
Project #3. Semantic Analysis

Symbol Table & Type Checker

Yongjun Park
Hanyang University



Project Goal

- **C-Minus Semantic Analyzer Implementation**
 - **Detect All Semantic Errors** using Symbol Table & Type Checker

Symbol Table & Type Checker

- Implement symbol table and type checker
- Traverse syntax tree created by parser
- Files to modify
 - globals.h
 - main.c
 - util.h, util.c
 - symtab.h, symtab.c
 - analyze.h, analyze.c



main.c

- Modify NO_ANALYZE, TraceParse, and TraceAnalyze to suit your assignment

```
1  /* File: main.c */
2  /* Main program for TINY compiler */
3  /* Compiler Construction: Principles and Practice */
4  /* Kenneth C. Louden */
5  /****** */
6
7  #include "globals.h"
8
9  /* set NO_PARSE to TRUE to get a scanner-only compiler */
10 #define NO_PARSE FALSE
11 /* set NO_ANALYZE to TRUE to get a parser-only compiler */
12 #define NO_ANALYZE FALSE
13
14 /* set NO_CODE to TRUE to get a compiler that does not
15  * generate code
16  */
17 #define NO_CODE FALSE
18
19 #include "util.h"
20 #if NO_PARSE
21 #include "scan.h"
22 #else
23 #include "parse.h"
24 #if !NO_ANALYZE
25 #include "analyze.h"
26 #if !NO_CODE
27 #include "cgen.h"
28 #endif
29 #endif
30 #endif
31
32 /* allocate global variables */
33 int lineno = 0;
34 FILE * source;
35 FILE * listing;
36 FILE * code;
37
38 /* allocate and set tracing flags */
39 int EchoSource = FALSE;
40 int TraceScan = FALSE;
41 int TraceParse = FALSE;
42 int TraceAnalyze = FALSE;
43 int TraceCode = FALSE;
```

```
9  /* set NO_PARSE to TRUE to
10 #define NO_PARSE FALSE
11 /* set NO_ANALYZE to TRUE t
12 #define NO_ANALYZE FALSE
13
14 /* set NO_CODE to TRUE to g
15  * generate code
16  */
17 #define NO_CODE FALSE
```

```
38 /* allocate and set tracing
39 int EchoSource = FALSE;
40 int TraceScan = FALSE;
41 int TraceParse = FALSE;
42 int TraceAnalyze = FALSE;
43 int TraceCode = FALSE;
```

Hint: symtab.h, symtab.c

- Add scope and type to symbol table
- Implement hash table

```
-void st_insert( char * name, int lineno, int loc );  
+void st_insert( char * scope, char * name, ExpType type, int lineno, int loc );  
  
/* Function st_lookup returns the memory  
 * location of a variable or -1 if not found  
 */  
-int st_lookup ( char * name );  
+BucketList st_lookup ( char * scope, char * name );  
+BucketList st_lookup_excluding_parent ( char * scope, char * name );
```

```
typedef struct BucketListRec  
{ char * name;  
  ExpType type;  
  LineList lines;  
  int memloc ; /* memory location for variable  
  struct BucketListRec * next;  
} * BucketList;  
  
/* The record for each scope,  
 * including name, its bucket,  
 * and parent scope.  
 */  
typedef struct ScopeListRec  
{ char * name;  
  BucketList bucket[SIZE];  
  struct ScopeListRec * parent;
```

Hint: analyze.c

- **Modify symbol table generation**
 - buildSymtab(), insertNode(), traverse(), ... , scope and type concept
- **Modify the checkNode() function to check the semantics of C-Minus**
- **Insert built-in function**
 - Input(), output()
- **This is just hint, not correct answer.**



Symbol Table in Tiny

```
1: { Sample program
2:   in TINY language -
3:   computes factorial
4: }
5: read x; { input an integer }
6: if 0 < x then { don't compute if x <= 0 }
7:   fact := 1;
8:   repeat
9:     fact := fact * x;
10:    x := x - 1
11:  until x = 0;
12:  write fact { output factorial of x }
13: end
```

<Location>
Counter for variable memory locations.
Never overlapped in a scope.

