Temp

Last edited by **김형순** less than a minute ago

Compiler project

3. Semantic Analysis Symbol Table & Type Checker

2015004493 김형순

프로젝트 개요

- C minus 라는 프로그래밍의 컴파일러를 만들기 위한 세번째 단계인 semantic analysis 위해 symbol table 과 type checker 를 만든다.
- symbol table 과 기존 parser 에서 만들었던 syntax tree 를 활용하여 semantic 오류를 찾아낸다.

컴파일 환경

- GCC 4.8.2 버전을 사용하여 컴파일 하였다.
- Ubuntu 14.04 LTS 환경에서 컴파일 및 실행 하였다.
- flex 2.5.35, bison 3.0.2 를 사용하였다.

실행 방법

- Makefile 을 이용하여 컴파일을 한다.
- cminus 라는 실행파일이 만들어지면 ./cminus [input file] 을 입력하여 실행한다.

구현방법

1. globals.h
   parsing 후 semantic analysis 를 할 때 integer array type 이 없기 때문에 type checking 에 어려움이 있었다. 그러므로 ExpType 에 IntegerArr 을 추가하여 정수형 배열도 비교 할 수 있도록 하였다.
2. main.c
   Parser 에서 출력하던 syntax tree 를 출력하지 않도록 traceParse 를 False 로 만든다. 구현중 symbol table 을 출력하여 볼 때는 traceAnalyze 를 True 로 만들어 출력해보았다.
3. symtab.h
4. Symbol table 을 만들기 위한 자료구조들과 함수의 선언을 모아놓은 헤더파일 이다.

5. Symbol table 은 bucket list 구조체로 symtab.c 내의 hash 함수에 의해 자료구조가 저장된다. 그러므로 같은 hash value 를 갖는 경우를 대비해 다음 bucket 을 가리키는 next pointer 를 attribute 로 갖고있다.
6. 또한 symbol table 의 여러 정보를 저장하기 위해 선언된 variable 의 자료형 type, variable 인지 parameter 인지 function 인지 알기 위해 만든 variable_type, 메모리위치를 표현하기 위해 memloc 을 사용하고, 코드의 몇번째 line 에서 사용되는지 linked list 로 저장 할 lines 가 함께 저장된다.

7. 각 symbol table 변수/함수 들은 몇번째 줄에서 사용될지 저장 되어야 한다. 그것을 위해 linked list 로 LineList 자료구조를 사용한다.

8. cminus는 scope 를 사용하여 variable/function 의 유효성을 판단하기 때문에 scope 정보를 저장할 Linked List 로 만들어진 ScopeList 자료구조를 사용한다.
9. ScopeList 에는 scope 의 이름 (name), scope 별 symbol 들 (bucket), parent scope pointer (parent), linked list 다음 scope pointer (next), scope 의 child scope 개수 (child_num), type checking 에서 현재까지 몇개의 child scope 를 확인 했는지 count 하는 current_child, memory location 을 표현하는 location 을 저장한다.

10. 마지막으로 type checking 에서 함수 관련 check 를 하기 위해 FunctionList 자료구조를 만들었다. FunctionList 는 함수 이름 (name), 파라미터 개수 (paramNum), 파라미터의 타입들 (paramType), return type(type), Linked list 로 구성되어 있으므로 next FunctionList 로의 pointer 를 저장한다.

```
typedef struct LineListRec
    { int lineno;
      struct LineListRec * next;
    } * LineList;

typedef struct BucketListRec
    { char * name;
      // 0: variable, 1: array, 2: function
      // 3: single parameter, 4: array parameter
      int variable_type;
      ExpType type;
      LineList lines;
      int memloc;
      struct BucketListRec * next;
    } * BucketList;

typedef struct ScopeListRec
    { char * name;
      BucketList bucket[SIZE];
      struct ScopeListRec * parent;
      struct ScopeListRec * next;
      int child_num;
      int current_child;  // Counter for if/while compound when type checking.
      int location;     // Counter for memory location.
    } * ScopeList;

typedef struct FunctionListRec
    { char * name;
      int paramNum;
      ExpType paramType[100];   //0: integer, 1: array
      ExpType type;
      struct FunctionListRec * next;
    } * FunctionList;
```

