Project #3. Semantic Analysis

2016025041 하태성

- Compilation Environment
  - macOS Monterery 12.2.1(M1 Pro 16GB)
  - clang : Apple clang version 13.1.6 (clang-1316.0.21.2.5)
  - flex : flext 2.6.4 Apple(flex-34)

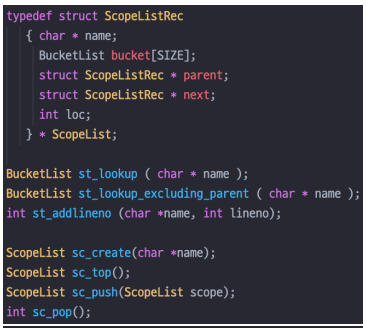
- C-Minus Semantic Analyzer Implementation
  - main.c
    - main.c 파일은 이전 프로젝트와 같이 flag만 수정하고 debugging을 할때 NO_PARSE / NO_ANALYZE flag만 변경하면서 사용하였습니다.
  - globals.h
    - global.h에서 type checking을 위한 ExpType에 Array를 추가하였고, Treenode structure안에 scope structure를 추가하였습니다. function call이 이루어지면서 이전 scope의 정보나 parameter 등을 어떻게 관리할지 여러 방법을 시도해보다 이런 방식으로 결정하였습니다.

```c
/* ExpType is used for type checking */
typedef enum {Void,Integer,Array} ExpType;

#define MAXCHILDREN 3

typedef struct treeNode
   { struct treeNode * child[MAXCHILDREN];
     struct treeNode * sibling;
     int lineno;
     NodeKind nodekind;
     union { StmtKind stmt; ExpKind exp; DecKind dec; } kind;
     union { TokenType op;
             int val;
             char * name; } attr;
     ExpType type; /* for type checking of exps */
     struct ScopeListRec * scope;
   } TreeNode;
```

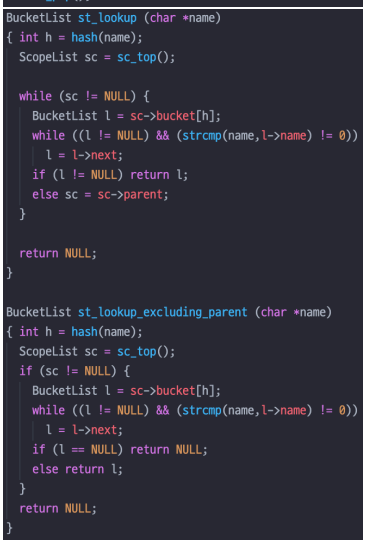
  - symtab.h & symtab.c
    - 이번 프로젝트에서 핵심 2가지 중 하나였습니다. symtab.c에서는 symbol table을 구성하는 table들과 symbol table에 사용되는 interface function들이 있습니다.
    - 먼저 기존에 주어진 함수와 구조를 수정하고 새로 scope를 관리하기 위한 scopetable을 추가하였습니다. 그리고 scope/function call은 실제 메모리에서도 stack에서 실행되고 FILO 이기때문에 마찬가지로 stack처럼 사용할 수 있게 sc_push(), sc_pop(), sc_top() 함수도 같이 구현하였습니다.
    - 그리고 주어진 예시와 비슷하게 scope를 올라가면서 탐색하는 st_lookup(), 현재 scope 내에서만 탐색하는 st_lookup_excluding_parent() 를 구현하였습니다. 그외엔 st_insert에 있는 이미 해당 name의 symbol이 존재할때 lineno를 추가해주는 부분이 제대로 작동하지 않아 해당 부분을 따로 밖으로 빼서 addlineno() 함수를 만들어주었습니다.
    - main.c에서 Traceanalysis 플래그를 켰을때 symbol table을 출력하는 printSymTab() 함수는 주어진 test result 처럼 모든 table을 출력하기보다 적당히 symbol table이 잘만들어지고 추가가 되었나를 확인 할 수 있을 정도로만 수정해주었습니다.



```c
typedef struct ScopeListRec
   { char * name;
     BucketList bucket[SIZE];
     struct ScopeListRec * parent;
     struct ScopeListRec * next;
     int loc;
   } * ScopeList;

BucketList st_lookup ( char * name );
BucketList st_lookup_excluding_parent ( char * name );
int st_addlineno (char *name, int lineno);

ScopeList sc_create(char *name);
ScopeList sc_top();
ScopeList sc_push(ScopeList scope);
int sc_pop();
```



```c
BucketList st_lookup (char *name)
{ int h = hash(name);
  ScopeList sc = sc_top();

  while (sc != NULL) {
    BucketList l = sc->bucket[h];
    while ((l != NULL) && (strcmp(name,l->name) != 0))
      l = l->next;
    if (l != NULL) return l;
    else sc = sc->parent;
  }

  return NULL;
}

BucketList st_lookup_excluding_parent (char *name)
{ int h = hash(name);
  ScopeList sc = sc_top();
  if (sc != NULL) {
    BucketList l = sc->bucket[h];
    while ((l != NULL) && (strcmp(name,l->name) != 0))
      l = l->next;
    if (l == NULL) return NULL;
    else return l;
  }
  return NULL;
}
```

