18231047 王肇凯

词法分析

最初设计

以一个函数`get_token()`为核心，产生token并输出至文件；主函数中读取输入文件所有字符，循环调用此函数读取字符生成token，直至文件末尾。

`get_token()`内部参考课本上的设计，读取非空白符，连续读取识别标识符并判断是否是保留字，连续读取整数，读取单个字符判断是否为一元分隔符，连续读取判断是否为二元分隔符。若以上一条满足，将当前token的字符串表示及分类分别存入`token`和`symbol`并输出至文件，否则产生异常。

实现与完善

为了方便以后扩展，将词法分析写为一个类`TokenAnalyze`，在初始化时指定是否输出分析结果至文件，并调用其`analyze()`方法进行词法分析主过程。将`token`、`symbol`等全局变量作为该类的成员变量。

```
class TokenAnalyze {
private:
    char ch{};
    string token;
    string symbol;
    string source;
    int pos = 0;
    int line_num = 1;
    int col_num = 1;
    bool save_to_file = false;
    int int_v;

public:
    int read_char();
    int analyze(const char *, const char *);
    int get_token();
    void retract();
    static string special(char);
    static string reserver(string);
    explicit TokenAnalyze(bool save_to_file): save_to_file(save_to_file)
{};
};
```

`analyze()`最开始将文件内容读入成员变量`source`，并维护整数值`pos`记录正在读取的字符串下标。这样做是为了方便回退，只需`pos--`即可。从而`read_char()`每次读取字符时只需更新`ch=source[pos]`并使`pos++`，更新行号`line_n um`和列号`col_num`（见后）。

在`reserver()`和`special()`分别维护两个`map`型变量记录保留字和非歧义分隔符。这里歧义指的是`>`与`>=`等需要多读取一个字符才能区分的分隔符，进行特判。

对于`INTCON`，记录其值在成员变量`int_v`中。

词法分析异常包含以下几类：

- 引号不匹配：读取到文件结尾仍未发现右双引号/右单引号
- 字符个数过多：两个单引号内字符多于一个
- 未知字符：所有判断条件均不满足的字符

为了更好的异常处理提示信息，记录了正在读取的字符的行数和列数，并在报错时输出行数列数。同时输出的还有提示信息，指出异常属于以上某个特定类别。