1. symtab.c
2. symtab.h 에서 정의 한 BucketList, ScopeList, FunctionList 를 insert 하고 lookup 하는 함수들을 정의 하였다.
3. st_insert : BucketList 를 이용해 symbol table 에 symbol을 insert 한다. scope 가 parameter 로 주어져 해당 scope 의 hash function 을 적용한 bucket 에 저장하게 된다.

```
void st_insert( ScopeList scope, char * name, ExpType type, int lineno, int variable
{ 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;
    l->lines = (LineList) malloc(sizeof(struct LineListRec));
    l->lines->lineno = lineno;
    l->memloc = scope->location++;
    l->variable_type = variable_type;
    l->type = type;
    l->lines->next = NULL;
    l->next = scope->bucket[h];
    scope->bucket[h] = l;
  }
  else /* found in table, so just add line number */
  { LineList t = l->lines;
    while (t->next != NULL) t = t->next;
    t->next = (LineList) malloc(sizeof(struct LineListRec));
    t->next->lineno = lineno;
    t->next->next = NULL;
  }
} /* st_insert */
```

- st_add : symbol table 의 이미 있는 symbol 에 line number 를 추가한다.

```
void st_add( ScopeList cur_scope, char * name, int lineno){
  int h = hash(name);
  ScopeList s = cur_scope;
  BucketList l;

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

  if (s == NULL){
    // There is no variable in st.
    fprintf(listing, "Fatal error. There is no variable in symtab.");
    return;
  }
  else{
    // Found bucket.
    LineList t = l->lines;
    while (t->next != NULL) t = t->next;
    t->next = (LineList) malloc(sizeof(struct LineListRec));
    t->next->lineno = lineno;
    t->next->next = NULL;
  }
}
```

- scope_insert : scope 를 새로 만들어 scopeTable 에 넣는다. 이때 부모 scope 를 parameter 로 받아 parent 로 지정해준다.

```
ScopeList scope_insert( char * scope, ScopeList parent)
{
  ScopeList s;

  int i, check = 0;

  for(i = 0; i<scope_num; i++){
    s = scopeTable[i];
    if(s != NULL && (strcmp(scope, s->name) ==0)){
      fprintf(listing,"Error. Scope already exist.");
      check =1;
      break;
    }
  }
 if(check == 0){
    s = (ScopeList) malloc(sizeof(struct ScopeListRec));
    s->name = copyString(scope);
      for(i = 0; i<SIZE; i++){
      s->bucket[i] = NULL;
    }

    s->parent = parent;
    s->child_num = 0;
    s->current_child = 0;
    s->location = 0;
    scopeTable[scope_num++] = s;

    if(parent != NULL){
      // This is possible when insert global scope.
      parent->child_num ++;
    }
    return s;
  }
  return NULL;
}
```

- st_lookup: parameter 로 주어진 scope 에서 parameter 에 해당하는 name 의 symbol 을 찾는다. 있다면 그 symbol 의 BucketList 를 반환하고, 없다면 NULL 을 반환한다.

```
BucketList st_lookup ( ScopeList scope, char * name )
{ int h = hash(name);
  ScopeList s = scope;

  while(s != NULL){
    // Search from currnt scope to global scope.
    BucketList l = s->bucket[h];
    while ((l != NULL) && (strcmp(name,l->name) != 0))
      l = l->next;
    if (l == NULL){
      s = s->parent;
    }
    else return l;
  }
  // There is no variable or function in st.
  return NULL;
}
```

- 나머지 scope_lookup, ft_insert, ft_lookup 은 위 함수들과 구현 방법이 비슷 하므로 생략 하겠다.
- Symbol table 이 잘 만들어졌는지 보기 위해, 그리고 앞으로 할 type check 가 잘 될것인지 확인하기 위해 symbol table, function table, global table, function locals and params table 을 출력하였다.
- printSymTab : Scope table 을 순회하며 만나는 scope 마다 모든 bucket 을 찾아 테이블로 출력한다. 이때 symbol 의 이름, 종류 (정수, 정수 배열, 함수 등), symbol 이 속한 scope 이름, memory location, line number 들을 출력한다.
- 과제 2 에 이어서 하다보니 integer array type 이 없는 상황에서 table 을 만들다보니 switch 문이 복잡해지게 되었다.

```
void printSymTab(FILE * listing)
{ int i, j;
  ScopeList s;

  fprintf(listing,"Variable Name  Variable Type  Scope Name  Location   Line Numbers
  fprintf(listing,"-------------  -------------  ----------  --------   ------------



  for (i=0; i<SIZE; ++i){
    if ((s = scopeTable[i]) != NULL){
      for (j=0; j<SIZE; ++j){
        BucketList l = s->bucket[j];
        while (l != NULL){
          LineList t= l->lines;
          fprintf(listing, "%-14s ", l->name);
          switch (l->variable_type){
            case 0:
            case 3:
              switch (l->type){
                case Integer:
```

```
            fprintf(listing, "%-14s ", "Integer");
            break;
          default:
            break;
        }
        break;
      case 1:
      case 4:
        switch (l->type){
          case Integer:
            fprintf(listing, "%-14s ", "IntegerArray");
            break;
          default:
            break;
        }
        break;

      case 2:
        fprintf(listing, "%-14s ", "Function");
        break;

      default:
        break;
    }
          fprintf(listing, "%-14s ", s->name);
          fprintf(listing, "%-8d  ", l->memloc);
          while (t != NULL){
            fprintf(listing, "%4d ", t->lineno);
            t = t->next;
          }
          fprintf(listing, "\n");
          l = l->next;
        }
      }
    }
  }
}
```

