



《编译技术》

实验一：词法分析程序编制

学 号 22920212204396

姓 名 黄子安

2024 年 4 月 10 日

实验一：词法分析程序编制

22920212204396 黄子安

一、实验目的

基本掌握计算机语言的词法分析程序的开发方法。

二、实验内容

编制一个能够分析三种整数、标识符、主要运算符和主要关键字的词法分析程序。

三、实验要求

1. 根据以下的正规式，编制正规文法，画出状态图；

基本要求：

单词类别	词法格式
标识符	<字母>(<字母> <数字符号>)*
十进制整数	(1 2 3 4 5 6 7 8 9)(0 1 2 3 4 5 6 7 8 9)* 0
八进制整数	0o(1 2 3 4 5 6 7)(0 1 2 3 4 5 6 7)*
十六进制整数	0x(0 1 2 3 4 5 6 7 8 9 a b c d e f)(0 1 2 3 4 5 6 7 8 9 a b c d e f)*
运算符和分隔符	+ - * / > < = () ;
关键字	if then else while do

附加要求：

单词类别	词法格式
标识符	<字母>(<字母> <数字符号>)*(ε _ .)(<字母> <数字字符>)*
十进制数	(1 2 3 4 5 6 7 8 9)(0 1 2 3 4 5 6 7 8 9)* 0(ε .(0 1 2 3 4 5 6 7 8 9)(0 1 2 3 4 5 6 7 8 9)*)
八进制数	0o(0 1 2 3 4 5 6 7)(0 1 2 3 4 5 6 7)*(ε .(0 1 2 3 4 5 6 7)(0 1 2 3 4 5 6 7)*)
十六进制数	0x(0 1 2 3 4 5 6 7 8 9 a b c d e f)(0 1 2 3 4 5 6 7 8 9 a b c d e f)* *(ε .(0 1 2 3 4 5 6 7 8 9 a b c d e f)(0 1 2 3 4 5 6 7 8 9 a b c d e f)*)

2. 根据状态图，设计词法分析函数 `int scan()`，完成以下功能：
 - 1) 从键盘读入数据，分析出一个单词。
 - 2) 返回单词种别（用整数表示），
 - 3) 返回单词属性（不同的属性可以放在不同的全局变量中）。
3. 编写测试程序，反复调用函数 `scan()`，输出单词种别和属性。

四. 实验环境

- 1、Windows 操作系统
- 2、GCC (C99) 编译器

五. 实验步骤

- 1、根据状态图，设计词法分析算法。

(1) 正规表达式

直接选择实现附加要求，对应的正规表达式即为实验要求所给出的内容：

单词类别	词法格式
标识符	<code><字母>(<字母> <数字符号>)*(ϵ $_$ $\.$)(<字母> <数字字符>)*</code>
十进制数	<code>(1 2 3 4 5 6 7 8 9)(0 1 2 3 4 5 6 7 8 9)*0(ϵ $\.$)(0 1 2 3 4 5 6 7 8 9)(0 1 2 3 4 5 6 7 8 9)*</code>
八进制数	<code>0o(0 1 2 3 4 5 6 7)(0 1 2 3 4 5 6 7)*(ϵ $\.$)(0 1 2 3 4 5 6 7)(0 1 2 3 4 5 6 7)*</code>
十六进制数	<code>0x(0 1 2 3 4 5 6 7 8 9 a b c d e f)(0 1 2 3 4 5 6 7 8 9 a b c d e f)*(ϵ $\.$)(0 1 2 3 4 5 6 7 8 9 a b c d e f)(0 1 2 3 4 5 6 7 8 9 a b c d e f)*</code>
运算符和分隔符	<code>+ - * / > < = () ;</code>
关键字	<code>if then else while do</code>

(2) 正规文法

字母表: $\{A \sim Z, a \sim z, 0 \sim 9, _ , ., \varepsilon, +, -, *, /, >, <, =, (,), ;, \}$

