词法分析-手工实现 实验报告

姓名：任泽高

学号：2022211181

1　实验题目与要求

实验题目：词法分析程序的C++实现

实验要求：能够识别C语言中的各种单词符号，并以记号形式输出，能够识别并跳过注释，分析后能够输出源程序语句行数，单词个数等统计信息，能够只对源程序进行一次扫描，检测出源程序中的语法错误，并报告错误位置。实现语言选择C++，源语言选择C语言。

2　程序设计说明

2.1　项目结构

- 源代码文件
  - `main.cpp`：主函数所在文件，负责调用词法分析器并处理文件输入输出。
  - `parser.h`：头文件，包含了所有必要的函数声明、符号表定义以及全局变量。
  - `parser.cpp`：词法分析逻辑。
- 主要功能模块
  - 缓冲区管理：两个缓冲区（`buffer1` 和 `buffer2`）用于处理输入的文件字符流，负责在切换缓冲区时处理 EOF 标记。
  - 词法分析核心：负责根据输入的字符流进行词法分析，识别不同的 Token 类型，如标识符、关键字、操作符、数字、注释等。
  - 符号表：维护了一个唯一标识符的集合（`symbolTable`），用于存储源代码中出现的唯一标识符。
  - 错误处理：当发现非法字符或未闭合的注释、字符常量等时，进行错误报告。

2.2　实现思路

词法分析器的工作流程可以概述如下：

1. **初始化**：程序启动后，`main` 函数打开指定的源代码文件，将其传递给 `lexicalAnalyzer` 函数。
2. **双缓冲区管理**：词法分析器使用两个缓冲区（`buffer1` 和 `buffer2`）来读取源代码文件中的字符。在扫描过程中，当一个缓冲区耗尽时，程序会自动切换到另一个缓冲区并从文件中读取更多字符。
3. **Token 识别**：通过 `lexicalAnalyzer` 函数，程序读取文件字符流并根据字符的不同类别调用不同的解析函数（如 `parseIdentifier`、`parseNumber`、`parseStringConstant` 等）。每个解析函数负责特定类型的 Token 识别并输出相应的 Token。
4. **符号表维护**：对于标识符，程序会检查符号表（`symbolTable`）中是否已经存在该标识符。如果不存在，则将其插入符号表，并增加标识符计数。
5. **错误处理**：在字符分析过程中，如果遇到非法字符、未闭合的字符常量或注释，程序会调用 `reportError` 函数并输出具体的错误信息。
6. **统计信息**：程序在扫描结束时会输出统计信息，包括代码行数、标识符数量、数字数量、操作符数量、注释数量等。

2.3 具体模块分析

- **双缓冲区管理**：双缓冲区的设计是本程序的核心，它使得文件读取更加高效，特别是在大文件处理时可以减少 I/O 操作。通过 `forward` 指针来遍历字符流，当遇到 EOF 时根据当前活跃的缓冲区选择下一个缓冲区进行填充，保持了连续的数据流。

- **Token 解析模块**：程序包含多个解析模块，每个模块负责处理特定类型的 Token：
  - `parseIdentifier()`：用于解析标识符和关键字。程序首先检查字符流中的第一个字符是否是字母或下划线，然后继续读取后续的字母、数字或下划线组成完整的标识符。程序还会检查标识符是否是关键字，如果是，则将其标记为关键字。
  - `parseNumber()`：用于解析整数、浮点数、十六进制和八进制数。程序通过状态机的方式处理不同进制的数字，并且能够识别浮点数的科学计数法表示法。
  - `parseStringConstant()` 和 `parseCharConstant()`：分别用于解析字符串和字符常量。程序处理转义字符并确保字符常量和字符串常量的正确闭合。
  - `parseComment()`：解析单行和多行注释。单行注释以 `//` 开头，多行注释以 `/* */` 包裹。
  - `parsePreprocessorDirective()` 和 `parseMacroDefinition()`：用于处理预处理指令和宏定义，解析 `#define` 或 `#include` 等预处理操作。
  - 其他函数类似，此处省略。

- **符号表**：符号表用一个 `set` 容器来存储唯一的标识符。由于 `set` 自动去重，它确保每个标识符只会被统计一次。这在词法分析器中有助于对标识符进行快速查询和统计。

- **错误处理**：程序在遇到错误时会调用 `reportError()` 函数输出错误信息，错误信息包括了具体的行号和列号，使得用户能够轻松定位错误源。常见的错误包括：未闭合的字符常量、非法字符、未闭合的注释等。

