|  |  |  |
| --- | --- | --- |
| **得分** | **教师签名** | **批改日期** |
|  |  |  |

课程编号： IB00064



**深圳技术大学课程设计报告**

**课程名称： 编译原理**

**实验名称： 课程设计--LR(1)分析法**

**班 级： 计算机科学与技术2班**

**指导教师： 袁明川、赵鹏**

**报 告 人： 黄荣权 学号： 202002020213**

**合 作 者： 组号：**

**实验地点： C5-359**

**实验时间： 2022 年 06 月 21 日 星期 二**

**提交时间： 2022/06/21**

# 设计目的与要求

构造LR(1)分析程序，利用它进行语法分析，判断给出的符号串是否为该文法识别的句子，了解LR（K）分析方法是严格的从左向右扫描和自底向上的语法分析方法。

# 设计内容与方法

1、对下列文法，用LR（1）分析法对任意输入的符号串进行分析：

（0）E->S

（1）S->BB

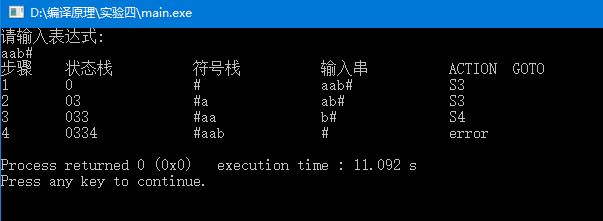
（2）B->aB

（3）B->b

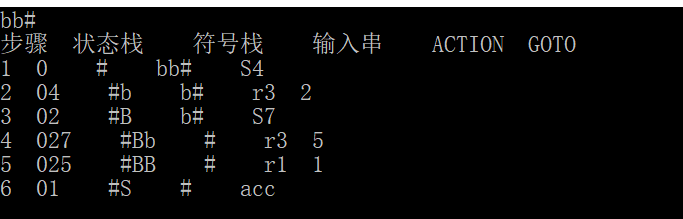
2、LR(1)分析表为：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 状态 | ACTION | | | GOTO | |
| a | b | # | S | B |
| S0 | S3 | S4 |  | 1 | 2 |
| S1 |  |  | acc |  |  |
| S2 | S6 | S7 |  |  | 5 |
| S3 | S3 | S4 |  |  | 8 |
| S4 | r3 | r3 |  |  |  |
| S5 |  |  | r1 |  |  |
| S6 | S6 | S7 |  |  | 9 |
| S7 |  |  | r3 |  |  |
| S8 | r2 | r2 |  |  |  |
| S9 |  |  | r2 |  |  |

1. 输入字符串aab#，则输出为：

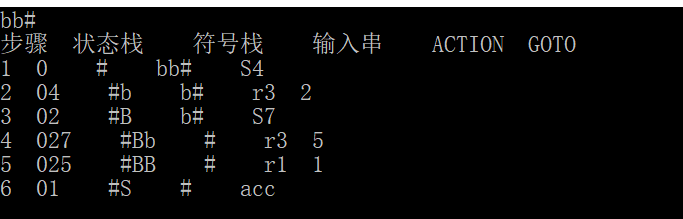


(2)若输入bb#，则输出为：



# 步骤与过程

**（1）根据样例分析输出的表项。**



①第一个是步骤计数器，从1开始；

②第二个是状态栈，从0开始；

③第三个是符号栈，以“#”开始；

④第四个是输入串，就是输入的表达式；

⑤第五个是ACTION GOTO，是当前输入串对应的解析表值。

**（2）以输入表达式“aab#”为例演绎实现流程。**

①状态栈和符号栈均为默认状态，分别为“0”和“#”，先打印输出串后pop第一个字符。

****

查阅LR(1)分析表可知此时的ACTION为S3。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 状态 | ACTION | | | GOTO | |
| a | b | # | S | B |
| S0 | S3 | S4 |  | 1 | 2 |

