

#include "math.h"

#include <iostream.h>

int Pas\_aut[8][5]={2,0,0,0,0,

2,3,5,0,8,

4,0,0,0,0,

4,0,5,0,8,

7,0,0,6,0,

7,0,0,0,0,

7,0,0,0,8,

0,0,0,0,0}; //状态转换矩阵

char line[30]; //常数字符串

int i\_line=0; //指针

class PascalCons

{

private:

int aut[8][5]; //状态转换矩阵

int s; //当前状态

int n,p,m,e,t; //尾数值，指数值，小数位数，指数符号，类型

double num; //常数

char ch; //当前符号

public:

PascalCons();

double number(int \*i); //

private:

void ProcError();

int map(char ch); //当前符号到自动机矩阵的列标记的映射

int find(int s, char ch); //查矩阵

void act(int s, char ch); //结点处的语义动作

};

PascalCons::PascalCons()

{

int i,j;

for (i=0;i<8;i++) //自动机转换矩阵初始化

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

aut[i][j]=Pas\_aut[i][j];

ch=' ';

};

void PascalCons::ProcError()

{

cout<<"err!"<<endl;

};

int PascalCons::map(char ch)

{ int j;

if (ch>='0' && ch<='9')

j=0;

else if (ch=='.')

j=1;

else if (ch=='E'||ch=='e')

j=2;

else if (ch=='+'||ch=='-')

j=3;

else

j=4;

return j;

}

int PascalCons::find(int s, char ch) //s---当前状态；ch---当前符号

{ int i,j; //行和列

i=s-1; //将s映射到行标记i

j=map(ch); //将ch映射到列标记j

return aut[i][j]; //返回下一个状态值

}

void PascalCons::act(int s, char ch)

{

switch (s)

{

case 1: n=0; m=0; p=0; t=0; e=1; num=0; break;

case 2: n=10\*n+ch-48; break;

case 3: t=1; break;

case 4: n=10\*n+ch-48; m++; break;

case 5: t=1; break;

case 6: if (ch=='-') e=-1; break;

case 7: p=10\*p+ch-48; break;

case 8: num=n\*pow(10,e\*p-m);

}

}

double PascalCons::number(int \*p)

{

s=1;

act(s,ch); //执行q1

while (s!=8)

{

ch=line[\*p]; //读取当前符号到ch中

(\*p)++;

s=find(s,ch); //查状态表

if (s==0)

break;

act(s,ch); //执行qs

}

if (s==8)

return num; //输出num

else

{

ProcError();

return 0; //错误处理

}

};

int main(int argc, char\* argv[])

{

PascalCons a;

cout<<"please input a number:";

cin>>line; //读入常数字符串

cout<<a.number(&i\_line)<<endl;

printf("Hello World!\n");

return 0;

}