Report

11910501 BaoZhiyuan 11910934 WuZemin 11810925 ZhouYicheng

1.Overview

In this project, we implement a compiler that will generate a particular intermediate representation (IR) for a given source program. And our compiler can translate the definition and the usage of n-D array.

2. IR data structure

The generated IRs are stored in memory using the data structure defined in `inter_code.h`. It is like this:

```
struct operand {
    enum { VARIABLE, CONSTANT, ... } kind;
    union {
        char* var_name;
        int value;
        ...
    } u;
}
struct inter_code {
    enum { ASSIGN, ADD, SUB, ... } kind;
    union {
        struct { operand right, left; } assign;
        struct { operand result, op1, op2; } binop;
        ...
    } u;
    inter_code* prev;
    inter_code* next;
}
```

It is just the enumeration of all kinds of three address code. In addition, to support the code catenate function, two pointers `prev` and `next` are added to make the inter_code doubly linked list. Therefore, we only need to traverse the linked list when printing the codes.

3.Translate schema

3.1 Basic translate schema

Most of the translate grammar flow the instruction in the released pdf file. And we add some other grammar to translate the following situation:

```
 #variable dec with assign
int num = 0, i = 1, k = 1;
#function dec
int sqr(int i1){
    return i1*i1;
}
```

```
translate_DefList(DefList)
    if Dec in the subtree of the DefList:
         code = code+ translate(Dec)

translate_Dec(dec)
     case vardec:
         return translate_VarDec(vardec)
     case vardec assign Exp:
         return translate_exp(Exp,vardec.id)

translate_VarDec(vardec_node){
     while( vardec1 LB INT RB)
         size *= INT
         vardec_node = vardec1
     return [DEC vardec_node.id size]
```

```
translate_FunDec(Fundec):
    case: ID LP RP
        return [FUNCTION ID:]
    case: ID LP VarList RP
        code1 = [FUNCTION ID:]
        code2 = translate_varlist(Varlist)
translate_varList(VarList)
     "get the id of param1"
     code1 = [param1 id size]
     if (VarList == ParamDec1 COMMA VarList2):
            return code1 + translate_varList(VarList)
```

3.2 Advance translate

We add a function that will calculate the addr of the place in a array recursively.

```
translate_exp_addr( exp, addr, base)
    case ID:
         return [addr := ID]
    case Exp1 LB Exp2 RB
         t1,t2,a1 = new place
         code1 = translate_exp(Exp2,t1)
         code1 = [t2 := t1*base]

         base *= symbol_tab.look(Exp1).size
         code2 = translate_addr_exp(Exp1,a1,base)
         code3 = [addr := a1 + t2 ]
         return code1+code2+code3
```

And we find out that only two way that will use the array or assign it, so we add following code.

```
translate_exp:
   case exp1 assign exp2
       switch exp1
       case exp3 [exp4]:
           t1,t2 = new place
           code1 = translate_exp_addr(exp1,t1,4)
           code2 = translate_exp(exp2,t2)
           code3 = [*t1 = t2]
       return code1+code2+code3
   case exp1[exp1]
       t1 = new place
       code1 = translate_exp_addr(exp1,t1,4)
       code2 = [place := *t1]
       return code1+code2
```