2022056262_project2

Goal / compilation environment and method

project 1에서 만들었던 scanner에서 나온 결과를 input으로 받아 AST를 output으로 내는

C-minus parser 만들기

컴파일 환경 및 방법

- Ubuntu 20.04.4 LTS
- 첨부되어 있던 Makefile을 사용 gcc로 컴파일, Makefile로 빌드

parsing은 parse 함수를 호출하면서 시작한다. parse 함수는 yyparse 함수를 호출한다.
yyparse 함수는 cminus.y, globals.h 등의 file을 토대로 만들어진다.
cminus.y에서 정한 rule들에 의해 결정. 즉 cminus.y에서 정의한 rule들에 의해 parsing된다.
LALR(1) parser는 bottom up parser이기 때문에 자식 node부터 만들어져서 shift, reduce rule에 의해 start symbol까지 온다. stack에 non-terminal에 해당하는 tree를 저장하고, reduction을 할 때 두 non-terminal이 하나로 줄어든다면 AST 역시 하나로 합쳐진다.

pre-processing

```
typedef enum {StmtK,ExpK, DeclK} NodeKind;
/*voidparamK 추가 (printTree에서 처리하기 용이 void parameter는 출력 형식이 다름) */
typedef enum {VarK, ArrK, FuncK, ParamK, VoidParamK, ArrParamK} DeclKind;
/*if else도 출력 형식이 달라서 If와 IfElse를 분리하는 게 용이*/
typedef enum {IfK,ElseK, WhileK, CompoundK, ReturnK } StmtKind;
typedef enum {CallK,AssignK,OpK,ConstK,IdK, TypeK, VarExpK} ExpKind;

typedef enum {Void,Integer,IntArray,VoidArray} ExpType;
union { StmtKind stmt; ExpKind exp; DeclKind decl;} kind;
```

globals.h에서 AST에 필요한 node들을 정의해준다.
grammar를 기반으로 결정하고, parsing code를 작성하면서 필요에 따라 수정하였다.
Decl에서 특이한 node는 var, arr, param, voidparam, arrparam 이다.
이 모든 node는 type, name을 속성으로 갖는다. 출력 시에 array는 int[], 그냥 var는 int로 출력해주어야 하기 때문에 둘을 분리해서 저장하였다.
Stmt에서도 역시 If, Else는 출력 형식이 달라서 둘을 분리해주었다. 그 외는 grammar 기반 결정
Exp에서는, IdK, TypeK가 특이하다. 사실 name, type은 별도의 node로 표현하지 않고 node의 속성으로 갖고 있는 요소들이다. 그러나 recursive descent parsing은 자식 node를 다 거친 후에 부모 node로 가기 때문에 처음 내려갈 때 저장했던 부모 node의 type과 name이 child에서 update되는 문제가 발생했다. 따라서 id와 type을 저장하기 위해 node를 만들고, 이 node의 type, name은 불변이기 때문에 부모 node에 저장할 때 이 node의 값을 갖고 와서 저장해주는 방식을 사용하였다.

```
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;
    t->type = Void;
  }
  return t;
}
```

위에서 정의한 node를 만들기 위한 함수이다. DeclNode는 새롭게 정의해준 거라서 기존 StmtNode 생성 코드를 복사해서 사용하였다. 수정해주어야 할 것은 nodekind를 DeclK로 바꿔주는 것이다.
globals.h에서 enum으로 typedef해둔 거라서 이름을 틀리면 안 된다... 마찬 가지로 kind 역시 DeclKind형으로 들어와야 한다.

```c
else if (tree->nodekind==DeclK){
  switch(tree->kind.decl){
    case VarK:
      fprintf(listing,"Variable Declaration: name = %s, type = %s\n",tree->attr.name,(tree->type==Integer)?"int":"void");
      break;
    case ArrK:
      fprintf(listing,"Variable Declaration: name = %s, type = %s\n",tree->attr.name,(tree->type==IntArray)?"int[]":"void[]");
      break;
    case FuncK:
      fprintf(listing,"Function Declaration: name = %s, return type = %s\n",tree->attr.name,(tree->type==Integer)?"int":"void");
      break;
    case ParamK:
      fprintf(listing,"Parameter: name = %s, type = %s\n",tree->attr.name, (tree->type==Integer)?"int":"void");
      break;
    case VoidParamK:
      fprintf(listing,"Void Parameter\n");
      break;
    case ArrParamK:
      fprintf(listing,"Parameter: name = %s, type = %s\n",tree->attr.name,(tree->type==IntArray)?"int[]":"void[]");
      break;
    default:
      fprintf(listing,"Unknown DeclNode kind\n");
      break;
  }
}
```

