C-Minus Semantic check

Environment : Ubuntu 18.04
Compilation method : gcc 를 이용해 make로 compile 해 동작을 수행하였다.

Main.c

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

/* set NO_CODE to TRUE to get a compiler that does not
 * generate code
 */
#define NO_CODE TRUE
```

실제 code를 generate 하지 않으므로 NO_CODE option을 TRUE로 변경

Symtab.h / symtab.c

```
typedef struct BucketListRec
   { char * name;
     LineList lines;
     int memloc ; /* memory location for variable */
     ExpType type;
     TreeNode* tree;
     struct BucketListRec * next;
   } * BucketList;

typedef struct ScopeListRec
   { char * name;
     BucketList bucket[SIZE];
     struct ScopeListRec * parent;
     int level;
   } * Scope;
```

Scope를 관리할 Scope Struct와 내용을 담을 Bucket Struct를 추가 한다.

```
Scope scope_top();
void scope_pop();
void scope_push(Scope sc);
Scope scope_make(char* scope_name);
```

```
static Scope Scope_Arr[SIZE];
static Scope Scope_Stack[SIZE];

static int Scope_Arr_Num = 0;
static int Scope_Stack_Num = 0;
```

Scope를 stack 과 array를 통해 관리한다.
이 때 stack은 scope들의 hierarchy를 위해 사용하고 array는 지금까지 나온 scope들을 모두 추가해 symbol table 출력을 할 때 사용한다.

```
void st_insert(char * name, ExpType type, int lineno, int loc, TreeNode *t);

BucketList st_lookup ( char * scope, char * name );
BucketList st_lookup_excluding_parent ( char * scope, char * name );
```

Symbol을 추가하는 함수 st_insert와
Symbol을 찾는 함수 st_lookup/st_lookup_excluding_parent
St_lookup 은 상위 스코프 까지 모두 찾고 st_lookup_excluding_parent 는 해당 scope에서만 찾는다.

Symbol table 출력은 scope별로 차례대로 출력을 하게 만들었다.

Analyze.c
Init

- 내장 함수인 input과 output을 global scope에 추가해준다.
- Output의 경우 parameter가 있으므로 output scope을 추가해주고 parameter value 를 추가해 준다.

InsertNode

- Function, variable이 선언 되었을 때 오류가 없다면 symbol table에 추가해 주고 function에 경우 새로운 scope를 만들어준다

- Compound statement가 나온 경우 새로운 scope를 만들어서 추가해준다.

checkNode

- 만들어진 symbol table을 바탕으로 오류가 있는지 없는 지 검사하여 오류가 있는 경우는 경우에 맞는 오류를 출력해준다.

결과

Test1.cm

```
1 /* A program to perform Euclid's
2    Algorithm to computer gcd **/
3
4 int gcd (int u, int v)
5 {
6   if (v == 0) return u;
7   else return gcd(v,u-u/v*v);
8   /* u-u/v*v == u mod v */
9 }
10
11 void main(void)
12 {
13   int x; int y;
14   x = input(); y = input();
15   output(gcd(x,y));
16 }
```

```
Scope Name : global (nested 0)
Variable Name Type      Data Type Location  Line Numbers
------------- --------- --------- --------- ------------
main          Function  Void      3         11
input         Function  Int       0         0 14 14
output        Function  Void      1         0 15
gcd           Function  Int       2         4 7 15

Scope Name : output (nested 1)
Variable Name Type      Data Type Location  Line Numbers
------------- --------- --------- --------- ------------
value         Varaible  Int       0         0

Scope Name : gcd (nested 1)
Variable Name Type      Data Type Location  Line Numbers
------------- --------- --------- --------- ------------
u             Varaible  Int       0         4 6 7 7
v             Varaible  Int       1         4 6 7 7 7

Scope Name : main (nested 1)
Variable Name Type      Data Type Location  Line Numbers
------------- --------- --------- --------- ------------
x             Varaible  Int       0         13 14 15
y             Varaible  Int       1         13 14 15
```

```
Checking Types...

Type Checking Finished
```

Test2.cm

```
1 void main(void)
2 {
3   int i; int x[5];
4
5   i = 0;
6   while( i < 5 )
7   {
8     x[i] = input();
9
10    i = i + 1;
11  }
12
13  i = 0;
14  while( i <= 4 )
15  {
16    if( x[i] != 0 )
17    {
18      output(x[i]);
19    }
20  }
21 }
```

```
Scope Name : global (nested 0)
Variable Name Type      Data Type Location  Line Numbers
------------- --------- --------- --------- ------------
main          Function  Void      2         1
input         Function  Int       0         0 8
output        Function  Void      1         0 18

Scope Name : output (nested 1)
Variable Name Type      Data Type Location  Line Numbers
------------- --------- --------- --------- ------------
value         Varaible  Int       0         0

Scope Name : main (nested 1)
Variable Name Type      Data Type Location  Line Numbers
------------- --------- --------- --------- ------------
i             Varaible  Int       0         3 5 6 8 10 10 13 14 16 18
x             Varaible  Int[]     1         3 8 16 18

Scope Name : main (nested 2)
Variable Name Type      Data Type Location  Line Numbers
------------- --------- --------- --------- ------------

Scope Name : main (nested 2)
Variable Name Type      Data Type Location  Line Numbers
------------- --------- --------- --------- ------------

Scope Name : main (nested 3)
Variable Name Type      Data Type Location  Line Numbers
------------- --------- --------- --------- ------------
```

```
Checking Types...

Type Checking Finished
```

Type_error.cm

```
1 int main(void)
2 {
3   int x;
4   int y[3];
5
6   x+y;
7   return 0;
8 }
```

```
Checking Types...
Type error at line 6: IntegerArr can't be Operand

Type Checking Finished
```

Void_var.cm

```
1 int main(void)
2 {
3   void x;
4   return 0;
5 }
```

```
symbol error at line 3: void can't be variable type
```

Func.cm

```
1 int x(int y)
2 {
3   return y +1;
4 }
5 int main(void)
6 {
7   int a;
8   int b;
9   int c;
10
11  return x(a,b,c);
12 }
```

```
Type error at line 11: parameter Number should be same with function definition
```

Undeclare.cm

```
1 int main(void)
2 {
3   return x;
4 }
```

```
Type error at line 3: Return type shoud be Integer
```

Indexing.cm

```
1 int main(void)
2 {
3   int x[5];
4   x[output(5)] = 3+5;
5
6   return 0;
7 }
```

```
Type error at line 4: array index should be Integer
```

Condition.cm

```
1 int main(void)
2 {
3   if(output(5)){}
4   return 0;
5   while(output(5)){
6   return 0;
7   }
8 }
```

```
Type error at line 3: if-test stmt can't be Void
Type error at line 5: while-test stmt can't be Void
```