Project3. Semantic analyzer

2015004302 CSE 곽상원

목차

---

1. Compile 환경

- Ubuntu14.04, gcc, flex, yacc
- make command를 입력해 “cminus_semantic” 빌드 및 생성

2. Project purpose

- 이전 project에서 만든 parser를 바탕으로 semantic analyzer를 구현
- **Semantic analyzer: scope, type check을 통해 의미상 오류를 찾아낸다.**

3. 기본설정

- Identifier는 function과 variable을 합쳐서 지칭한다.
- Type은 void,int,int[] 3종류뿐이다.
- Void type은 function에만 사용가능하다.
- int[] type variable에 대한 사칙연산은 불가능하다.
- “global” scope가 root scope가 되어, 전역변수 및 함수는 global의 scope에 포함된다.
- int input(void), void output(int) 이 두 개의 함수는 기본적으로 내장된 함수이다.

4. semantic error 관리

▶symbol table 생성

- parser를 통해 만들어진 AST를 traverse하면서 각 노드에 맞는 action을 통해 symbol table을 생성한다. 생성된 형태는 다음과 같이 각 identifier의 Name, Type, Scope name, Location(해당 scope에서 몇 번째로 table에 들어갔는지), Line Numbers(해당 identifier가 나오는 모든 line) 정보를 출력한다. 이때 함수의 Line Numbers는 declaration의 경우 중괄호가 닫히는 위치이다.

```
< Symbol Table >
Variable Name Variable Type  Scope Name  Location   Line Numbers
------------- -------------  ----------  --------   ------------
x             Integer        main        0           13   14   15
y             Integer        main        1           13   14   15
main          Function       global      3           16
input         Function       global      0            0   14   14
output        Function       global      1            0   15
gcd           Function       global      2            9    7   15
u             Integer        gcd         0            4    6    7    7
v             Integer        gcd         1            4    6    7    7    7
```

symbol table 구현은 hash table과 linked list방식을 이용했다. ScopeList란 하나의 scope 안에 BucketList를 추가하고, ScopeList에 parent를 정해서 hierarchy 구조를 유지했다. 그리고 이 ScopeList를 관리하는 변수가 scopeTable이다.

insertNode함수에서 노드의 종류에 대한 action은 다음과 같다.

1. **FunK(함수선언)**: global scope내에 현재 name에 맞는 BucketList를 추가한다. 또한 name에 맞는 새로운 scope를 생성하고 함수의 반환형 등등의 정보를 넣어준다.

```
case FuncK:
    bucket = st_lookup("global",t->attr.name);
    if(bucket != NULL){
        sprintf(message_box,"Redefined function(%s)",t->attr.name);
       pre_Error(t,message_box);
    }
    st_insert(scope,t->attr.name,t->type,t->lineno,location++);
    scope_make(t->attr.name,scope);
    strcpy(scope,t->attr.name);//scope change
    cur_loc = location;
    cur_num = number;
    location = 0;
    number = -1;
    break;
```

2. **VarK, VararrK(변수선언)**: 현재 scope에서 name에 맞는 BucketList를 추가한다.

```
case VarK:
case VararrK:
    bucket = st_lookup(scope,t->attr.name);
    if(bucket != NULL){
        sprintf(message_box,"Redefined variable(%s)",t->attr.name);
        pre_Error(t,message_box);
    }
    st_insert(scope,t->attr.name,t->type,t->lineno,location++);
    cur_loc = location;
    cur_num = number;
    break;
default:
    break;
```

3. **ParK(함수 선언시의 parameter)**: 현재 scope에서 name에 맞는 BucketList를 추가하고, 함수의 argument정보에 자신의 정보를 추가한다.

```
case SinpK:
    if(t->type == Void){
        cur_loc = location;
        break;
    }
    st_insert(scope,t->attr.name,t->type,t->lineno,location++);
    Parm_add(scope,t,location-1);
    cur_loc = location;
    break;
```

4. **CmstmtK(compound statement)**: 함수 선언 시에 쓰이는 중괄호 이외의 경우, scope를 생성해준다.

```
case CmstmtK:
    if(number != -1){
    char* temp_child =(char*)malloc(sizeof(char)*(strlen(scope)+10));
    sprintf(temp_child,"%s.%d",scope,number++);
    scope_make(temp_child,scope);
```

5. **CallK(함수호출), IdK(선언된 변수이용)**: 현재 위치를 선언된 변수의 BucketList Line Numbers에 추가한다.

```
st_insert("global",t->attr.name,t->type,t->lineno,0);
```

