

编译原理

**实验名称**

学 院： 詹天佑

专 业： 计算机

学生姓名： 程维森

学 号： 21231264

北京交通大学

2023年10月

1. **程序功能描述**

这是一个C++程序，用于分析输入文本文件（`test.txt`）中的字符序列，并将分析结果输出到另一个文件（`output.txt`）中。程序的主要功能是识别输入文本中的关键字、数字、特殊字符、算术符号和注释，并将它们分类输出到`output.txt`文件中。

以下是程序的主要逻辑：

1. 包含头文件：程序使用了 `<iostream>`, `<cstring>`, 和 `<fstream>` 头文件。

2. 关键字数组：程序定义了一个字符串数组 `s[110]`，其中包含C++语言的关键字。

3. 文件输入输出：程序打开了一个输入文件流（`test.txt`）和一个输出文件流（`output.txt`）。

4. 字符读取和分类：程序使用一个循环来逐字符读取输入文件，然后根据字符的类型进行分类：

   - 空格和制表符：被忽略，继续读取下一个字符。

   - 换行符：输出到输出文件，并继续读取下一个字符。

   - 字母：识别单词，如果是关键字则输出为关键字，否则输出为标识符。

   - 数字：识别数字，输出到输出文件。

   - 特殊字符：识别特殊字符，输出到输出文件。

   - 算术符号：识别各种算术符号，输出到输出文件。

   - 注释：识别注释，包括行注释和块注释，输出到输出文件。

   - 其他字符：输出错误信息。

5. 文件操作结束：文件读取完毕后，关闭输入输出文件流，并输出"Analysis completed!"。

需要注意的是，该程序对于注释的处理比较复杂，包括了行注释（`//`）和块注释（`/\* \*/`）的处理。它会将注释内容一并输出到输出文件中。

此程序是一个简单的词法分析器，用于初步识别C++代码中的基本元素。但请注意，它可能并**不适用**于所有情况，例如，它无法处理嵌套注释等特殊情况。

1. **主要数据结构**

|  |  |
| --- | --- |
| 变量及类型 | 用途 |
| string s[110] | 字符串数组，存储C++关键字 |
| char c | 字符变量，用于存储从输入文件读取的字符 |
| char str[100] | 字符数组，用于存储识别到的字母串（关键字或标识符） |
| char str1[100] | 字符数组，用于存储识别到的数字串 |
| ifstream input | 输入文件流对象，用于读取输入文件（test.txt） |
| ofstream output | 输出文件流对象，用于写入输出文件（output.txt） |
| int amt | 整数变量，用于记录字符数组中的字符数量 |
| int mid | 整数变量，用于标记是否识别到了关键字 |
| int flag | 整数变量，用于标记块注释中的注释结束符号是否已经出现 |
| char s[110] | 字符数组，用于存储注释中的字符，用于输出到输出文件 |

1. **程序结构描述**
2. **设计方法**

这个程序是一个简单的C++词法分析器，它通过逐字符地读取输入文本文件，识别并分类字符，然后将结果输出到另一个文件中。以下是程序的**设计方法**：

1. 文件操作：程序使用C++的文件操作类（`ifstream`和`ofstream`）来处理输入和输出。它打开一个输入文件流（`test.txt`）用于读取文本内容，并打开一个输出文件流（`output.txt`）用于将分析结果写入文件。文件操作是程序的基础。

2. 字符读取与分类：程序通过一个主循环逐字符读取输入文件。根据字符的类型，程序将字符分类为关键字、标识符、数字、特殊字符、算术符号、注释等。这是通过一系列的条件判断语句来实现的。例如，使用`isalpha()`和`isdigit()`函数来判断字符是字母还是数字，然后将字符存储到相应的字符数组中。

3. 关键字识别：程序使用一个预定义的字符串数组`s[110]`存储C++关键字。当程序识别到一个字母串时，它会在这个数组中查找是否存在相应的关键字。如果存在，将其输出为关键字；否则，将其输出为标识符。

