

## Programming Assignment #2 – Due Tuesday, April 3.

This is a pair programming assignment. You can choose a partner and work in pairs, or if you prefer you are allowed to work alone. The program must be implemented in “C” using Lex and Yacc.

You are to extend your integer calculator *icalc* from Program #1 to be a more powerful calculator, called *scalc* (*super calc*), which includes support for double precision floating point constants, variables, and functions. This calculator should *strongly typed*, meaning that it is an error to do mixed-mode arithmetic operations. For example, the expression  $3.14 + 2$  is an error because the operands to the  $+$  operator are not the same type. The expression  $x+2$  is an error if variable  $x$  contains a floating point number, but is legal if  $x$  contains an integer. It is recommended to use dynamic type checking.

Extend the previous calculator with the following features (not listed in any particular order):

1. Add named variables so that users can assign values to variables and use those variables in subsequent expressions in the calculator. Variables require that you implement assignment expressions of the form `var = expr`. Each assignment expression evaluates to the value of the expression on the RHS, and updates the variable on the LHS. Variable names should use the same form as in C or Java: a letter followed by letters or digits (but no more than 10 characters). It is an error to use a variable before it has been given a value. Note that the assignment operator is right associative operators so assignment expressions of the form: `var = var = ... = var = expr` are legal.

```
scalc> a2=5
5
scalc> a2+7
12
scalc> a = b = 9
9
```

2. Add the unary pre/post-increment (`++`) and pre/post-decrement (`--`) C operators so that they can only be applied to variables. Note that these unary operators have higher precedence than binary operators. The result of applying these operators should have the same semantics as in C.
3. Implement relational operators `==`, `!=`, `<`, `<=`, `>`, `>=`. The value of  $3>4$  is 0, while  $5.0>1.0$  is 1.

```
scalc> x=1
1
scalc> x++
1
scalc> x
2
scalc> ---x
syntax error
scalc> --(--x)
-1
..
```

4. Add a conditional expression of the form: `if expr then expr else expr end`. Use 0 for “false” and let any other value be considered true.

5. Add support for double precision floating point constants in standard decimal notation (e.g., 1.0, -2., 3.14). Also add the real valued constants  $\text{PI} = 3.141592$  and  $\text{E} = 2.718281$ . Extend all the operators to apply to floating point numbers in addition to integers. Remember that mixing integers and floating point is an error unless an explicit conversion is given (round and float).

```
scalc> min(2, 3)
2
scalc> min(max(1.0, 2), 3)
type error
scalc> sqrt(10)
3.162776600
scalc> round(sqrt(10))
3
scalc> float(3)
3.0
```

6. Add built-in overloaded functions `abs(x)`, `min(x,y)` and `max(x,y)` to the calculator that work on integer or floating point values.

- Implement conversion function `round(x)` and `float(n)` that convert between real numbers and integers. Define `sqrt(x)` to accept integers or floating point but always return floating point. Note that the built-in functions are called using the same syntax as user-defined functions.
7. Implement an overloaded side-effecting procedure `swap(x,y)` that swaps the contents of its two integer or two floating-point variable arguments.
  8. Implement user-defined functions with parameters passed by value. These functions are defined as follows: **define** *name*(*var*, ..., *var*) = *expr* where *name* is an identifier, *var*, ..., *var* is a list of variable names (the list may contain 0, 1, or 2 variables) and *expr* is an expression. To call a function, use the form *name*(*expr*, ..., *expr*). The variables are passed by value and are considered local variables whose usage does not affect global variable with the same name. Use static scoping for variable names not mentioned in the variable list of a function, so that all non-local variables refer to the root global variable. Since *scal* is line-oriented, the complete function definition must fit on one line.

```

scalc> define square(n) = n * n
scalc> square(3)
9
scalc> x=1
1
scalc> define dec(n) = n-x
scalc> define fact(n)= if n>0 then n*fact(dec(n)) else 1 end
scalc> fact(3)
6
scalc> square(fact(3))
36
scalc> define update(x) = (++x)
scalc> update(x)
2
scalc> x
1

```

The update example illustrates that `x` is a local variable. Incrementing `x` within the function does not affect the value of the global variable `x`.

To implement functions correctly you will have to create an activation record each time a function is called to store the values of local variables. Note that since the assignment does **not** require functions to be passed as arguments or returned as values, you do not need to implement closures for this assignment.

9. NOTE: One of the key issues you should pay careful attention to is how you manage the binding of names to values. This functionality is needed for both global variables and local variables. You should design a data structure, called an *environment*, for storing the bindings between names and their values. Your implementation can then include functions for storing a variable/value into an environment and also for looking up the value of a variable.
10. NOTE: You are not required to perform garbage collection or other storage management. You may simply allocate storage as needed. It will be reclaimed when the program exits.

### Helpful Flex/Bison Programming Information

In flex and bison, the array `yytext` contains the currently matched token and the variable `yyval` is used to pass the value of `yytext` defined in flex to the parser generated by bison. By default, `yyval` is declared to be a C int, but you can change this by defining `YYSTYPE` in the `.y` file header section before the grammar rules section denoted by the `%%` directive.

For example, if you want `yyval` to be able to contain values of type `AST` that is defined in a header `"ast.h"`, you define the following in the `.y` file for your grammar:

```
%{
#include "ast.h"

#define YYSTYPE AST
%}
```

Likewise, in the lex specification file must be modified to return values of type `AST`. You must be sure to convert a string token to the correct type of value before `yyval` is set. For example, the following two lexical rules match integer and floating point constants, converting each to an appropriately typed binary value using standard C library functions `ascii-to-int` (`atoi`) and `string-to-double` (`strtod`):

```
[0-9]+      { yyval = MakeInt(atoi(yytext)); return INT; }
[0-9]+ "." [0-9]* { yyval = MakeReal(strtod(yytext, 0)); return REAL; }
```

ALSO NOTE: When printing floating-point values in `scal`, it is recommended that you use the `printf` `"%e"` format selector. For example: `printf("%e", PI)` would print `3.141592e+00`.

### Submission Requirements:

You are to hand in at the [start of class on the due date](#) a source code listing of your program with your name and comments included. You are also to submit an electronic copy of your source program to the course TA using the **turnin** program on CS Department machines BEFORE class begins on the due date. Late assignments will not be accepted and a grade of zero will be assigned.

### Honor Policy:

This assignment is a personal learning opportunity that will be evaluated based on your ability to think independently, work through a problem in a logical manner and implement a software program on your own (or in a team with one other student). You may however discuss verbally or via email the general nature of the conceptual problem to be solved with your classmates, the course TA or the course instructor, but you are to complete the actual programming for this assignment without resorting to help from any other person or other resources that are not authorized as part of this course. If in doubt, ask the course instructor. *Any indication of improper cooperation or the use of non-approved course materials will result in a grade of zero being assigned for this project and referral to the College for possible disciplinary action with respect to your continued participation in this course.* **By submitting a solution to this assignment under your name, you are asserting that the solution was done by you under this honor policy.**