2. 词法分析与语法分析

本章实验为**实验一**，任务是编写一个程序对使用C--语言书写的源代码进行词法分析和语法分析（C--语言的文法参见附录A），并打印分析结果。实验要求使用词法分析工具GNU Flex和语法分析工具GNU Bison，并使用C语言来完成。在两个强大工具的帮助下，编写一个能进行词法分析和语法分析的程序是一件相当轻松愉快的事情。

需要注意的是，由于在后面的实验中还会用到本次实验已经写好的代码，因此保持一个良好的代码风格、系统地设计代码结构和各模块之间的接口对于整个实验来讲相当重要。

2.1 实验内容

2.1.1 实验要求

你的程序要能够查出C--源代码中可能包含的下述几类错误：

1) **词法错误（错误类型A）：**即出现C--词法中未定义的字符以及任何不符合C--词法单元定义的字符；

2) **语法错误（错误类型B）。**

除此之外，你的程序可以选择完成以下部分或全部的要求：

1) **要求1.1：**识别八进制数和十六进制数。若输入文件中包含符合词法定义的八进制数（如0123）和十六进制数（如0x3F），你的程序需要得出相应的词法单元；若输入文件中包含不符合词法定义的八进制数（如09）和十六进制数（如0x1G），你的程序需要给出输入文件有词法错误（即错误类型A）的提示信息。八进制数和十六进制数的定义参见附录A。

2) **要求1.2：**识别指数形式的浮点数。若输入文件中包含符合词法定义的指数形式的浮点数（如1.05e-4），你的程序需要得出相应的词法单元；若输入文件中包含不符合词法定义的指数形式的浮点数（如1.05e），你的程序需要给出输入文件有词法错误（即错误类型A）的提示信息。指数形式的浮点数的定义参见附录A。

3) **要求1.3：**识别“//”和“/*…*/”形式的注释。若输入文件中包含符合定义的“//”和“/*…*/”形式的注释，你的程序需要能够滤除这样的注释；若输入文件中包含不符合定义的注释（如“/*…*/”注释中缺少“/*”），你的程序需要给出由不符合定义的注释所引发的错误的提示信息。注释的定义参见附录A。

你的程序在输出错误提示信息时，需要输出具体的错误类型、出错的位置（源程序行号）以及相关的说明文字。

2.1.2 输入格式

你的程序的输入是一个包含C--源代码的文本文件，程序需要能够接收一个输入文件名作为参数。例如，假设你的程序名为cc、输入文件名为test1、程序和输入文件都位于当前目录下，那么在Linux命令行下运行./cc test1即可获得以test1作为输入文件的输出结果。

2.1.3 输出格式

实验一要求通过标准输出打印程序的运行结果。对于那些包含词法或者语法错误的输入文件，只要输出相关的词法或语法有误的信息即可。在这种情况下，注意不要输出任何与语法树有关的内容。要求输出的信息包括错误类型、出错的行号以及说明文字，其格式为：

```
Error type [错误类型] at Line [行号]: [说明文字].
```

说明文字的内容没有具体要求，但是错误类型和出错的行号一定要正确，因为这是判断输出的错误提示信息是否正确的唯一标准。请严格遵守实验要求中给定的错误分类（即词法错误为错误类型A，语法错误为错误类型B），否则将影响你的实验评分。注意，输入文件中可能会包含一个或者多个错误（但输入文件的同一行中保证不出现多个错误），你的程序需要将这些错误全部报告出来，每一条错误提示信息在输出中单独占一行。

对于那些没有任何词法或语法错误的输入文件，你的程序需要将构造好的语法树按照先序遍历的方式打印每一个结点的信息，这些信息包括：

1) 如果当前结点是一个语法单元并且该语法单元没有产生ε（即空串），则打印该语法单元的名称以及它在输入文件中的行号（行号被括号所包围，并且与语法单元名之间有一个空格）。所谓某个语法单元在输入文件中的行号是指该语法单元产生出的所有词素中的第一个在输入文件中出现的行号。

