MAB471 - Compiladores I Compilador de mini javascript

- Description
- Submission
- Edit
- Submission view

Grade

Reviewed on Friday, 15 November 2019, 4:36 PM by Automatic grade
Grade 0 / 100
Assessment report
[+]**Failed tests**

[-]**Test 1: #1 Função sem parâmetros**

Incorrect program result
--- Input ---

```
let str = "hello";
function valor() { return str + ", world"; }
let a = valor();
```

--- Program output ---

```
Erro: a variável 'function' não foi declarada.
```

--- Expected output (exact text)---

```
=== Console ===
=== Vars ===
|{ a: hello, world; str: hello; undefined: undefined; valor: { &funcao: ##; }; }|
=== Pilha ===
```

[-]**Test 2: #2 Função com parâmetros**

Incorrect program result
--- Input ---

```
function sqr(x) { return x*x; }
let a = sqr( 5.2 );
```

--- Program output ---

```
Erro: a variável 'function' não foi declarada.
```

--- Expected output (exact text)---

```
=== Console ===
=== Vars ===
|{ a: 27.04; sqr: { &funcao: ##; }; undefined: undefined; }|
=== Pilha ===
```

[-]**Test 3: #3 mdc**

Incorrect program result
--- Input ---

```
function mdc( a, b ) {
  if( b == 0 )
    return a;
  else
    return mdc( b, a % b );
}

let a = mdc( 24, 33  );
```

--- Program output ---

```
Erro: a variável 'function' não foi declarada.
```

--- Expected output (exact text)---

```
=== Console ===
=== Vars ===
|{ a: 3; mdc: { &funcao: ##; }; undefined: undefined; }|
=== Pilha ===
```

[-]**Test 4: #4 função sem valor de retorno**

Incorrect program result
--- Input ---

```
function teste( a, b ) {
  if( a > b )
    return a;
}

let um = teste( 3, 4 ),
    dois = teste( 4, 3 );
```

--- Program output ---

```
Erro: a variável 'function' não foi declarada.
```

--- Expected output (exact text)---

```
=== Console ===
=== Vars ===
|{ dois: 4; teste: { &funcao: ##; }; um: undefined; undefined: undefined; }|
=== Pilha ===
```

[-]**Test 5: #5 asm**

Incorrect program result
--- Input ---

```
function log( msg ) {
  msg asm{println # undefined};
}

let r = log( 'Hello, world!' );
```

--- Program output ---

```
Erro: a variável 'function' não foi declarada.
```

--- Expected output (exact text)---

```
=== Console ===
Hello, world!
=== Vars ===
|{ log: { &funcao: ##; }; r: undefined; undefined: undefined; }|
=== Pilha ===
```

[-]**Test 6: #6 métodos**

Incorrect program result
--- Input ---

```
let console = {};
let Number = {};
```

```
function log( msg ) {
  msg asm{println # undefined};
}

function number_to_string( msg ) {
  msg asm{to_string # '&retorno' @ ~};
}

console.log = log;
Number.toString = number_to_string;

let a = "Saida: ";
let b = 3.14;

console.log( a + Number.toString( b ) );
```

--- Program output ---

```
Erro: a variável 'function' não foi declarada.
```

--- Expected output (text)---

```
=== Console ===
Saida: 3.14
=== Vars ===
|{ Number: { toString: { &funcao: ##; }; }; a: Saida: ; b: 3.14; console: { log: {
=== Pilha ===
"
```

[-]**Test 7: #7 argumentos**

Incorrect program result
--- Input ---

```
let console = {};

function exit( n ) {
  'Codigo de erro: ' asm{print # undefined};
  n asm{println # undefined};
  0 asm{.};
}

function teste( a, b, c ) {
  exit( b );
}

let a = "Saida: ";
let b = 3.14;

console.teste = {};
console.teste.log = [];
console.teste.log[1] = teste;

console.teste.log[1]( a, b, "5" );
```

--- Program output ---

```
Erro: a variável 'function' não foi declarada.
```

--- Expected output (exact text)---

```
=== Console ===
Codigo de erro: 3.14
=== Vars ===
|{ a: Saida: ; b: 3.14; console: { teste: { log: [ 0: undefined; 1: { &funcao: ##;
|{ &retorno: ##; a: Saida: ; arguments: [ 0: Saida: ; 1: 3.14; 2: 5; ]; b: 3.14; c
|{ &retorno: ##; arguments: [ 0: 3.14; ]; n: 3.14; }|
=== Pilha ===
|0|
```