- **统计信息输出**：在词法分析结束后，程序会输出详细的统计信息，包括行数、唯一标识符的数量、数字的数量、操作符的数量和注释的数量。

3 测试报告

3.1 简单程序测试

`test.c` 内容：

```c
#include <stdio.h>
#include <string.h>

// 变量定义
int main() {
    // 单行注释：变量定义
    int a = 10;             // 整数
    float f = 3.14;         // 浮点数
    double e = 2.71828e10;  // 科学计数法
    unsigned int hex = 0x1A3; // 十六进制
    int oct = 0123;         // 八进制

    // 字符常量和字符串常量
    char ch = 'A';          // 字符常量
    char str[] = "Hello, world!";  // 字符串常量

    // 多行注释
    /* 这是一段
```

```
       多行注释 */

    // 操作符测试
    a = a + 1;                // 加法
    a -= 5;                   // 复合操作符
    a *= 2;                   // 乘法
    a /= 3;                   // 除法
    a++;                      // 自增操作符
    a--;                      // 自减操作符
    int b = (a > 5) ? 1 : 0;  // 三元操作符
    if (a == b && b != 0) {   // 比较操作符和逻辑操作符
        printf("%s\n", str);  // 字符串输出
    }

    // 未闭合的字符常量和字符串常量，用于错误处理
    char badChar = 'X;        // 错误：未闭合的字符常量

    // 未闭合的注释
    /* 这是一个未闭合的注释

    return 0;
}
```

程序输出结果：

```
<PREPROCESSOR_DIRECTIVE, #include>
<HEADER_FILE, stdio.h>
<PREPROCESSOR_DIRECTIVE, #include>
<HEADER_FILE, string.h>
<COMMENT,  变量定义>
<KEYWORD, int>
<IDENTIFIER, main>
<SEPARATOR, (>
<SEPARATOR, )>
<SEPARATOR, {>
<COMMENT,  单行注释：变量定义>
<KEYWORD, int>
<IDENTIFIER, a>
<OPERATOR, =>
<NUMBER, 10>
<SEPARATOR, ;>
<COMMENT,  整数>
<KEYWORD, float>
<IDENTIFIER, f>
<OPERATOR, =>
<NUMBER, 3.14>
<SEPARATOR, ;>
<COMMENT,  浮点数>
<KEYWORD, double>
<IDENTIFIER, e>
<OPERATOR, =>
<NUMBER, 2.71828e+010>
<SEPARATOR, ;>
<COMMENT,  科学计数法>
<KEYWORD, unsigned>
<KEYWORD, int>
```

```
<IDENTIFIER, hex>
<OPERATOR, =>
<NUMBER, 419>
<SEPARATOR, ;>
<COMMENT,  十六进制>
<KEYWORD, int>
<IDENTIFIER, oct>
<OPERATOR, =>
<NUMBER, 83>
<SEPARATOR, ;>
<COMMENT,  八进制>
<COMMENT,  字符常量和字符串常量>
<KEYWORD, char>
<IDENTIFIER, ch>
<OPERATOR, =>
<CHAR_CONSTANT, A>
<SEPARATOR, ;>
<COMMENT,  字符常量>
<KEYWORD, char>
<IDENTIFIER, str>
<SEPARATOR, [>
<SEPARATOR, ]>
<OPERATOR, =>
<STRING_CONSTANT, Hello, world!>
<SEPARATOR, ;>
<COMMENT,  字符串常量>
<COMMENT,  多行注释>
<COMMENT,  这是一段
       多行注释 >
<COMMENT,  操作符测试>
<IDENTIFIER, a>
<OPERATOR, =>
<IDENTIFIER, a>
<OPERATOR, +>
<NUMBER, 1>
<SEPARATOR, ;>
<COMMENT,  加法>
<IDENTIFIER, a>
<OPERATOR, -=>
<NUMBER, 5>
<SEPARATOR, ;>
<COMMENT,  复合操作符>
<IDENTIFIER, a>
<OPERATOR, *=>
<NUMBER, 2>
<SEPARATOR, ;>
<COMMENT,  乘法>
<IDENTIFIER, a>
<OPERATOR, /=>
<NUMBER, 3>
<SEPARATOR, ;>
<COMMENT,  除法>
<IDENTIFIER, a>
<OPERATOR, ++>
<SEPARATOR, ;>
<COMMENT,  自增操作符>
```

```
<IDENTIFIER, a>
<OPERATOR, -->
<SEPARATOR, ;>
<COMMENT,  自减操作符>
<KEYWORD, int>
<IDENTIFIER, b>
<OPERATOR, =>
<SEPARATOR, (>
<IDENTIFIER, a>
<OPERATOR, >>
<NUMBER, 5>
<SEPARATOR, )>
<OPERATOR, ?>
<NUMBER, 1>
<OPERATOR, :>
<NUMBER, 0>
<SEPARATOR, ;>
<COMMENT,  三元操作符>
<KEYWORD, if>
<SEPARATOR, (>
<IDENTIFIER, a>
<OPERATOR, ==>
<IDENTIFIER, b>
<OPERATOR, &&>
<IDENTIFIER, b>
<OPERATOR, !=>
<NUMBER, 0>
<SEPARATOR, )>
<SEPARATOR, {>
<COMMENT,  比较操作符和逻辑操作符>
<IDENTIFIER, printf>
<SEPARATOR, (>
<STRING_CONSTANT, %s\n>
<SEPARATOR, ,>
<IDENTIFIER, str>
<SEPARATOR, )>
<SEPARATOR, ;>
<COMMENT,  字符串输出>
<SEPARATOR, }>
<COMMENT,  未闭合的字符常量和字符串常量，用于错误处理>
<KEYWORD, char>
<IDENTIFIER, badChar>
<OPERATOR, =>
Error at line 34, column 23: Unclosed character constant
<UNKNOWN, =>
<SEPARATOR, ;>
<COMMENT,  错误：未闭合的字符常量>
<COMMENT,  未闭合的注释>
Error at line 41, column 1: Unclosed comment
<COMMENT,  这是一个未闭合的注释

    return 0;
}
>

--- Statistics ---
```

```
Number of lines: 41
Number of unique identifiers: 17
Number of numbers: 13
Number of operators: 22
Number of comments: 26
Total number of valid characters (excluding whitespace): 599
```