2) 如果当前结点是一个语法单元并且该语法单元产生了ε，则无需打印该语法单元的信息。

3) 如果当前结点是一个词法单元，则只要打印该词法单元的名称，而无需打印该词法单元的行号。

a) 如果当前结点是词法单元ID，则要求额外打印该标识符所对应的词素；

b) 如果当前结点是词法单元TYPE，则要求额外打印说明以该类型为int还是float；

c) 如果当前结点是词法单元INT或者FLOAT，则要求以十进制的形式额外打印该数字所对应的数值；

d) 词法单元所额外打印的信息与词法单元名之间以一个冒号和一个空格隔开。

每一条词法或语法单元的信息单独占一行，而每个子结点的信息相对于其父结点的信息来说，在行首都要求缩进2个空格。具体输出格式可参见后续的样例。

2.1.4 测试环境

你的程序将在如下环境中被编译并运行：

1) GNU Linux Release: Ubuntu 20.04, kernel version 5.13.0-44-generic；
2) GCC version 7.5.0；
3) GNU Flex version 2.6.4；
4) GNU Bison version 3.0.4。

一般而言，只要避免使用过于冷门的特性，使用其它版本的Linux或者GCC等，也基本上不会出现兼容性方面的问题。我们建议你使用上述版本的工具，以避免在编译过程中产生不必要的问题，你需要在实验报告中注明你所使用的工具版本。注意，实验一的检查过程中不会去安装或尝试引用各类方便编程的函数库（如glib等），因此请不要在你的程序中使用它们。

2.1.5 提交要求

实验一要求提交如下内容：

1) Flex、Bison以及C语言的可被正确编译运行的源程序。
2) 一份PDF格式的实验报告，内容包括：
   a) 你的程序实现了哪些功能？简要说明如何实现这些功能。清晰的说明有助于助教对你的程序所实现的功能进行合理的测试。
   b) 你的程序应该如何被编译？可以使用脚本、makefile或逐条输入命令进行编译，请详细说明应该如何编译你的程序。无法顺利编译将导致助教无法对你的程序所实现的功能进行任何测试，从而丢失相应的分数。
   c) 实验报告的长度不得超过三页！所以实验报告中需要重点描述的是你的程序中的亮点，是你认为最个性化、最具独创性的内容，而相对简单的、任何人都可以做的内容则可不提或简单地提一下，尤其要避免大段地向报告里贴代码。实验报告中所出现的最小字号不得小于五号字（或英文11号字）。

2.1.6 样例（必做内容）

实验一的样例包括**必做内容样例**与**选做要求样例**两部分，分别对应于实验要求中的必做内容和选做要求。请仔细阅读样例，以加深对实验要求以及输出格式要求的理解。这节列举必做内容样例。

样例1:

输入（行号是为标识需要，并非样例输入的一部分，后同）：

```
1  int main()
2  {
3    int i = 1;
4    int j = ~i;
5  }
```

输出:

这个程序存在词法错误。第4行中的字符“~”没有在我们的C--词法中被定义过，因此你的程序可以输出如下的错误提示信息:

```
Error type A at Line 4: Mysterious character "~".
```

样例2:

输入:

```
1  int main()
2  {
3    float a[10][2];
4    int i;
5    a[5,3] = 1.5;
6    if (a[1][2] == 0) i = 1 else i = 0;
7  }
```

输出:

这个程序存在两处语法错误。其一，虽然我们的程序中允许出现方括号与逗号等字符，但二维数组正确的访问方式应该是a[5][3]而非a[5,3]。其二，第6行的if-else语句在else之前少了一个分号。因此你的程序可以输出如下的两行错误提示信息:

```
Error type B at Line 5: Missing "]".
Error type B at Line 6: Missing ";".
```

样例3:

输入:

```
1  int inc()
2  {
3    int i;
4    i = i + 1;
5  }
```

输出:

