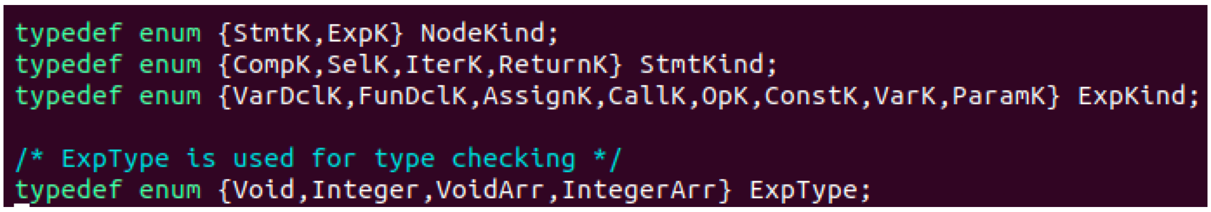
C-Minus Scanner

Environment : Ubuntu 18.04

Cimpilation method : c언어 file과 lex를 위한 .l 파일을 수정하고 Makefile을 이용해 gcc로 한번에 Compile 했다.

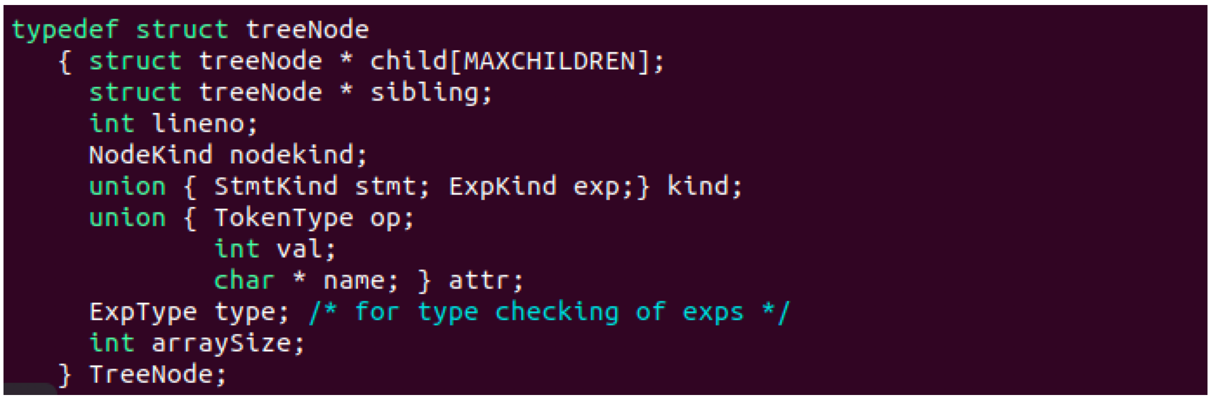
Globals.h



```
typedef enum {StmtK,ExpK} NodeKind;
typedef enum {CompK,SelK,IterK,ReturnK} StmtKind;
typedef enum {VarDclK,FunDclK,AssignK,CallK,OpK,ConstK,VarK,ParamK} ExpKind;

