编译器设计文档

——22373362杜启嵘

一、参考编译器介绍

二、编译器总体设计

> 使用cpp进行编译器实现

三、词法分析设计

词法分析部分的主要任务是将读入的文件(字符串)进行一遍“字符串解析”，将读入的字符串中的单词按照种类识别出来。

单词名称	类别码	单词名称	类别码	单词名称	类别码	单词名称	类别码
Ident	IDENFR	else	ELSETK	void	VOIDTK	;	SEMICN
IntConst	INTCON	!	NOT	*	MULT	,	COMMA
StringConst	STRCON	&&	AND	/	DIV	(	LPARENT
CharConst	CHRCON	||	OR	%	MOD	)	RPARENT
main	MAINTK	for	FORTK	<	LSS	[	LBRACK
const	CONSTTK	getint	GETINTTK	<=	LEQ	]	RBRACK
int	INTTK	getchar	GETCHARTK	>	GRE	{	LBRACE
char	CHARTK	printf	PRINTFTK	>=	GEQ	}	RBRACE
break	BREAKTK	return	RETURNTK	==	EQL		
continue	CONTINUETK	+	PLUS	!=	NEQ		
if	IFTK	-	MINU	=	ASSIGN		

对于表格中给出的token种类，笔者使用一个枚举类型`TokenType`进行记录，该枚举类型内嵌在`Token`类中，`Token`类中有三个属性变量：token的字符串表示(`string`)，token的种类(`TokenType`)，token所属的行(`line_number`)，并定义其to_string方法用于输出。

```
public:
enum TokenType {
    IDENFR, INTCON, STRCON, CHRCON, MAINTK, CONSTTK, INTTK, CHARTK, BREAKTK, CONTINUETK,
    IFTK, ELSETK, NOT, AND, OR, FORTK, GETINTTK, GETCHARTK, PRINTFTK, RETURNTK, PLUS, MINU,
    VOIDTK, MULT, DIV, MOD, LSS, LEQ, GRE, GEQ, EQL, NEQ, ASSIGN, SEMICN, COMMA,
    LPARENT, RPARENT, LBRACK, RBRACK, LBRACE, RBRACE
};
```

词法分析由`Lexer`类完成，其**属性**定义为：

- source(string)：读入的程序字符串
- line_number(int)：当前分析到的行号
- pos(int)：当前分析的字符串索引位置
- errors(`vector<Error>`)：记录错误的数组(在词法分析阶段只有a类错误)
- tokens(`vector<TOken>`)：解析字符串得到的Token数组
- reverse_words(`unordered_map<std::string,Token::TokenType>`)：`SysY`保留字表

```
// lexer.h
private:
        std::string source;
        int pos;
        Token::TokenType token_type;
        int line_number;
        std::unordered_map <std::string, Token::TokenType> reserve_words;
        std::vector <Error> errors;  // 保存a类错误
        std::vector <Token> tokens;
```

`Lexer`类中的**方法**定义为：

- `initialize_reverse_word_map()`：建立保留字表，用于后续查找
- 构造方法/析构方法
- `next()`：按照读入的字符`ch=source[pos]`,从字符串中解析token，存入其自身属性tokens
  - `intcon()`：解析`INTCON`种类token的私有方法
  - `idenfr()`：解析`IDENFR`种类token的私有方法
  - `skip_single_line_comment()`：处理单行注释的私有方法
  - `skip_multi_line_comment()`：处理多行注释的私有方法(状态机设计)
  - `chrcon()`：解析`CHRCON`种类token的私有方法(尤其要注意转义字符的处理`\`)
- `run()`：按格式输出`tokens/errors`数组到文件中

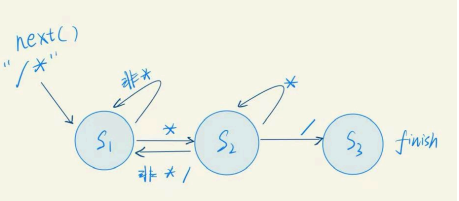
```
    public:
        Lexer(std::string source);
        ~Lexer();
        void next();
        void run();

    private:
            void intcon();
            void idenfr();
            void strcon();
            void chrcon();
            void skip_single_line_comment();
            void skip_multi_line_comment();
            void initialize_reverse_word_map();
```

处理多行注释的状态机设计：借鉴了课程组提供的词法分析`ppt`中的思路，但由于笔者的设计中只有连续读到`/,*`符号才会进入多行注释处理程序，因此只需要一个三状态状态机。

```
// multi_line_comment_fsm.h
enum State {
    S1,S2,S3
};
```

状态转移图如下图：



词法分析阶段的错误处理：在词法分析阶段只会有a类错误：将`&& / ||`记为`& / |`，为了可扩展性考虑，笔者建立了`Error`类来管理错误，其属性为：行号(`line_number`)，错误类型(error_type)，并定义相应的to_string方法用于输出Error类实例对象的字符串格式。