这个程序非常简单，也没有任何词法或语法错误，因此你的程序需要输出如下的语法树结点信息（行号是为标识需要，并非程序输出的一部分，后同）：

```
 1  Program (1)
 2    ExtDefList (1)
 3      ExtDef (1)
 4        Specifier (1)
 5          TYPE: int
 6        FunDec (1)
 7          ID: inc
 8          LP
 9          RP
10        CompSt (2)
11          LC
12          DefList (3)
13            Def (3)
14              Specifier (3)
15                TYPE: int
16              DecList (3)
17                Dec (3)
18                  VarDec (3)
19                    ID: i
20              SEMI
21          StmtList (4)
22            Stmt (4)
23              Exp (4)
24                Exp (4)
25                  ID: i
26                ASSIGNOP
27                Exp (4)
28                  Exp (4)
29                    ID: i
30                  PLUS
31                  Exp (4)
32                    INT: 1
33              SEMI
34          RC
```

样例4:

输入:

```
1  struct Complex
2  {
3    float real, image;
4  };
5  int main()
6  {
7    struct Complex x;
8    y.image = 3.5;
9  }
```

输出:

这个程序虽然包含了语义错误（即使用了未定义的变量y），但不存在任何词法或语法错误，因此你的程序不能报错而是要输出相应的语法树结点信息。至于把该语义错误检查出来的任务，我们则放到实验二中去做。本样例输入所对应的正确输出应为:

```
1  Program (1)
2    ExtDefList (1)
3      ExtDef (1)
4        Specifier (1)
```

```
 5        StructSpecifier (1)
 6          STRUCT
 7          OptTag (1)
 8            ID: Complex
 9          LC
10          DefList (3)
11            Def (3)
12              Specifier (3)
13                TYPE: float
14              DecList (3)
15                Dec (3)
16                  VarDec (3)
17                    ID: real
18                COMMA
19                DecList (3)
20                  Dec (3)
21                    VarDec (3)
22                      ID: image
23              SEMI
24          RC
25      SEMI
26    ExtDefList (5)
27      ExtDef (5)
28        Specifier (5)
29          TYPE: int
30        FunDec (5)
31          ID: main
32          LP
33          RP
34        CompSt (6)
35          LC
36          DefList (7)
37            Def (7)
38              Specifier (7)
39                StructSpecifier (7)
40                  STRUCT
41                  Tag (7)
42                    ID: Complex
43              DecList (7)
44                Dec (7)
45                  VarDec (7)
46                    ID: x
47              SEMI
48          StmtList (8)
49            Stmt (8)
50              Exp (8)
51                Exp (8)
52                  Exp (8)
53                    ID: y
54                  DOT
55                  ID: image
56                ASSIGNOP
57                Exp (8)
58                  FLOAT: 3.500000
59              SEMI
60          RC
```

2.1.7 样例（选做要求）

这节列举选做要求样例。

样例1:

输入:

```
1  int main()
```

```
2  {
3    int i = 0123;
4    int j = 0x3F;
5  }
```

输出：

如果你的程序需要完成要求1.1，该样例输入不包含任何词法或语法错误，其对应的输出为：

```
1  Program (1)
2    ExtDefList (1)
3      ExtDef (1)
4        Specifier (1)
5          TYPE: int
6        FunDec (1)
7          ID: main
8          LP
9          RP
10       CompSt (2)
11         LC
12         DefList (3)
13           Def (3)
14             Specifier (3)
15               TYPE: int
16             DecList (3)
17               Dec (3)
18                 VarDec (3)
19                   ID: i
20                 ASSIGNOP
21                 Exp (3)
22                   INT: 83
23             SEMI
24           DefList (4)
25             Def (4)
26               Specifier (4)
27                 TYPE: int
28               DecList (4)
29                 Dec (4)
30                   VarDec (4)
31                     ID: j
32                   ASSIGNOP
33                   Exp (4)
34                     INT: 63
35               SEMI
36         RC
```

