11712639-Project2-Report

1.Basic Requirement

1.1 Implements

Based on the syntax tree built in previous project, tranverse this tree to do semantic analysis. The main idea of traversal is to implement `visit` functions. It point out what to do when traverse different tree node. (for example:)

```
// when meet node VarList
// it has two productions: ParamDec | ParamDec COMMA VarList
// so it will invoke visit_ParamDec() and visit_VarList()
FieldList* visit_VarList(Node *node){
    FieldList *list = create_FieldList();
    list = visit_ParamDec(node->children[0]);
    if (node->childrenNum == 3){
        list->next = visit_VarList(node->children[2]);
    }
    return list;
}
```

My implementation for subsequent errors:

For an undefined identifier, it type is `UNKNOWN`. Any operation on unknow type identifier is valid.

```
for example: an undefined identifier a
a.size
a()
a[2]
return a
int b = a
a + 1
...
```

For all above cases, my semantic analyser only report `type1 error`, since a's type is unknown and maybe it can be defined by user with correct type later. Only report type1 error to user is enough, I think. If user later defines it as a wrong type, then question turns into another case that type checking handles.

1.2 Self-written test cases

1. Continuous definitions (eg. int a, b[2], c;) and array definition.
2. Nested structure definition.
3. Information of firstly defined variable is stored and can't be covered later when redifined variable occours.
4. Funtion components should be checked even it is redefined function.
5. Check return type.
6. Functions call each other.
7. Cases of rvalues on the left of "=".
8. Both function and variable are undefined when function calls.
9. Structure components should be checked even it is redefined structure.

10. Structure oprands.

2. Bonus features

revoke Assumption 6

2.1 Implements

Main idea: functional- style. According to project file:

> Under such implementation, we organize all these symbol tables into a scope stack, the innermost scope is stored at the top of the stack, with the next containing scope that is underneath it, etc.. When a new scope is opened, a new symbol table is created and the variables declared in that scope are inserted into the new table. We then push the symbol table on the scope stack. When a scope is closed, the top symbol table is popped. To find a symbol, we start at the top of the stack and work our way down until we find it. If we do not find it, the variable is not accessible and an error should be generated.

When to insert a new symbol table into to stack?

- enter a program (global scope)
- a function define
- structure define
- if, else, while blocks

code segment: (visit.c)

```
void visit_Program(Node *node, FILE* fp1){
    ...
    printf("visit_Program\n");
    scope_stack = stack_init();
    push(scope_stack, symtab_init());
    visit_ExtDefList(node->children[1]);
    ...
}
TypeTuple* visit_StructSpecifier(Node *node){
    printf("visit_StructSpecifier\n");
    if (node->childrenNum == 5)
    {
        ...
        printf("after insert struct name, create new table\n");
        push(scope_stack, symtab_init());//再建一个table，结构体里的scope
        ...
    }
    return create_TypeTuple(STRUCTURE, node->children[1]->text);
}
IDType* visit_FunDec(Node *node, TypeTuple *type){
    printf("visit_FunDec\n");
    ...
    printf("after insert func name, create new table\n");
    push(scope_stack, symtab_init());
    ...
    return id;
}
```

```
void visit_Stmt(Node *node, IDType* func){
    printf("visit_Stmt\n");
    if (node->children[0]->isTerminal)//retrun, if, while
    {
        ...
        else//if, while
        {
            printf("enter if or while, create new table\n");
            visit_Exp(node->children[2]);
            push(scope_stack, symtab_init());
            ...
            if (node->childrenNum == 7)//带有else的情况
            {
                printf("enter else, create new table\n");
                push(scope_stack, symtab_init());
                ...
}
```

When to insert a new id into a symbol table?

- global definiton
- function parameters
- function decleared list
- structure decleared list
- decleared list in block ({})

When to pop a symbol table?

- after structure define
- after function define
- after if, else, while block
- end of program

code segment:

```
void visit_ExtDef(Node *node){
    ...
            IDType *func = visit_FunDec(node->children[1], type);
            visit_CompSt(node->children[2], func);
            printf("after visit func, pop table\n");
            pop(scope_stack);
    ...
}
TypeTuple* visit_StructSpecifier(Node *node){
    if (node->childrenNum == 5)
    {   ...
        printf("after visit struct, pop table\n");
        pop(scope_stack);//当结构体建立完成就pop最顶部的table
    }
    ...
}
void visit_Stmt(Node *node, IDType* func){
    ...
            printf("after if or while scope, pop table\n");
            pop(scope_stack);
            if (node->childrenNum == 7)//带有else的情况
            {
                ...
```

```
                printf("after else scope, pop table\n");
                pop(scope_stack);
            }
    ...
}
```