Variable Name	Location	Line Numbers					
x	0	5	6	9	10	10	11
fact	1	7	9	9	12		

Symbol Table in C-Minus

```

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: }

```

Name	Type	Location	Scope	Line Numbers
output	Void	0	global	0 15
Input	Integer	1	global	0 14 14
gcd	Integer	2	global	4 7 15
main	Void	3	global	11
u	Integer	0	gcd	4 6 7 7
v	Integer	1	gcd	4 6 7 7 7
x	Integer	0	main	13 14 15
y	Integer	1	main	13 14 15

Symbol Table in C-Minus

- Build with TraceAnalyze = TRUE; in main.c

C-MINUS COMPILATION: ../../submission/2019-02/scanner/grading/testcase/gcd.cm

Building Symbol Table...

< Symbol Table >

Variable Name	Variable Type	Scope Name	Location	Line Numbers				
x	Integer	main	0	13	14	15		
y	Integer	main	1	13	14	15		
main	Function	global	3	16				
input	Function	global	0	0	14	14		
output	Function	global	1	0	15			
gcd	Function	global	2	9	7	15		
u	Integer	gcd	0	4	6	7	7	
v	Integer	gcd	1	4	6	7	7	7

Symbol Table in C-Minus

- Build with TraceAnalyze = TRUE; in main.c

< Function Table >

Function Name	Scope Name	Return Type	Paramter Name	Paramter Type
main	global	Void		Void
input	global	Integer		Void
output	global	Void		Integer
gcd	global	Integer	u v	Integer Integer

< Function and Global Variables >

ID Name	ID Type	Data Type
main	Function	Void
input	Function	Integer
output	Function	Void
gcd	Function	Integer



Symbol Table in C-Minus

- Build with TraceAnalyze = TRUE; in main.c

< Function Parameters and Local Variables >

Scope Name	Nested Level	ID Name	Data Type
-----	-----	-----	-----
main	1	x	Integer
main	1	y	Integer
gcd	1	u	Integer
gcd	1	v	Integer

