컴파일러설계 프로젝트2 (Parser) 보고서

2019019016 서시언

1. Compilation method and environment

- Ubuntu 20.04.3 LTS

2. Explanation about how to implement and how to operate

- 제공된 Makefile 사용

[main.c]

- 다음과 같이 설정한다.

```
/* set NO_PARSE to TRUE to get a scanner-only
#define NO_PARSE FALSE
/* set NO_ANALYZE to TRUE to get a parser-on
#define NO_ANALYZE FALSE

/* set NO_CODE to TRUE to get a compiler tha
 * generate code
 */
#define NO_CODE TRUE
```

```
/* allocate and set tracing flags */
int EchoSource = FALSE;
int TraceScan = FALSE;
int TraceParse = FALSE;
int TraceAnalyze = FALSE;
int TraceCode = FALSE;
```

[symtab.h]

- 함수의 parameter 개수 및 타입 저장을 위한 ParamListRec, Scope정보를 저장하기 위한 ScopeListRec, Scope 관리를 위한 ScopeStackRec을 추가하고, BucketListRec에는 함수의 파라미터 정보를 저장하기 위해 ParamList params와 타입을 저장하기 위한 ExpType type를 추가해준다.

```
typedef struct ParamListRec
   { ExpType type;
     struct ParamListRec * next;
   } * ParamList;

/* The record in the bucket lists for
 * each variable, including name,
 * assigned memory location, and
 * the list of line numbers in which
 * it appears in the source code
 */
typedef struct BucketListRec
   { char * name;
     // add type info
     ExpType type;
     LineList lines;
     ParamList params;
     int memloc ; /* memory location for variable */
     struct BucketListRec * next;
   } * BucketList;

// add scope info
typedef struct ScopeListRec
   { char * name;
     BucketList bucket[SIZE];
     struct ScopeListRec * parent;
     struct ScopeListRec * next;
   } * Scope;

typedef struct ScopeStackRec
   { Scope scope;
     struct ScopeStackRec * next;
     int location;
   } * ScopeStack;
```

[symtab.c]

- scope, name, type, lineno, loc 정보를 받아 해당 scope에 새로운 심볼을 추가

```
void st_insert( Scope scope, char * name, ExpType type, int lineno, int loc )
{ int h = hash(name);
  BucketList l =  scope->bucket[h];
  while ((l != NULL) && (strcmp(name,l->name) != 0))
    l = l->next;
  if (l == NULL) /* variable not yet in table */
  { l = (BucketList) malloc(sizeof(struct BucketListRec));
    l->name = name;
    // save type info
    l->type = type;
    l->lines = (LineList) malloc(sizeof(struct LineListRec));
    l->lines->lineno = lineno;
    l->memloc = loc;
    l->params = NULL;
    l->lines->next = NULL;
    l->next = scope->bucket[h];
    scope->bucket[h] = l; }
  else /* found in table, so just add line number */
    addLineno( lineno, l);
} /* st_insert */
```

- 넘겨받은 scope부터 시작해서 parent scope로 옮겨가며 name에 해당하는 심볼을 찾는다.

```
BucketList st_lookup ( Scope scope, char * name )
{ while (scope)
  { int h = hash(name);
    BucketList l =  scope->bucket[h];
    while ((l != NULL) && (strcmp(name,l->name) != 0))
      l = l->next;
    if (l == NULL) scope = scope->parent;
    else return l;
  }
  return NULL;
}
```

- 넘겨받은 scope에서 name에 해당하는 심볼을 찾는다.

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

- scope를 스택으로 관리하기 위해 push, pop, top 함수를 만든다.

```
Scope scopeList = NULL;
ScopeStack scopeStack = NULL;

Scope push ( char * scope )
{ if (!scopeStack)
  { scopeStack = (ScopeStack)malloc(sizeof(struct ScopeStackRec));
    memset(scopeStack, 0, sizeof(struct ScopeStackRec));
  }
  Scope s = (Scope)malloc(sizeof(struct ScopeListRec));
  memset(s, 0, sizeof(struct ScopeListRec));

  s->name = scope;

  if (scopeList)
  { Scope iter = scopeList;
    while(iter->next) iter = iter->next;
    iter->next = s;
  }
  else scopeList = s;

  ScopeStack topOfStack = top();
  if (topOfStack) s->parent = topOfStack->scope;

  ScopeStack tmp = (ScopeStack)malloc(sizeof(struct ScopeStackRec));
  tmp->scope = s;
  tmp->next = scopeStack->next;
  tmp->location = 0;
  scopeStack->next = tmp;

  return s;
}

ScopeStack top()
{ if (!scopeStack)
  { scopeStack = (ScopeStack)malloc(sizeof(struct ScopeStackRec));
    memset(scopeStack, 0, sizeof(struct ScopeStackRec));
  }
  return scopeStack->next;
}

void pop()
{ if (scopeStack && scopeStack->next)
    scopeStack->next = scopeStack->next->next;
}
```

