



题 目:	编译原理上机报告
姓 名:	刘宇鹏
院 系:	计算机学院
班 级:	191181
学 号:	20181003174
指导老师:	刘远兴

2020 年 12 月 5 日

一、 作者简介

刘宇鹏，计算机学院计算机科学与技术专业 18 级学生，班号 191181，学号 20181003174.

二、 报告摘要

本报告为对编译原理课程第一次上机实验的分析总结，包括实验内容及要求、实验代码、实验小组分工实验具体设计思路以及实验结果。

三、 报告目录

目录

一、	作者简介.....	2
二、	报告摘要.....	2
三、	报告目录.....	2
四、	报告正文.....	3
1.	小组分工：	3
2.	需求分析：	3
3.	设计：	3
3.1	总体设计：	3
3.2	自己负责的模块设计——DFA 的模拟以及功能的综合	6
五、	总结.....	14
六、	致谢.....	14

四、 报告正文

题目：C-Minus 语言的编译实验问题

实习时间：2020.11.24

【问题描述】

C-Minus 语言的编译实验问题的一种描述是：设计一个词法分析器，对 C-Minus 语言所编写的程序进行词法分析。

【基本要求】

- ① 设计各单词的状态转换图，并为不同的单词设计种别码。将词法分析器设计成供语法分析器调用的子程序。
- ② 具备预处理功能。将不翻译的注释等符号先滤掉，只保留要翻译的符号串，即要求设计一个供词法分析调用的预处理子程序；
- ③ 能够拼出语言中的各个单词；
- ④ 返回（种别码， 属性值）。
- ⑤ 设计处正则表达式转 NFA、NFA 转 DFA、DFA 最小化模块

1. 小组分工：

191181 第三组 组长：吴勇

正则表达式转 NFA： 陶叶、黄李波

NFA 转 DFA：吴勇、解天宇

DFA 最小化：杨彤

DFA 模拟及功能综合与输入输出设计：刘宇鹏

2. 需求分析：

- 1) 输入一个正则表达式，进行词法分析，识别输入符号串，根据相关规则构造 NFA
- 2) 在 NFA 的基础上，使用子集法构造 DFA
- 3) 使用状态划分法，使 DFA 最小化
- 4) 设计输入输出的界面，可以读入文件中的 C 程序进行分析

3. 设计：

3.1 总体设计：

- 1) 设计思想：

根据实验需求，需要写出至少四个模块：

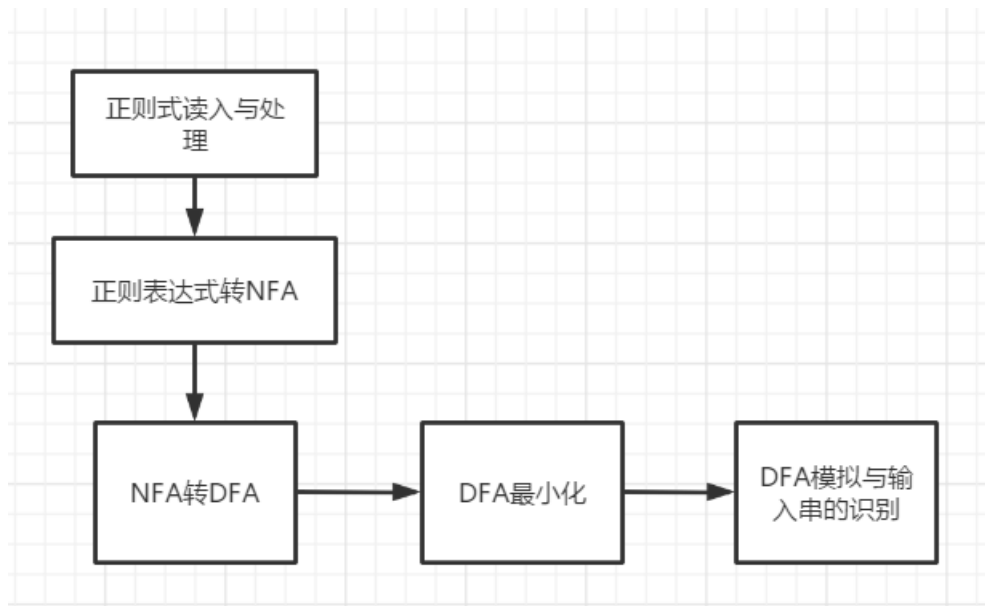
正则式转 NFA 模块

NFA 转 DFA 模块

DFA 化简模块

输入输出（系统界面）模块

- 2) 流程图：



3) 文件结构:

