中间代码生成

name: 谷莘,杨杰 **id**: 221220092, 221220100

运行方式

编译过程

```
cd Code
make
```

生成的程序是出现在Code目录下的parser，以下是parser的使用方法：

```
./parser ./path_to_Test/test.cmm ./path_to_Test/test.ir #test.cmm是等待被处理的C--代码,test.ir是处理后的中间代码
```

实现思路

基本架构

使用了三个struct来主要存储与ir相关的数据，分别存储中间代码块，单个中间代码，单个运算符，并在Intercode中存储每条中间代码类型，具体代码如下：

```
struct InterCode{
    enum { LABEL, FUNCTION, ASSIGN, PLUS, MINUS, STAR, DIV,
    GET_ADDR, READ_ADDR, WRITE_ADDR,
    IF_GOTO, GOTO, RETURN, DEC, ARGS, CALL, PARAM,
    READ, WRITE} kind;
    char* op1, *op2, *result;
    InterRelop* relop;
    int size;
    InterCode* prev;
    InterCode* next;
};

struct InterCodeList{
    InterCode* head;
    InterCode* tail;
};

struct Offset{
    InterCodeList* code;
    Type* ty;
};
```

除了和ir直接生成有关的translate函数，我还提供了一些封装函数，具体代码如下：

```
char* new_temp();
char* new_label();

InterCodeList* empty_InterCodeList();
InterCode* newintercode(int kind, ...);
InterCodeList* getInterCodewrapped(InterCode* code);
void appendInterCode(InterCodeList* codes, InterCode* code);
void appendInterCodeList(InterCodeList* codes, InterCodeList* codes2);

int mulog10_int(int x);
void writeonParam(char *name);
bool isonParam(char *name);

void printInterRelop(InterRelop* relop, FILE* file);
void printInterCode(InterCode* code, FILE* file);
void printInterCodeList(InterCodeList* codes, FILE* file);
```

实现了代码创建，代码插入，变量创建，标号创建等函数封装，通过使用此处的封装和复用syn_hash_list中的封装提高代码复用性。

中间代码生成过程

主要思路是先进行一次语义分析，在过程中生成IR，然后再调用printInterCodeList输出结果，对于大部分已给出的翻译方案直接翻译即可，对于未给出但是需要完成的额外任务struct此处进行思路说明：

在syn_hash_list修改了一些原有的结构体：在Type上加入structure_size存储结构体的字节数，并且在Type内部原有的存储array的struct中加入element_size统计数组字节数，在FieldList上加入offset存储偏移量。

对于另一个任务的多维数组直接输出报错信息并且返回，对于结构体主要处理过程在Exp中，产生式为`Exp->Exp DOT ID`，处理方式与数组`Exp->Exp LB Exp RB`类似，进行合并处理，主要思路是计算偏移之后处理数组和结构体中的每个元素，我实现了translateExpOffset函数，主要作用是递归地累加偏移至place项，并最终返回累加所需的IR和当前的Type，当遇到左值为ID时返回并在返回途中完成IR拼接。对于offset的计算以上两者比较类似，下面展示结构体的translateExpOffset：

```
if(DOT_ != NULL){ //Exp DOT ID
        Offset info = translateExpOffset(now->down, place);
        assert(info.ty->kind == STRUCTURE);
        FieldList* field = info.ty->structure;
        while(strcmp(field->name, now->down->next->next->text)){
            field = field->tail;
        }
        char* fieldoffset = malloc(mylog10_int(field->offset) + 2);
        memset(fieldoffset, 0, sizeof fieldoffset);
        fieldoffset[0] = '#';
        sprintf(fieldoffset + 1, "%d", field->offset);
```

```
        InterCode* code1 = newintercode(PLUS, place, place, fieldoffset);
        appendInterCode(info.code, code1);
        info.ty = field->type;
        return info;
}
```

在代码完成过程中遇到的困难主要有以下：结构体作为函数参数时代码经常报错，具体为IR中经常有中间变量前面有错误的&，后来改进了ParamTable用来统计已经出现的结构体类型，在遇到ID时查询，如果在表中则直接调用ASSIGN生成IR，解决了这个问题。因为我在结构体作为函数参数时传入的是地址，不需要额外进行取地址操作。