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

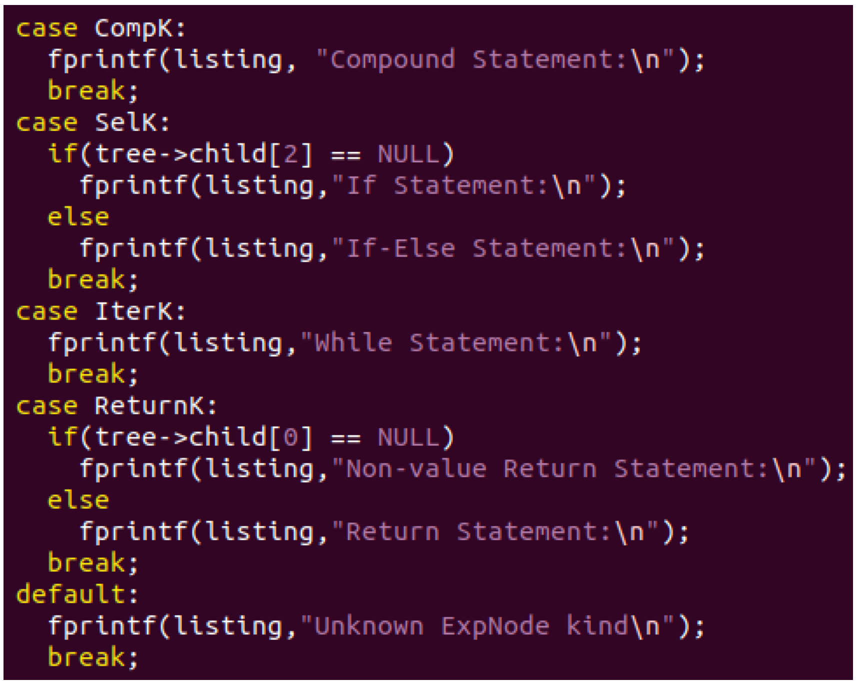
Cminus 문법에 맞는 stmt들을 추가해주고 ExpType을 Void, Integer, VoidArr, IntegerArr로 한다.



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

Array 변수의 size를 저장할 수 있는 arraySize를 추가해준다.

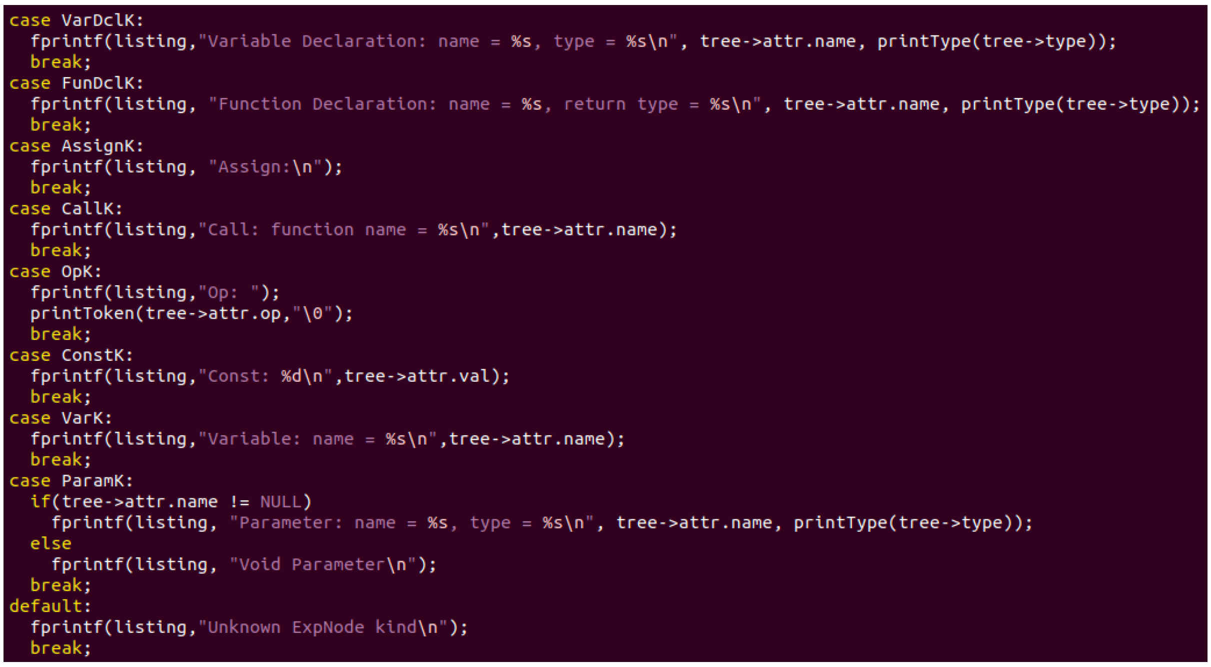
Util.c



