C-Minus Compiler Implementation - 2_Parser

2018008531 송연주

Compilation method and environment

실행환경 : VMWare Workstation 16 Player, Ubuntu 18.04.5 LTS

컴파일 및 실행 방법

```
$ yacc -d cminus.y
$ make
$ ./cminus_parser ./test.1.txt
```

Explanation about how to implement and how to operate & Some explanation about the modified code

1. `Makefile`, `main.c` : 제공받은 Makefile을 사용했으며 `printTree()`에서 parsing 결과를 출력할 때 'C-MINUS COMPILATION'을 출력하도록 `main.c`를 수정했다.

2. `globals.h`

   `cminus.y`에서 parsing을 하기 위한 enum type들을 새롭게 정의하였다. 또한 `ExpType`에서 사용되지 않는 타입인 Boolean을 제거하였다. 그리고 array를 표현하는데에 사용될 `ArrayAttr`구조체를 새롭게 정의하였다. 그리고 `TreeNode`구조체가 위에서 정의한 enum type들과 `ArrayAttr`을 갖고 있도록 추가하고 parsing에 필요한 여러 변수들을 `attr`에 추가하였다.

```c
typedef enum {StmtK,ExpK,DeclK,ParamK} NodeKind;
typedef enum {IfNEK,IfEK,WhileK,ReturnK,CompK} StmtKind;
typedef enum {AssignK,OpK,ConstK,CallK,IdK,ArrIdK} ExpKind;
typedef enum {VarK,ArrVarK,FunK,VoidK,IntK} DeclKind;
typedef enum {ArrParamK,NArrParamK} ParamKind;

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

#define MAXCHILDREN 3

typedef struct arrayAttr
  { char* name;
    int size;
  } ArrayAttr;

typedef struct treeNode
   { struct treeNode * child[MAXCHILDREN];
     struct treeNode * sibling;
     int lineno;
     NodeKind nodekind;

     union { StmtKind stmt; ExpKind exp; DeclKind decl; ParamKind param; } kind;
     union {
        TokenType op;
        int val;
        char * name;
        TokenType * type;
        ArrayAttr arr;
        /* To check void parameter */
        int check_void;
      } attr;

     ExpType type; /* for type checking of exps */
   } TreeNode;
```

3. `util.c`

   static 변수들을 새롭게 추가하였고 이 함수들은 `printTree()` 함수에서 사용된다. `var_array` 는 variable이 declaration될 때 declaration된 변수가 array인지 나타낸다. `par_array` 는 parameter가 배열인지 아닌지를 나타낸다. 해당 변수들을 통해 `int[]`, `void[]` 등을 나타내게 된다.

   `arr_size` 와 `check_type` 에 대한 내용은 아래의 `ConstK` 에 대한 내용에 서술하였다.

```c
#include "globals.h"
#include "util.h"

static int var_array = 0;
static int par_array = 0;

static int arr_size = 0;
static int check_type = 0;
```

   `globals.h` 에서 정의한 DeclKind, ParamKind를 위해 새로운 노드를 생성하는 함수를 새롭게 추가하였다.

```
TreeNode * newDeclNode(DeclKind kind)
{ TreeNode * t = (TreeNode *) malloc(sizeof(TreeNode));
  int i;
  if (t==NULL)
    fprintf(listing,"Out of memory error at line %d\n",lineno);
  else {
    for (i=0;i<MAXCHILDREN;i++) t->child[i] = NULL;
    t->sibling = NULL;
    t->nodekind = DeclK;
    t->kind.decl = kind;
    t->lineno = lineno;
  }
  return t;
}

TreeNode * newParamNode(ParamKind kind)
{ TreeNode * t = (TreeNode *) malloc(sizeof(TreeNode));
  int i;
  if (t==NULL)
    fprintf(listing,"Out of memory error at line %d\n",lineno);
  else {
    for (i=0;i<MAXCHILDREN;i++) t->child[i] = NULL;
    t->sibling = NULL;
    t->nodekind = ParamK;
    t->kind.param = kind;
    t->lineno = lineno;
  }
  return t;
}
```

정해진 방식대로 parsing의 결과를 출력하기 위해서 `printTree()` 함수를 수정하였다. 현재 node의 nodeKind와 어떤 stmt를 가졌는지에 따라 적절한 문구를 출력한다.