样例2：

输入：

```
1  int main()
2  {
3    int i = 09;
4    int j = 0x3G;
5  }
```

输出：

如果你的程序需要完成要求1.1，该样例程序中的“09”为错误的八进制数，其会因被识别为十进制整数“0”和“9”而引发语法错误；同样，“0x3G”为错误的十六进制数，其也

会因被识别为十进制整数“0”和标识符“x3G”而引发语法错误。因此你的程序可以输出如下的错误提示信息：

```
Error type B at Line 3: Missing ";".
Error type B at Line 4: Missing ";".
```

你的程序也可以直接将“09”和“0x3G”分别识别为错误的八进制数和错误的十六进制数，此时你的程序也可以输出如下的错误提示信息：

```
Error type A at Line 3: Illegal octal number '09'.
Error type A at Line 4: Illegal hexadecimal number '0x3G'.
```

样例3：

输入：

```
1  int main()
2  {
3    float i = 1.05e-4;
4  }
```

输出：

如果你的程序需要完成要求1.2，该样例程序不包含任何词法或语法错误，其对应的输出为：

```
 1  Program (1)
 2    ExtDefList (1)
 3      ExtDef (1)
 4        Specifier (1)
 5          TYPE: int
 6        FunDec (1)
 7          ID: main
 8          LP
 9          RP
10        CompSt (2)
11          LC
12          DefList (3)
13            Def (3)
14              Specifier (3)
15                TYPE: float
16              DecList (3)
17                Dec (3)
18                  VarDec (3)
19                    ID: i
20                  ASSIGNOP
21                  Exp (3)
22                    FLOAT: 0.000105
23              SEMI
24          RC
```

样例4：

输入：

```
1  int main()
2  {
3    float i = 1.05e;
4  }
```

输出：

如果你的程序需要完成要求1.2，该样例程序中的“1.05e”为错误的指数形式的浮点数，其会因被识别为浮点数“1.05”和标识符“e”而引发语法错误。因此你的程序可以输出如下的错误提示信息：

```
Error type B at Line 3: Syntax error.
```

你的程序也可以直接将“1.05e”识别为错误的指数形式的浮点数，此时你的程序也可以输出如下的错误提示信息：

```
Error type A at Line 3: Illegal floating point number "1.05e".
```

样例5：

输入：

```
1  int main()
2  {
3    // line comment
4    /*
5   block comment
6    */
7    int i = 1;
8  }
```

输出：

如果你的程序需要完成要求1.3，该样例程序不包含任何词法或语法错误，其对应的输出为：

```
 1  Program (1)
 2    ExtDefList (1)
 3      ExtDef (1)
 4        Specifier (1)
 5          TYPE: int
 6        FunDec (1)
 7          ID: main
 8          LP
 9          RP
10        CompSt (2)
11          LC
12          DefList (7)
13            Def (7)
14              Specifier (7)
15                TYPE: int
16              DecList (7)
17                Dec (7)
18                  VarDec (7)
19                    ID: i
20                  ASSIGNOP
21                  Exp (7)
22                    INT: 1
23              SEMI
24          RC
```

注意，助教检查你的程序的时候，所使用的测试用例中“//”注释的范围与“/*…*/”注释的范围不会重叠（以减少问题的复杂性），因此你的程序不需要专门处理“//”注释的范围与“/*…*/”注释的范围相重叠的情况。

样例6:

输入:

```
 1  int main()
 2  {
 3    /*
 4    comment
 5    /*
 6    nested comment
 7    */
 8    */
 9    int i = 1;
10  }
```

输出:

样例输入中程序员主观上想使用嵌套的“/*…*/”注释，但C--语言不支持嵌套的“/*…*/”注释。如果你的程序需要完成要求1.3，该样例输入中第8行的“*/”会因被识别为乘号“*”和除号“/”而引发语法错误（你只需为每行报告一个语法错误），你的程序可以输出如下的错误提示信息:

```
Error type B at Line 8: Syntax error.
```