- 나머지 3개의 table 을 출력하는 함수도 위와 비슷하므로 구현에 대한 설명을 생략한다.

- analyze.c
- symtab.c 에서 symbol table 을 만드는 함수와 자료구조들을 정의 했으므로 analyze.c 에서 syntax tree 의 노드 type 에 따라 적절하게 symbol 들을 insert 해주어야 한다.
- analyze.c 는 크게 두 부분으로 구성되어 있는데 첫번째 부분은 symbol table 을 만드는 insertNode 의 부분이다. symbol table 은 syntax tree 를 순회 (traverse) 하며 만들어지게 되는데, 미리 구현되어 있는 traverse 함수를 이용하면 현재 노드에서 다음 노드로 넘어가기 전에 처리하는 preproc 과 다음 노드들을 처리하고나서 실행하는

postproc 으로 나누어 순회한다.

- symbol table 을 만들기위해 buildSymbolTable() 을 실행하게 된다.
- buildSymbolTable() 은 우선 모든 symbol 의 부모가 되는 global scope 를 insert 하고, 기본적으로 사용 할 수 있는 built-in function 들과 필요한 parameter 들을 insert 한다.
- 그 후 syntax tree 를 순회 하는데, preproc 으로 insert_node 를 하고, postproc 으로 exitScope 함수를 실행하여 compound expression 이 종료 되었을 때 현재 scope 를 나와 부모 scope 로 이동 할 수 있도록 하였다.

```
void buildSymtab(TreeNode * syntaxTree)
{
  ScopeList temp;
  FunctionList fun;

  cur_scope = scope_insert("global", NULL);
  // Insert built - in function.
  st_insert(cur_scope, "input", Integer, 0, 2);
  scope_insert("input", cur_scope);
  ft_insert("input", Integer);
  st_insert(cur_scope, "output", Void, 0, 2);
  temp = scope_insert("output", cur_scope);
  fun = ft_insert("output", Void);
  // Insert no-name parameter.
  st_insert(temp, "", Integer, 0, 3);
  fun->paramType[fun->paramNum++] = Integer;

  temp_scope = cur_scope;
  traverse(syntaxTree,insertNode,exitScope);
  if (TraceAnalyze)
  { fprintf(listing,"\n< Symbol Table >\n");
    printSymTab(listing);
    fprintf(listing, "\n< Function Table >\n");
    printFunTab(listing);
    fprintf(listing, "\n< Function And Global Variables >\n");
    printGlobalTab(listing);
    fprintf(listing, "\n< Function Parameters and Local Variables >\n");
    printFunVarTab(listing);

  }
}
```

- insertNode 함수는 각 node 의 종류에 따라 적절한 작업을 하여 symbol table 을 구축 하는 함수 이다.
- 예시로 몇개 node 를 설명하면, FunK node 같은 경우에는 함수를 define 하는 부분 이므로 새로운 scope 가 만들어질 필요가 있다. 그러므로 scope_insert 를 이용해 새로운 scope 를 만들고, st_insert 와 ft_insert 로 symbol table 과 function table에 기록한다.

```
case FunK:
  if( (check_scope = scope_insert(t->attr.name, cur_scope)) == NULL){
    // Scope already exist which means
    // function name is invalid. Redefinition error.
    sprintf(buf, "Function name %s is already defined", t->attr.name);
     redefineError(t, buf);
  }
  else{
    // Insert function to st.
    st_insert(cur_scope, t->attr.name, t->type, t->lineno, 2);
    // Change scope.
    // Temp scope and check_scope is diffrent now.
    temp_scope = cur_scope;
    cur_scope = check_scope;
    // Insert function table.
    ft_insert(t->attr.name, t->type);
}
```