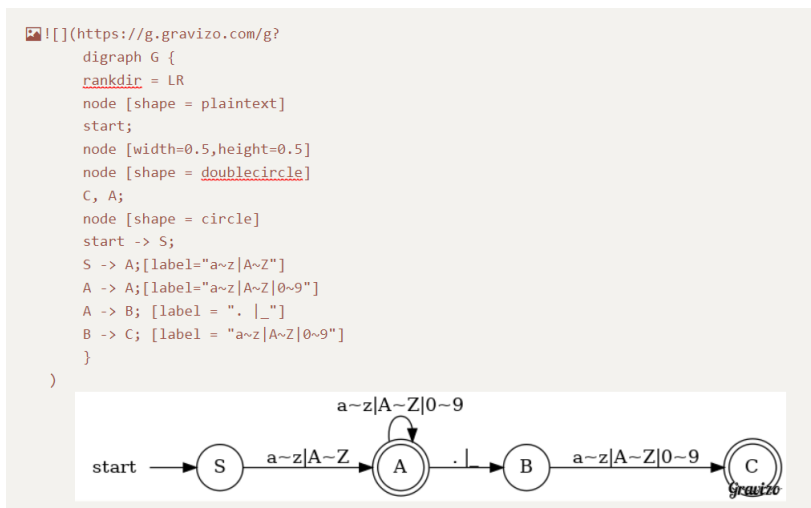
为了防止混淆状态表示字符与大写字母终结符, 约定:

$\langle \text{字母} \rangle \Sigma$ 表示 $a\Sigma | b\Sigma \dots z\Sigma | A\Sigma | \dots Z\Sigma$ 其中 Σ 为非终结符

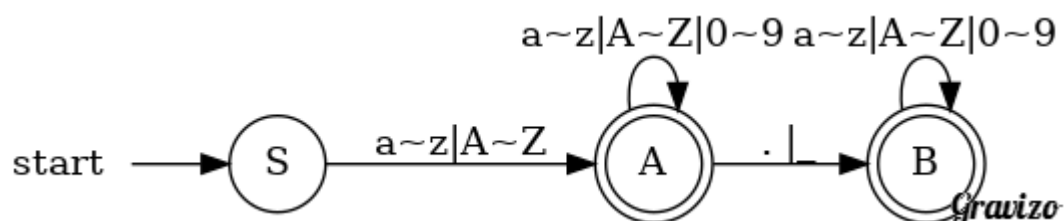
单词类型	正规文法
标识符	$S \rightarrow \langle \text{字母} \rangle A$ $A \rightarrow \langle \text{字母} \rangle A \langle \text{数字字符} \rangle A$ $A \rightarrow \langle \text{字母} \rangle B \langle \text{数字字符} \rangle B _ B . B$ $B \rightarrow \langle \text{字母} \rangle B \langle \text{数字字符} \rangle B \varepsilon$
十进制数	$S \rightarrow 0A \langle 1 \sim 9 \rangle B$ $A \rightarrow \varepsilon . C$ $B \rightarrow \varepsilon . C$ $B \rightarrow \langle 0 \sim 9 \rangle B$ $C \rightarrow \langle 0 \sim 9 \rangle D$ $D \rightarrow \langle 0 \sim 9 \rangle D \varepsilon$
八进制数	$S \rightarrow 0A$ $A \rightarrow oB OB$ $B \rightarrow \langle 0 \sim 7 \rangle C$ $C \rightarrow \langle 0 \sim 7 \rangle C$ $C \rightarrow \varepsilon . D$ $D \rightarrow \langle 0 \sim 7 \rangle E$ $E \rightarrow \varepsilon \langle 0 \sim 7 \rangle E$
十六进制	$S \rightarrow 0A$ $A \rightarrow xB XB$ $B \rightarrow \langle 0 \sim 9, a \sim f \rangle C$ $C \rightarrow \langle 0 \sim 9, a \sim f \rangle C$ $C \rightarrow \varepsilon . D$ $D \rightarrow \langle 0 \sim 9, a \sim f \rangle E$ $E \rightarrow \varepsilon \langle 0 \sim 9, a \sim f \rangle E$
运算符和分隔符	$S \rightarrow + - * / > < = () ;$
关键字	$S \rightarrow \text{if} \text{then} \text{else} \text{while} \text{do}$

