VC:
A Virtual Classroom Language

Team:
Carolyn Fine
Marissa Golden
Michelle Levine

Motivation

- Programming language designed for teachers to create arithmetic tests.
- Simple creation of a gradable test with a variety of multiple choice, true/false, and fill in the blank questions.
- Generate options for multiple choice answers.

Language Functionality

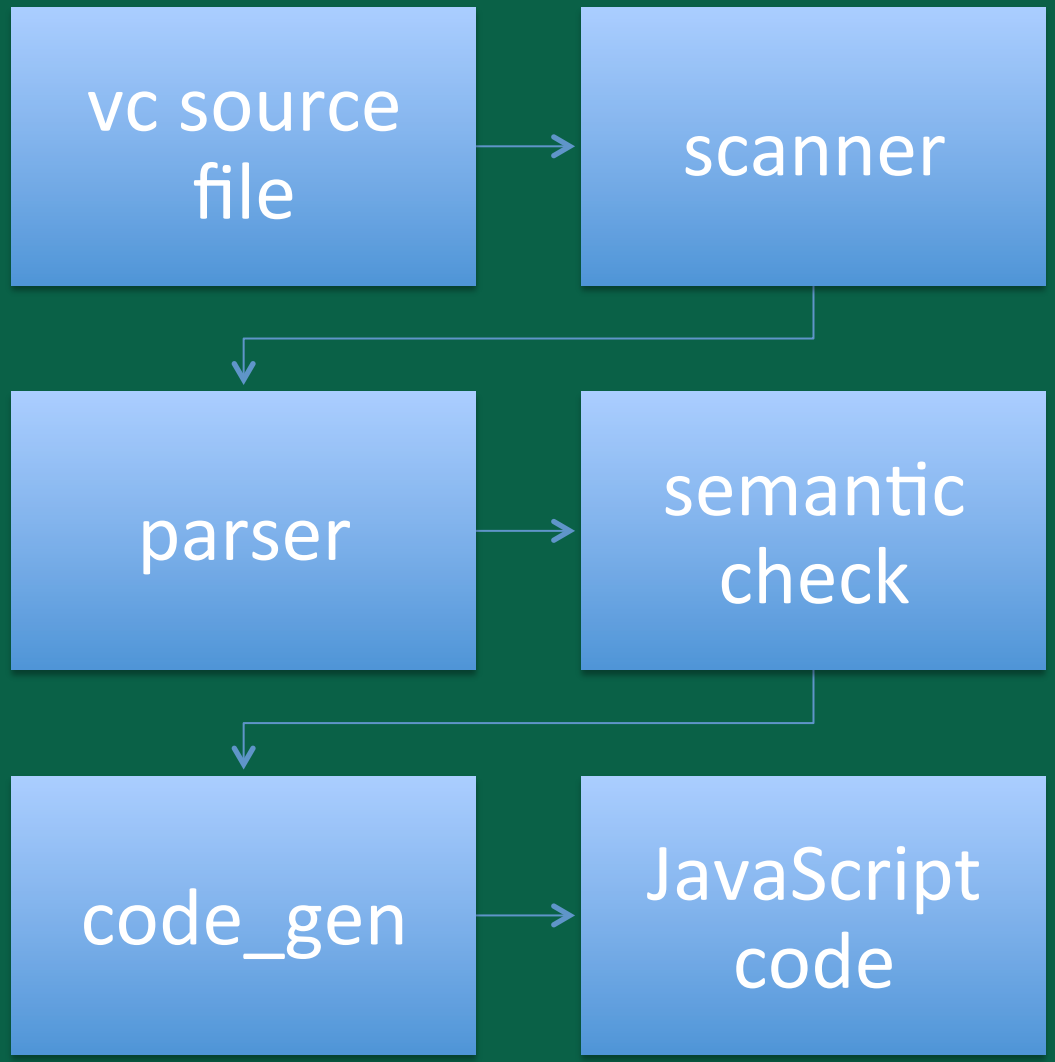
- Language allows the teacher to write functions for generating questions, create multiple choice options, and shuffle answers.
- Built-in functions to print, convert to string, generate random integers, string manipulations, etc. for the teacher to use in functions.
- Final test is displayed in an HTML/JavaScript format

Built-In Functions

- print
  - print to console
- rand
  - generate random integers
  - allows teacher to generate questions with similar formats and varying values
- str
  - convert type of variable or expression to a string
  - used to create string versions of questions
- evalDouble
  - evaluate a string expression into a double
  - used to evaluate string questions to determine the correct answer
- evalInt
  - evaluate a string expression into an integer
  - used to evaluate string questions to determine the correct answer