- CompK 는 function definition 에서 사용 될때는 scope 를 만들지 않지만, if, while 문등으로 사용 될 때는 scope 를 만들어야 한다. FunK 에서는 새로운 scope 가 만들어져 전역변수로 선언된 temp_scope 가 현재 scope 와 다르기 때문에 그 점을 이용하여 scope 를 만들지 만들지 않을지 결정한다.

```
case CompK:
  if (cur_scope == temp_scope){
    // It is if/while statement. Insert, change scope.
    sprintf(buf, "%s.%d", cur_scope->name, cur_scope->child_num);
    cur_scope = scope_insert(buf, cur_scope);
  }
  temp_scope = cur_scope;
  break;
```

- 다른 node 들에서도 symbol table 에 symbol 을 추가하거나 line number 를 기존 symbol 에 추가한다. 동시에 symbol table 을 이용하여 symbol 이 재정의 되는지 (redefinition error), 정의 되지 않은 symbol 이 사용되는지 (undefinition error) 를 검사한다.
- 상세한 코드 구현은 생략한다.
- Symbol table 구축이 완료되면 type check 를 할 수 있다. 만들어진 symbol table 과 syntax tree node 를 이용하여 checkNode 함수에서 type check 를 한다.
- type check 함수는 syntax tree 를 순회하며 preproc 으로 enter_scope, postproc 으로 checkNode 를 실행하는데 insertNode 와 다르게 checkNode 를 postporc 으로 실행하는 이유는 syntax tree 의 밑에쪽부터 type check 를 해야 여러 symbol 들이 연산 후의 type 을 알 수 있기 때문이다.
- checkNode 함수는 insertNode 함수와 마찬가지로 syntax tree 의 node 종류에 따라 알맞은 동작을 하는데 OpK 나 AssignK 등 자식 node 에 따라 type 이 달라지는 node 들에 type 을 부여 하며 type check 를 한다.
- 몇개 node 의 예를 들어보면 다음과 같다.
- OpK 는 두 operand 의 type 이 같은지 판단하고, type 이 같다면 type 을 두 operand 의 type 으로 node 의 type 을 설정한다. 현재 operation 이 가능한 type 은 integer 밖에 없다.

```
case OpK:
 if((t->child[0]->type != Integer) || (t->child[1]->type != Integer)){
   typeError(t, "Operation between diffrent type.");
 }
 else{
    t->type = Integer;
 }
 break;
```

- CallK 는 현재 scope 에 함수와 같은 이름의 변수가 선언되어 있는지 확인하고, 같은 이름이 있으면 error 를 표시한다. (이와같은 경우 gcc 에서 에러가 나는것을 확인하였습니다.) 또, parameter 개수, 각 parameter 의 type, parameter 가 void 인지 아닌지 확인 한다. 이때 function table 을 많이 사용하게 된다.

```
case CallK:
  if(st_excluding_parent(cur_scope, t->attr.name) != NULL){
            // When a variable with same name of function exist,
  typeError(t, "invalid function call. It's not a function in current scope");
  }
  fun = ft_lookup(t->attr.name);
  t->type = fun->type;

  if(fun->paramNum == 0){
    if(t->child[0] != NULL){
      typeError(t, "invalid function call. Argument exist on void function.");
     }
  }
  else{ // Parameter exist.
   child = t->child[0];
   for(i=0; i<fun->paramNum; i++){ //check each parameter
     if(child == NULL){
       typeError(t, "invalid function call. Number of argument mismatch.");
       break;
     }
     if(fun->paramType[i] != child->type){
       typeError(t, "invalid function call. Argument type mismatch.");
       break;
     }
     else{
       if(child == NULL){
         typeError(t, "invalid function call. Number of parameter mismatch");
         break;
       }
       child = child->sibling;
     }
   }
   if((child != NULL) && (child->sibling != NULL)){
    typeError(t, "invalid function call. Number of parameter mismatch");
   }
}
break;
```

- 비슷한 방법으로 type check 를 진행하여 semantic 오류가 있는지 check 한다.

