# 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* iff *x* is true

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

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

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

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

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

**formal** *name**n* allocate the formal *name* having *n* bytes

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

**localloc** *name**n* allocate the local *name* having *n* bytes

**func** *name type* begin function *name with* type – 1 for int type and 2 for double type

**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,

**double 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)

A picture containing text, receipt, screenshot

Description automatically generated

* Your assignment is to write the semantic actions for the csem program to produce the desired intermediate code. The following files, which will comprise part of your program, are in the *˜uh/cis4930/proj5* directory.

cc.h. - include file

cgram.y. - yacc grammar for subset of C

makefile. - csem makefile

scan.c. - lexical analyzer

scan.h. - defines prototypes for scan routines sem.h. - defines prototypes for routines in sem.c semutil.c - utilitity routines for the semantic actions semutil.h - defines prototypes for semutil routines

sym.c - symbol table management

sym.h - defines prototypes for routines in sym.c

* The **makefile** will create an executable called **csem** in the current directory. You should copy the **~uh/cis4930/proj5/semdum.c** file into your directory as **sem.c**. This file contains stubs for the semantic action routines. While I have given you read access to the other \*.c and \*.h files and the makefile, you ***should not*** copy them into your directory. You are only allowed to update the file **sem.c** and will not be allowed to update any other files. You should make additional functions in this file to abstract common operations. When making your executable, refer to the **makefile** by using the command **make -f ˜uh/cis4930/proj5/makefile**, which uses the other \*.c and \*.h files when producing the executable. This will allow me to make updates (and perhaps occasional fixes to problems) that everyone will instantly receive. The **run.sh** script in **~uh/cis4930/proj5** takes a single intermediate test file as a command line argument and attempts to execute the intermediate code. You should test your intermediate code on the machine ***program****.*
* Submit only the file **sem.c** to Canvas course assignment link.

Another Example

This example shows the intermediate code generation for a test function with multiple formal parameters, locals, and actual arguments.

**test(int a, int b)**

**{**

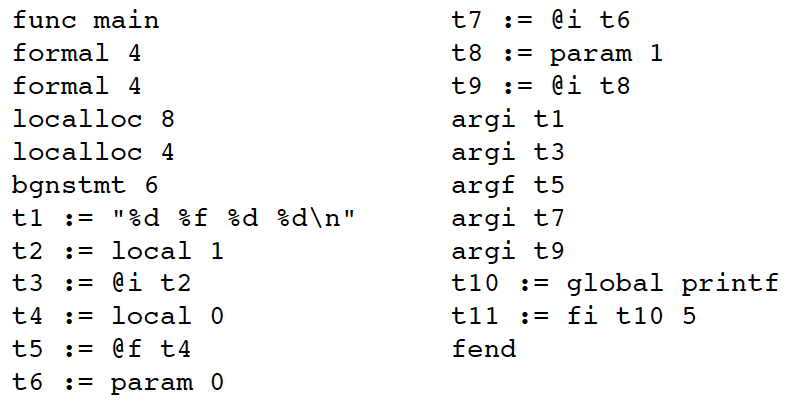
**double d;**

**int i;**

**printf("%d %f %d %d\n", i, d, a, b);**

**}**

compiles into



* 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**

**dogoto**