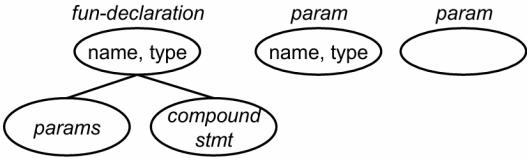
printTree는 기존 구조를 그대로 갖고 와서 형식만 바꾼 거라서 DecIK case 설명으로 대체하였다.
nodekind는 바로 위에서 DeclNode 생성하면서 DecIk로 설정했으니 당연히 해당 node는 이 조건에 걸릴 것이다. DecIK에 있는 node들마다 출력문을 정해주었다.
type은 int형이라서 삼중문으로 조건을 걸어주었다.
array같은 경우, array size까지 출력해야 하는 경우가 있는데, 그건 array node의 child에 저장되어 있다. 그런데 재귀를 통해 자식 node로 다 searching하기 때문에 여기서 자식 노드의 값을 출력할 필요는 없다.

rules (handle dangling else problem)

모든 rule은 ppt에 있는 grammar를 그대로 사용하였다.
id, num, int, void만 별도의 rule을 추가해서 다루었고 selection-stmt도 dangling else 문제를 방지하기 위해 수정하였다.

```
func_decl   : type_spec id LPAREN params RPAREN comp_stmt
                { $$ = newDeclNode(FuncK);
                  $$->type = $1->type;
                  $$->attr.name = $2->attr.name;
                  $$->lineno = lineno;
                  $$->child[0] = $4;
                  $$->child[1] = $6;
                }
                ;
```

`function-declarations → type-specifier ID (params) compound-stmt` 이 production rule을 기반으로 작성한 코드이다.
terminal은 scanner에서 return하는 token 이름 그대로 넣어주면 되고, non-terminal은 cminus file 안에서 이름을 동일하게 맞춰주면 된다.



fun-declaration은 이런 node 형태를 가진다. func의 name은 id, type은 type_spec에 의해 결정된다.
child[0]은 params, child[1]은 compound_stmt이다. Yacc에서 LHS는 $$로, RHS는 순서대로 $1, $2로 참조할 수 있다. 따라서 child[0]에는 params인 $4, child[1]에는 comp_stmt인 $6을 넣는다.
원래 savedName, savedType이라는 local에 type_spec의 type과 id의 name을 저장하고 funcK에 넣으려고 하였으나, 계속해서 다른 곳에서 그 값들이 update되어서 제대로 저장이 안 되는 문제가 있었다. 따라서 아예 local 변수 대신에 type, id를 다 node로 만들어서 이름과 type을 저장해두고, func_decl에서 직접적으로 갖고 올 수 있게 code를 수정하였다.
$$→type=savedType; 이 형태에서 $$→type=$1→type 이 형태로 변경.

```
params      : param_list { $$ = $1; }
            | VOID { $$ = newDeclNode(VoidParamK); }
            ;
```

non-terminal로만 가는 경우, LHS에 RHS를 할당하고, 그 non-terminal로 다시 rule을 적용할 수 있게
$$=$1;로 적어주면 된다. node에 속성이 있는 경우엔 그에 맞는 값을 넣어주면 되고, void parameter처럼 아무런 속성도 안 갖는 경우엔 그냥 node를 만들어주기만 하면 된다.

```
param_list : param_list COMMA param
                { YYSTYPE t = $1;
                  if (t != NULL)
                  { while (t->sibling != NULL)
                       t = t->sibling;
                    t->sibling = $3;
                    $$ = $1; }
                    else $$ = $3;
                }
```

```
            | param { $$ = $1; }
            ;
```

재귀를 갖는 경우엔 param_list를 linked-list로 만든다. parameter가 추가될 때마다 형제를 추가한다.
YYSTYPE은 scanner에 사용되는 모든 token, non-terminal 기호의 data type을 정의한다.
param_list라는 linked-list에 sibling이 null이 될 때까지 sibling으로 이동하고, 끝까지 갔으면 새로 생긴 $3 param을 추가한다. 그리고 $$은 여전히 %1이다. 만약 param_list가 null이었다면 param만 취급하면 돼서 $3을 할당한다.

select stmt를 제외한 모든 rule은 위와 같은 방식으로 다 지정해주면 된다.