分析：该程序是一个简单示例，程序中有两个错误，分别是34行未闭合的字符常量，还有下面未闭合注释，词法分析均可以检查出来，而且可以在一趟中将这两个问题都检测出来。其他词法类型在源程序注释中写的很清楚，不再重复。

3.2 字符常量测试

`testcharconst.c`内容：

```c
//
// Created by Trenchance on 2024/9/27.
//
char c1 = 'a';
char c2 = '\n';
char c3 = '\\';
char c4 = 'b;
```

程序输出结果：

```
<COMMENT, >
<COMMENT,  Created by Trenchance on 2024/9/27.>
<COMMENT, >
<KEYWORD, char>
<IDENTIFIER, c1>
<OPERATOR, =>
<CHAR_CONSTANT, a>
<SEPARATOR, ;>
<KEYWORD, char>
<IDENTIFIER, c2>
<OPERATOR, =>
<CHAR_CONSTANT, \n>
<SEPARATOR, ;>
<KEYWORD, char>
<IDENTIFIER, c3>
<OPERATOR, =>
<CHAR_CONSTANT, \\>
<SEPARATOR, ;>
<KEYWORD, char>
<IDENTIFIER, c4>
<OPERATOR, =>
Error at line 7, column 14: Unclosed character constant
<UNKNOWN, =>
<SEPARATOR, ;>

--- Statistics ---
Number of lines: 8
Number of unique identifiers: 5
Number of numbers: 0
Number of operators: 4
Number of comments: 3
Total number of valid characters (excluding whitespace): 73
```

分析：这是一个针对字符串常量的测试样例，可见测法分析程序可以识别到字符串常量，以及未闭合符号。

3.3 注释测试

`testcomment.c`内容：

```
//
// Created by Trenchance on 2024/9/27.
//
// This is a simple single-line comment
int a = 5;
// Comment with special characters !@#$%^&*()
/*
This comment contains another comment start symbol /*
But it is not closed properly
*/
int b = 10;
int a = 5;/*Comment right after code*/int b = 10;
/*
This comment is not closed
int d = 20;
```

程序输出结果：

```
<COMMENT, >
<COMMENT,  Created by Trenchance on 2024/9/27.>
<COMMENT, >
<COMMENT,  This is a simple single-line comment>
<KEYWORD, int>
<IDENTIFIER, a>
<OPERATOR, =>
<NUMBER, 5>
<SEPARATOR, ;>
<COMMENT,  Comment with special characters !@#$%^&*()>
<COMMENT,
This comment contains another comment start symbol /*
But it is not closed properly
>
<KEYWORD, int>
<IDENTIFIER, b>
<OPERATOR, =>
<NUMBER, 10>
<SEPARATOR, ;>
<KEYWORD, int>
<IDENTIFIER, a>
<OPERATOR, =>
<NUMBER, 5>
<SEPARATOR, ;>
<COMMENT, Comment right after code>
```

```
<KEYWORD, int>
<IDENTIFIER, b>
<OPERATOR, =>
<NUMBER, 10>
<SEPARATOR, ;>
Error at line 14, column 1: Unclosed comment
<COMMENT,
This comment is not closed
int d = 20;
>

--- Statistics ---
Number of lines: 14
Number of unique identifiers: 3
Number of numbers: 4
Number of operators: 4
Number of comments: 8
Total number of valid characters (excluding whitespace): 295
```

分析：这是一份针对注释的测试，测试了各种可能出现的注释情况，末尾没有闭合的注释也能够很好的识别出来。

3.4 数字测试

`testnum.c`内容：

```c
//
// Created by Trenchance on 2024/9/27.
//
#include <stdio.h>

int main() {
    // 正常的十进制整数
    int dec1 = 123;
    int dec2 = 0;           // 零
    int dec3 = -456;        // 负数

    // 八进制整数
    int oct1 = 0123;        // 八进制的83
    int oct2 = 00;          // 八进制的0
    int oct3 = -077;        // 八进制的负63

    // 十六进制整数
    int hex1 = 0x1A3;       // 十六进制的419
    int hex2 = 0XABC;       // 十六进制的2748
    int hex3 = -0xFF;       // 十六进制的负255

    // 浮点数
    float flt1 = 1.23;
    float flt2 = 0.0;
    float flt3 = -456.789;
    float flt4 = 123.0e10;  // 科学计数法
    float flt5 = 1.23E-10;  // 科学计数法，负指数
    // 非法数字（测试应当在词法分析中捕获这些错误）
    int ill1 = 08;        // 非法的八进制数（8不合法）
    int ill2 = 0xG123;    // 非法的十六进制数（G不合法）

    return 0;
}
```