(3) 状态图

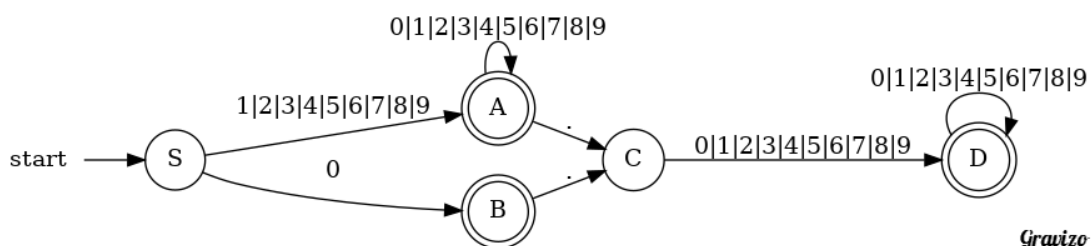
先分开绘制上述六种单词的状态图，之后进行汇总整合，从而进行判断，同时为了方便后续编写代码，直接绘制对应的确定有穷自动机 DFA，以下自动机图片中水印来自因为图片使用 Markdown 语法+Gravizo 绘制



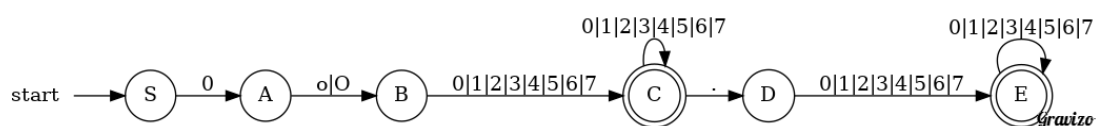
a.标识符（对于关键字将在识别标识符完成后进行特判）:



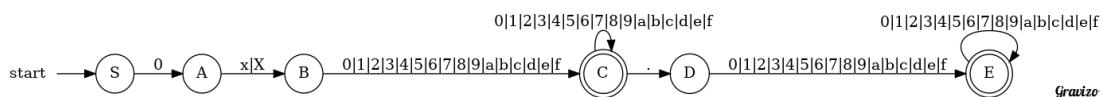
b.十进制数:



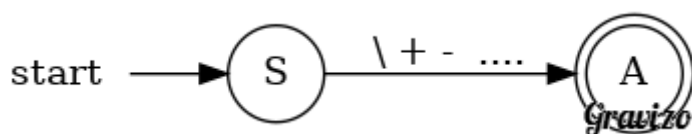
c.八进制数:



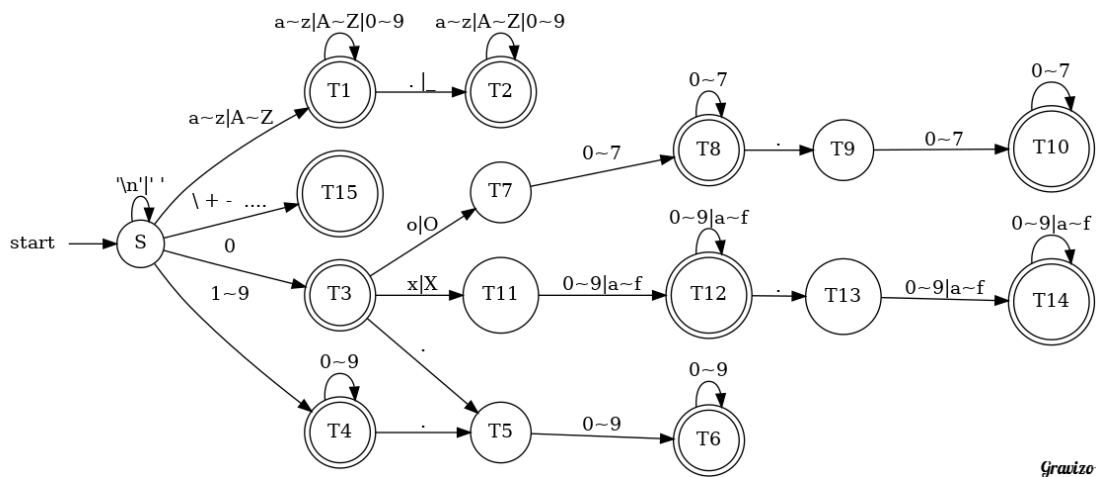
d. 十六进制数:



e. 运算符和分隔符



最后将其整合为一个总的 DFA, 适当合并冗余的节点, 同时加入对于空格字符和换行字符的处理, 此外其实有一个隐藏状态, 当给定状态在给定的某个条件下无法转移时将跳转到该隐藏状态, 结束本次状态转移