- `VoidK`의 경우

> `cminus.y`에서 void parameter일 경우 void_param을 통해 변수가 선언되며 이 때 새로 만들어진 DeclNode의 `check_void`가 1로 표시되어 void parameter임을 나타낸다.
>
> 1. `check_void`가 1이 아닐 때, parameter가 array인 경우나 variable이 array인 경우 `void[]`을 출력한다. 그리고 `par_array`변수를 0으로 초기화한다.
> 2. `check_void`가 1이 아닐 때, 위의 경우에 해당하지 않는다면 `void`를 출력한다.
> 3. `check_void`가 1이라면 void parameter이므로 `Void Parameter`를 출력한다.

- `IntK`의 경우

> 1. parameter가 array인 경우나 variable이 array인 경우 `int[]`를 출력한다. 그리고 `par_array`변수를 0으로 초기화한다.
> 2. array가 아닌 경우 `int`를 출력한다.

```
case VoidK:
  if(tree->attr.check_void != 1)
  {
    if(par_array == 1 || var_array == 1)
    {
      fprintf(listing,"type = void[]\n");
      if(par_array == 1) par_array = 0;
      //if(var_array == 1) var_array = 0;
    }
    else
      fprintf(listing,"type = void\n");
  }
  else
  {
    printSpaces();
    fprintf(listing,"Void Parameter\n");
  }
  check_type = 1;
  break;
case IntK:
  if(par_array == 1 || var_array == 1)
  {
    fprintf(listing,"type = int[]\n");
      if(par_array == 1) par_array = 0;
      //if(var_array == 1) var_array = 0;
  }
  else
    fprintf(listing,"type = int\n");
  check_type = 1;
  break;
```

- `ConstK`의 경우

> `cminus.y`를 보면 variable declaration 이후에 type_specifier에서 배열의 size를 const로 출력하기 위해 `newExpNode(ConstK)`를 통해 새로운 ExpNode를 생성한다. variable이 배열일 경우 `arr_size`에 배열의 크기를 저장한다. `check_type`은 `VoidK`와 `IntK`에서 1로 set된다.
>
> 1. `check_type`이 1인 경우, type출력 이후에 `ConstK`임을 의미하므로 배열(`var_array == 1`)이고, 배열의 크기가 0이 아닐 경우(`arr_size != 0`)에 출력한다. 출력하고 나면 `arr_size`와 `var_size`를 0으로 초기화한다.
> 2. `check_type`이 0인 경우, 일반적인 경우의 const 출력이므로 그냥 출력해주면 된다.

```
case ConstK:
  if(check_type == 1)
  {
    if(var_array == 1)
    {
      if(arr_size != 0)
      {
        printSpaces();
        fprintf(listing,"Const: %d\n",arr_size);
        arr_size = 0;
        if(var_array == 1) var_array = 0;
      }
    }
    check_type = 0;
  }
  else
  {
    printSpaces();
    fprintf(listing,"Const: %d\n",tree->attr.val);
  }
  break;
```

4. `cminus.y`

   과제 명세의 BNF grammar에 맞춰서 구현하였다. 그 중 추가한 문법과 conflict 해결에 대해 아래에 설명한다.

   - `ID`와 `NUM`의 경우

     `savedNum`을 추가로 정의하였다.

     ```
     static char * savedName; /* for use in assignments */
     static int savedNum;
     ```

     `savedName`과 `savedNum`을 바로 사용하면 전역변수인 값들이 overwrite될 수 있기 때문에 추가적인 문법을 정의하여 사용하게 되었다.

     ```
     identifier          : ID
                         { savedName = copyString(tokenString);
                           savedLineNo = lineno;
                         }
                         ;
     number              : NUM
                         { savedNum = atoi(tokenString);
                           savedLineNo = lineno;
                         }
                         ;
     ```

   - `void_param`의 경우

     parameter가 void인 경우 params에서 void_param으로 가도록 void_param이라는 새로운 문법을 추가하였다.

```
params              : param_list
                      { $$ = $1; }
                    | void_param
                      { $$ = $1; }
                    ;
```

출력할 때 void를 나타내는 것이 두가지가 있었다. 하나는 `void[]`이고, 하나는 'Void Parameter'이다. 따라서 이 둘을 구분하여 출력하기 위해 기존의 type_specifier는 `void[]` 출력을 할 때 사용하였고 void_param은 'Void Parameter'를 출력하는 데에 사용하였다. `printTree()`에서 void_param에서 1로 set한 `check_void`의 값을 이용한다.

```
void_param          : VOID
                      { $$ = newDeclNode(VoidK);
                        $$->attr.type = VOID;
                        $$->attr.name = "Void Parameter";
                        $$->attr.check_void = 1;
                      }
                    ;
```