```
case CompK:
  fprintf(listing, "Compound Statement:\n");
  break;
case SelK:
  if(tree->child[2] == NULL)
    fprintf(listing,"If Statement:\n");
  else
    fprintf(listing,"If-Else Statement:\n");
  break;
case IterK:
  fprintf(listing,"While Statement:\n");
  break;
case ReturnK:
  if(tree->child[0] == NULL)
    fprintf(listing,"Non-value Return Statement:\n");
  else
    fprintf(listing,"Return Statement:\n");
  break;
default:
  fprintf(listing,"Unknown ExpNode kind\n");
  break;
```

SelK 에서는 child[2]의 유무에 따라 if문과 if-else 문을 구분한다.

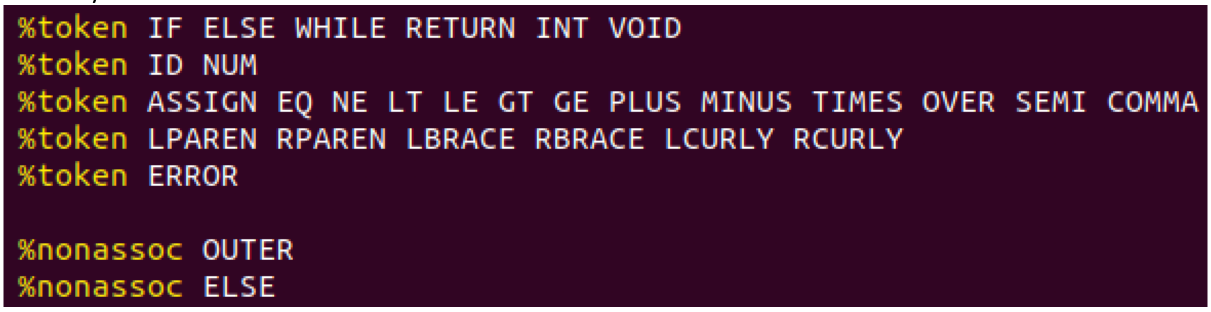
return 에서는 child[0]의 유무에 따라 Non-value Return을 구분한다.



```
case VarDclK:
  fprintf(listing,"Variable Declaration: name = %s, type = %s\n", tree->attr.name, printType(tree->type));
  break;
case FunDclK:
  fprintf(listing, "Function Declaration: name = %s, return type = %s\n", tree->attr.name, printType(tree->type));
  break;
case AssignK:
  fprintf(listing, "Assign:\n");
  break;
case CallK:
  fprintf(listing,"Call: function name = %s\n",tree->attr.name);
  break;
case OpK:
  fprintf(listing,"Op: ");
  printToken(tree->attr.op,"\0");
  break;
case ConstK:
  fprintf(listing,"Const: %d\n",tree->attr.val);
  break;
case VarK:
  fprintf(listing,"Variable: name = %s\n",tree->attr.name);
  break;
case ParamK:
  if(tree->attr.name != NULL)
    fprintf(listing, "Parameter: name = %s, type = %s\n", tree->attr.name, printType(tree->type));
  else
    fprintf(listing, "Void Parameter\n");
  break;
default:
  fprintf(listing,"Unknown ExpNode kind\n");
  break;
```

ParamK에서는 attr.name의 유무에 따라서 Void Parameter를 구분한다.

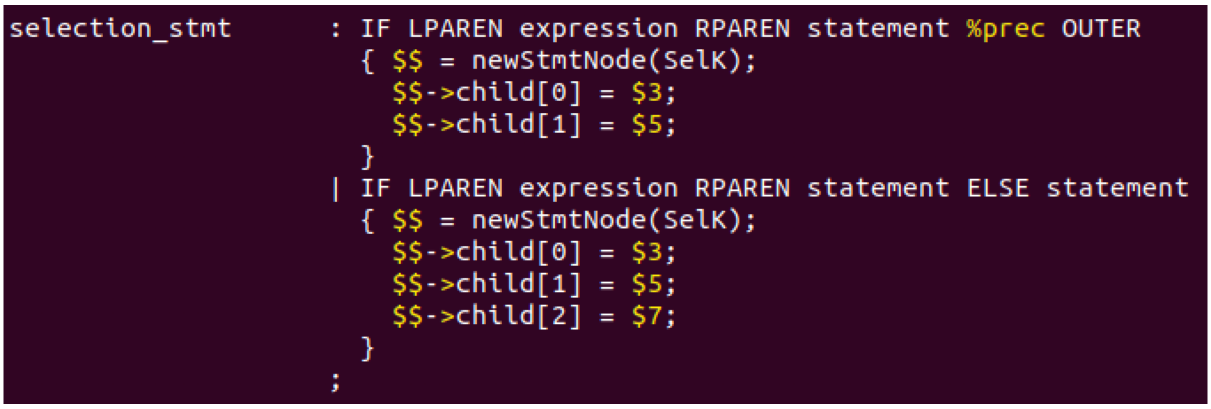
Cminus.y