실행결과

1) 예시 테스트 파일 sort.cm

- 파일 정보 :

```
parallels@ubuntu:~/Desktop/compiler/git/3_Semantic$ cat sort.cm
/* A program to perform selection sort on a 10 element array */

int x[10];
int minloc( int a[], int low, int high ) {
        int i;
        int x;
        int k;
        k = low;
        x = a[low];
        i = low +  1;
        while( i < high ) {
                if( a[i] < x ) {
                        x = a[i];
                        k = i;
                }
                i = i + 1;
        }
        return k;
}

void sort(int a[], int low, int high) {
        int i;
        int k;
        i = low;
        while( i < high -  1 ) {
                int t;
                k = minloc( a, i, high );
                t = a[k];
                a[k] = a[i];
                a[i] = t;
                i = i + 1;
        }
}

void main(void) {
        int i;
        i = 0;
        while( i < 10 ) {

                x[i] = input();
                i = i + 1;
        }
        sort(x, 0, 10);
        i = 0;
        while( i < 10 ) {
                output(x[i]);
                i = i + 1;
        }
}
```

(https://hconnect.hanyang.ac.kr/2019_ELE4029_12214/2019_ELE4029_2015004493/uploads/8a410baf6814018f4b700a70bfd91cb4/%EC%8A%A4%ED%81%AC%EB%A6%B0%EC%83%B7_2019-12-12_%EC%98%A4%ED%9B%84_11.29.52.png)

- Semantic analysis

```
parallels@ubuntu:~/Desktop/compiler/git/3_Semantic$ ./cminus sort.cm

C-minus COMPILATION: sort.cm

Building Symbol Table...

< Symbol Table >
Variable Name  Variable Type  Scope Name  Location   Line Numbers
-------------  -------------  ----------  --------   ------------
main           Function       global         5          35
sort           Function       global         4          21   43
input          Function       global         0           0   40
minloc         Function       global         3           4   27
output         Function       global         1           0   46
x              IntegerArray   global         2           3   40   43   46
               Integer        output         0           0
low            Integer        minloc         1           4    8    9   10
a              IntegerArray   minloc         0           4    9   12   13
i              Integer        minloc         3           5   10   11   12   13   14   16   16
k              Integer        minloc         5           7    8   14   18
x              Integer        minloc         4           6    9   12   13
high           Integer        minloc         2           4   11
low            Integer        sort           1          21   24
a              IntegerArray   sort           0          21   27   28   29   29   30
i              Integer        sort           3          22   24   25   27   29   30   31   31
k              Integer        sort           4          23   27   28   29
high           Integer        sort           2          21   25   27
t              Integer        sort.0         0          26   28   30
i              Integer        main           0          36   37   38   40   41   41   44   45   46   47   47
```

(https://hconnect.hanyang.ac.kr/2019_ELE4029_12214/2019_ELE4029_2015004493/uploads/c33ca3f1e1c4ae210b5a57173c0fe66c/%EC%8A%A4%ED%81%AC%EB%A6%B0%EC%83%B7_2019-12-12_%EC%98%A4%ED%9B%84_11.28.14.png)

```
< Function Table >
Function Name  Scope Name  Return Type  Parameter Name   Parameter Type
-------------  ----------  -----------  --------------   --------------
main           global         Void                         Void
sort           global         Void
                                           low             Integer
                                           a               IntegerArray
                                           high            Integer
input          global         Integer                      Void
minloc         global         Integer
                                           low             Integer
                                           a               IntegerArray
                                           high            Integer
output         global         Void
                                                           Integer

< Function And Global Variables >
ID Name  ID Type  Data Type
-------  -------  ---------
main           Function       Void
sort           Function       Void
input          Function       Integer
minloc         Function       Integer
output         Function       Void
x              Variable       Integer Array
```

(https://hconnect.hanyang.ac.kr/2019_ELE4029_12214/2019_ELE4029_2015004493/uploads/6d4f405d3de9d10f3f28620ecbe04eea/%EC%8A%A4%ED%81%AC%EB%A6%B0%EC%83%B7_2019-12-12_%EC%98%A4%ED%9B%84_11.28.26.png)

```
< Function Parameters and Local Variables >
Scope Name  Nested Level  ID Name  Data Type
----------  ------------  -------  ---------

output         1                     Integer

minloc         1        low          Integer
minloc         1        a            IntegerArray
minloc         1        i            Integer
minloc         1        k            Integer
minloc         1        x            Integer
minloc         1        high         Integer



sort           1        low          Integer
sort           1        a            IntegerArray
sort           1        i            Integer
sort           1        k            Integer
sort           1        high         Integer

sort           2        t            Integer

main           1        i            Integer
```

(https://hconnect.hanyang.ac.kr/2019_ELE4029_12214/2019_ELE4029_2015004493/uploads/81341f28a1896e3a05cc111eb0d72dc3/%EC%8A%A4%ED%81%AC%EB%A6%B0%EC%83%B7_2019-12-12_%EC%98%A4%ED%9B%84_11.28.33.png)

주어진 예시와 거의 동일하게 symbol table 을 만들 수 있었다. 다른점은 주어진 예시에서는 symbol table 에 output parameter 를 보이지 않게 했지만 이름 없는 output parameter 도 symbol table 에서 의미가 있다고 생각해 출력해주었다.

