代码高亮与提示程序文档

1.词法分析部分

通过读取源文件(`person . cpp`)为一个 `string` 变量,依次处理该字符串数组的每个元素为 `cvalue` ，记录当前的 `token` 为 `value` 。识别 `cvalue` 种类并分类处理。
如果为空格或者换行则进行 `token` 的分割处理，使用 `checkKeyword` 识别字母串为 `KEYWORD` 还是 `INDENTIFIER` 类型，统一将 `KEYWORD` , `IDENTIFIER` 和 `NUMBER` 类型的 `token` 加入到 `tokenlist` 中。
如果 `cvalue` 为数字，若 `value` 未定义则定义为 `NUMBER` 类型，若为 `NUMBER` 类型，字母串类型或者无效类型则维持原类型。之后将 `cvalue` 加入到 `value` 末尾。
如果 `cvalue` 为字母，若 `value` 未定义、为字母串或者 `NUMBER` 类型，则改为字母串类型。若为无效类型则维持原类型。之后将 `cvalue` 加入到 `value` 末尾。
如果为符号类型，向后查询得到完整的符号序列并加入 `tokenlist` 中。
最后将得到的 `tokenlist` 输出到命令行中，之后会再进行相应的语法处理。

2.语法分析部分

这一阶段还是使用到了 `antlr` 来进行语法分析。首先定义 `cppParser.g4` 文件，该文件用于定义语法规则和词法规则。语法规则中，包括以下内容：

库包含、类定义、类中的成员、访问修饰符、构造函数、成员、函数、函数参数、函数体、语句块/函数快、语句块、初始化语句、赋值语句、 `if` 语句、 `while` 语句、 `for` 语句、 `return` 语句、 `const` 修饰变量名、指针、常见函数、成员访问、多层成员访问、类型定义、忽略行和块注释及空白符号等

词法规则中，对用到的操作符、变量等进行了定义，包括：

`IDENTIFIER` 、 `INT` 、 `DOUBLE` (包含 `FLOAT`)、 `CHAR` 、 `STRING` 、 `LIB` 、逻辑运算符、二元运算符、一元运算符等

接下来生成 python 文件，以及 `listener` 和 `visitor` 遍历器。

```
antlr4 cppParser.g4 -Dlanguage=Python3
antlr4 cppParser.g4 -Dlanguage=Python3 -visitor
```

新建一个 visitor 继承生成的 visitor 遍历器。在每个节点存储如下信息

```
new_node = {
    "type": type(node).__name__.replace('Context', ''), #节点类型
    "children": [], #子节点
    "parent": self.current_node,#父节点，用于遍历，遍历结束后删除
}

# 如果遍历所有子节点后 "children" 列表为空，那么删除 "children" 属性，并添加 "content" 属性‘
# 叶子节点与代码内容一一对应
if not new_node["children"]:
    new_node["content"] = node.getText()
```

遍历方式为深度优先遍历。最后将生成的 JSON 数据写入到 `result.json` 中

```
with open('result.json', 'w') as f:
    json.dump(visitor.json_output, f, indent=4)
```

3.完整代码高亮与提示程序

3.1 实现功能

- 使用 antlr 生成 Lexer 和 Parser
- 通过命令行来实现支持 python 的代码高亮

3.2 使用指南

在实现过程中，我们将代码高亮和代码提示功能分为了两个不同 js 文件来分别进行实现。
对于代码高亮功能，执行：

```
node highlight.js ./person.cpp
```

即可对 person.cpp 中的内容进行高亮，并在命令行中进行输出；
对于代码提示功能，
执行：

```
node mycode.js ./person.cpp
```

可以对 person.cpp 中的参数以及函数进行提取，此时再在命令行中输入不完整的代码或者函数名时，即可进行代码提示，并在命令行中输出

3.3 实现方法

`lexer/parser`

首先由于 cpp 语法的复杂性，我们参考了(https://github.com/antlr/grammars-v4/tree/master/cpp)，同时在原有的 `Parser` 基础上补充，编写了与 C++ 相关的文法和语法，然后通过 antlr 生成 `CPP14Lexer` 和 `CPP14Parser`。