程序输出结果:

```
<COMMENT, >
<COMMENT,  Created by Trenchance on 2024/9/27.>
<COMMENT, >
<PREPROCESSOR_DIRECTIVE, #include>
<HEADER_FILE, stdio.h>
<KEYWORD, int>
<IDENTIFIER, main>
<SEPARATOR, (>
<SEPARATOR, )>
<SEPARATOR, {>
<COMMENT,  正常的十进制整数>
<KEYWORD, int>
<IDENTIFIER, dec1>
<OPERATOR, =>
<NUMBER, 123>
<SEPARATOR, ;>
<KEYWORD, int>
<IDENTIFIER, dec2>
<OPERATOR, =>
<NUMBER, 0>
<SEPARATOR, ;>
<COMMENT,  零>
<KEYWORD, int>
<IDENTIFIER, dec3>
<OPERATOR, =>
<OPERATOR, ->
<NUMBER, 456>
<SEPARATOR, ;>
<COMMENT,  负数>
<COMMENT,  八进制整数>
<KEYWORD, int>
<IDENTIFIER, oct1>
<OPERATOR, =>
<NUMBER, 83>
<SEPARATOR, ;>
<COMMENT,  八进制的83>
<KEYWORD, int>
<IDENTIFIER, oct2>
<OPERATOR, =>
<NUMBER, 0>
<SEPARATOR, ;>
<COMMENT,  八进制的0>
<KEYWORD, int>
<IDENTIFIER, oct3>
<OPERATOR, =>
<OPERATOR, ->
```

```
<NUMBER, 63>
<SEPARATOR, ;>
<COMMENT,  八进制的负63>
<COMMENT,  十六进制整数>
<KEYWORD, int>
<IDENTIFIER, hex1>
<OPERATOR, =>
<NUMBER, 419>
<SEPARATOR, ;>
<COMMENT,  十六进制的419>
<KEYWORD, int>
<IDENTIFIER, hex2>
<OPERATOR, =>
<NUMBER, 2748>
<SEPARATOR, ;>
<COMMENT,  十六进制的2748>
<KEYWORD, int>
<IDENTIFIER, hex3>
<OPERATOR, =>
<OPERATOR, ->
<NUMBER, 255>
<SEPARATOR, ;>
<COMMENT,  十六进制的负255>
<COMMENT,  浮点数>
<KEYWORD, float>
<IDENTIFIER, flt1>
<OPERATOR, =>
<NUMBER, 1.23>
<SEPARATOR, ;>
<KEYWORD, float>
<IDENTIFIER, flt2>
<OPERATOR, =>
<NUMBER, 0>
<SEPARATOR, .>
<NUMBER, 0>
<SEPARATOR, ;>
<KEYWORD, float>
<IDENTIFIER, flt3>
<OPERATOR, =>
<OPERATOR, ->
<NUMBER, 456.789>
<SEPARATOR, ;>
<KEYWORD, float>
<IDENTIFIER, flt4>
<OPERATOR, =>
<NUMBER, 1.23e+012>
<SEPARATOR, ;>
<COMMENT,  科学计数法>
<KEYWORD, float>
<IDENTIFIER, flt5>
<OPERATOR, =>
<NUMBER, 1.23e-010>
<SEPARATOR, ;>
<COMMENT,  科学计数法，负指数>
<COMMENT,  非法数字（测试应当在词法分析中捕获这些错误）>
<KEYWORD, int>
```

```
<IDENTIFIER, ill1>
<OPERATOR, =>
Error at line 30, column 17: Invalid octal number
<NUMBER, 0>
<SEPARATOR, ;>
<COMMENT,  非法的八进制数（8不合法）>
<KEYWORD, int>
<IDENTIFIER, ill2>
<OPERATOR, =>
Error at line 31, column 19: Invalid hexadecimal number
<NUMBER, 0>
<NUMBER, 123>
<SEPARATOR, ;>
<COMMENT,  非法的十六进制数（G不合法）>
<KEYWORD, return>
<NUMBER, 0>
<SEPARATOR, ;>
<SEPARATOR, }>

--- Statistics ---
Number of lines: 35
Number of unique identifiers: 20
Number of numbers: 19
Number of operators: 20
Number of comments: 20
Total number of valid characters (excluding whitespace): 551
```