2) PDF 의 type error 예시들

- PDF 에 type error 를 출력하는 4개의 예시들을 semantic check 해보았다.

```
parallels@ubuntu:~/Desktop/compiler/git/3_Semantic$ cat test.txt
int x(int y){
    return y+1;
}

int main(void){
    int a;
    int b;
    int c;
    return x(a,b,c);
}
parallels@ubuntu:~/Desktop/compiler/git/3_Semantic$ ./cminus test.txt

C-minus COMPILATION: test.txt
Type error at line 9: invalid function call. Number of parameter mismatch
parallels@ubuntu:~/Desktop/compiler/git/3_Semantic$
```

(https://hconnect.hanyang.ac.kr/2019_ELE4029_12214/2019_ELE4029_2015004493/uploads/41e200ea8b999e362af025a56cd709c1/%EC%8A%A4%ED%81%AC%EB%A6%B0%EC%83%B7_2019-12-

12_%EC%98%A4%ED%9B%84_11.39.26.png)

```
parallels@ubuntu:~/Desktop/compiler/git/3_Semantic$ cat test.txt
int main(void){
    void x;
    return 0;
}
parallels@ubuntu:~/Desktop/compiler/git/3_Semantic$ ./cminus test.txt

C-minus COMPILATION: test.txt
Type error at line 2: type 'void' is not avaliable for variable
parallels@ubuntu:~/Desktop/compiler/git/3_Semantic$
```

(https://hconnect.hanyang.ac.kr/2019_ELE4029_12214/2019_ELE4029_2015004493/uploads/ae5ee4ba6a39b4706711d4f83d280042/%EC%8A%A4%ED%81%AC%EB%A6%B0%EC%83%B7_2019-12-12_%EC%98%A4%ED%9B%84_11.38.01.png)

```
parallels@ubuntu:~/Desktop/compiler/git/3_Semantic$ cat test.txt
int main(void){
  return x;
}
parallels@ubuntu:~/Desktop/compiler/git/3_Semantic$ ./cminus test.txt

C-minus COMPILATION: test.txt
Undefinition error at line 2: Variable x is not defined
Type error at line 2: invalid return type. Return type should be int
parallels@ubuntu:~/Desktop/compiler/git/3_Semantic$
```

(https://hconnect.hanyang.ac.kr/2019_ELE4029_12214/2019_ELE4029_2015004493/uploads/63ba94f78b515578df414081accba450/%EC%8A%A4%ED%81%AC%EB%A6%B0%EC%83%B7_2019-12-12_%EC%98%A4%ED%9B%84_11.37.18.png)

```
parallels@ubuntu:~/Desktop/compiler/git/3_Semantic$ cat test.txt
int main(void){
  int x;
  int y[3];

  x+y;

  return 0;
}
parallels@ubuntu:~/Desktop/compiler/git/3_Semantic$ ./cminus test.txt

C-minus COMPILATION: test.txt
Type error at line 5: Operation between diffrent type.
parallels@ubuntu:~/Desktop/compiler/git/3_Semantic$
```

(https://hconnect.hanyang.ac.kr/2019_ELE4029_12214/2019_ELE4029_2015004493/uploads/d45fa86606ea7526b18152e3a909a893/%EC%8A%A4%ED%81%AC%EB%A6%B0%EC%83%B7_2019-12-12_%EC%98%A4%ED%9B%84_11.36.24.png)

- 함수 call 에서 argument 의 개수가 mismatch 일때
- void 형 변수를 선언 했을 때
- 선언되지 않은 변수를 사용 했을 때
- operation 에서 operand 의 type 이 같지 않을 때
- 위 네 상황 모두에서 오류를 잘 발견 하였다.
- 3번째 상황에서 int 형이 아닌 return 값이라는 오류는 선언되지 않은 변수는 int 형이 아니기 때문에 undefinition error 와 type error 모두 출력된 것이다.