```
%token IF ELSE WHILE RETURN INT VOID
%token ID NUM
%token ASSIGN EQ NE LT LE GT GE PLUS MINUS TIMES OVER SEMI COMMA
%token LPAREN RPAREN LBRACE RBRACE LCURLY RCURLY
%token ERROR

%nonassoc OUTER
%nonassoc ELSE
```

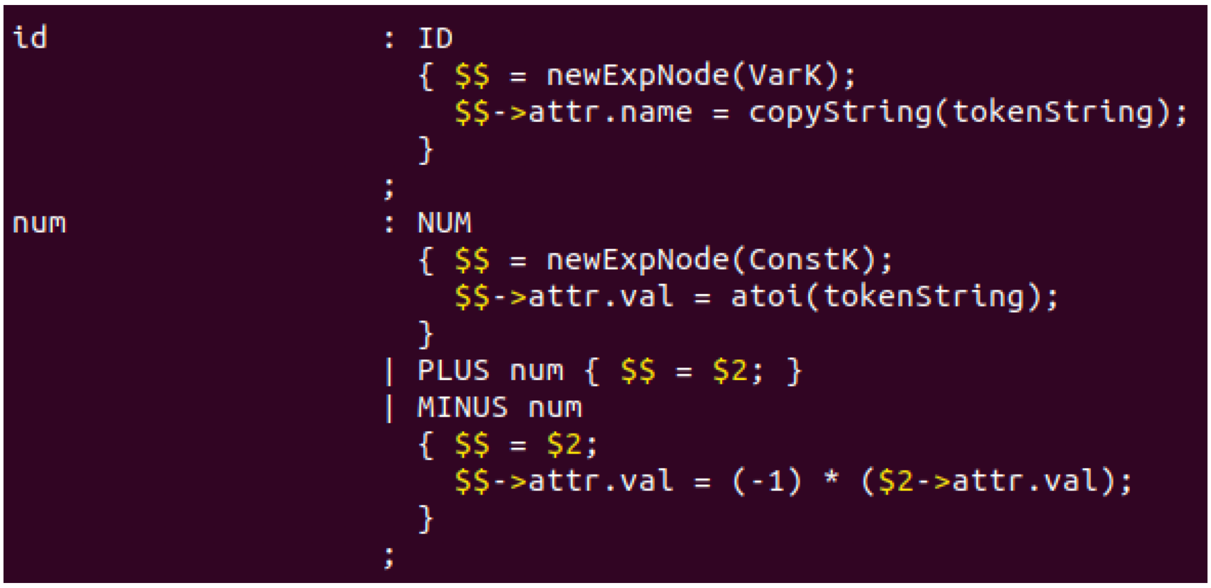
Cminus 에서 사용하는 token들을 입력해준다



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

Dangling else 문제를 해결하기 위해서 ambiguity를 해소해야 한다.

따라서 OUTER라는 token을 등록해 precednece를 주어 해결한다.