分析：这是一个较为全面的针对数字的测试，包括了常见整数，浮点数，科学计数法，八进制以及十六进制数。程序可以很好的识别出这些内容，同时能够对错误的数字（错误的八进制和十六进制数）做出相应处理。

3.5 字符常量测试

`teststringconst.c`内容：

```c
//
// Created by Trenchance on 2024/9/27.
//
#include <stdio.h>

int main() {
    // 正常的字符串常量
    char *str1 = "Hello, world!";

    // 包含转义字符的字符串常量
    char *str2 = "Line 1\nLine 2\tTabbed";

    // 包含双引号和反斜杠的字符串常量
    char *str3 = "She said, \"Hello!\" and used a backslash \\";

    // 未闭合的字符串常量
    char *str4 = "This string is not closed


    printf("%s\n", str1);
    printf("%s\n", str2);
    printf("%s\n", str3);
    return 0;
}
```

程序输出结果:

```
<COMMENT, >
<COMMENT,  Created by Trenchance on 2024/9/27.>
<COMMENT, >
<PREPROCESSOR_DIRECTIVE, #include>
<HEADER_FILE, stdio.h>
<KEYWORD, int>
<IDENTIFIER, main>
<SEPARATOR, (>
<SEPARATOR, )>
<SEPARATOR, {>
<COMMENT,  正常的字符串常量>
<KEYWORD, char>
<OPERATOR, *>
<IDENTIFIER, str1>
<OPERATOR, =>
<STRING_CONSTANT, Hello, world!>
<SEPARATOR, ;>
<COMMENT,  包含转义字符的字符串常量>
<KEYWORD, char>
<OPERATOR, *>
<IDENTIFIER, str2>
<OPERATOR, =>
<STRING_CONSTANT, Line 1\nLine 2\tTabbed>
<SEPARATOR, ;>
<COMMENT,  包含双引号和反斜杠的字符串常量>
<KEYWORD, char>
<OPERATOR, *>
<IDENTIFIER, str3>
<OPERATOR, =>
<STRING_CONSTANT, She said, \"Hello!\" and used a backslash \\>
<SEPARATOR, ;>
<COMMENT,  未闭合的字符串常量>
<KEYWORD, char>
<OPERATOR, *>
<IDENTIFIER, str4>
<OPERATOR, =>
Error at line 17, column 42: Unclosed string constant
<UNKNOWN, =>
<IDENTIFIER, printf>
<SEPARATOR, (>
<STRING_CONSTANT, %s\n>
<SEPARATOR, ,>
<IDENTIFIER, str1>
<SEPARATOR, )>
<SEPARATOR, ;>
<IDENTIFIER, printf>
<SEPARATOR, (>
<STRING_CONSTANT, %s\n>
<SEPARATOR, ,>
<IDENTIFIER, str2>
<SEPARATOR, )>
<SEPARATOR, ;>
<IDENTIFIER, printf>
<SEPARATOR, (>
<STRING_CONSTANT, %s\n>
<SEPARATOR, ,>
<IDENTIFIER, str3>
<SEPARATOR, )>
<SEPARATOR, ;>
<KEYWORD, return>
<NUMBER, 0>
<SEPARATOR, ;>
<SEPARATOR, }>

--- Statistics ---
Number of lines: 25
Number of unique identifiers: 9
Number of numbers: 1
Number of operators: 8
Number of comments: 7
Total number of valid characters (excluding whitespace): 338
```

分析：这是一个对于字符串常量的测试，经过测试，说明该程序能够检查出错误，并正确识别字符串常量。

3.6 复杂测试

`test2.c`内容：

```
//
// Created by Trenchance on 2024/9/27.
//
#include <stdio.h>
#include <math.h>
#include <string.h>  // 字符串操作库

#define PI 3.14159   // 定义常量 PI
#define MAX(a,b) ((a) > (b) ? (a) : (b))  // 宏定义最大值

// 枚举类型的定义
enum Days { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY };

// 结构体的定义
struct Point {
    int x;
    int y;
};

// 函数声明
void greet();
int factorial(int n);
float calculate_circle_area(float radius);
void swap(int *a, int *b);
```

```c
int main() {
    // 变量声明与初始化
    int a=10,b=20;
    float radius = 5.5;
    double large_num = 1.23e+10;
    char letter = 'A';
    char str[] = "Hello, World!";
    int hex = 0x1A3;      // 十六进制
    int octal = 0123;     // 八进制
    unsigned long factorial_result = 0;

    // 打印一些信息
    printf("Hello, World!\n");
    greet();

    // 数学运算
    float area = calculate_circle_area(radius);
    printf("Area of circle: %.2f\n", area);

    // 循环与数组
    int array[5] = {0, 1, 2, 3, 4};
    for (int i = 0; i < 5; i++) {
        printf("Array[%d] = %d\n", i, array[i]);
    }

    // 枚举的使用
    enum Days today = MONDAY;
    switch (today) {
        case SUNDAY:
            printf("It is Sunday!\n");
            break;
        case MONDAY:
            printf("It is Monday!\n");
            break;
        default:
            printf("It is another day of the week.\n");
            break;
    }

    // 位运算
    int bitwise_and = a & b;
    int bitwise_or = a | b;
    int bitwise_xor = a ^ b;

    // 指针与地址
    swap(&a, &b);

    // 递归函数调用
    factorial_result = factorial(5);
    printf("Factorial of 5 is: %lu\n", factorial_result);

    // 多行注释的示例
    /* 这是一段多行注释
       用来测试词法分析器对注释的处理
       这里不会被解析为代码
    */

    // 退出程序

    return 0;
}

// 问候函数
void greet() {
    printf("Greetings from the C program!\n");
}

// 递归计算阶乘
int factorial(int n) {
    if (n == 0 || n == 1) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}

// 计算圆的面积
float calculate_circle_area(float radius) {
    return PI * radius * radius;
}

// 交换两个变量的值
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

程序输出：