②状态栈push进入上一状态ACTION中的Si的i；

符号栈push进入上一状态输入进的字符a，打印输出串的首字母a后将其pop，此时字符变为b#；



输入串变为ab#，首字符依旧是a，此时到状态3寻找对应a的状态，所以此时ACTION依旧是S3。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 状态 | ACTION | | | GOTO | |
| a | b | # | S | B |
| S0 | S3 | S4 |  | 1 | 2 |
| S1 |  |  | acc |  |  |
| S2 | S6 | S7 |  |  | 5 |
| S3 | S3 | S4 |  |  | 8 |

③状态栈push进上一步骤ACTION的Si中的i--S3；

符号栈push进上一输入串的首字符a，此时变为#aa；

打印输出串的首字母b后将其pop，此时字符变为#；



当前状态依然为S3，在表中查找b对应的状态可知ACTION变为S4。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 状态 | ACTION | | | GOTO | |
| a | b | # | S | B |
| S0 | S3 | S4 |  | 1 | 2 |
| S1 |  |  | acc |  |  |
| S2 | S6 | S7 |  |  | 5 |
| S3 | S3 | S4 |  |  | 8 |

④状态栈push进上一步骤ACTION的Si中的i--S4；

符号栈push进上一输入串的首字符b，此时变为#aab；

输入串pop出当前首字符#（先打印后pop）

打印输出串的首字母#后将其pop，



当前状态为S4，在表中查询得到为空，输出error。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 状态 | ACTION | | | GOTO | |
| a | b | # | S | B |
| S0 | S3 | S4 |  | 1 | 2 |
| S1 |  |  | acc |  |  |
| S2 | S6 | S7 |  |  | 5 |
| S3 | S3 | S4 |  |  | 8 |
| S4 | r3 | r3 |  |  |  |

**（3）根据（2）的演绎进行代码实现设计。**

①需要保存终结符、非终结符和产生式；

②需要保存ACTION表和GOTO表以供查询；

③需要两个栈用来保存状态和符号，一个字符串保存输入串；

④需要设计查询ACTION和GOTO值的功能函数，其中需要处理移进、归约；

**（4）代码实现。**

①定义终结符、非终结符以及产生式为全局变量和设计一些变量

char terminator[3] = {'a', 'b', '#'};                         //存放终结符

char non\_terminator[2] = {'S', 'B'};                          //存放非终结符

char \*generative[4] = {"E->S#", "S->BB#", "B->aB#", "B->b#"}; //存放产生式

int a[10];             // 分析栈

char b[10], c[10], c1;

int top1, top2, top3, top, m, n;

string s1;

②保存ACTION表和GOTO表供查询用

// ACTION表

char \*action[10][3] = {"S3#", "S4#", NULL,

                        NULL, NULL, "acc",

                        "S6#", "S7#", NULL,

                        "S3#", "S4#", NULL,

                        "r3#", "r3#", NULL,

                        NULL, NULL, "r1#",

                        "S6#", "S7#", NULL,

                        NULL, NULL, "r3#",

                        "r2#", "r2#", NULL,

                        NULL, NULL, "r2#"};

// GOTO表

\_\_attribute\_\_((unused)) vector<int> v1;

int goto1[10][2] = {1, 2,

                    0, 0,

                    0, 5,

                    0, 8,

                    0, 0,

                    0, 0,

                    0, 9,

                    0, 0,

                    0, 0,

                    0, 0};

③初始化变量，读取表达式，输出表单。

    int g, h, i, j, k, l, p, y, z, count;

    char x, copy[10], copy1[10];

    top1 = 0;

    top2 = 0;

    top3 = 0;

    top = 0; //左字符

    a[0] = 0;

    b[0] = '#';

    count = 0; //步骤计数

    z = 0;

    cout << "请输入表达式:" << endl;