2、采用 C 语言，设计函数 scan(), 实现该算法。

函数 scan 主要任务就是根据当前状态和输入条件进行对应的状态转移，因为之前给出已经是 DFA，因此可以直接进行状态转移，对应的函数如下：

```
void scan(char s[], int len, int line)
{
    int j = 0, i = 0;
    while(s[j]){
        int T = 0, finished = 0;
        for(i = j; s[i]; ++i){
            int next_T = get_next(s[i], T);
            if(next_T == -2){
                finished = 1;
                break;
            }
            if(next_T == -1) {
                printf("\nComplie error in line %d , column %d:\n", line, i + 1);
                printf("| %s\n", s);
                for(int k = 0; k ≤ i + 1; ++k) putchar(' ');
                printf("^ \n\n");
                return;
            }
            else if(next_T == 0){
                T = next_T;
                ++j;
            }
            else T = next_T;
        }
        if(!s[i]) finished = 1;
        if(finished)
        {
            put_type(T, s, j, i - 1);
            j = i;
        }
    }
}
```

Scan 函数会依次扫描字符，同时进行状态的转移

- 对于结束状态如果没有转移回到自身便返回-2，代表结束状态转移，之后根据对应的状态类型进行输出
- 如果转移到-1 说明出现了不在转移矩阵中的情况，此时编译错误，进行对应的定位与报错
- 如果转移状态回到 0 只有一种情况，即对应的单词前面有多余的空格，这个时候会进行空格的过滤，避免影响后续的输出
- 如果状态还需要继续转移，就修改现态，之后继续进行转移

在 scan 函数执行完毕之后便会完成对字符串 s 的扫描与识别工作

之后是状态转移部分，根据前面的有穷自动机，可以确定转移矩阵，但是实验的时候发现自动机中有很多条件式有交集，不是很方便转换为转移矩阵，因此没有直接构造矩阵，而是以 switch-case 和 if 语的形式实现

```
int get_next(char c, int T) //return -1 means error, return -2 means terminal
{
    switch (T)
    {
        case 0:
            if(c == '0') return 3;
            else if(isdigit(c) && c != '0') return 4;
            else if(isupper(c) || islower(c)) return 1;
            else if(is_split(c)) return 15;
            else if(c == '\n' || c == ' ') return 0;
            break;

        case 1:
            if(isdigit(c) || isupper(c) || islower(c)) return 1;
            else if(c == '.' || c == '_') return 2;
            else return -2;
            break;

        case 2:
            if(isdigit(c) || isupper(c) || islower(c)) return 2;
            else return -2;
            break;

        case 3:
            if(c == 'o' || c == 'O') return 7;
            else if(c == 'x' || c == 'X') return 11;
            else if(c == '.') return 5;
            else return -2;
            break;

        case 4:
            if(isdigit(c)) return 4;
            else if(c == '.') return 5;
            else return -2;
            break;

        case 5:
            if(isdigit(c)) return 6; break;

        case 6:
            if(isdigit(c)) return 6;
            else return -2;
            break;

        case 7:
            if(c >= '0' && c <= '7') return 8;
            break;

        case 8:
            if(c == '.') return 9;
            else if(c >= '0' && c <= '7') return 8;
            else return -2;
            break;
    }
}
```



```

    case 9:
        if(c ≥ '0' && c ≤ '7') return 10; break;

    case 10:
        if(c ≥ '0' && c ≤ '7') return 10;
        else return -2; break;

    case 11:
        if((c ≥ '0' && c ≤ '9') || (c ≤ 'f' && c ≥ 'a')) return 12; break;

    case 12:
        if(c == '.') return 13;
        else if((c ≥ '0' && c ≤ '9') || (c ≤ 'f' && c ≥ 'a')) return 12;
        else return -2; break;

    case 13:
        if((c ≥ '0' && c ≤ '9') || (c ≤ 'f' && c ≥ 'a')) return 14; break;

    case 14:
        if((c ≥ '0' && c ≤ '9') || (c ≤ 'f' && c ≥ 'a')) return 14;
        else return -2;
        break;

    case 15:
        return -2; break;

    default:
        return -1; break;
}
return -1;
}