4. 特殊字符与算术符号识别：程序使用一系列的条件判断语句来识别特殊字符和算术符号。例如，`if (c == '>')`用于识别大于号，然后根据下一个字符再判断是大于等于号（`>=`）还是右移符号（`>>`）。

5. 注释处理：程序能够处理行注释（以`//`开头）和块注释（以`/\*`开始，以`\*/`结束）。

6. 错误处理：如果程序遇到不认识的字符，它会输出错误信息（`[ERROR!!!] CHECK INPUT!!!`）。

7. 输出结果：程序将识别到的各类字符和字符串，以及其类型，按照一定的格式输出到输出文件中。

8. 循环控制：程序使用一个主循环，逐字符地读取输入文件，直到文件结束。在循环中使用了`input.get(c)`语句来获取文件中的字符。

总体来说，这个程序采用了**顺序处理**的方法，根据字符的不同类型，采取不同的处理逻辑。它使用了条件判断语句和循环结构，通过逐字符的方式进行分析和分类。需要注意的是，虽然这个程序可以识别基本的C++代码元素，但它并不是一个完整的编译器，因为它只处理了词法层面的内容。

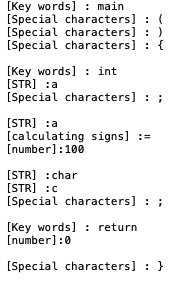
1. **函数定义及函数间的调用关系**

**没有显式定义函数**。整个程序是在一个主函数 main 中执行的，没有独立的函数定义。因此，没有函数间的调用关系。以下是主函数 main 的功能描述：

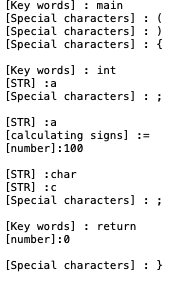
主函数，包含了整个程序的执行逻辑。打开输入文件，逐字符分析并分类，将结果输出到输出文件，处理关键字、标识符、数字、特殊字符、算术符号和注释。最后输出分析完成的消息到控制台。

1. **程序测试**

测试用例1：



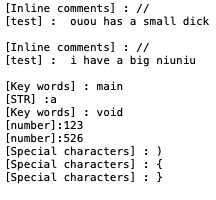
程序执行结果：



测试用例2：



程序执行结果：



5. 附件（源代码列表）

#include<bits/stdc++.h>

**using** **namespace** std;

string s[110] = {"void","int","float","double","if","else","for","do","while","return","break","main","include"};

**int** main(){

    ifstream input("test.txt");

    ofstream output("output.txt", ios::app);

**char** ch;

    input.get(ch);

**while**(input){

**if**(ch == ' '||ch == '\t'){

    input.get(ch);

**continue**;

}

**else** **if**(ch == '\n'){

output<<endl;

input.get(ch);

**continue**;

}

**else** **if**(isalpha(ch)){

**char** str[100];

**int** amt = 0;

**int** mid = 0;

str[amt++] = ch;

**do**{

input.get(ch);

str[amt++] = ch;

}**while**(isalnum(ch));

str[amt - 1] = '\0';

**for**(**int** i = 0;i <= 12;i++){

**if**(str == s[i]){

output << "[Key words] : "<< str <<endl;

                mid = 1;

                }

}

**if**(mid == 0)

                output << "[STR] :" << str << endl;

}

**else** **if**(isdigit(ch)){

**char** str1[100];

**int** amt = 0;

str1[amt++] = ch;

**do**{

input.get(ch);

str1[amt++] = ch;

}**while**(isdigit(ch));

str1[amt - 1] = '\0';

output << "[number]:" << str1 <<endl;

input.get(ch);

}

**else** **if**(ch == '!' || ch == '@' || ch == '#' || ch == '$' || ch == '%'||ch == '^' || ch == '&' || ch == '\*' || ch == '(' || ch == ')'||ch == '{' || ch == '}'||ch == '[' ||  ch == ']' || ch == ';'){

output << "[Special characters] : "<< ch << endl;

input.get(ch);