- **select stmt는 dangling else 문제를 방지하기 위해 추가적인 설정을 해주어야 한다.**
  해결 방법은 else를 가장 가까운 if와 연결해주는 것이다.
  ppt에 있는 mif, uif로 해결할까 하였으나 그냥 else를 righ associative하게 설정하고, 추가 조건을 달아준다면 else는 늘 else의 왼쪽에 있는 if들 중에서 가장 오른쪽 즉 else와 가장 가까이 있는 if와 결합할 것이다. 만약 if (exp) if (exp) return ; else return; 이라는 구문이 있다면, 이 else는 두 번째 if와 결합해야 한다. ELSE는 right 결합이므로, 두 번째 if와 정상적으로 결합한다.
  if (exp) if (exp) return ; else return; else return;
  이 경우에도 첫 번째 else는 두 번째 if, 두 번째 else는 첫 번째 if와 결합한다. right asso는 아직 결합이 완료되지 않은 곳부터 right 결합을 하기 때문에 첫 번째 else부터 결합을 한다.

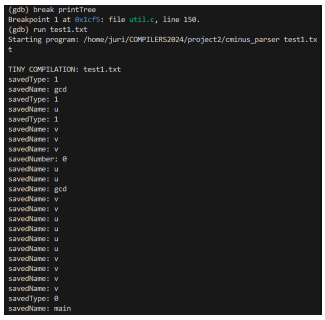
```
%token IF ELSE INT RETURN VOID WHILE
%token ID NUM
%token LT LE GT GE EQ NE SEMI
%token LPAREN RPAREN LCURLY RCURLY LBRACE RBRACE
%left PLUS MINUS
%left TIMES OVER COMMA
%right ASSIGN
%right THEN ELSE
%token ERROR
select_stmt: IF LPAREN expression RPAREN statement
                 { $$ = newStmtNode(IfK);
                   $$->child[0] = $3;
                   $$->child[1] = $5;
                 } %prec THEN
           | IF LPAREN expression RPAREN statement ELSE statement
                { $$ = newStmtNode(ElseK);
                  $$->child[0] = $3;
                  $$->child[1] = $5;
                  $$->child[2] = $7;
                }
           ;
```

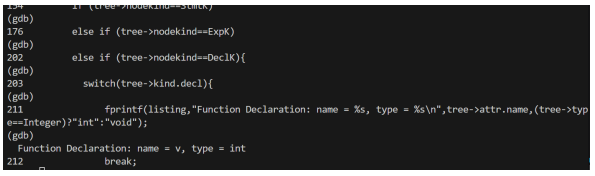
%prec THEN은 첫 번째 rule의 우선순위를 THEN의 우선순위로 지정하겠다는 의미이다.
근데 여기서 THEN은 ELSE와 달리 실제로 사용하는 token이 아니기 때문에 ELSE보다 우선순위가 낮다.
따라서 첫 번째 rule과 ELSE가 만나면 ELSE를 우선 결합한다.

Trouble shooting

- **Segmentation fault가 printTree에 도달하기도 전에 발생하였다.**
  ⇒ gdb를 사용해서 어디서 seg fault 문제가 생기는 지 확인해보았다.
  자식 node를 생성하는 코드를 부모 rule에 작성했었는데, 실제로는 자식 node부터 접근해서 속성을 정해주기 때문에 계속 존재하지 않는 node에 접근하려고 했던 것이다.
  자식 node는 자식 rule에서 생성할 수 있도록 code를 수정하여 seg fault 문제를 해결하였다.

- **function 이름과 type이 parameter의 이름과 type으로 출력되는 문제가 발생하였다.**
  ⇒ gdb를 사용해서 savedName과 type을 출력하고, 한 줄씩 실행해보았다.
  처음 접근할 때는 gcd로 잘 저장되어 있는데 print할 때는 다른 값이 출력되었다.
  savedName과 type이 다른 rule에 의해 덮어씌워지는 것이 문제였다.
  예를 들어, type_specifier의 type이 function의 type이 되어야 하지만, params에도 type_specifier가 존재하기 때문에 여기서 savedType이 덮어 씌워졌던 것.
  $$→type=$1 이런 식으로 저장하려고 하였으나 type이 맞지 않아, 문제가 생겼다.
  따라서 type을 저장하는 node를 만들고 그 node의 type에 해당 type을 저장해주었다.
  이 node과 func node를 연결하지만 않으면 AST 구조에 차이는 없기 때문에 적절한 접근법이라고 판단하였다.