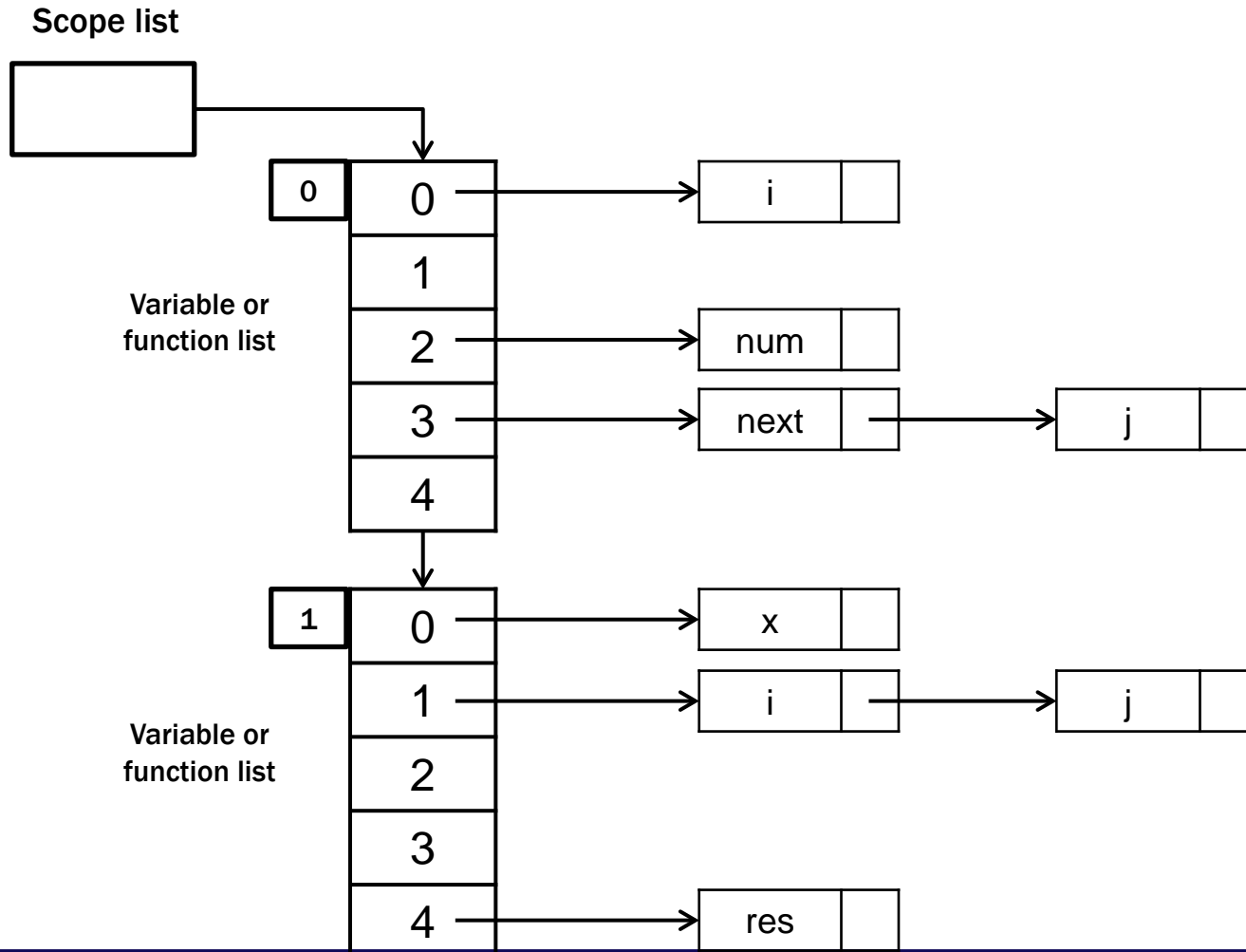


Built-in function

- **int input(void)**
 - One integer value is input from the user.
- **void output(int)**
 - Prints the value of arg.
- **These two functions are considered to be global functions defined by default.**



Symbol Table in C-Minus (Hint)



Implementation Notes

- Variables follow scope of each compound statement.
- Throws an error when an undeclared variable is used.
- Built-in functions should be always accessible.
- As long as the scope concept is implemented properly, you can use any implementation. This is not output.



Type Checker

- **Type checking for functions and variables.**
 - The type “void” is only available for functions.
 - Check return type.
 - Verify the type match of two operands when assigning.
 - Check the argument number when calling function.
 - Check if conditional of “If” or “While” has a value.
 - Check other things by referring to C-Minus syntax.
 - Note: C-minus Type → void, int, int[]



Goal: Semantic Error Detection

- Undefined Variables/Function
- Redefined Variables/Function
- Type Check
 - Expression: “int[] + int[]” and “void + void” not allowed
 - Return Type, Assignment Type, Void Variable...
 - Function Parameter(Number of Parameter, Type)
- ... And everything you can consider
- Output Requirements: **Line Number, Error Type**



Examples

```
1  int main(void)
2  {
3      int x;
4      int y[3];
5
6      x + y;
7
8      return 0;
9  }
```

C-MINUS COMPILATION: ./test/simple/type_error.cm
Error: Type error at line 6: invalid expression

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

C-MINUS COMPILATION: ./test/simple/void_var.cm
Error: Variable Type cannot be Void at line 3 (name : x)

Examples

```
1  int x(int y)
2  {
3      return y + 1;
4  }
5
6  int main(void)
7  {
8      int a;
9      int b;
10     int c;
11
12     return x( a, b, c );
13 }
```

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

C-MINUS COMPILATION: ./test/simple/func.cm
Error: Type error at line 12: invalid function call

C-MINUS COMPILATION: ./test/simple/undeclare.cm
Error: Undeclared Variable "x" at line 3
Error: Type error at line 3: invalid return type

Report

- **Guideline**

- Build environment(OS, compiler, ...).
- Semantic analysis implementation process and source code description of principal parts.

- **File format**

- MS Word, HWP, PDF, ...
- GitLab Wiki Not Allowed
(If you want, write report in markdown and **take screenshot** and submit in other formats(PDF, JPEG, ...))



Submission

- **Submission directory in repository: 3_Semantic**
(Please submit all your codes and reports into the submission directory)
- **Questions**
 - compiler.teachingassistant@gmail.com
- **Submission deadline**
 - Push until Sunday, December 15, 2019, 23:59:59.
 - Master branch will be cloned at 0:00 on Monday, December 16, 2019



Contact (Prof. Yongjun Park)

- **Submission**

- Where: Using GitLab

- <https://hconnect.hanyang.ac.kr>

- Git Project:

- https://hconnect.hanyang.ac.kr/2019_ELE4029_12214/2019_ELE4029_Student#.git

- Example URL: https://hconnect.hanyang.ac.kr/2019_ELE4029_12214/2019_ELE4029_2019000000.git

- The Submission Directory is in Repo: 1_Scanner, 2_Parser, **3_Semantic**, ...

- Teaching Assistant

- compiler.teachingassistant@gmail.com

- If you don't have the GITLAB account, **please let him know** the account information after creation.

- **What to submit**

- **All the source codes and the report**

Q & A