Built-In Functions continued…

- `get_char_at(<string>, <index>)`
  - get the character at the specified index
  - use cases: teacher can manipulate questions
  - e.g. `var op = get_char_at(question1, 2)`
- `length(<string | list>)`
  - use cases: for loops, creating functions, manipulate answers array
- `strReplace(<string>, <string_to_replace>, <string_replace_with>)`
  - finds a <string_to_replace> within the input <string> and replaces it with <string_replace_with>
  - allows teacher to manipulate string questions in order to generate incorrect multiple choice answers
  - e.g. `strReplace( question1, "(" , "" )` and `strReplace (question2, ")", "" )`


vc source file
scanner
parser
semantic check
code_gen
JavaScript code

vc source file

```
function evalWithoutParens returns double
(string q){
     string newq=strReplace(q, "(", "");
     newq=strReplace(newq, ")","");
     double ans = evalDouble(newq);
     return ans;
}
function produceWrongAns returns
double(string q){
     double b=evalDouble(q);
     b=b+rand(5);
     return b;
}
string list operators = ["+", "-", "*",
"/"];
function createQ returns string (){
     int len=length(operators);
     int randInd= rand(len)-1;
     string q;
     string b= operators[randInd];
     randInd=rand(len) -1;
     string c = operators[randInd];
     q=
str(rand(100))~b~"("~str(rand(100))~c~str(ra
nd(100))~")";
     return q;
}
```

```
function run returns none (){

/*string q1=createQ();
int len=length(operators);
int randInd= rand(len)-1;
string b= operators[randInd];
q1=q1~b~createQ();
int a1 = evalInt(q1);
int wa = evalWithoutParens(q1);
int wa2 = produceWrongAns(q1);
display_radio(q1,[wa,a1, wa2,a1],"q1");*/
int a;
loop conditions (start: a=0; check: a < 10;
change: a=a+1) do {
     string name = "q"~str(a);
     string q1=createQ();
     int len=length(operators);
     int randInd= rand(len)-1;
     string b= operators[randInd];
     q1=q1~b~createQ();
     double a1 = evalDouble(q1);
     double wa = evalWithoutParens(q1);
     double wa2 = produceWrongAns(q1);
     display_radio(q1,[wa,a1, wa2,a1],name);
}

}
```

The Environment

```
type environment = {
    functions: func_decl list;
    scope: string;
    locals: var_scope;
    globals: var_scope;
    has_return: bool;
    return_val: expr;
    return_type: var_type;
}

and var_scope = {
    prims: (string * var_type * expr) list;
    lists: (string * var_type * expr list) list
(*Form (type, list_id, list contents) *)
}
```

AST -> SAST
via Semantic Check

```
and get_sexpr env ex = match ex with
    Literal(l) -> (match l with
        Int_Literal(i) -> SLiteral(SInt_Literal(i), Int)
      | Double_Literal(d)-> SLiteral(SDouble_Literal(d), Double)
      | String_Literal(s) -> SLiteral(SString_Literal(s), String)
      | Bool_Literal(b) -> SLiteral(SBool_Literal(b), Bool)
      | _ -> raise(Error("Can't get SLiteral of this")))
      (*| Any -> SLiteral(SAny, String))*)
  | Id(v) -> SId(v, check_expr env ex)
  | Unop(u, e) -> SUnop(u, get_sexpr env e, check_expr env ex)
  | Binop(e1, op, e2) -> SBinop(get_sexpr env e1, op, get_sexpr env
  | Call(str, el) -> SCall(str, List.map (fun e -> get_sexpr env e)
```

```
and check_stmt env stmt =
    prerr_string("Calling check_stmt\n"); match stmt with
    | Block(stmt_list) ->
        prerr_string("Calling Block from check_stmt\n");
        let new_env = env in
        let (checked_stmts, up_env) = List.fold_left (fun (l, e) s
                                                        (check
```

Semantic Error Check

Var_Decl_Assign: Checking operators and type is string
Starting to check function: evalWithoutParens.
Calling check_stmt
Calling Var_decl from check_stmt
Var_Decl_Assign: Checking newq type is string
strReplace function is being called
check_expr: q id called
strReplace function is being called
check_expr: q id called
check_expr: q id called
Calling check_stmt
Calling expression from check_stmtAssign being called from check_expr
strReplace function is being called
check_expr: newq id called
tl = string tr = string.
strReplace function is being called
check_expr: newq id called
check_expr: newq id called
Calling check_stmt
Calling Var_decl from check_stmt
Var_Decl_Assign: Checking ans type is double
Evaluate function is being called
check_expr: newq id called
Evaluate function is being called
check_expr: newq id called
check_expr: newq id called
Calling check_stmt
Return from check_stmtcheck_expr: ans id called
check_expr: ans id called
Starting to check function: produceWrongAns.
Calling check_stmt
Calling Var_decl from check_stmt
Var_Decl_Assign: Checking b type is double
Evaluate function is being called
check_expr: q id called
Evaluate function is being called
check_expr: q id called
check_expr: q id called
Calling check_stmt
Calling expression from check_stmtAssign being called from check_expr
check_expr: b id called
Rand function is being called
tl = double tr = double.
check_expr: b id called
Rand function is being called
Rand function is being called
check_expr: b id called
Calling check_stmt
Return from check_stmtcheck_expr: b id called
check_expr: b id called
Starting to check function: createQ.
Calling check_stmt
Calling Var_decl from check_stmt
Var_Decl_Assign: Checking len type is int
length function is being called
check_expr: operators id called
length function is being called
check_expr: operators id called
check_expr: operators id called
Calling check_stmt