    //读取表达式

    do

    {

        cin >> c1;

        c[top3] = c1;

        top3 = top3 + 1;

    } while (c1 != '#');

    //输出步骤、状态栈、符号栈、输入串、ACTION、GOTO表单

    cout << left << setw(12) << "步骤";

    cout << left << setw(12) << "状态栈";

    cout << left << setw(13) << "符号栈";

    cout << left << setw(14) << "输入串";

    cout << left << setw(10) << "ACTION";

    cout << left << setw(8) << "GOTO" << endl;

④打印输出序列号。

        y = z; // y,z指向状态栈栈顶

        m = 0;

        n = 0;

        g = top; // 输入的串长度

        j = 0;

        k = 0;

        x = c[top]; // 当前字符

        count++;

        // 打印序列号

        cout << left << setw(10) << count;

⑤输出状态栈、符号栈、输出串。

        //输出状态栈

        while (m <= top1)

        {

            s1.clear();

            cout << a[m];

            m = m + 1;

        }

        for (int u = m; u < 10; ++u)

        {

            cout << " ";

        }

        //输出符号栈

        while (n <= top2)

        {

            s1.push\_back(b[n]);

            n = n + 1;

        }

        cout << left << setw(10) << s1;

        s1.clear();

        //输出输入串

        while (g <= top3)

        {

            s1 += c[g];

            g = g + 1;

        }

        cout << left << setw(10) << s1;

        s1.clear();

⑥输出对应的action值和goto值。

//查询动作表

        switch (x)

        {

        case 'a':

            j = 0;

            break;

        case 'b':

            j = 1;

            break;

        case '#':

            j = 2;

            break;

        }

        // 当action表项为空

        if (action[y][j] == NULL)

        {

            cout << left << setw(10) << "error";

            getchar();

            getchar();

            return 0;

        }

        // 当action表项不为空

        else

        {

            strcpy(copy, action[y][j]);

        }

        //处理移进

        if (copy[0] == 'S')

        {

            z = copy[1] - '0'; // 状态栈栈顶元素，移除结尾符号

            top1 = top1 + 1;

            top2 = top2 + 1;

            a[top1] = z;

            b[top2] = x;

            top = top + 1;

            i = 0;

            while (copy[i] != '#')

            {

                s1 += copy[i];

                i++;

            }

            //输出对应的ACTION值

            cout << left << setw(10) << s1 << endl;

            s1.clear();

        }

        //处理归约

        else if (copy[0] == 'r')

        {

            i = 0;

            while (copy[i] != '#')

            {

                s1 += copy[i];

                i++;

            }

            cout << left << setw(10) << s1;

            s1.clear();

            //输出对应的goto值

            h = copy[1] - '0';

            strcpy(copy1, generative[h]);

            if (copy1[0] == 'S')

            {

                k = 0;

            }

            else if (copy1[0] == 'B')

            {

                k = 1;

            }

            l = strlen(generative[h]) - 4;

            top1 = top1 - l + 1;

            top2 = top2 - l + 1;

            y = a[top1 - 1];

            p = goto1[y][k];

            a[top1] = p;

            b[top2] = copy1[0];

            z = p;

            cout << left << setw(10) << p << endl;

        }

    } while (action[y][j] != "acc");

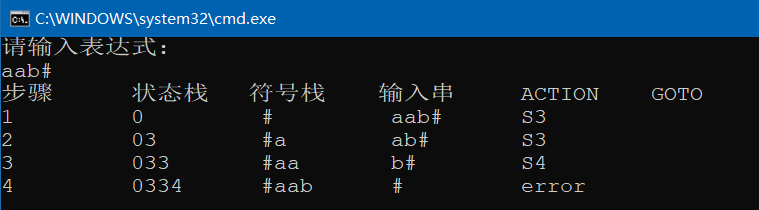
    cout << left << setw(10) << "acc" << endl;