```

在每次状态转移到达终态之后，可以根据对应的终态编号进行类别输出

```

void put_type(int T, char s[], int i, int j)
{
    switch (T)
    {
        case 1:
        case 2: put_word(s, i, j);break;
        case 3:
        case 4: printf("INT10    "),put_INT(s, i, j, 10);break;
        case 6: printf("FLOAT10  "),put_FLOAT(s, i, j, 10);break;
        case 8: printf("INT8     "),put_INT(s, i, j, 8);break;
        case 10: printf("FLOAT8   "),put_FLOAT(s, i, j, 8);break;
        case 12: printf("INT16    "),put_INT(s, i, j, 16);break;
        case 14: printf("FLOAT16  "),put_FLOAT(s, i, j, 16);break;
        case 15: printf("%c\t _\n",s[i]);break;
    }
}

```

此外要进行关键字的特判:

```

        char keywords[][10] = {
            "if",
            "then",
            "else",
            "while",
            "do"
        };

void put_word(char str[], int i, int j)
{
    char sub[1024];
    strncpy(sub, str + i, j - i + 1);
    sub[j - i + 1] = '\0';

    int is_keyword = 0;
    for (int k = 0; k < sizeof(keywords) / sizeof(keywords[0]); k++) {
        if (strcmp(sub, keywords[k]) == 0) {
            is_keyword = 1;
            break;
        }
    }
    if (is_keyword) {
        for(int k = 0; k < strlen(sub); ++k) sub[k]=toupper(sub[k]);
        printf("%s",sub);
        for(int i = 8 - strlen(sub); i ≥ 0 ; --i) putchar(' ');
        printf("_\n");
    } else {
        printf("IDN      %s\n",sub);
    }
}

```

根据输出样例, 对于非十进制数要进行进制转换

```

void put_INT(char s[], int i, int j, int p)
{
    if(p ≠ 10) i += 2;
    int x = 0;
    for(int k = i; k ≤ j; ++k){
        x = x * p + s[k] - (isdigit(s[k]) ? '0' : ('a' - 10));
    }
    printf("%d\n",x);
}

void put_FLOAT(char s[], int i, int j, int p)
{
    char sub[2024];
    strncpy(sub, s + i, j - i + 1);
    sub[j - i + 1] = '\0';
    if(p = 10) printf("%s\n",sub);
    else{
        int idx = i + 2;
        for(; idx ≤ j; ++idx){
            if(s[idx] == '.') break;
        }
        double x = 0, y = 0;
        for(int k = i + 2; k < idx; ++k){
            x = x * p + s[k] - (isdigit(s[k]) ? '0' : ('a' - 10));
        }
        for(int k = idx + 1; k ≤ j; ++k){
            y = y / p + s[k] - (isdigit(s[k]) ? '0' : ('a' - 10));
        }
        printf("%.2lf\n",x + y);
    }
}

```

3、编制测试程序（主函数 main）。

采用文件输入的方式进行快速测试

```
int main()
{
    FILE *fp;
    char filename[] = "test.txt";
    char buffer[2024];
    fp = fopen(filename, "r");
    if (fp == NULL) {
        printf("faile to open file:%s\n", filename);
        exit(1);
    }
    int line = 1;
    while(fgets(buffer, sizeof(buffer), fp) != NULL)
    {
        scan(buffer, strlen(buffer), line++);
    }
    fclose(fp);
    return 0;
}
```

