编译原理实验报告

实验题目：		学号：
日期：	班级：	姓名：
Email：		

实验目的：

你需要完成一个简单的C++风格的编译器。这个实验分为若干部分。在本实验中，需要完成该编译器的词法分析器。

实验环境介绍：

硬件环境：

Legion R7000P2021 AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz

软件环境：

Clion

解决问题的主要思路：

本实验中将会给定C++语言风格的源程序，我们需要将其从字符流转换为词语流，同时在转换过程中还需判断是否有词法分析错误，如果没有的话，最后输出单词流以及每个单词的类型即可。

本实验解决过程分以下几个步骤：

1. 由于题目中要求我们设计的编译器不应当对用户输入程序进行假设，故我们采用getline的方式进行读入。

2. 在读入完所有程序后，先对其进行预处理，即删除注释、过滤空白符等。

3. 之后开始识别关键字、标识符、数字以及运算符，这里设置三种状态（后面有详细介绍），通过不断识别关键字，从而完成整个词法分析过程。

4. 如果在识别的过程中发现了错误，那么直接结束即可。如若没有的话，在最后输出单词流以及每个单词的类型。

实验步骤：

在本实验中，错误有以下四种：

错误描述	输出内容
浮点数中不止一个小数点	Malformed number: More than one decimal point in a floating point number.
小数点在浮点数的开始或者结尾	Malformed number: Decimal point at the beginning or end of a floating point number.
整数或小数的整数部分中有前导零	Malformed number: Leading zeros in an integer.
非注释部分有不能识别的字符,包括单独出现一次的&,| 字符	Unrecognizable characters.

注意：前三种数字错误类型的输出优先级为 1>> 2 >> 3。

关键字表如下：

关键字	对应类型
int	INTSYM
double	DOUBLESYM
scanf	SCANFSYM
printf	PRINTFSYM
if	IFSYM
then	THENSYM
while	WHILESYM
do	DOSYM

符号表如下：

关键字	对应类型
=	AO
==	RO
>	RO
>=	RO
<	RO
<=	RO
||	LO
&&	LO
!	LO
!=	RO
+	PLUS
-	MINUS
*	TIMES
/	DIVISION

,	COMMA
(	BRACE
)	BRACE
{	BRACE
}	BRACE
;	SEMICOLON

标识符以及数字
标识符应符合如下文法：
< 标识符 > -> < 字母 >{< 字母 >|< 数字 >}
{}:出现 0 次或多次

标识符或者数字	对应类型
标识符（如 x）	IDENT
整数（如 3）	INT
小数（如 2.1）	DOUBLE

对于注释，与 C++类似，有两种方式，如下表所示：

//	单行注释
/* */	多行注释

假如只有/*而没有匹配的*/，则认为从/*往后的内容都是注释。
在词法分析中，注释的内容不应输出，即需要将这些内容过滤掉。

1.首先对符号和关键字进行抽象，根据关键字表和符号表将关键字和符号进行一一映射，如下所示：

```
// 关键字
const map<string,string> mp = {
        {"int","INTSYM"},{"double","DOUBLESYM"},{"scanf","SCANFSYM"},
        {"printf","PRINTFSYM"},{"if","IFSYM"},{"then","THENSYM"},
        {"while","WHILESYM"},{"do","DOSYM"}
};

// 符号
const map<string,string> mp1 = {
        {"=","AO"},{"==","RO"},{">","RO"},{">=","RO"},
        {"<","RO"},{"<=","RO"},{"||","LO"},{"&&","LO"},
        {"!","LO"},{"!=","RO"},{"+","PLUS"},{"-","MINUS"},
        {"*","TIMES"},{"/","DIVISION"},{",","COMMA"},{"(","BRACE"},
        {")","BRACE"},{"{","BRACE"},{"}","BRACE"},{";","SEMICOLON"}
};
```

2. 定义状态，在读入 C++语言源程序的时候，存在三种状态，分别是 Begin，Readletter，Readnumber，如下所示：

```
enum status {
    Begin, // 初始状态，表示开始读入一个新的数字或单词
    Readletter, // 读字母
    Readnumber // 读数字，在该状态会读入数字
};
```

3. 词法分析器的设计

（1）首先要考虑的是输入问题，由于对用户输入程序不设限制，所以采用 while 不断读入，同时考虑到输入程序是以多行输入，故本实验中我们采用 while 加 getline 的方式配合读入程序，如下所示：

```
ios::sync_with_stdio(false);
string str1;
while (getline(cin, str1)) { // 不断读入字符
    str += str1; // 存入到str中
    str += '\n';// str末尾置空，否则最后一个元素很有可能无法正常输出
}
```