[-]**Test 8: #8 Função com variáveis locais**

Incorrect program result
--- Input ---

```
function f( x ) {
  let b = 5 * x;
  let c = {};

  c.num = b;
  c.arr = [];
  c.arr[1] = 0;
  return c ;
}

let res = f( 11 );
```

--- Program output ---

```
Erro: a variável 'function' não foi declarada.
```

--- Expected output (exact text)---

```
=== Console ===
=== Vars ===
|{ f: { &funcao: ##; }; res: { arr: [ 0: undefined; 1: 0; ]; num: 55; }; undefined
=== Pilha ===
```

[-]**Test 9: #9 Função que se anula**

Incorrect program result
--- Input ---

```
function f( x ) {
  let b = f;
  f = x;
  return b;
}

let a;
```

```
let g = f( a = [], {} );
```

--- Program output ---

```
Erro: a variável 'function' não foi declarada.
```

--- Expected output (exact text)---

```
=== Console ===
=== Vars ===
|{ a: [ ]; f: [ ]; g: { &funcao: ##; }; undefined: undefined; }|
=== Pilha ===
```

[-]**Test 10: #10 Super teste**

Incorrect program result
--- Input ---

```
function getNome( obj ) { return obj.nome; }

function getClass( obj ) { return obj.class; }

function criaAluno( nome, celular, email ) {
  let aluno = {};

  aluno.nome = nome;
  aluno.celular = celular;
  aluno.email = email;
  aluno.super = prototipoAluno;
  aluno.getNome = getNome;

  return aluno;
}

function log( msg ) {
  msg asm{println # undefined};
}

function invoke( obj, metodo ) {
  if( toString( obj[metodo] ) == 'undefined' )
    return obj.super[metodo]( obj.super );
  else
    return obj[metodo]( obj );
}

function toString( msg ) {
  msg asm{to_string # '&retorno' @ ~};
}

let prototipoAluno = {};

prototipoAluno.class = 'Classe Aluno';
prototipoAluno.getClass = getClass;

let joao = criaAluno( 'Joao', '123456', 'eu@aqui.com' );
let maria = criaAluno( 'Maria', '123457', 'voce@la.com' );
```

```
log( invoke( joao, 'getNome' ) );
log( invoke( joao, 'getClass' ) );
log( invoke( maria, 'getNome' ) );
log( invoke( maria, 'getClass' ) );
```

--- Program output ---

```
Erro: a variável 'function' não foi declarada.
```

--- Expected output (exact text)---

```
=== Console ===
Joao
Classe Aluno
Maria
Classe Aluno
=== Vars ===
|{ criaAluno: { &funcao: ##; }; getClass: { &funcao: ##; }; getNome: { &funcao: ##
=== Pilha ===
```

[+]**Summary of tests**

Submitted on Friday, 15 November 2019, 4:36 PM (Download)

mini_js.l

```
1 DIGITO  [0-9]
2 LETRA   [A-Za-z_]
3 DOUBLE  {DIGITO}+("."{DIGITO}+)?
4 NEGDOUBLE ("-"{DOUBLE})
5 ID      {LETRA}({LETRA}|{DIGITO})*
6 STR       (\"([^\"\n]|(\\\")|\"\")+\")|('([^'\n]|(\\')|'')+')
7
8 %%
9
10 "\t"       { coluna += 4; }
11 " "        { coluna++; }
12 "\n"         { linha++; coluna = 1; }
13
14 {DOUBLE}   { return retorna( NUM ); }
15 {NEGDOUBLE}   { return retorna( NEGNUM ); }
16
17 {STR}        { return retorna( STR ); }
18
19 "let"        { return retorna( LET );}
20 "if"       { return retorna( IF );}
21 "else"     { return retorna( ELSE );}
22 "{}"         { return retorna( NEWOBJ );}
23 "[]"         { return retorna( NEWARRAY );}
24 "=="       { return retorna( IGUAL );}
25
26
27 {ID}       { return retorna( ID ); }
28
```

```
29 .          { return retorna( *yytext ); }
30
31 %%
```