4、调试程序：输入一组单词，检查输出结果。

六、基本测试数据

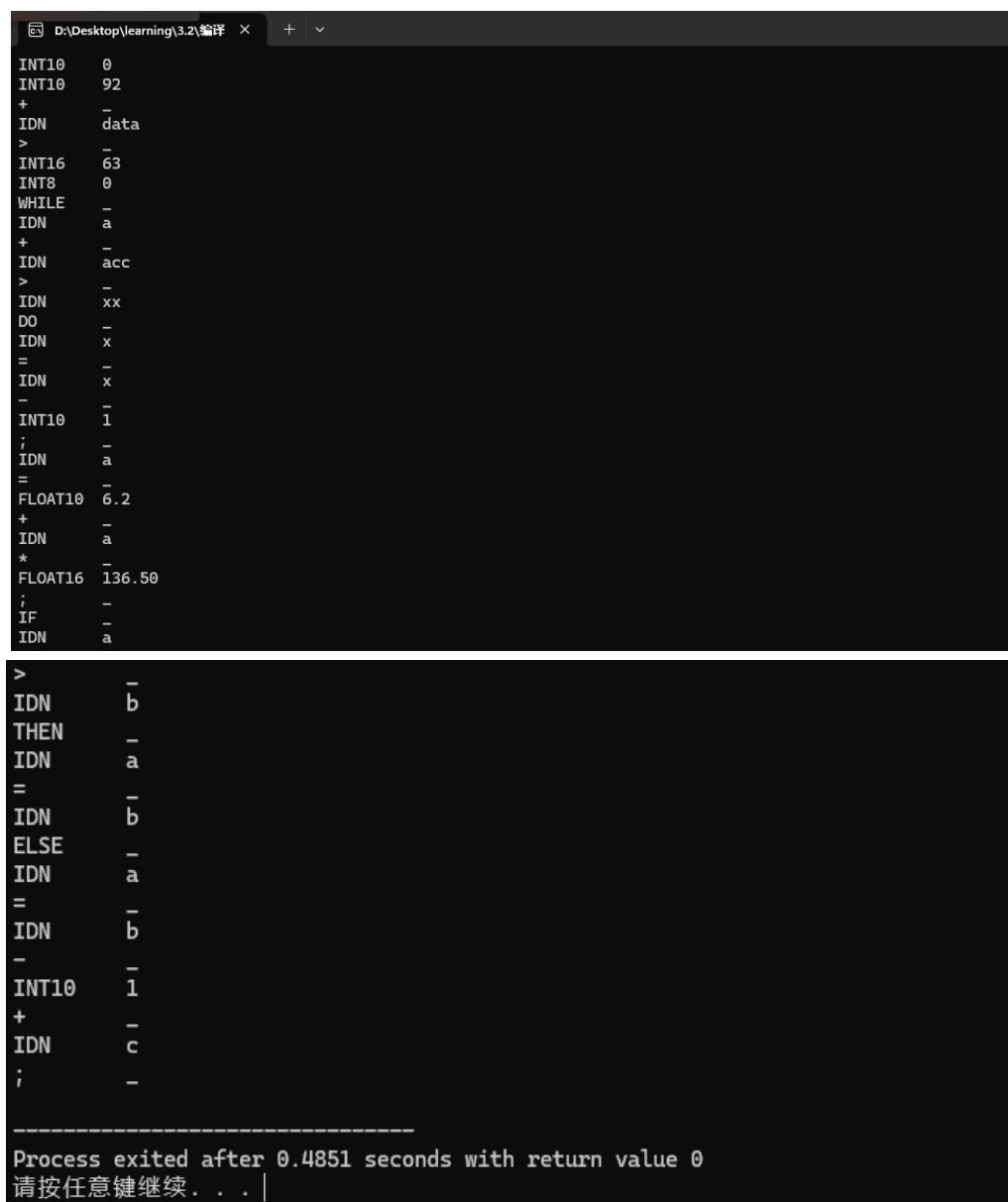
输入数据示例：

```
0 92+data> 0x3f 0o0 while a+acc>xx do x=x-1;
```

```
a=6.2+a*0X88.80;
```

```
if a>b then a=b else a=b-1+c;
```