由于我们是在一起讨论进行的编程，所以大家写完自己的部分就都交给我来整合。我们将所有操作都写入了一个文件：源.cpp

4) 数据结构:

由于输入的符号串长度不定，NFA、DFA 的状态数也不确定，所以使用图来储存，最终我们选用的是图的邻接表的结构来存储数据。代码和说明如下：

// 定义 DFA 的构造类

```

class DFA
{
public:
    DFA();
    ~DFA();
    void GetRegExp();
    void InsertCatNode();
    void RegExpToPost();
    void GetEdgeNumber();
    void ThompsonConstruction();
    void SubsetConstruction();
    void FindMatchingPatternInFile();
private:
    char *exp;
    char *post;
    char *edge;
    int edgeNumber;
    int **DStates;
    int **Dtran;
    int *AcceptStates;
    int DStatesNumber;
    int DtranNumber;
  
```

```

    int NFASStatesNumber;
    int DFASStatesNumber;
    AdjacentTable *NFATable;
    TransitionTable *DFATable;
    int Precedence(char symbol);
    int CompArray(int *t1, int *t2);
    int MinimizeDFASStates(int **Dtran, int *AcceptStates, int DtranNumber, int
edgeNumber);
    void RemoveFirstSymbol(char *buf, int &len);
};
// 用邻接表定义的图类
class AdjacentTable
{
    private:
        Vertex *startVertex;
        int numOfVertices;
        int numOfEdges;
    public:
        AdjacentTable();
        ~AdjacentTable();
        int GetValueByPos(int pos) const;
        int GetPosByValue(int value) const;
        char GetWeightByPos(int v1, int v2) const;
        char GetWeightByValue(int value1, int value2) const;
        void SetValue(int value, int pos);
        void InsertVertex(int value);
        void InsertEdgeByPos(int v1, int v2, char weight);
        void InsertEdgeByValue(int value1, int value2, char weight);
        void RemoveAllEdges(void);
        void Clear(void);
        int* Closure(int *T);
        int* Move(int *T, char ch);
        void OutputNFA(void);
};
// 定义邻接表的边表类
class Edge
{
    public:
        int number;
        int position;
        char weight;
        Edge *link;
        Edge();
        Edge(int num, int pos, char ch);
};

```

```
};
// 定义邻接表的顶点类
class Vertex
{
public:
    int number;
    Vertex *next;
    Edge *out;
    Vertex();
    Vertex(int num);
};
```

Exp 输入的正规式
 Post 逆波兰式
 Edge 终结符
 edgeNumber 终结符个数
 DStates 集合
 Dtran DFA 对应的状态迁移表
 AcceptStates 接受状态
 DStatesNumber 集合数量
 DtranNumber DFA 最小化状态数
 NFASStatesNumber NFA 状态数
 DFASStatesNumber DFA 状态数

a) 对正则表达式的处理:

正则表达式先存入一个 `char*` 类型的指针变量中, 然后加入 `cat-node` 作为连接点标志, 用来区分保留字和普通字母。

然后用符号栈来将正则表达式转为逆波兰式, 扫描逆波兰式中的字符, 作为字母表

b) 生成 NFA 的存储结构:

用一个邻接表类指针对象来保存生成的 NFA

将改写后的正则表达式中的字符放入链栈中进行 NFA 的生成并保存到邻接表的类的指针对象中。

c) NFA 转 DFA 的存储结构

用子集法来求 DFA, 先构造 DState 表来存储 ϵ -闭包, 然后构造 Dstran 状态转换表, 这两个表都是用邻接表的方式存储。

d) DFA 最小化的存储结构

通过对 DState 表和 Dstran 表的划分得出最小化的 DFA, 用邻接表的结构来存储。

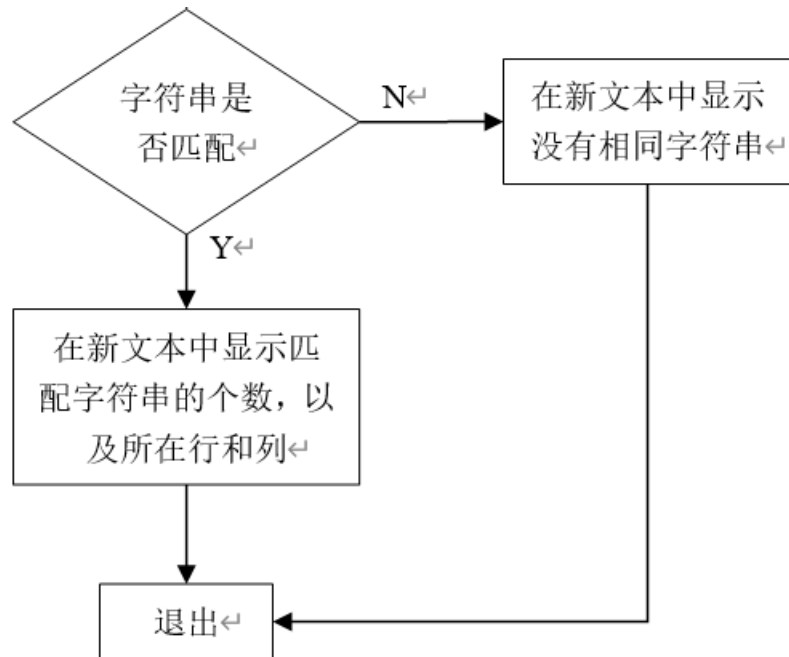
3.2 自己负责的模块设计——DFA 的模拟以及功能的综合

DFA 的模拟即为扫描输入串, 看其中是否有符合正则表达式的部分, 由于其他几位同学已经实现了正则表达式转 NFA, NFA 转 DFA 和 DFA 最小化的过程, 在这里我只需要设计一个算法过程, 使其能够调用其他同学已经设计好的函数接口来扫描输入的

字符串。此外我还需要实现从文件中读取字符串的过程，方便用户使用。

1) DFA 模拟和字符串扫描模块：

思想流程图：



```

void DFA::FindMatchingPatternInFile()
{
    cout << "\n 第八步：查找匹配字符串\n\n";

    char filePath[128];
    fstream infile;
    infile.open("Test.txt", ios::in);
    Test.txt 中存放的是要识别的 C-minus 语言编写的字符串
    if (!infile)
    {
        cout << "\n 打开文件" << filePath << "失败！" << endl;
        _getch();
        exit(1);
    }
    提示用户输入保存结果的文件名：
    cout << "请输入保存记录的文件" << endl;
    cin >> filePath;
    fstream outfile;
    outfile.open(filePath, ios::out | ios::trunc);
    if (!outfile)
    {
        cout << "\n 建立文件" << filePath << "失败！" << endl;
        _getch();
        exit(1);
    }
}
  
```

```
}
```

开始验证正则表达式的匹配串：

```
int ln = 1, col = 0, bufHead = 0;
```

```
int state = 0;
```

```
int count = 0;
```

```
char buf[512];
```

```
int curr = 0;
```

```
int len = 0;
```

```
char ch;
```

顺序读入存放输入串的 **txt** 文件中内容：

```
while (infile.get(ch))
```

```
{
```

遇到空格或换行符时的操作：

```
    if ((ch == '\n') || (ch == ' '))
```

```
    {
```

```
        while (curr < len)
```

```
        {
```

```
            state = 0;
```

写入这行的行号以及调用最小化 **DFA** 函数识别后的结果到输出文件：

```
            while ((state != -1) && (curr < len))
```

```
            {
```

```
                state = DFATable->Transit(state, buf[curr], edge);
```

```
                if (AcceptStates[state] == 1)
```

```
                {
```

```
                    outfile << "\nLn " << ln << ", Col " << bufHead << ": ";
```

```
                    outfile.write(buf, curr + 1);
```

```
                    count++;
```

```
                }
```

```
                curr++;
```

```
            }
```

```
            RemoveFirstSymbol(buf, len);
```

```
            bufHead++;
```

```
            curr = 0;
```

```
        }
```

```
    if (ch == '\n')
```

```
    {
```

```
        ln++;
```

```
        col = 0;
```

```
    }
```

```
    if (ch == ' ')
```

```
    {
```

```
        col++;
```

```
    }
```

```
}
```


读取到的非换行符或空格时进行的操作：

```
else
{
    col++;
    if (len == 0)
    {
        bufHead = col;
    }
    buf[len++] = ch;
    if (len >= 256)
    {
        cout << "读取字符串的长度超过最大限额！" << endl;
        _getch();
        exit(1);
    }
}
```

将识别出来的关键字或正则表达式可以表示的变量写入文件：

```
while (curr < len)
{
    state = 0;
    curr = 0;
    while ((state != -1) && (curr < len))
    {
        state = DFATable->Transit(state, buf[curr], edge);
        if (AcceptStates[state] == 1)
        {
            outfile << "\nLn " << ln << ", Col " << bufHead << ": ";
            outfile.write(buf, curr + 1);
            count++;
        }
        curr++;
    }
    RemoveFirstSymbol(buf, len);
    bufHead++;
}
```