mini_js.y

```
 1 %{
 2 #include <string>
 3 #include <iostream>
 4 #include <map>
 5 #include <vector>
 6
 7
 8 using namespace std;
 9
10 struct Atributos {
11   vector<string> v;
12   int id;
13 };
14
15 #define YYSTYPE Atributos
16
17 #define st first
18 #define nd second
19
20 void erro( string msg );
21 void New_Inst( string st ); //nova instrução
22 void New_Inst( vector<string> &st );
23 void Pula_linha();
24 void imprime_codigo();
25 void get_var(vector<string> v);
26 void let_var(vector<string> v);
27 string gera_ini_label(string pref,int id);
28 string gera_fim_label(string pref,int id);
29 int get_id();
30 // protótipo para o analisador léxico (gerado pelo lex)
31 int yylex();
32 void yyerror( const char* );
33 int retorna( int tk );
34
35 int linha = 1;
36 int coluna = 1;
37
38
39 vector<string> codigo;
40 map<string,int> var_globais;
41
42 %}
43
44 %token NUM NEGNUM STR ID PRINT LET NEWOBJ NEWARRAY IF ELSE IGUAL
45
46 %left '.'
47 %left '+' '-'
48 %left '*' '/'
49
50 %right '='
51
52
```

```
%%


START : CMD
      ;

CMD : LET Decl CMD
  | P CMD
  | IF_LINHA CMD
  |
  ;


CMD_LINHA : A';'
          | LET Decl
          | IF_LINHA CMD
          ;
BLOCO: CMD_LINHA
   | '{' CMD '}'
   |
   ;

IF_LINHA : IF '('COND')' {$1.id=get_id();}
           {New_Inst(gera_ini_label("INI_IF",$1.id));New_Inst("?");
           Pula_linha();
           New_Inst(gera_ini_label("ELSE_IF",$1.id));New_Inst("#");
           Pula_linha();
           New_Inst(gera_fim_label("INI_IF",$1.id));}
           BLOCO
           {New_Inst(gera_ini_label("FIM_IF",$1.id));New_Inst("#");}
           ELSE_LINHA
           {New_Inst(gera_fim_label("ELSE_IF",$1.id));}
           BLOCO{New_Inst(gera_fim_label("FIM_IF",$1.id));}
         ;

ELSE_LINHA:
            |ELSE
            ;

Decl : ID { New_Inst( $1.v ); let_var($1.v); New_Inst("&"); Pula_linha();} '
   | ID { New_Inst( $1.v ); let_var($1.v); New_Inst("&"); Pula_linha();} ';'
   | Adecl ',' Decl
   | Adecl ';'
   ;

Adecl : ID { New_Inst( $1.v ); let_var($1.v); New_Inst("&"); New_Inst( $1.v
    ;

P : A ';' P
  | A ';'
  ;

A : ID { New_Inst( $1.v ); get_var($1.v);} '=' RVALUE { New_Inst( "=" ); New
  | LVALUEPROP '=' RVALUE { New_Inst( "[=]" ); New_Inst( "^" ); Pula_linha()
  | PRINT RVALUE { New_Inst( "print" ); New_Inst( "#" ); Pula_linha(); }
  ;

LVALUEPROP : E '.' LVALUEPROPSUFFIX
             | E['E']''.'{New_Inst("[@]");}LVALUEPROPSUFFIX
```

```
112             | E['E']'
113             ;
114 LVALUEPROPSUFFIX: ID { New_Inst( $1.v );}
115                         | ID { New_Inst( $1.v );}{New_Inst("[@]");}'['ELPR
116                         | ID { New_Inst( $1.v );}'.'{New_Inst("[@]");}LVAL
117                         | ID { New_Inst( $1.v );}'.'{New_Inst("[@]");}LVAL
118                         ;
119
120
121 RVALUE : E
122      | NEWOBJ {New_Inst("{}");}
123      | NEWARRAY {New_Inst("[]");}
124      | A { New_Inst( $1.v ); get_var($1.v); New_Inst("@");}
125      ;
126 E : E '+' E { New_Inst( "+" ); }
127   | E '-' E { New_Inst( "-" ); }
128   | E '*' E { New_Inst( "*" ); }
129   | E '/' E { New_Inst( "/" ); }
130   | F
131   ;
132
133 COND : E '<' E { New_Inst( "<" ); }
134      | E '>' E { New_Inst( ">" ); }
135      | E IGUAL E { New_Inst( "==" ); }
136      ;
137
138 ELPROP: ELPROP '+' ELPROP { New_Inst( "+" ); }
139     | ELPROP '-' ELPROP { New_Inst( "-" ); }
140       | ELPROP '*' ELPROP { New_Inst( "*" ); }
141     | ELPROP '/' ELPROP { New_Inst( "/" ); }
142     | FLPROP
143     ;
144
145 FLPROP: ID { New_Inst( $1.v ); }
146           | NUM { New_Inst(       $1.v ); }
147           | NEGNUM {int sz=$1.v.size()-1; $1.v[sz]=$1.v[sz].substr(1);New_In
148           | STR { New_Inst(       $1.v ); }