（2）接下来就是预处理的过程，根据实验内容可知，首先我们需要把程序中的注释全部去除，而注释又分为//和/* */两种。

针对//类型的注释，该类注释注释的是一行的内容，故我们先在字符串中找到//，然后在其后面查找到换行符，找到之后将中间内容全部删除即可，如下所示：

```
// 去除 //
while (str.find("//") != -1) {
    int start = str.find("//");
    int end = 0;
    for (int i = start + 2; i < str.size(); i++) {
        if (str[i] == '\n') {
            end = i;
            break;
        }
    }
    str.erase(start, end - start);
}
```

针对/* */类型的注释，我们先在程序中查找到/*，如若没有则无需处理该类注释。若有，则往后查找*/，并将中间所有内容全部删除，如下所示：

```
// 去除 /**/
while (str.find("/*") != -1) {
    int start = str.find("/*");
    int end = str.find("*/");
    if (end != -1)str.erase(start, end - start + 2);
    else {
        str.erase(start, str.size() - start);
        str += '\n';//如果只有/*，而没有*/，在最后加一个换行符
    }
}
```

去除之后，由于在输入程序的过程中涉及到回车换行等，所以还需将输入的字符串中的回车换行等全部去除，本实验中我们的处理方式是，将这些全部用空格替代，如下所示：

```
for (int i = 0; i < str.size(); i++)
    if (str[i] == '\t' || str[i] == '\r' || str[i] == '\n')
         str[i] = ' ';
```

（3）在预处理之后，就开始进入词法分析，在词法分析过程中，不断读出字符，并判断是否存在错误。首先定义一个 GetString 函数，该函数的作用是识别每一个单词以便在后面使用，具体如下：定义一个变量 state 来记录当前所在状态（即前面所说的 Begin，Readletter，Readnumber 三种状态），一个变量 token 代表识别出的单词。根据当前处于不同的状态，进行分析。

Begin 状态：每次读入的初始状态，首先判断当前正在读入的是字符还是数字，如果是字符就转换到 Readletter 状态，是数字就转换到 Readnumber 状态，如果都不是的话继续处于该状态，如果遇到无法识别的字符则输出错误，程序结束，如下所示：

```
            case Begin: // 每一个初始状态
                if (IsLetter(str[now])) { // 读字符
                    token += str[now];
                    state = Readletter;
```

```
                } else if (IsDigit(str[now]) || str[now] == '.') { // 读数字
                    token += str[now];
                    state = Readnumber;
                } else if (str[now] == '>' || str[now] == '<' || str[now] == '!'
|| str[now] == '=') {
                    token += str[now];
                    now++;
                    if (str[now] == '=') {
                        token += str[now];
                        now++;
                    }
                    return token;
                } else if (str[now] == '=' || str[now] == '&' || str[now] == '|')
{ // 这三个符号是可以重复被定义的
                    token += str[now];
                    now++;
                    if (str[now - 1] == str[now]) {
                        token += str[now];
                        now++;
                    }
                    return token;
                } else if (Single(str[now])) {
                    token += str[now];
                    now++;
                    return token;
                } else {
                    cout << "Unrecognizable characters.";
                    exit(0);
                }
                now++;
                break;
```

其中 IsLetter()函数是判断读入的是否为字符，如下所示：

```
// 判断是否是字符
bool IsLetter(char c) {
    if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || c == '_') return true;
    else return false;
}
```

IsDigit()函数是判断读入的是否为数字，如下所示：

```
// 判断是否是数字
bool IsDigit(char c) {
    if (c >= '0' && c <= '9') return true;
    else return false;
}
```

如果既不是字符也不是数字，则判断是否为两个运算符组成（>=、<=、!=）及可重复符号（=、&、|），如果都不是的话在判断是否为单运算符，即 Single()函数，如下所示：

```
// 判断是否有可能是单独符号
bool Single(char c) {
    if (c == '!' || c == '(' || c == ')' || c == '{' || c == '}' || c == ';' || c
== ',' || c == '/' || c == '+' ||
        c == '-' || c == '*' || c == '>' || c == '<' || c == '=')
        return true;
    else return false;
}
```