进行统计输出，即共找到的匹配字符串数目：

```
if (count > 0)
{
    outfile << "\n\n 一共找到" << count << "匹配字符串";
}
else
{
    outfile << "\n\n 没有找到任何匹配字符串";
}
```

```
}  
最后关闭打开的文件并提示用户扫描结果保存的位置:
```

```
infile.close();  
outfile.close();  
cout << "\n 查找结果已保存在" << filePath << "中\n"  
      << "\n-----" << endl;  
}
```

2) 资源整合:

将正则表达式的获取模块、正则表达式转 NFA 模块、NFA 转 DFA 模块、DFA 最小化模块以及 DFA 模拟与字符串读取模块都整合到主函数中, 依次进行调用。

```
void main()
```

```
{
```

先实例化一个 DFA 对象以备后用:

```
DFA dfa;
```

获取正则表达式的输入:

```
dfa.GetRegExp();
```

对正则表达式进行处理:

a. 加入 cat-node 作为连结点标志:

```
dfa.InsertCatNode();
```

b. 将处理过的正则表达式转为逆波兰式:

```
dfa.RegExpToPost();
```

c. 获取字符集合:

```
dfa.GetEdgeNumber();
```

d. 用 Thompson 构造法构造 NFA:

```
dfa.ThompsonConstruction();
```

e. 用子集构造法构造 DFA 并进行最小化操作:

```
dfa.SubsetConstruction();
```

f. 读入包含输入串的文件, 从中找出所有的匹配式并输出到新文件:

```
dfa.FindMatchingPatternInFile();
```

```
system("pause");
```

```
return;
```

```
}
```

3) 调试分析:

(1) 优点分析: 此程序可以完美实现题目中的要求, 对 C-Minus 语言编写的程序识别扫描结果较为准确, 并且可以输出到文件中便于查看保存。

(2) 缺点分析: 程序编写较为冗长, 可读性较差, 不能一次识别全部正规式

(3) 改进方法: 扩充邻接表, 将程序所能容纳的正则表达式字符数增加

4) 使用手册:

a) 运行程序, 根据提示输入正则表达式

b) 等待程序运行读取包含输入串的文件

c) 输入计算的结果要保存的路径文件, 结果即保存到该文件中

5) 测试结果:

输入测试数据 (输入串保存在 Test.txt 中);

输入正则表达式:

```
在下面输入正则表达式
else|if|int|return|void|while_
```

输入保存路径名:

```
请输入保存记录的文件
out.txt

查找结果已保存在out.txt中

-----
请按任意键继续. . .
```

最终结果如下:

```
-----
第一步: 加入连结点
e.l.s.e|i.f|i.n.t|r.e.t.u.r.n|v.o.i.d|w.h.i.l.e
字符串长度: 47
```

```
-----
第二步: 转为后缀式
e.l.s.e.if.|in.t.|re.t.u.r.n.|vo.i.d.|wh.i.l.e.|
字符串长度: 47
```

```
-----
第三步: 获取字符集
e l s i f n t r u v o d w h
字符个数: 14
```

```
-----
第四步: 构造NFA
```

状态	边(权值)
0	57(~)
1	2(e)
2	3(~)
3	4(1)

```

3      4(l)
4      5(~)
5      6(s)
6      7(~)
7      8(e)
8      14(~)
9      10(i)
10     11(~)
11     12(f)
12     14(~)
13     1(~) 9(~)
14     22(~)
15     16(i)
16     17(~)
17     18(n)
18     19(~)
19     20(t)
20     22(~)
21     13(~) 15(~)
22     36(~)
23     24(r)
24     25(~)
25     26(e)
26     27(~)
27     28(t)
28     29(~)
29     30(u)
30     31(~)
31     32(r)
32     33(~)
33     34(n)
34     36(~)
35     21(~) 23(~)
36     46(~)
37     38(v)
38     39(~)
39     40(o)
40     41(~)
41     42(i)
42     43(~)

```

```

43     44(d)
44     46(~)
45     35(~) 37(~)
46     58(~)
47     48(w)
48     49(~)
49     50(h)
50     51(~)
51     52(i)
52     53(~)
53     54(l)
54     55(~)
55     56(e)
56     58(~)
57     45(~) 47(~)
58     END

```

第五步：构造DStates表

```

项目0:  0 57 45 47 35 37 21 23 13 15 1 9
项目1:  2 3
项目2:  16 17 10 11
项目3:  24 25
项目4:  38 39
项目5:  48 49
项目6:  4 5
项目7:  12 14 22 36 46 58
项目8:  18 19
项目9:  26 27
项目10: 40 41
项目11: 50 51
项目12: 6 7
项目13: 20 22 36 46 58
项目14: 28 29
项目15: 42 43
项目16: 52 53
项目17: 8 14 22 36 46 58