```
<COMMENT, >
<COMMENT,  Created by Trenchance on 2024/9/27.>
<COMMENT, >
<PREPROCESSOR_DIRECTIVE, #include>
<HEADER_FILE, stdio.h>
<PREPROCESSOR_DIRECTIVE, #include>
<HEADER_FILE, math.h>
<PREPROCESSOR_DIRECTIVE, #include>
<HEADER_FILE, string.h>
<COMMENT,  字符串操作库>
<PREPROCESSOR_DIRECTIVE, #define>
<MACRO_DEFINITION, PI>
<MACRO_VALUE, 3.14159   >
<COMMENT,  定义常量 PI>
<PREPROCESSOR_DIRECTIVE, #define>
<MACRO_DEFINITION, MAX>
<MACRO_PARAMETER, a,b>
<MACRO_VALUE, ((a) > (b) ? (a) : (b))  >
<COMMENT,  宏定义最大值>
<COMMENT,  枚举类型的定义>
```

```
<KEYWORD, enum>
<IDENTIFIER, Days>
<SEPARATOR, {>
<IDENTIFIER, SUNDAY>
<SEPARATOR, ,>
<IDENTIFIER, MONDAY>
<SEPARATOR, ,>
<IDENTIFIER, TUESDAY>
<SEPARATOR, ,>
<IDENTIFIER, WEDNESDAY>
<SEPARATOR, ,>
<IDENTIFIER, THURSDAY>
<SEPARATOR, ,>
<IDENTIFIER, FRIDAY>
<SEPARATOR, ,>
<IDENTIFIER, SATURDAY>
<SEPARATOR, }>
<SEPARATOR, ;>
<COMMENT,  结构体的定义>
<KEYWORD, struct>
<IDENTIFIER, Point>
<SEPARATOR, {>
<KEYWORD, int>
<IDENTIFIER, x>
<SEPARATOR, ;>
<KEYWORD, int>
<IDENTIFIER, y>
<SEPARATOR, ;>
<SEPARATOR, }>
<SEPARATOR, ;>
<COMMENT,  函数声明>
<KEYWORD, void>
<IDENTIFIER, greet>
<SEPARATOR, (>
<SEPARATOR, )>
<SEPARATOR, ;>
<KEYWORD, int>
<IDENTIFIER, factorial>
<SEPARATOR, (>
<KEYWORD, int>
<IDENTIFIER, n>
<SEPARATOR, )>
<SEPARATOR, ;>
<KEYWORD, float>
<IDENTIFIER, calculate_circle_area>
<SEPARATOR, (>
<KEYWORD, float>
<IDENTIFIER, radius>
<SEPARATOR, )>
<SEPARATOR, ;>
<KEYWORD, void>
<IDENTIFIER, swap>
<SEPARATOR, (>
<KEYWORD, int>
<OPERATOR, *>
<IDENTIFIER, a>
```

```
<SEPARATOR, ,>
<KEYWORD, int>
<OPERATOR, *>
<IDENTIFIER, b>
<SEPARATOR, )>
<SEPARATOR, ;>
<KEYWORD, int>
<IDENTIFIER, main>
<SEPARATOR, (>
<SEPARATOR, )>
<SEPARATOR, {>
<COMMENT,  变量声明与初始化>
<KEYWORD, int>
<IDENTIFIER, a>
<OPERATOR, =>
<NUMBER, 10>
<SEPARATOR, ,>
<IDENTIFIER, b>
<OPERATOR, =>
<NUMBER, 20>
<SEPARATOR, ;>
<KEYWORD, float>
<IDENTIFIER, radius>
<OPERATOR, =>
<NUMBER, 5.5>
<SEPARATOR, ;>
<KEYWORD, double>
<IDENTIFIER, large_num>
<OPERATOR, =>
<NUMBER, 1.23e+010>
<SEPARATOR, ;>
<KEYWORD, char>
<IDENTIFIER, letter>
<OPERATOR, =>
<CHAR_CONSTANT, A>
<SEPARATOR, ;>
<KEYWORD, char>
<IDENTIFIER, str>
<SEPARATOR, [>
<SEPARATOR, ]>
<OPERATOR, =>
<STRING_CONSTANT, Hello, World!>
<SEPARATOR, ;>
<KEYWORD, int>
<IDENTIFIER, hex>
<OPERATOR, =>
<NUMBER, 419>
<SEPARATOR, ;>
<COMMENT,  十六进制>
<KEYWORD, int>
<IDENTIFIER, octal>
<OPERATOR, =>
<NUMBER, 83>
<SEPARATOR, ;>
<COMMENT,  八进制>
<KEYWORD, unsigned>
```