测试代码

在 person.cpp 中，在文件中实现引入其他包的语句，类的定义及对应变量的初始化，函数定义及其调用，类变量对应的成员函数的调用，字符串常量与数值型常量，算术运算符和逻辑运算符，条件判断语句与循环语句这七种测试代码，用于测试实现的功能

代码高亮

通过调用 `Lexer`，将文件中的代码转化为 `tokens`，根据 `token` 的 `type` 对不同的词进行颜色分类，因为 js 可以比较方便的实现命令行输出不同颜色的标识，所以我们在实现的过程中采用了 javascript 来进行实现。

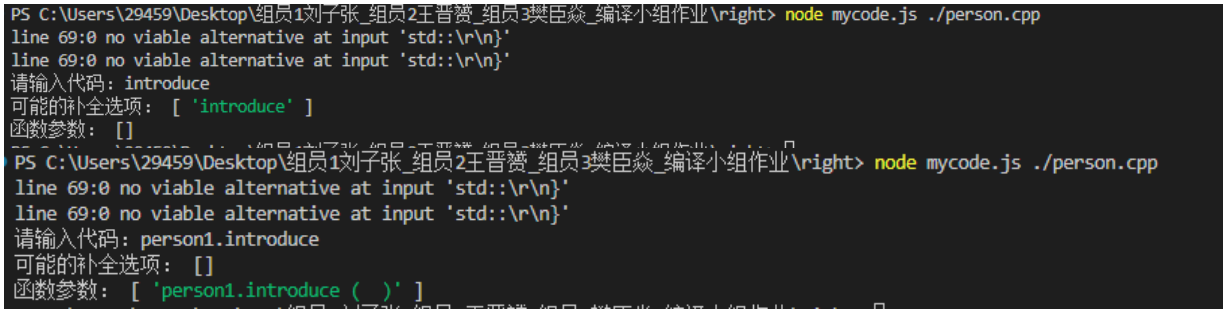
```
PS C:\Users\29459\Desktop\组员1刘子张_组员2王晋赞_组员3樊臣燚_编译小组作业\right> node highlight.js ./person.cpp
#include <stdio.h>
#include < string.h>
// a
int test(int a,int b){
    return a+b;
}
class Person {
public:
    // b
    Person(const char* name, int age) {
        this->name = strdup(name);
        this->age = age;
    }

    // c
    void introduce() {
        printf("Name: %s, Age: %d\n", name, age);
    }

    // d
    char* name;
    int age;
};

int main() {
    // e

    // f
    Person person1("Alice", 30);
    Person person2("Bob", 40);

    // g
    printf("Hello, World!\n");

    // h
    person1.introduce();
    person2.introduce();

    // i
    const char* str = "Hello";
    int num = 10;

    // j
    int a = 5;
    int b = 10;
    int sum = a + b;
    printf("Sum: %d\n", sum);

    // k
    bool condition = (a > b) && (sum > 15);
    printf("Condition: %s\n", condition ? "True" : "False");

    // l
    if (a > b) {
```

代码提示

首先使用 `Parser` 遍历整个文件中的代码，将代码中定义的函数，类和变量加入补全选项中，然后将内置的函数和变量也加入补全选项

对于参数提示而言，首先收集代码中的所有定义的函数，同时记录函数的参数和作用域，当命令行中输入的内容与某个定义的函数名匹配，同时作用域没有错误时，输出函数的参数提示。

```
请输入代码: person
可能的补全选项:  [ 'person1', 'person2' ]
函数参数:  []
```



4.遇到的问题

(1) 函数作用域问题

在参数提示部分中，存在着函数作用域问题，也就是在类中定义的函数，如果直接进行引用，是不可以给出参数提示的。为了解决这一问题，我们对文件中每个定义的类变量，都向补全选项添加其对应的所有类函数，从而在引用时可以正确地进行识别

(2) 变量重复问题

在变量补全部分中，可能会存在类，函数以及变量同名的问题，这样在补充时可能会产生误解。为解决这一问题，我们在收集定义的函数，类和变量时添加收集到的种类，然后如果存在同名的情况，则在输出时加上种类这一属性，如果没有则直接输出

5.小组分工

人员	分工
刘子张	词法分析，参数提示的完善与优化，测试代码编写
樊臣焱	语法分析，代码补全的完善与优化
王晋赟	代码高亮，代码提示功能的基础框架实现，以及编写文档