149           | '(' RVALUE ')'
150           | ID '(' PARAM ')' { New_Inst( $1.v ); New_Inst("#") ;}
151           ;
152
153
154 F : ID { New_Inst( $1.v ); get_var($1.v); New_Inst("@"); }
155   | NUM { New_Inst(       $1.v ); }
156   | NEGNUM {int sz=$1.v.size()-1; $1.v[sz]=$1.v[sz].substr(1);New_Inst("0");
157   | STR { New_Inst(       $1.v ); }
158   | '(' RVALUE ')'
159   | ID '(' PARAM ')' { New_Inst( $1.v ); New_Inst("#"); }
160   | LVALUEPROP {New_Inst("[@]");}
161   ;
162
163 PARAM : ARGs
164                   |
165                   ;
166
167 ARGs : RVALUE ',' ARGs
168    | RVALUE
169    ;
170
```

```
171 %%
172
173 #include "lex.yy.c"
174
175 map<int,string> nome_tokens = {
176   { PRINT, "print" },
177   { STR, "string" },
178   { ID, "nome de identificador" },
179   { NUM, "número" }
180 };
181
182 string nome_token( int token ) {
183   if( nome_tokens.find( token ) != nome_tokens.end() )
184           return nome_tokens[token];
185   else {
186           string r;
187
188           r = token;
189           return r;
190   }
191 }
192
193 int retorna( int tk ) {
194   yylval.v.push_back(yytext);
195   coluna += strlen( yytext );
196
197   return tk;
198 }
199
200 void yyerror( const char* msg ) {
201   cerr << msg << endl;
202   exit( 1 );
203 }
204
205
206
207 void Pula_linha(){
208   codigo.push_back("\n");
209 }
210
211 void imprime_codigo(vector<string> &codigo){
212   int cont=0;
213   for(int i=0;i<codigo.size();i++){
214           //if(codigo[i]!="\n")
215                   //cerr<<cont++<<":";
216           cout<<codigo[i];
217           if(codigo[i]!="\n")
218                   cout<<' ';
219   }
220 }
221
222 void get_var(vector<string> v){
223   string w = v[v.size()-1];
224   if(var_globais.count(w)==0){
225           string msg = "Erro: a variável '"+w+"' não foi declarada.";
226           yyerror(msg.c_str());
227   }
228
229 }
```

```
230 void let_var(vector<string> v){
231   string w = v[v.size()-1];
232   if(var_globais.count(w)!=0){
233           string msg = "Erro: a variável '"+w+"' já foi declarada na linha "
234           yyerror(msg.c_str());
235   }
236   else{
237           var_globais[w]=linha;
238   }
239 }
240
241 vector<string> operator+( vector<string> a, vector<string> b ) {
242   a.insert( a.end(), b.begin(), b.end() );
243   return a;
244 }
245
246 vector<string> operator+( vector<string> a, string b ) {
247   a.push_back(b);
248   return a;
249 }
250 vector<string> operator+( string a, vector<string> b ) {
251   return b+a;
252 }
253
254 void New_Inst( string st ) {
255   codigo.push_back(st);
256 }
257
258
259 void New_Inst( vector<string> &st ){
260   New_Inst(st[st.size()-1]);
261 }
262
263 int get_id(){
264   static int id;
265   return id++;
266 }
267 string gera_ini_label(string pref,int id){
268   return pref+to_string(id)+":";
269 }
270 string gera_fim_label(string pref, int id){
271   return ":"+pref+to_string(id)+":";
272
273 }
274
275 vector<string> resolve_enderecos( vector<string> &entrada ) {
276   map<string,int> label;
277   vector<string> saida;
278   int cont=0;
279
280   for( int i = 0; i < entrada.size(); i++ )
281     if( entrada[i][0] == ':' )
282         label[entrada[i].substr(1)] = cont;
283     else{
284       if(entrada[i]!="\n")cont++;
285       saida.push_back( entrada[i] );
286           }
287   for( int i = 0; i < saida.size(); i++ )
288     if( label.count( saida[i] ) > 0 )
```

```
289         saida[i] = to_string(label[saida[i]]);
290
291   return saida;
292 }
293
294 int main() {
295   yyparse();
296   //imprime_codigo(codigo); cout<<"\n\n";
297   codigo = resolve_enderecos(codigo);
298   imprime_codigo(codigo);
299   cout<<".\n";
300   return 0;
301 }
```