```
<KEYWORD, long>
<IDENTIFIER, factorial_result>
<OPERATOR, =>
<NUMBER, 0>
<SEPARATOR, ;>
<COMMENT,  打印一些信息>
<IDENTIFIER, printf>
<SEPARATOR, (>
<STRING_CONSTANT, Hello, World!\n>
<SEPARATOR, )>
<SEPARATOR, ;>
<IDENTIFIER, greet>
<SEPARATOR, (>
<SEPARATOR, )>
<SEPARATOR, ;>
<COMMENT,  数学运算>
<KEYWORD, float>
<IDENTIFIER, area>
<OPERATOR, =>
<IDENTIFIER, calculate_circle_area>
<SEPARATOR, (>
<IDENTIFIER, radius>
<SEPARATOR, )>
<SEPARATOR, ;>
<IDENTIFIER, printf>
<SEPARATOR, (>
<STRING_CONSTANT, Area of circle: %.2f\n>
<SEPARATOR, ,>
<IDENTIFIER, area>
<SEPARATOR, )>
<SEPARATOR, ;>
<COMMENT,  循环与数组>
<KEYWORD, int>
<IDENTIFIER, array>
<SEPARATOR, [>
<NUMBER, 5>
<SEPARATOR, ]>
<OPERATOR, =>
<SEPARATOR, {>
<NUMBER, 0>
<SEPARATOR, ,>
<NUMBER, 1>
<SEPARATOR, ,>
<NUMBER, 2>
<SEPARATOR, ,>
<NUMBER, 3>
<SEPARATOR, ,>
<NUMBER, 4>
<SEPARATOR, }>
<SEPARATOR, ;>
<KEYWORD, for>
<SEPARATOR, (>
<KEYWORD, int>
<IDENTIFIER, i>
<OPERATOR, =>
<NUMBER, 0>
```

```
<SEPARATOR, ;>
<IDENTIFIER, i>
<OPERATOR, <>
<NUMBER, 5>
<SEPARATOR, ;>
<IDENTIFIER, i>
<OPERATOR, ++>
<SEPARATOR, )>
<SEPARATOR, {>
<IDENTIFIER, printf>
<SEPARATOR, (>
<STRING_CONSTANT, Array[%d] = %d\n>
<SEPARATOR, ,>
<IDENTIFIER, i>
<SEPARATOR, ,>
<IDENTIFIER, array>
<SEPARATOR, [>
<IDENTIFIER, i>
<SEPARATOR, ]>
<SEPARATOR, )>
<SEPARATOR, ;>
<SEPARATOR, }>
<COMMENT,  枚举的使用>
<KEYWORD, enum>
<IDENTIFIER, Days>
<IDENTIFIER, today>
<OPERATOR, =>
<IDENTIFIER, MONDAY>
<SEPARATOR, ;>
<KEYWORD, switch>
<SEPARATOR, (>
<IDENTIFIER, today>
<SEPARATOR, )>
<SEPARATOR, {>
<KEYWORD, case>
<IDENTIFIER, SUNDAY>
<OPERATOR, :>
<IDENTIFIER, printf>
<SEPARATOR, (>
<STRING_CONSTANT, It is Sunday!\n>
<SEPARATOR, )>
<SEPARATOR, ;>
<KEYWORD, break>
<SEPARATOR, ;>
<KEYWORD, case>
<IDENTIFIER, MONDAY>
<OPERATOR, :>
<IDENTIFIER, printf>
<SEPARATOR, (>
<STRING_CONSTANT, It is Monday!\n>
<SEPARATOR, )>
<SEPARATOR, ;>
<KEYWORD, break>
<SEPARATOR, ;>
<KEYWORD, default>
<OPERATOR, :>
```