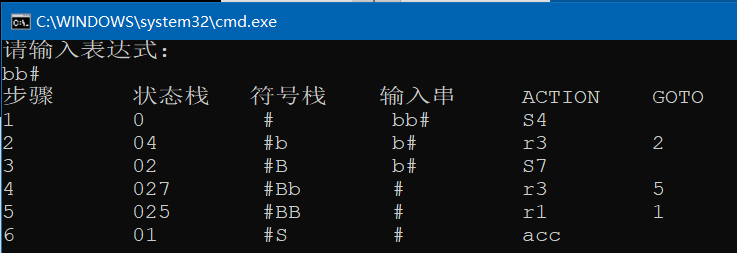
    getchar();

    getchar();

    return 0;

**（5）样例验证。**





# 结论或体会

在构造LR(1)分析程序得实验中，通过利用它进行语法分析，判断给出的符号串是否为该文法识别的句子，了解LR（K）分析方法是严格的从左向右扫描和自底向上的语法分析方法，加深了我对LR(1)分析法的理解运用。

1. **课程总结**

**查资料学习编译程序发展过程，指出课本中哪些内容你认为与其它计算机课程差异最大，并展望未来编译器（程序）可能的变化。（要求字数不少于800字）**

刚开始学习触编译原理这门课的体会是晦涩难懂和枯燥乏味，但随着持续地深入学习，逐渐构建起编译原理的知识体系，让我深入认识到这是一门介绍编译程序构造的一般原理和基本方法的课程。前七章的学习内容包括语言和文法、语法分析、自顶向下和自底向上的语法分析、语法制导翻译与属性文法、语义分析与中间代码生成，系统性地介绍了编译原理的基本概念、基本理论和基本方法。

我认为其中自顶向下和自底向上的语法分析的内容对其他计算机课程差异最大。这部分内容是编译程序的核心部分，通过检查词法分析器输出的单词序列是否构成源语言中的句子来判断是否符合源语言的语法规则。这部分内容让我了解了机器是如何具体实现语言的编译过程。同时自底向上的分析是从具体到抽象的过程；自顶向下的分析是从抽象到具体的过程，这两种分析思路恰恰又是哲学思考问题的两大方向。与其他计算机课程比如数据结构、算法设计与分析、面向对象程序设计只是对一个方法的理解和应用，编译原理最大的不同地地方在于这是一门追寻程序设计语言的本质、更深入理解程序和语言在底层的实现的课程。

编译原理及技术从本质上来讲就是一个算法问题，不过这个问题过于复杂而且其解决也非常复杂。相比数据结构与算法讲授的基础算法，编译原理比较专注的是解决问题本质的算法。掌握编译原理的思想，可以更加容易理解一种语言的哪些写法是等价的，哪些是有差异的，同时学习新的一门语言的效率也更高，在理解和运用如何优化程序运行也有更多手段。

在机器与软件的日益复杂化、多核处理器的介入以及对安全的关注的背景下，我认为能更好地实现程序优化、确保编译的正确和安全是未来编译器可能的变化。

在当今的多核时代，使时钟频率还能提高但也是非常缓慢的，因此要想改进应用程序的性能和能力，最关键的就是更好地利用大规模并行的硬件。这方面对于编译研究的选择可能就是采用自适应的编译策略尽可能地利用目标机器的潜力，使之能基于脚本代码语法和优化的潜能来发掘编译性能的提升和编译器结构的简化。

对于编译器而言，检查程序中的编码缺陷是一项重要任务。对于未来的编译器而言，在确保正确性和安全性下，检测工具必须在保证软件可靠性和安全性的前提下尽量减少误报，以避免浪费程序员的时间，而且应能在显示检测结果时排出优先顺序，并加以分类。这是未来编译器重要的变化方向。

|  |
| --- |
| 指导教师批阅意见： |
| 备注： |

注：1、教师批改学生实验报告时间应在学生提交实验报告时间后10日内。