最终的输出结果如下所示

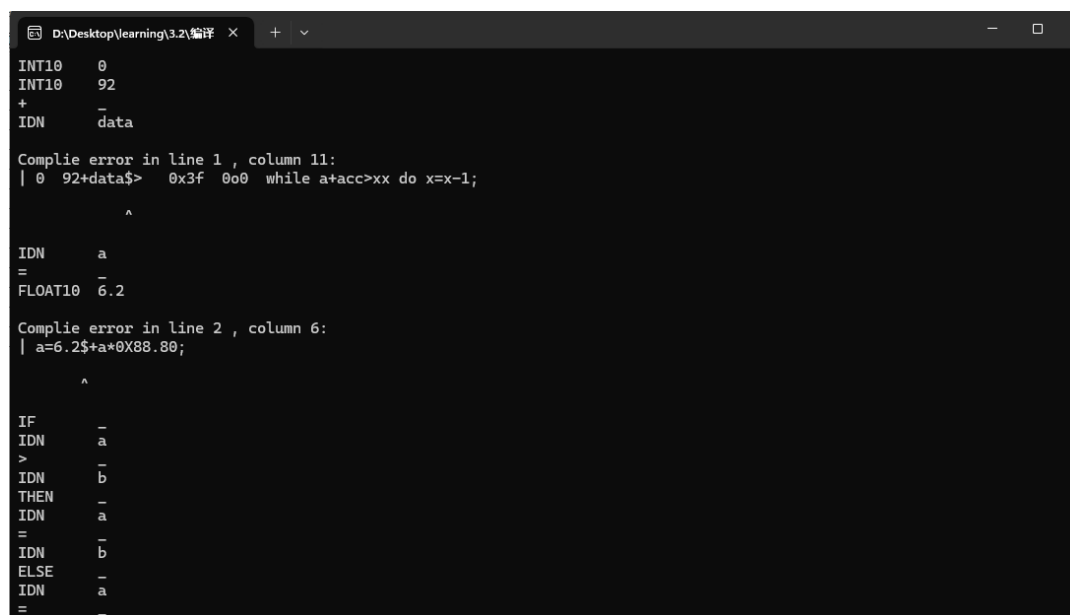


```
D:\Desktop\learning\3.2\编译 X + v
INT10 0
INT10 92
+ -
IDN data
> -
INT16 63
INT8 0
WHILE -
IDN a
+ -
IDN acc
> -
IDN xx
DO -
IDN x
= -
IDN x
- -
INT10 1
; -
IDN a
= -
FLOAT10 6.2
+ -
IDN a
* -
FLOAT16 136.50
; -
IF -
IDN a

> -
IDN b
THEN -
IDN a
= -
IDN b
ELSE -
IDN a
= -
IDN b
- -
INT10 1
+ -
IDN c
; -

-----
Process exited after 0.4851 seconds with return value 0
请按任意键继续. . . |
```

此外程序中进行了一定的出错定位与显示，将测试数据进行修改运行后结果如下所示：



```
D:\Desktop\learning\3.2\编译 x + v
INT10 0
INT10 92
+ -
IDN data

Complie error in line 1 , column 11:
| 0 92+data$> 0x3f 0o0 while a+acc>xx do x=x-1;
      ^

IDN a
= -
FLOAT10 6.2

Complie error in line 2 , column 6:
| a=6.2$a+a*0X88.80;
      ^

IF -
IDN a
> -
IDN b
THEN -
IDN a
= -
IDN b
ELSE -
IDN a
= -
```

七、思考题

1. 词法分析能否采用空格来区分单词？

词法分析可以使用空格来区分单词，空格在大多数情况下被视为单词之间的分隔符。然而，在某些情况下空格可能不会被视为分隔符，比如在双引号内的字符串中的空格通常不会被视为单词的分隔符，这涉及到了语义分析

2. 程序设计中哪些环节影响词法分析的效率？如何提高效率？

- **单词的数量和复杂度：**单词的数量越多，分析器需要处理的可能性就越多，从而影响效率。
- **正则表达式的复杂度：**词法分析器通常使用正则表达式来识别单词，复杂的正则表达式会导致分析器性能下降。
- **数据结构的选择：**本次实验中存在大量的 `switch` 和 `if` 语句，对于现代指令流水线架构 CPU 过多的跳转语句会导致流水线出现气泡，使用转移矩阵配合 `Cache` 可以实现更高的效率

要提高词法分析的效率，可以采取以下措施：

- **优化正则表达式：**简化正则表达式或使用更高效的匹配算法。

- **选择合适的数据结构：**根据实际需求选择最适合的数据结构，以提高查找和存储的效率。
- **实现算法优化：**对词法分析算法进行优化，减少不必要的操作和重复计算。
- **使用缓存：**利用缓存来存储已经处理过的结果，避免重复计算。
- **并行处理：**将文本分成多个部分并行处理，以提高处理速度。

本次博客地址：[【编译技术】实验一：词法分析程序编制-CSDN 博客](#)