```
id                    : ID
                        { $$ = newExpNode(VarK);
                          $$->attr.name = copyString(tokenString);
                        }
                      ;
num                   : NUM
                        { $$ = newExpNode(ConstK);
                          $$->attr.val = atoi(tokenString);
                        }
                      | PLUS num { $$ = $2; }
                      | MINUS num
                        { $$ = $2;
                          $$->attr.val = (-1) * ($2->attr.val);
                        }
                      ;
```

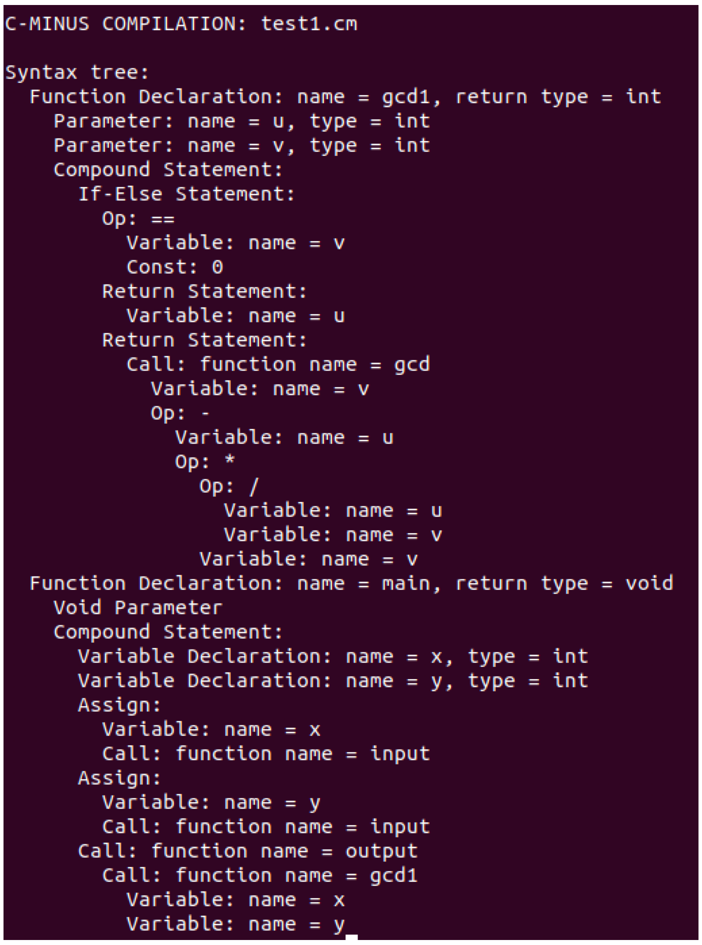
Statement들에 사용하는 ID와 num을 처리하는 부분이다.

이때 +10, -4 같은 Num을 처리하기 위한 별도의 과정을 추가하였다.

그 밖에 다른 부분들은 tiny.y에서 사용된 부분들을 참고하여 cminus 형식에 맞게 추가 및 변경 하였다.

Result

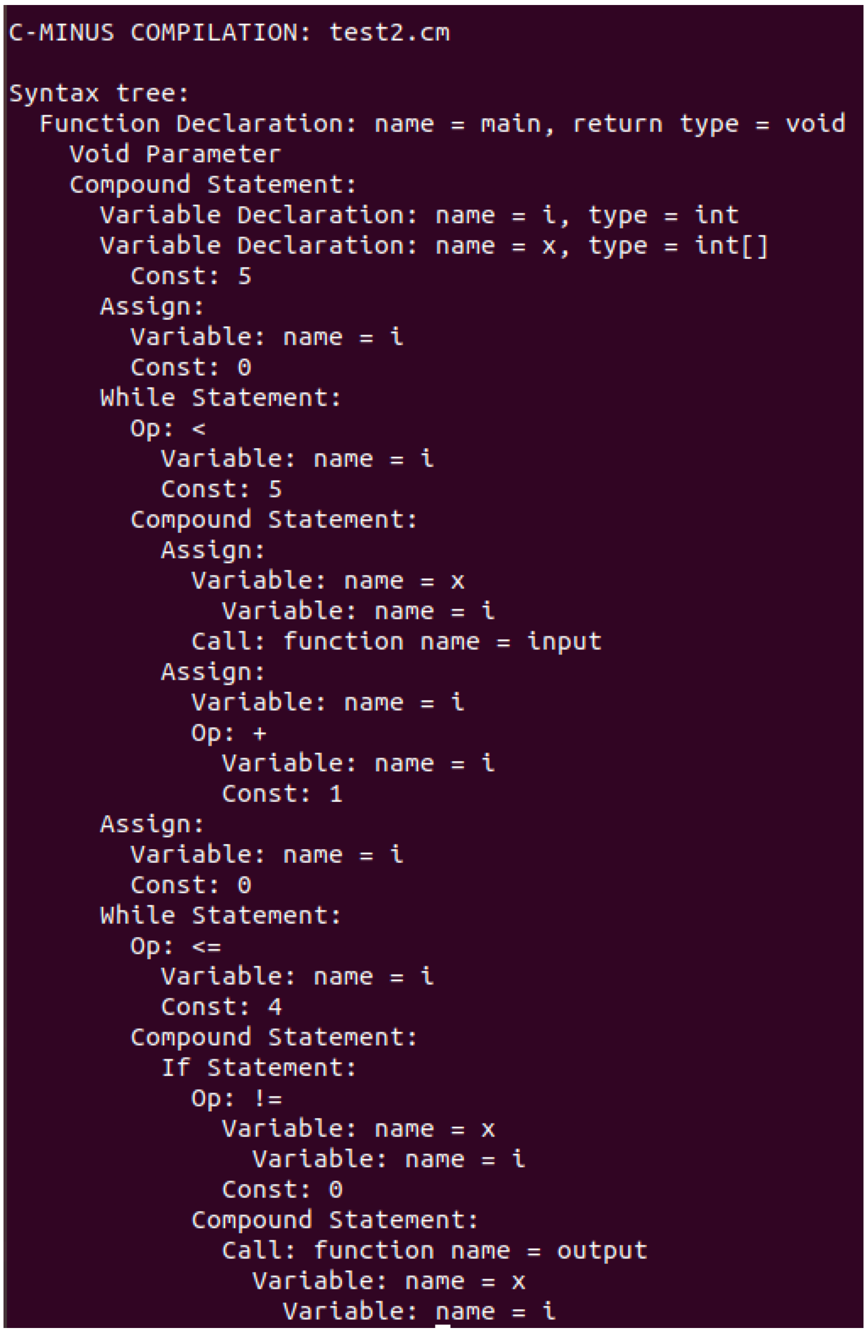
Test1.cm