}

**else** **if** (ch == '>') {

            input.get(ch);

**if** (ch == '=') {

                output << "[calculating signs] :>= \n";

                input.get(ch);

            } **else** **if** (ch == '>') {

                output << "[calculating signs] :>> \n";

                input.get(ch);

            } **else** {

                output << "[calculating signs] :> \n";

            }

        }

**else** **if** (ch == '<') {

            input.get(ch);

**if** (ch == '=') {

                output << "[calculating signs] :<= \n";

                input.get(ch);

            } **else** **if** (ch == '<') {

                output << "[calculating signs] :<< \n";

                input.get(ch);

            } **else** {

                output << "[calculating signs] :< \n";

            }

        }

**else** **if** (ch == '!') {

            input.get(ch);

**if** (ch == '=') {

                output << "[calculating signs] :!= \n";

                input.get(ch);

            } **else** {

                output << "[calculating signs] :! \n";

            }

        }

**else** **if** (ch == '=') {

            input.get(ch);

**if** (ch == '=') {

                output << "[calculating signs] :== \n";

                input.get(ch);

            } **else** {

                output << "[calculating signs] := \n";

            }

        }

**else** **if** (ch == '&') {

            input.get(ch);

**if** (ch == '&') {

                output << "[calculating signs] :&& \n";

                input.get(ch);

            } **else** {

                output << "[calculating signs] :& \n";

            }

        }

**else** **if** (ch == '|') {

            input.get(ch);

**if** (ch == '|') {

                output << "[calculating signs] :|| \n";

                input.get(ch);

            } **else** {

                output << "[calculating signs] : | \n";

            }

        }

**else** **if** (ch == '+') {

            input.get(ch);

**if** (ch == '=') {

                output << "[calculating signs] : +=\n";

                input.get(ch);

            } **else** **if** (ch == '+') {

                output << "[calculating signs] :++ \n";

                input.get(ch);

            } **else** {

                output << "[calculating signs] :+ \n";

            }

        }

**else** **if** (ch == '-') {

            input.get(ch);

**if** (ch == '=') {

                output << "[calculating signs] :-=\n";

                input.get(ch);

            } **else** **if** (ch == '-') {

                output << "[calculating signs] :--\n";

                input.get(ch);

            } **else** {

                output << "[calculating signs] :-\n";

            }

        }

**else** **if** (ch == '\*') {

            input.get(ch);

**if** (ch == '=') {

                output << "[calculating signs] :\*=\n";

                input.get(ch);

            } **else** {

                output << "[calculating signs] :\*\n";

            }

        }

**else** **if** (ch == '/') {

            input.get(ch);

**if** (ch == '\*') {

                output << "[Start annotation] :/\*\n [test] : ";

**do** {

**int** flag = 0;

**int** amt = 0;

**char** s[110];

                    input.get(ch);

**while**(ch != '\*'){

input.get(ch);

s[amt++] = ch;

}

**while**(ch == '\*') {

                        input.get(ch);

**if** (ch == '/') {

s[amt - 1] = '\0';

output << s;

output << "\n";

                            output << "[end annotation] : \*/ \n";

                            input.get(ch);

                            flag = 1;

                        }

                    }

**if** (flag == 1)

**break**;

                } **while** (**true**);

            }

**else** **if** (ch == '/') {

                output << "[Inline comments] : // \n[test] : ";

**int** amt = 0;

**char** s[110];

**do** {

                    input.get(ch);

s[amt++] = ch;

                } **while** (ch != '\n');

amt++;

s[amt - 1] = '\0';

output << s;

                output << "\n";

                input.get(ch);

            }

**else** **if** (ch == '=') {

                output << "[calculating signs] :/= \n";

                input.get(ch);

            }

**else** {

                output << "[calculating signs] : / \n";

            }

        }

**else** {

            output << "[ERROR!!!] CHECK INPUT!!!\n";

            input.get(ch);

        }

}

input.close();

    output.close();

cout << "Analysis completed!\n";

**return**  0;

}