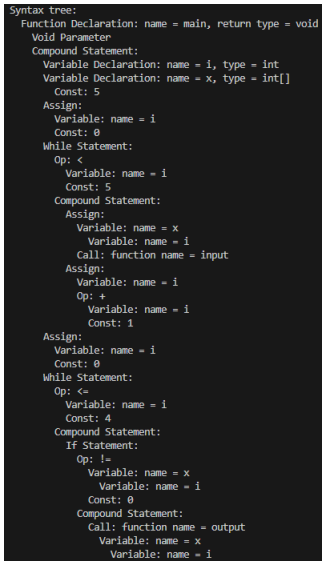


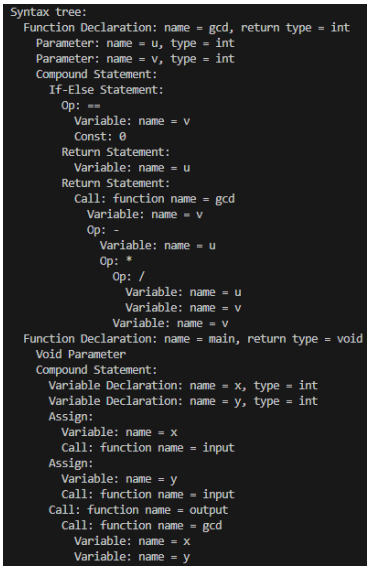


testCase

기본 testcase

lms에 올라온 testCase 실행 결과는 첨부되어 있던 result와 동일하게 나왔다.



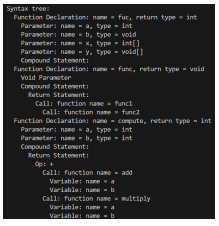


각종 함수 관련 문제

```
#1번 여러 type의 인자
int fuc
(int a, void b, int x[], void
y[]){}
```

```
#2번 함수의 인자로 함수 호출
void func(void) {
return func1(func2());
}
```

```
#3번 함수와 함수의 연산
int compute
(int a,int b){ return
add(a, b) +
multiply(a,b);}
```

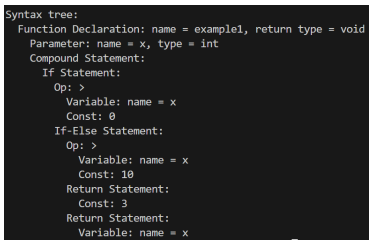


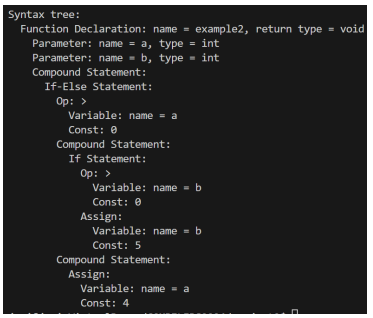
dangling else 문제...

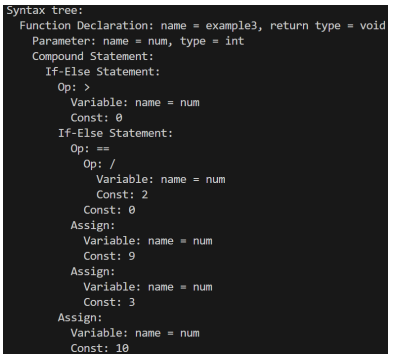
```
#1번 (if(if else))
void example1(int x) {
    if (x > 0)
        if (x > 10) return 3;
        else return x;
}
```

```
#2번 (if(if)else)
void example2
(int a, int b) {
    if (a > 0) {
        if (b > 0)
            b=5;
    } else { a=4;
    }
}
```

```
#3번 (if(if else)else)
void example3(int num) {
    if (num > 0)
        if (num / 2 == 0) num=9;
        else num =3;
    else num=10;
}
```








```
#4번 (if(if)else)인데 {}가 없는
void example4(void){
if(a>0)
   if(b>0)
else a=4;
```

이 경우엔 결합법칙에 의해
두 번째 else와 결합한다.