# Assignment 5

**csem** – C semantic analyzer

This assignment is designed by **Prof. Whalley** in FSU Computer Science, Tallahassee and revised by Gang-Ryung Uh, Ph.D at FSUPC Computer Science

* ***csem*** reads a C program (actually a subset of C) from its standard input and compiles it into a list of intermediate language quadruples on its standard output. The form of the quadruple operators appears below:

*x* **:=** *y op z* operate on *y* and *z* and place result in *x*

**bt** *x lab* branch to *lab* if *x* is true

**br** *lab* branch to *lab*

*x* **:= global** *name* yield address of global identifier *x* **:= local** *name**n* yield address of local name *nth*

*x* **:= param** *name n* yield address of parameter *name* *nth*

*x* **:=** *c* yield value of constant value *c*

*x* **:=** *s* yield address of character string *s*

**formal** *name**type**n* allocate the formal *name of type* (either int or double)having *n* bytes

**alloc** *name type n* allocate the global *name* of *type* (either int or double)having *n* bytes

**localloc** *name*type*n* allocate the local *name* *of type* (either int or double) having *n* bytes

**func** *name type* begin function *name with* type (either int or double)

**fend** end function

*lab***=***y* define *lab* to be *y*

**bgnstmt** *n* beginning of statement at line *n*

* ***name*** denotes an identifier from the C program. ***n*** denotes an integer. ***c*** denotes a C integer constant. ***s*** denotes a string enclosed by double quotes. ***x, y,*** and ***z*** denote quadruple temporaries. ***lab*** denotes the location of a quadruple or a reference to a symbol defined later by a "**lab=y**" command. ***op*** denotes any of the C operators below:

**== != <= >= < > = | ˆ & << >> + - \* / %**

operate on *x* and *y*

**˜** invert *x*

**-** negate *x*

**@** dereference *x*

**cv** convert *x*

**f** call function *y* with *n* arguments

**arg** pass *x* as an argument

**ret** return *x*

**[]** index *z* into *y*

followed by ***i*** (for the integer version of the operator) or by ***f*** (for the floating-point version). *y* is omitted for unary operators. You should assume all bitwise operators (**| ˆ & << >> ˜**) and **%** only operate on integer values.

* For example,

**int m[6];**

**scale(double x) {**

**int i;**

**if (x == 0) return 0;**

**for(i = 0; i < 6; i += 1) m[i] \*= x;**

**return 1;**

**}**

compiles into the intermediate operations below (actually only one column)

|  |  |  |
| --- | --- | --- |
| alloc m 17 24  func scale 1  formal x 1 4  localloc i 1 4  bgnstmt 6  t1 := param x 0  t2 := @i t1  t3 := 0  t4 := t2 ==i t3  bt t4 B1  br B2  label L1  bgnstmt 7  t5 := 0  reti t5  label L2  B1=L1  B2=L2 | bgnstmt 8  t6 := local i 0  t7 := 0  t8 := t6 =i t7  label L3  t9 := local i 0  t10 := @i t9  t11 := 6  t12 := t10 <i t11  bt t12 B3  br B4  label L4  t13 := local i 0  t14 := 1  t15 := @i t13  t16 := t15 +i t14  t17 := t13 =i t16  br B5  label L5  bgnstmt 9  t18 := local i 0  t19 := @i t18 | t20 := global m  t21 := t20 []i t19  t22 := param x 0  t23 := @i t22  t24 := @i t21  t25 := t24 \*i t23  t26 := t21 =i t25  br B6  label L6  B3=L5  B4=L6  B5=L3  B6=L4  bgnstmt 10  t27 := 1  reti t27  fend |

* Your assignment is to write the semantic actions for the **csem** program to produce the desired intermediate code in quadruples.
* You first need to download project files from [**https://github.com/gangryunguh/csem-student-template.git**](https://github.com/gangryunguh/csem-student-template.git)
* Submit only the file **semdum.c** to Canvas course assignment link.

|  |  |
| --- | --- |
| cc.h | include file |
| cgram.y | yacc grammar for subset of C |
| scan.c | lexical analyzer |
| sem.h | declares prototypes for routines which being referenced in cgram.y |
| **semdum.c** | defines prototypes for routines being listed in sem.h, which you need to implement |
| semutil.c | defines utility routines for the semantic actions |
| semutil.h | declares prototypes for semutil routines |
| sym.c | defines symbol table management |
| sym.h | declares prototypes for routines in sym.c |

* Submit only the file **semdum.c** to Canvas course assignment link.
* Below is the order in which I recommend you implement the semantic routines.

**fname**

**fhead**

**ftail**

**bgnstmt**

**id**

**string**

**op1**

**exprs**

**call**

**-------------- enough to get through the second example**

**con**

**m**

**doret**

**set**

**op2**

**index**

**ccexpr**

**rel**

**n**

**backpatch**

**doif**

**dofor**

**-------------- enough to get through the first example**

**doifelse dowhile**

**dodo**

**ccand**

**ccor**

**ccnot**

**opb**

**startloopscope endloopscope docontinue**

**dobreak**

**labeldcl**

**dogo**