- if문에서의 shift-reduce conflict의 경우

  `%nonassoc`을 통해 `NO_ELSE`와 `ELSE`를 추가하였다.

```
%nonassoc NO_ELSE
%nonassoc ELSE
```

  `%prec NO_ELSE`를 통해 else가 가까운 if와 associate되도록 우선순위를 부여하였다.

```
selection_stmt      : IF LPAREN expression RPAREN statement %prec NO_ELSE
                      { $$ = newStmtNode(IfNEK);
                        $$->child[0] = $3;
                        $$->child[1] = $5;
                        $$->child[2] = NULL;
                      }
                    | IF LPAREN expression RPAREN statement ELSE statement
                      { $$ = newStmtNode(IfEK);
                        $$->child[0] = $3;
                        $$->child[1] = $5;
                        $$->child[2] = $7;
                      }
                    ;
```

Example and Result Screenshot

- example 1. `test.1.txt`

```
/* A program to perform Euclid's
   Algorithm to computer gcd */

int gcd (int u, int v)
{
    if (v == 0) return u;
    else return gcd(v,u-u/v*v);
    /* u-u/v*v == u mod v */
}
```

```
void main(void)
{
    int x; int y;
    x = input(); y = input();
    output(gcd(x,y));
}
```

`test.1.txt`에 대한 output

```
C-MINUS COMPILATION: ./test.1.txt

Syntax tree:
  Function Declaration: name = gcd, return type = int
    Parameter: name = u, type = int
    Parameter: name = v, type = int
    Compound Statement:
      If-Else Statement:
        Op: ==
          Variable: name = v
          Const: 0
        Return Statement:
          Variable: name = u
        Return Statement:
          Call: function name = gcd
            Variable: name = v
            Op: -
              Variable: name = u
              Op: *
                Op: /
                  Variable: name = u
                  Variable: name = v
                Variable: name = v
  Function Declaration: name = main, return type = void
    Void Parameter
    Compound Statement:
      Variable Declaration: name = x, type = int
      Variable Declaration: name = y, type = int
      Assign:
        Variable: name = x
        Call: function name = input
      Assign:
        Variable: name = y
        Call: function name = input
      Call: function name = output
        Call: function name = gcd
          Variable: name = x
          Variable: name = y
```

- example 2. `test.2.txt`

```
void main(void)
{
    int i; int x[5];

    i = 0;
    while( i < 5 )
    {
```

```
        x[i] = input();

        i = i + 1;
    }

    i = 0;
    while( i <= 4 )
    {
        if( x[i] != 0 )
        {
            output(x[i]);
        }
    }
}
```

`test.2.txt`에 대한 output

```
C-MINUS COMPILATION: ./test.2.txt

Syntax tree:
  Function Declaration: name = main, return type = void
    Void Parameter
    Compound Statement:
      Variable Declaration: name = i, type = int
      Variable Declaration: name = x, type = int[]
          Const: 5
      Assign:
        Variable: name = i
        Const: 0
      While Statement:
        Op: <
          Variable: name = i
          Const: 5
        Compound Statement:
          Assign:
            Variable: name = x
              Variable: name = i
            Call: function name = input
          Assign:
            Variable: name = i
            Op: +
              Variable: name = i
              Const: 1
      Assign:
        Variable: name = i
        Const: 0
      While Statement:
        Op: <=
          Variable: name = i
          Const: 4
        Compound Statement:
          If Statement:
            Op: !=
              Variable: name = x
                Variable: name = i
              Const: 0
            Compound Statement:
              Call: function name = output
                Variable: name = x
                  Variable: name = i
```

- example 3. `test.3.txt`

```
/* Semantic Error Example */
/* (1) void-type variable a, b
 * (2) uninitialized variable c (and b)
 * (3) undefined variable d */

int main ( void a[] )
{
    void b;
    int c;
    d[1] = b + c;
}
```

`test.3.txt`에 대한 output

```
C-MINUS COMPILATION: ./test.3.txt

Syntax tree:
  Function Declaration: name = main, return type = int
    Parameter: name = a, type = void[]
    Compound Statement:
      Variable Declaration: name = b, type = void
      Variable Declaration: name = c, type = int
      Assign:
        Variable: name = d
          Const: 1
        Op: +
          Variable: name = b
          Variable: name = c
```