编译原理实验报告三

151220135 许丽军 xulj.cs@gmail.com

零、实验进度描述

完成了所有的必做内容和选做要求**3.1 3.2**

一、实验内容

从不存在词法错误、语法错误和语义错误的C--源代码,生成特定的中间代码

二、小组成员

151220135 许丽军

三、实现细节

数据结构

- 中间代码

```
typedef struct InterCode {

    enum {ASSIGN , ASSIGN_ADDR,ASSIGN_STAR,STAR_ASSIGN,ADD , SUB , MUL , DIV
 ,FUNC_DEC, PARAM, ARG , RET, IF, GOTO , LABEL ,FUNC_CALL, READ ,WRITE , DEC
} kind;
    union{
        struct {Operand left,right;} assign;
        struct {Operand op;}unop;
        struct {Operand result,op1,op2;}binop;
        struct {Operand left,right;const char *op;}cond;
        struct {Operand place;const char *func_name;}func;
        struct {Operand op;int size;}dec;
        const char *name; //func_name var_name
        Operand op;
    };
}InterCode;
```

- 采用线性结构（双向循环链表）链接中间代码

```
typedef struct InterCodes {
    InterCode *code;
    struct InterCodes *prev,*next;
}InterCodes;
```

- 操作数的结构

```
typedef struct Operand_* Operand;
struct Operand_{
    enum {VARIABLE , CONSTANT , ADDRESS  } kind;
    union {

        const char *info;

    };
};
```

函数接口

- 用于生成中间代码

```
InterCodes *ic_gen_func_dec(const char *);
InterCodes *ic_gen_arg(const char *);
InterCodes *ic_gen_varlist(FieldList para);
InterCodes *ic_gen_assign_star(const char *,const char*);
InterCodes *ic_gen_assign_addr(const char *,const char*);
InterCodes *ic_gen_assign(const char *,const char*);
InterCodes *ic_gen_ari(const char *,const char*,const char *,const char *);
InterCodes *ic_gen_neg(const char*,const char *);
InterCodes *ic_gen_ret(const char *);
InterCodes *ic_gen_goto(const char *);
InterCodes *ic_gen_if(const char *,const char *,const char*);
InterCodes *ic_gen_label(const char*);
InterCodes *ic_gen_read(const char*);
InterCodes *ic_gen_write(const char*);
InterCodes *ic_gen_func_call(const char*,const char*);
InterCodes *ic_gen_dec(const char *,int);
```

- 用于中间代码的连接和调整

```
InterCodes *IC_2_ICs(InterCode *);
InterCodes* ICs_concat(int,...);
InterCodes *ICs_pop_back(InterCodes*);
```

- 用于打印中间代码

```
void print_IC(InterCode*);
```

翻译过程

- 类似实验二，从ExtDef开始，为语法树的每一个节点定义一个对应的翻译函数，用来对以此节点为根的子树进行翻译，并调用子节点对应的函数.
- 通过函数的返回值来返回中间代码，用参数来传递中间信息.
- 所有翻译函数的函数体模式均为

```
InterCodes *translate_xxx(Node *p,...);
```

四、实验亮点

- 将中间代码生成和打印的功能封装成函数接口，翻译过程中只需要调用即可，模块性强.
- 利用可变参数来一次连接任意多条中间代码.

```
InterCodes *ICs_concat(int num,...)
```

- 对中间代码采用指针操作(链表结点的域采用中间代码的指针，而不是中间代码本身），减小数据传递的开销.

```
typedef struct InterCodes {
    InterCode *code;
    struct InterCodes *prev,*next;
}InterCodes;
```

五、编译和运行

- 编译并生成可执行目标文件：make
- 运行可执行目标文件： ./mycc input_file output_file
- 清除中间生成文件：make clean-temp

- 清除所有生成文件: make clean