如果都不是，则当前单词无法识别，输出错误即可。

Readletter 状态:读字母状态，进入该状态后，只需判断当前读入的是否仍为数字或者字母即可，如果是的话，就继续读入，如下所示：

```
            case Readletter: // 读字母
                if (IsLetter(str[now]) || IsDigit(str[now])) // 如果还是字母或者数
字，接下往下读
                    token += str[now];
                else
                    return token;
                now++;
                break;
```

Readnumber 状态：读数字状态，进入该状态后，需判断当前读入的是否仍为数字或者.，如果是的话，就继续读入，如下所示：

```
            case Readnumber:
                if (IsDigit(str[now]) || str[now] == '.') // 判读是否是数字或者.
                    token += str[now];
                else
                    return token;
                now++;
```

（4）得到一个单词以后，需要检查该单词是否合法，如果合法，则需给出对应单词的类型，如果不合法，则输出错误。具体如下：先检查是否是关键字，接着检查是否为符号，如果都不是的话，那就是标识符或者数字。接着在判断是否为数字，如果是的话判断是否合法即可，不是数字则为标识符，如下所示：

```
if (tep.size() != 0 && tep[0] != ' ') {
       if (mp.find(tep) != mp.end()) {// 说明它是关键字
             ans.push_back(tep);
             ans1.push_back(mp.at(tep));
       } else if (mp1.find(tep) != mp1.end()) { // 说明它是符号
             ans.push_back(tep);
             ans1.push_back(mp1.at(tep));
       } else { // 说明是标识符或者数字
              if (tep.size() != 0 && tep[0] == '.') {// 说明第一个字符是小数点
                  cout << "Malformed number: Decimal point at the beginning or end
of a floating point number.";
                  exit(0);
```

```
            } else if (tep.size() != 0 && tep[0] >= '0' && tep[0] <= '9') { //
说明是数字
                int cnt = 0;//用于记录字符串有多少个小数点
                for (int i = 0; i < tep.size(); i++)
                    if (tep[i] == '.')
                        cnt++;
                if (cnt == 0) { // 说明是整数
                    if (tep[0] == '0' && tep.size() != 1) { // 说明有前导0
                        cout << "Malformed number: Leading zeros in an
integer.";
                        exit(0);
                    }
                    ans.push_back(tep);
                    ans1.push_back("INT");
                } else if (cnt == 1) { // 说明是小数
                    if (tep.size() >= 2 && tep[0] == '0' && tep[1] != '.') {
                        cout << "Malformed number: Leading zeros in an
integer.";
                        exit(0);
                    }
                    if (tep[tep.size() - 1] == '.') {
                        cout << "Malformed number: Decimal point at the
beginning or end of a floating point number.";
                        exit(0);
                    }
                    ans.push_back(tep);
                    ans1.push_back("DOUBLE");
                } else { // 说明有多个小数点
                    cout << "Malformed number: More than one decimal point in a
floating point number.";
                    exit(0);
                }
```

```
            } else { // 说明是字母
                ans.push_back(tep);
                ans1.push_back("IDENT");
            }
        }
}
```

至此，我们完成了词法分析的整个过程，我们在分析的过程中，用两个数组进行存储，如果分析过程中无语法错误的话，在最后输出数组中的内容即可。

实验结果展示及分析：

1. 对于如下输入程序：

```
int a, b;
double c=1.2; // This is a comment
scanf(a);
scanf(b);
printf(c);
```

由于该程序并无词法分析错误，所以当整个程序输入完成后，我们可以得到如下输出

```
int INTSYM
a IDENT
, COMMA
b IDENT
; SEMICOLON
double DOUBLESYM
c IDENT
= AO
1.2 DOUBLE
; SEMICOLON
scanf SCANFSYM
( BRACE
a IDENT
) BRACE
; SEMICOLON
scanf SCANFSYM
( BRACE
b IDENT
) BRACE
; SEMICOLON
printf PRINTFSYM
( BRACE
c IDENT
) BRACE
; SEMICOLON
```

2. 对于如下输入程序：

```
int a = 6037210
int b = 06356417
```

由于该程序中在定义 b 的时候，右值存在前导 0，所以此处存在词法分析错误，故程序输出 Malformed number: Leading zeros in an integer.

3. 对于如下输入程序：

```
12910
1223.219
27912.120921
2181123.
2810.
12123
```

可以看到的是在 2181123（以及下一行）处多出一个小数点，所以此处存在词法分析错误，故程序输出 Malformed number: Leading zeros in an integer.

4. 对于如下输入程序：

```
int a,b;
c=2;
d=123.21;
?
*
~
```

由于我们设计的词法分析器不能识别？，所以此处程序输出 Unrecognizable characters.