- analyze.c
  - analyze.c에서는 본격적으로 parser에서 가져온 AST에서 노드를 가져와 insertNode() 함수를 통해 symbol table에 먼저 쭉 넣습니다. 그 후 typecheck() 함수를 통해 다시 syntax tree를 traverse하면서 semantic error가 있는지 check하게됩니다.
  - built-in function은 AST에서 define되지않기때문에 insertBuiltInFunNode() 함수를 통해 Treenode를 만들어서 insert(), output() 함수에 해당하는 definition node를 symbol table에 추가해주었습니다.
  - insertNode() 함수에서는 syntax tree를 traverse하면서 노드에서 각 노드의 종류와 이름에 따라 symbol table에 추가하거나 이미 있을경우 lineno만 추가하거나 에러처리를 해줍니다. 해당 과정에서 가장 까다로웠던 것은 function에 따른 scope관리와 parameter관리였습니다.
  - scope는 global variable을 만들어 이전 scope 이름과 location을 저장하는 방식으로 구현하였습니다. 이런 방식을 선택할 경우 간단한 프로그램에서는 정상적으로 작동하나 nested function call에선 문제가 발생할 수 있을 거라고 생각을 해서 stack에 해당 정보를 저장하는 방식을 생각했으나 시간이 부족해 구현은 못하였습니다. scope가 기존의 tiny에서 추가되었기때문에 nullproc()은 삭제하고 해당 함수대신 sc_pop하고 트리에서 부모의 scope로 돌아가는 함수를 추가하였습니다.
  - parameter는 Scopelistrec 구조체에 paramlist 구조체를 새로 만들어 추가하는 방식으로 구현을 시도하였으나 현재는 그냥 treenode 자체를 scopelist 안에 추가해 child를 확인하는 방식으로 변경하였습니다. 이러한 방식이 parameter를 따로 관리하는 것보다 메모리를 많이 차지하는 단점이 있을거라 생각하지만 이번 과제에는 그러한 제한은 없고 node를 직접 넣으면 앞서 parser에서 다루던 방식 그대로 parameter를 확인 할 수 있어 보다 직관적이였기때문에 이러한 방식을 선택하였습니다.
  - checknnode()에서는 다시한번 traverse를 하면서 기존의 symbol table을 사용하여 주어진 error_message에 해당하는 case가 있는지를 확인합니다. 여기서도 function과 paramater와 관련된 부분이 가장 까다로웠습니다.
  - function과 관련해서 function call을 하는 callK노드를 처리할때 parameter가 type과 갯수가 맞는지를 확인하는 부분이 상당히 어려웠습니다. 어느정도 구현은 하였으나 솔직히 완벽하게 function call에 대한 semantic check를 하는지는 확실치 않습니다.
  - 그리고 function call에 대한 return부분도 처리가 쉽진 않았는데 return을 처리하는 것이 결국 이전 scope의 정보를 올바르게 저장하고 똑바로 가져올 수 있는가에 대한 문제였고 이전에 scope 처리를 생각한 구현을 완성까진 못하였기때문에 아마 다른 테스트 케이스에서는 버그가 발생할 수 있지않을까 생각합니다.

- Result & Conclusion
  - 주어진 테스트 케이스는 모두 정상적으로 에러를 검출하는 것을 확인했습니다.
  - 이번 프로젝트는 parser와 연관되어 syntax tree와 treenode을 모두 이해하고 새로 scope를 bucketlist에 추가하여 구현하여야해서 상당히 까다로웠던 것 같습니다.
  - 감사합니다.