```

```
项目17: 8 14 22 36 46 58
项目18: 30 31
项目19: 44 46 58
项目20: 54 55
项目21: 32 33
项目22: 56 58
项目23: 34 36 46 58

-----

第六步: 构造Dtran表

状态  e    l    s    i    f    n    t    r    u    v    o    d    w    h    是否接收
0     1                                3     4     5
1                                     6
2                                     7    8
3     9
4                                     10
5                                     11
6                                     12
7                                     13
8                                     14
9                                     15
10                                    16
11     17
12
13                                     18
14                                     19
15
16     20
17                                     21
18
19                                     22
20     22
21                                     23
22
23                                     Acc
                                     Acc

-----
```

```
第七步: 构造最小DFA

状态  e    l    s    i    f    n    t    r    u    v    o    d    w    h    是否接收
0     1                                3     4     5
1                                     6
2                                     7    8
3     9
4                                     10
5                                     11
6                                     12
7                                     13
8                                     14
9                                     15
10                                    16
11     7
12                                     17
13
14     12
15
16                                     7
17                                     17

-----

第八步: 查找匹配字符串

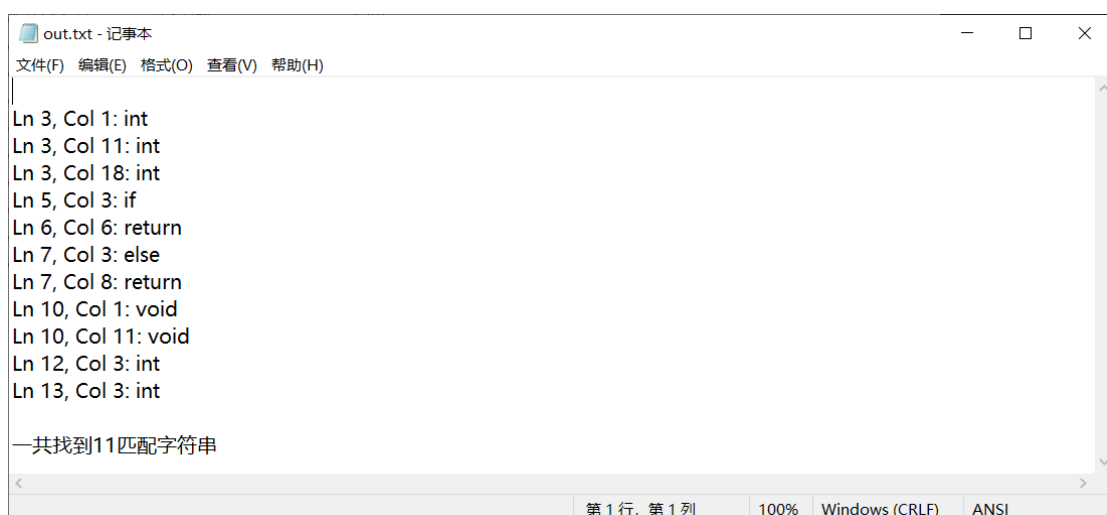
请输入保存记录的文件
out.txt

查找结果已保存在out.txt中

-----

请按任意键继续. . .
```

out.txt 中:



```
Ln 3, Col 1: int
Ln 3, Col 11: int
Ln 3, Col 18: int
Ln 5, Col 3: if
Ln 6, Col 6: return
Ln 7, Col 3: else
Ln 7, Col 8: return
Ln 10, Col 1: void
Ln 10, Col 11: void
Ln 12, Col 3: int
Ln 13, Col 3: int

一共找到11匹配字符串
```

第 1 行, 第 1 列 100% Windows (CRLF) ANSI

五、 总结

编译原理是一门很重要的课程。我们平常写小的 C 语言程序会感到困难，而编译原理则是关于编写编译器的技术，难度之大可想而知。在整个程序中，我认为其中逆波兰式的构造和自动机的构造都是比较难的，是通过对基础知识的详细深入回顾得出，特别是自动机的确定化过程烦琐又耗时，采用了邻接表进行存储。上机的调试时间也不是很长，但程序的最大问题就是可读性不强，虽基本功能能达到理想的效果。本人对此次课程设计最大的感受理论是实践的基础。

六、 致谢

感谢刘老师几个月以来的辛勤付出与谆谆教导，编译原理这门课让我受益匪浅，懂得了很多底层编译知识，让我能在以后的学习中得到很大帮助，希望以后有机会能与您合作。