- 심볼(BucketList) 와 파라미터의 타입을 넘겨받아, 해당 심볼에 parameter 정보를 추가한다.

```
void addParam ( BucketList l, ExpType type )
{ if (!l->params)
  { l->params = (ParamList) malloc(sizeof(struct ParamListRec));
    l->params->type = type;
    l->params->next = NULL;
  }
  else
  { ParamList tmp = l->params;
    while (tmp->next) tmp = tmp->next;
    tmp->next = (ParamList) malloc(sizeof(struct ParamListRec));
    tmp->next->type = type;
    tmp->next->next = NULL;
  }
}
```

[analyze.c]

- traverse에 popStack을 넘겨줘서 scope가 끝나면 pop()을 호출하도록 한다. 또한 해당 현재 scopeName을 현재 scopeName의 parent로 업데이트 해준다.

```
void buildSymtab(TreeNode * syntaxTree)
{ init();
  traverse(syntaxTree,insertNode,popStack);
  if (TraceAnalyze)
  { fprintf(listing,"\nSymbol table:\n\n");
    printSymTab(listing);
  }
}
```

```
static void popStack(TreeNode * t)
{ if (t->nodekind == StmtK)
  { if (t->kind.stmt == CompoundK) pop();
    else if (t->kind.stmt == FuncK) scopeName = t->scope->parent->name;
  }
}
```

- 또한 buildSymtab의 처음에 init()dmf ghcnfgo output, input 함수를 테이블에 추가해준다.

```
// initialize scope stack (insert input, output to global)
static void init()
{ Scope global = push(scopeName);
  ScopeStack topOfStack = top();

  st_insert(global, "input", IntFunc, 0, topOfStack->location++);
  st_insert(global, "output", VoidFunc, 0, topOfStack->location++);

  Scope output = push("output");
  topOfStack = top();
  st_insert(output, "value", Int, 0, topOfStack->location++);
  addParam(st_lookup(global,"output"),Int);
  pop();
}
```

- insertNode에서는 stack의 top에 있는 scope를 찾아 선언되는 변수를 해당 scope에 저장하며, 변수 선언 시에는 해당 scope에 같은 이름의 심볼이 이미 있는지 확인해서 redefine에러를 찾는다. 또한 compound statement가 나오면 scope를 새로 푸시해준다.

```
static void insertNode( TreeNode * t)
{ ScopeStack topOfStack = top();
  Scope scope = topOfStack->scope;
  BucketList l;

  switch (t->nodekind)
  { case StmtK:
      switch (t->kind.stmt)
      { case CompoundK:
          if (!isFunction) t->scope = push(scopeName);
          isFunction = 0;
          break;
        case VarDeclK:
          // not in scope
          if (st_lookup_excluding_parent(scope, t->attr.name) == NULL)
          { // void varable
            if (t->type == Void)
            { fprintf(listing, "Error: Variable Type cannot be void at line %d (name : %s)\n", t->lineno, t->attr.name);
              Error = TRUE;
              st_insert(scope, t->attr.name, SemanticError, t->lineno, topOfStack->location++);
            }
            else st_insert(scope, t->attr.name, Int, t->lineno, topOfStack->location++);
          }
          // already defined
          else
          { fprintf(listing, "Error: redefined symbol '%s' at line %d\n", t->attr.name, t->lineno);
            Error = TRUE;
          }
          t->scope = scope;
          break;
        case VarArrDeclK:
          // not in scope
          if (st_lookup_excluding_parent(scope, t->attr.name) == NULL)
          { // void varable
            if (t->type == Void)
            { fprintf(listing, "Error: Variable Type cannot be void at line %d (name : %s)\n", t->lineno, t->attr.name);
              Error = TRUE;
              st_insert(scope, t->attr.name, SemanticError, t->lineno, topOfStack->location++);
            }
            else st_insert(scope, t->attr.name, IntArr, t->lineno, topOfStack->location++);
          }
```

- 파라미터의 경우 해당 function을 찾아 function에 parameter정보를 추가해준다.

```
case ParamK:
  // not in scope
  l = st_lookup_excluding_parent(scope->parent, scope->name);
  if (st_lookup_excluding_parent(scope, t->attr.name) == NULL)
  { // void varable
    if (t->type == Void)
    { fprintf(listing, "Error: Parameter Type cannot be void at line %d (name : %s)\n", t->lineno, t->attr.name);
      Error = TRUE;
      st_insert(scope, t->attr.name, Void, t->lineno, topOfStack->location++);
      addParam(l, Void);
    }
    else
    { st_insert(scope, t->attr.name, Int, t->lineno, topOfStack->location++);
      addParam(l, Int);
    }
  }
  // already defined
  else
  { fprintf(listing, "Error: redefined symbol '%s' at line %d\n", t->attr.name, t->lineno);
    Error = TRUE;
  }
  t->scope = scope;
  break;
```

- 변수를 사용하거나, 함수를 call하는 경우 lineno를 추가해준다.

```
case ExpK:
  switch (t->kind.exp)
  { case VarK:
    case CallK:
      // if already defined in scope, just add lineno
      if ((l = st_lookup(scope, t->attr.name)) != NULL)
      { addLineno(t->lineno, l);
        t->scope = scope;
      }
      break;
    default:
      t->scope = scope;
      break;
  }
```