```
C-MINUS COMPILATION: test1.cm

Syntax tree:
  Function Declaration: name = gcd1, 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 = gcd1
          Variable: name = x
          Variable: name = y
```

Test2.cm

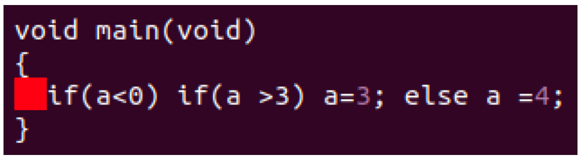


```
C-MINUS COMPILATION: test2.cm

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
```

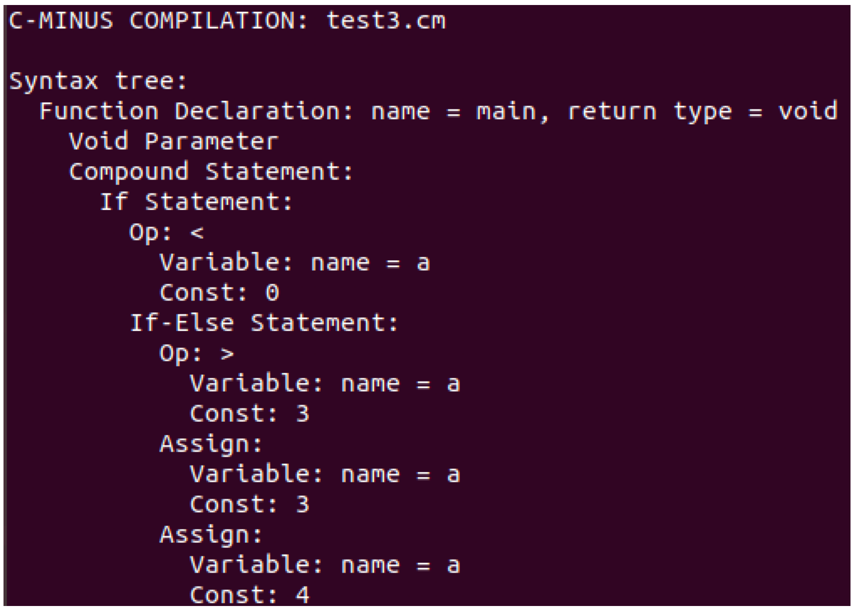
Dangling-else

code