Code Generation

```
let gen_literal lit = match lit with
  | SInt_Literal(i)      -> string_of_int i
  | SDouble_Literal(d)   -> string_of_float d
  | SBool_Literal(b)     -> string_of_bool b
  | SString_Literal(str) -> "\"" ^ str ^ "\""

let rec gen_literal_list ll = match ll with
  | [] -> ""
  | head::[] -> gen_literal head
  | head::tail -> gen_literal head ^ ", " ^ gen_literal_list tail

let rec gen_expr expr = match expr with
  | SLiteral(l,t)           -> gen_literal l
  | SId(v,t)                -> v
  | SUnop(u, e, t)          -> gen_unop u ^ "(" ^ gen_expr e ^ ")"
  | SBinop(e1, op, e2, t)   -> (match op with
              | Concat -> gen_expr e1 ^ ".concat(" ^ gen_expr e2 ^ ")"
              | Exp    -> "Math.pow(" ^ gen_expr e1 ^ ", " ^ gen_expr e2 ^ ")"
              | _ ->gen_expr e1 ^ gen_binop op ^ gen_expr e2)
  | SCall(id, e1, t)        -> if(id="print") then gen_print e1
```

JavaScript Source Code

```
$( document ).ready(function() {
    var operators = ["+","-","*","/"];

var a;

        for (a = 0; a<10; a = a+1){
        var name = "q".concat(a.toString());

    var q1 = createQ();

    var len = operators.length;

    var randInd = Math.floor((Math.random() * len + 1))-1;

    var b = operators[randInd];

    q1 = q1.concat(b).concat(createQ());
    var a1 = eval(q1);

    var wa = evalWithoutParens(q1);

    var wa2 = produceWrongAns(q1);

    display_radio(q1, [wa,a1,wa2,a1], name);
    }
    function createQ() {
    var len = operators.length;

    var randInd = Math.floor((Math.random() * len + 1))-1;

    var q;

    var b = operators[randInd];

    randInd = Math.floor((Math.random() * len + 1))-1;
    var c = operators[randInd];

    q = Math.floor((Math.random() * 100 + 1)).toString().conc
        return q
    }
    function produceWrongAns(q) {
    var b = eval(q);
```

HTML output

1. 52/(93+46)+39-(6+61)
   ○ 18.55913978494624 ○ -27.62589928057554 ○ -26.62589928057554

2. 57+(12+96)*47/(14*37)
   ○ 77.71042471042472 ○ 66.7992277992278 ○ 70.7992277992278

3. 8+(5*38)+78*(90*50)
   ○ 351198 ○ 351198 ○ 351202

4. 96*(29+77)+84-(3+67)
   ○ 2875 ○ 10190 ○ 10193

5. 86+(66/46)/49*(91-69)
   ○ 86.64418811002662 ○ 86.64418811002662 ○ 87.64418811002662

6. 57+(91+90)-10-(67+56)
   ○ 105 ○ 105 ○ 108

7. 11/(58*94)+85+(81+48)
   ○ 231.82758620689657 ○ 214.00201760821716 ○ 216.00201760821716

8. 54/(70/16)/19/(33+55)
   ○ 0.000028836295283663707 ○ 0.007382091592617909 ○ 5.007382091592618

9. 27*(13-58)+56-(39/20)
   ○ 347.05 ○ -1160.95 ○ -1159.95

10. 5/(61/23)-38+(17+35)
    ○ 14.003563791874555 ○ 15.885245901639344 ○ 17.885245901639344

Submit

To compile and run

```
open Unix
open String
open Filename
type action = Ast | Sast | JavaScript | Help

let usage (name:string) =
  "usage:\n" ^ name ^ "\n" ^
    "        -a source.vc                (Print AST of an vc source)\n"^
    "        -s source.vc                (Run Semantic Analysis over source)\n"^
    "        -j source.vc [target.js]   (Generate JavaScript code for vc)\n"^
    "        -c source.vc [target.out]    (Compile vc to executable)\n" ^
    "        -h                           (Shows this menu)"

let backend_path = "../backend/src/"
let target_path = backend_path ^ "com/vc/"
```

do: make

then: ./vc -[option] source.vc

options:

- a to print AST
- s to run semantic analysis on source code
- j to compile and output JavaScript and HTML source code
- h for help

Lessons Learned