- checkNode에서는 각각의 expression에서 알맞은 타입이 사용되었는지 체크하고, 에러를 출력하며, expression의 type을 업데이트 해준다.

```
static void checkNode(TreeNode * t)
{ BucketList l;
  char* funcName;
  switch (t->nodekind)
  { case ExpK:
      switch (t->kind.exp)
      { case OpK:
          // child's type must be int
          if ( t->child[0]->type != Int || t->child[1]->type != Int )
            typeError(t, "invalid expression");
          else t->type = Int;
          break;
        case ConstK:
          t->type = Int;
          break;
        case VarK:
          if ( (l = st_lookup(t->scope, t->attr.name))==NULL || l->lines->lineno > t->lineno )
          { fprintf(listing, "Error: Undeclared Variable '%s' at line %d\n", t->attr.name, t->lineno);
            Error = TRUE;
            t->type = SemanticError;
          }
          else
          { if (t->child[0] && t->child[0]->type != Int)
            { fprintf(listing, "Error: Invalid array indexing at line %d (name: '%s'). Indices should be integer\n", t->lineno, t->attr.name);
              Error = TRUE;
            }
            else if (!t->child[0]) t->type = l->type;
            else t->type = Int;
          }
          break;
        case AssignK:
          if ( (l = st_lookup(t->child[0]->scope, t->child[0]->attr.name))==NULL || l->lines->lineno > t->child[0]->lineno )
          { fprintf(listing, "Error: Undeclared Variable '%s' at line %d\n", t->attr.name, t->lineno);
            Error = TRUE;
            t->type = SemanticError;
          }
          else if ( t->child[0]->type != t->child[1]->type )
          { fprintf(listing,"Error: Assginment type error at line %d (name: '%s')\n", t->lineno, t->child[0]->attr.name);
            Error = TRUE;
            t->type = SemanticError;
          }
          else t->type = t->child[0]->type;
          break;
```

[실행]

make

./cminus_semantic [input_file]

3. Example and Result ScreenShot

[type_error.txt]

```
siun@siun-QEMU-Virtual-Machine:~/2021_ele4029_2019019016/3_Semantic$ cat type_error.txt
int main(void)
{
        int x;
        int y[3];

        x+y;

        return 0;
}
siun@siun-QEMU-Virtual-Machine:~/2021_ele4029_2019019016/3_Semantic$ make
make: 'all'을(를) 위해 할 일이 없습니다.
siun@siun-QEMU-Virtual-Machine:~/2021_ele4029_2019019016/3_Semantic$ ./cminus_semantic type_error.txt

C-MINUS COMPILATION: type_error.txt
Error: Type error at line 6: invalid expression
```

[void_var.txt]

```
siun@siun-QEMU-Virtual-Machine:~/2021_ele4029_2019019016/3_Semantic$ cat void_var.txt
int main(void)
{
       void x;
       return 0;
}
siun@siun-QEMU-Virtual-Machine:~/2021_ele4029_2019019016/3_Semantic$ ./cminus_semantic void_var.txt

C-MINUS COMPILATION: void_var.txt
Error: Variable Type cannot be void at line 3 (name : x)
```

[func.txt]

```
siun@siun-QEMU-Virtual-Machine:~/2021_ele4029_2019019016/3_Semantic$ cat func.txt
int x(int y)
{
        return y+1;
}

int main(void)
{
        int a;
        int b;
        int c;

        return x( a, b, c);
}
siun@siun-QEMU-Virtual-Machine:~/2021_ele4029_2019019016/3_Semantic$ ./cminus_semantic func.txt

C-MINUS COMPILATION: func.txt
Error: Parameter error at line 12: invalid function call (name: 'x')
```

[undeclare.txt]

```
siun@siun-QEMU-Virtual-Machine:~/2021_ele4029_2019019016/3_Semantic$ cat undeclare.txt
int main(void)
{
        return x;
}
siun@siun-QEMU-Virtual-Machine:~/2021_ele4029_2019019016/3_Semantic$ ./cminus_semantic undeclare.txt

C-MINUS COMPILATION: undeclare.txt
Error: Undeclared Variable 'x' at line 3
Error: Type error at line 3: invalid return type
```

[indexing.txt]

```
siun@siun-QEMU-Virtual-Machine:~/2021_ele4029_2019019016/3_Semantic$ cat indexing.txt
int main(void) {
        int x[5];
        x[output(5)] = 3 + 5;

        return 0;
}
siun@siun-QEMU-Virtual-Machine:~/2021_ele4029_2019019016/3_Semantic$ ./cminus_semantic indexing.txt

C-MINUS COMPILATION: indexing.txt
Error: Invalid array indexing at line 3 (name: 'x'). Indices should be integer
```

[condition.txt]

```
siun@siun-QEMU-Virtual-Machine:~/2021_ele4029_2019019016/3_Semantic$ cat condition.txt
int main(void) {
        if(output(5)) {}
        return 0;
}
siun@siun-QEMU-Virtual-Machine:~/2021_ele4029_2019019016/3_Semantic$ ./cminus_semantic condition.txt

C-MINUS COMPILATION: condition.txt
Error: Invalid condition at line 3
```