```
void main(void)
{
 if(a<0) if(a >3) a=3; else a =4;
}
```

Result

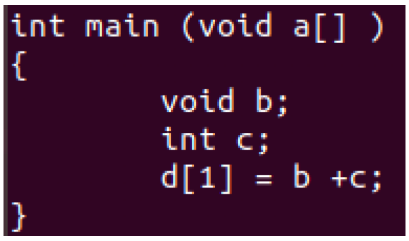


```
C-MINUS COMPILATION: test3.cm

Syntax tree:
  Function Declaration: name = main, return type = void
    Void Parameter
    Compound Statement:
      If Statement:
        Op: <
          Variable: name = a
          Const: 0
        If-Else Statement:
          Op: >
            Variable: name = a
            Const: 3
          Assign:
            Variable: name = a
            Const: 3
          Assign:
            Variable: name = a
            Const: 4
```

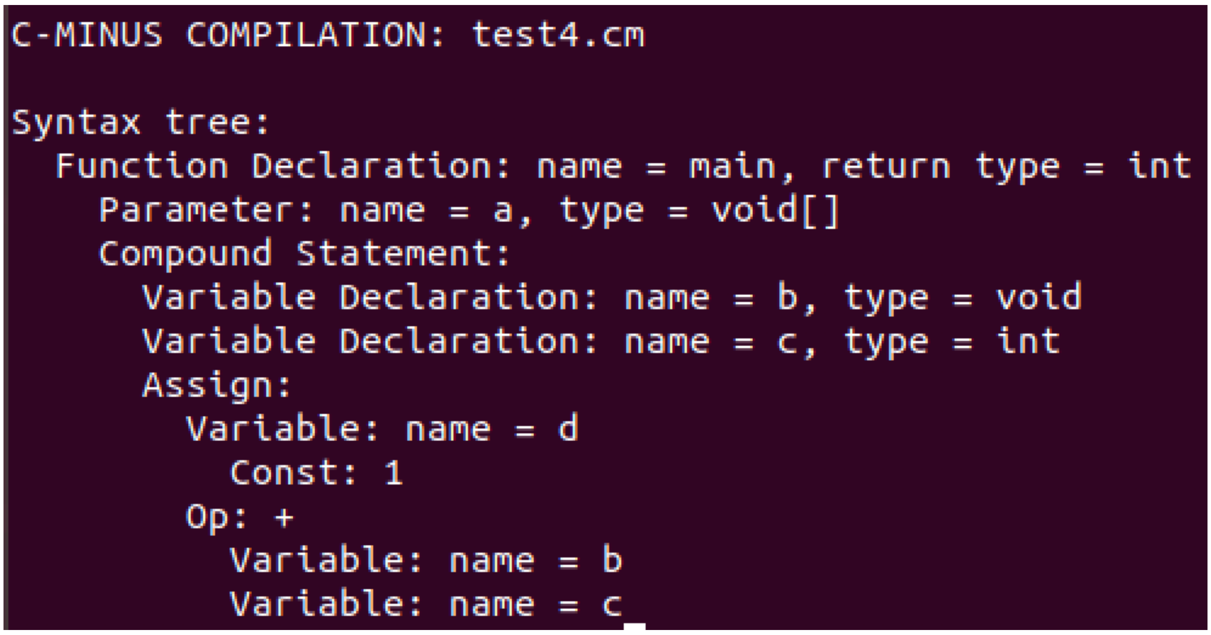
Semantic error

code



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

result



```
C-MINUS COMPILATION: test4.cm

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
```