▶Error check

- **error check는 symbol table을 생성하면서 하는 것과 symbol table을 다 생성하고 하는 것으로 나누었다.**
- **<u>같은 line에서는 최초 1회의 Error만 출력한다.</u> 이미 이전에 error가 발생한 line에 대해 error check는 더 이상의 의미를 갖지 못하기 때문이다.**

1. Symbol table을 생성하면서 Error check

① **"Redefined function(name)"**: 이미 앞에 같은 name의 identifier가 있는 상황에서 함수 선언 시에 띄워지는 문구이다.
② **"Redefined variable(name)"**: 같은 scope 내에서, 이미 앞에 같은 name의 identifier가 있다면 변수 선언 시에 띄워지는 문구이다.

2. Symbol table을 생성하고 Error check

① **"Void is not invalid type in variable declaration(name)"**: 변수가 void형으로 선언되면 나타난다.
② **"Void is not invalid type in parameter(name)"**: 함수선언 시에 argument가 void형이면 나타난다.
③ **"Return type Error"**: 현재 함수의 반환형과 'return'뒤에 오는 data의 type이 맞지않으면 나타난다.
④ **"'return' with a value, in function returning void"**: 반환형이 void인 함수에서 'return' 뒤에 data가 있다면 나타난다.
⑤ **"Inside [] type error"**: []내에는 int type만 가능한데, 그 이외의 type이 들어올 경우 나

타난다.
⑥ “(name) should be IntegerArray”: a[3]과 같은 형태에서 a가 int[] type이 아닌 경우에 나타난다.
⑦ “Undefined variable(name)”: 앞서 선언되지 않은 변수를 사용하면 나타난다.
⑧ “Types of operands are not equal”: operator에 대해 두 피연산자의 type이 다르면 나타난다.
⑨ “Arithmetic operation on IntegerArray type!”: int[]형 변수에 대해 사칙연산이 있을 시에 나타난다.
⑩ “Undefined function(name)”: 선언되지 않은 함수를 호출 시에 나타난다.
⑪ “Argument number is not equal”: 함수호출 시에 argument수가 동일하지 않으면 나타난다.
⑫ “Argument type error”: 함수호출 시에 argument type이 동일하지 않으면 나타난다.
⑬ “Assign type is not equal”: assign operator의 피연산자 type이 동일하지 않으면 나타난다.

5. 실행예제 및 결과

< cminus_semantic 빌드 >

```
ksw@ubuntu:~/2019_ELE4029_2015004302/3_Semantic/source$ make
yacc -d -v cminus.y
gcc  -c main.c
gcc  -c util.c
flex cminus.l
gcc  -c lex.yy.c
gcc  -c y.tab.c
gcc  -c symtab.c
gcc  -c analyze.c
gcc  -c code.c
gcc  main.o util.o lex.yy.o y.tab.o symtab.o analyze.o code.o -o cminus_semantic -lfl
```

< test1.c >

▶test file

```
1 int temp;
2
3 int gcd(int a){
4     int c[3];
5     a + b;
6     return c;
7 }
8
```

▶ 실행결과

```
ksw@ubuntu:~/2019_ELE4029_2015004302/3_Semantic/source$ ./cminus_semantic tes
t1.c

C-MINUS COMPILATION: test1.c

Building Symbol Table...

Symbol table:

< Symbol Table >
Variable Name  Variable Type  Scope Name  Location  Line Numbers
-------------  -------------  ----------  --------  ------------
temp           Integer        global      2         1
input          Function       global      0         0
output         Function       global      1         0
gcd            Function       global      3         7
a              Integer        gcd         0         3  5
c              IntegerArray   gcd         1         4  6

Checking Types...
Error at line 5: Undefined variable(b)
Type error at line 6: Return type Error

Type Checking Finished
```

< test2.c >

▶test file

```
1 int arr[3];
2
3 void main(void){
4     int a;
5     int b;
6
7     output(arr[1]);
8     return a;
9 }
```

▶실행결과

```
ksw@ubuntu:~/2019_ELE4029_2015004302/3_Semantic/source$ ./cminus_semantic test2.c

C-MINUS COMPILATION: test2.c

Building Symbol Table...

Symbol table:

< Symbol Table >
Variable Name  Variable Type  Scope Name  Location  Line Numbers
-------------  -------------  ----------  --------  ------------
a              Integer        main        0         4  8
b              Integer        main        1         5
main           Function       global      3         9
input          Function       global      0         0
output         Function       global      1         0  7
arr            IntegerArray   global      2         1

Checking Types...
Type error at line 8: 'return' with a value, in function returning void

Type Checking Finished
```