```
<IDENTIFIER, printf>
<SEPARATOR, (>
<STRING_CONSTANT, It is another day of the week.\n>
<SEPARATOR, )>
<SEPARATOR, ;>
<KEYWORD, break>
<SEPARATOR, ;>
<SEPARATOR, }>
<COMMENT,  位运算>
<KEYWORD, int>
<IDENTIFIER, bitwise_and>
<OPERATOR, =>
<IDENTIFIER, a>
<OPERATOR, &>
<IDENTIFIER, b>
<SEPARATOR, ;>
<KEYWORD, int>
<IDENTIFIER, bitwise_or>
<OPERATOR, =>
<IDENTIFIER, a>
<OPERATOR, |>
<IDENTIFIER, b>
<SEPARATOR, ;>
<KEYWORD, int>
<IDENTIFIER, bitwise_xor>
<OPERATOR, =>
<IDENTIFIER, a>
<OPERATOR, ^>
<IDENTIFIER, b>
<SEPARATOR, ;>
<COMMENT,  指针与地址>
<IDENTIFIER, swap>
<SEPARATOR, (>
<OPERATOR, &>
<IDENTIFIER, a>
<SEPARATOR, ,>
<OPERATOR, &>
<IDENTIFIER, b>
<SEPARATOR, )>
<SEPARATOR, ;>
<COMMENT,  递归函数调用>
<IDENTIFIER, factorial_result>
<OPERATOR, =>
<IDENTIFIER, factorial>
<SEPARATOR, (>
<NUMBER, 5>
<SEPARATOR, )>
<SEPARATOR, ;>
<IDENTIFIER, printf>
<SEPARATOR, (>
<STRING_CONSTANT, Factorial of 5 is: %lu\n>
<SEPARATOR, ,>
<IDENTIFIER, factorial_result>
<SEPARATOR, )>
<SEPARATOR, ;>
<COMMENT,  多行注释的示例>
```

```
<COMMENT,  这是一段多行注释
        用来测试词法分析器对注释的处理
        这里不会被解析为代码
    >
<COMMENT,  退出程序>
<KEYWORD, return>
<NUMBER, 0>
<SEPARATOR, ;>
<SEPARATOR, }>
<COMMENT,  问候函数>
<KEYWORD, void>
<IDENTIFIER, greet>
<SEPARATOR, (>
<SEPARATOR, )>
<SEPARATOR, {>
<IDENTIFIER, printf>
<SEPARATOR, (>
<STRING_CONSTANT, Greetings from the C program!\n>
<SEPARATOR, )>
<SEPARATOR, ;>
<SEPARATOR, }>
<COMMENT,  递归计算阶乘>
<KEYWORD, int>
<IDENTIFIER, factorial>
<SEPARATOR, (>
<KEYWORD, int>
<IDENTIFIER, n>
<SEPARATOR, )>
<SEPARATOR, {>
<KEYWORD, if>
<SEPARATOR, (>
<IDENTIFIER, n>
<OPERATOR, ==>
<NUMBER, 0>
<OPERATOR, ||>
<IDENTIFIER, n>
<OPERATOR, ==>
<NUMBER, 1>
<SEPARATOR, )>
<SEPARATOR, {>
<KEYWORD, return>
<NUMBER, 1>
<SEPARATOR, ;>
<SEPARATOR, }>
<KEYWORD, else>
<SEPARATOR, {>
<KEYWORD, return>
<IDENTIFIER, n>
<OPERATOR, *>
<IDENTIFIER, factorial>
<SEPARATOR, (>
<IDENTIFIER, n>
<OPERATOR, ->
<NUMBER, 1>
<SEPARATOR, )>
<SEPARATOR, ;>
<SEPARATOR, }>
<SEPARATOR, }>
<COMMENT,  计算圆的面积>
<KEYWORD, float>
<IDENTIFIER, calculate_circle_area>
<SEPARATOR, (>
<KEYWORD, float>
<IDENTIFIER, radius>
<SEPARATOR, )>
<SEPARATOR, {>
<KEYWORD, return>
<IDENTIFIER, PI>
<OPERATOR, *>
<IDENTIFIER, radius>
<OPERATOR, *>
<IDENTIFIER, radius>
<SEPARATOR, ;>
<SEPARATOR, }>
<COMMENT,  交换两个变量的值>
<KEYWORD, void>
<IDENTIFIER, swap>
<SEPARATOR, (>
<KEYWORD, int>
<OPERATOR, *>
<IDENTIFIER, a>
<SEPARATOR, ,>
<KEYWORD, int>
<OPERATOR, *>
<IDENTIFIER, b>
<SEPARATOR, )>
<SEPARATOR, {>
<KEYWORD, int>
<IDENTIFIER, temp>
<OPERATOR, =>
<OPERATOR, *>
<IDENTIFIER, a>
<SEPARATOR, ;>
<OPERATOR, *>
<IDENTIFIER, a>
<OPERATOR, =>
<OPERATOR, *>
<IDENTIFIER, b>
<SEPARATOR, ;>
<OPERATOR, *>
<IDENTIFIER, b>
<OPERATOR, =>
<IDENTIFIER, temp>
<SEPARATOR, ;>
<SEPARATOR, }>

--- Statistics ---
Number of lines: 113
Number of unique identifiers: 53
Number of numbers: 21
Number of operators: 45
Number of comments: 26
```

```
Total number of valid characters (excluding whitespace): 1582
```

分析：这是一个复杂的C程序，几乎可以涵盖C词法的全部部分，每类单词的类型在注释中有所标注，这里不再重复。