

Nov 09, 01 6:54

## Makefile

Page 1/1

```

1 # SUPER majikal makefile
2
3 OCAMLMAKEFILE = OcamlMakefile
4
5 SOURCES = bnode.ml parser.mly lexer.mll main.ml
6
7 RESULT = basic2c
8
9 all: native-code
10
11 test: all
12     cat testing.bas | ./basic2c > out.c && gcc out.c
13
14
15 -include $(OCAMLMAKEFILE)
16
17

```

Nov 09, 01 6:55

## lexer.mll

Page 1/1

```

1 (*
2  * lexer.mll - Does the lexical analysis
3  *
4  * Copyright (c) 2001 Brock Wilcox
5  * Released under the terms of the GNU GPL
6  * See copying.txt for details
7  *)
8
9
10 {
11   open Parser
12   let debug = true
13 }
14
15 rule token = parse
16   | { (' ' | '\t' | '+') | (('REM' | '\n') | '\t' | [^ '\n.']* )
17   | { token lexbuf }
18
19
20 (* NewLines and end-of-files are needed *)
21 | '\n' { EOL }
22 | eof { EOF }
23
24
25 (* Variable assignment *)
26 | "let" { LET }
27 | '=' { EQUAL }
28
29
30 (* Looping constructs *)
31 | "do" { DO }
32 | "if" { IF }
33 | "then" { THEN }
34 | "else" { ELSE }
35 | "end" { END }
36 | "for" { FOR }
37 | "to" { TO }
38 | "while" { WHILE }
39 | "goto" { GOTO }
40 | ':' { COLON }
41
42
43 (* Commands *)
44 | ',' { COMMA }
45 | { ('a'-'z' | 'A'-'Z' | '_' | ['a'-'z' | 'A'-'Z' | '_' | '0'-'9']* ) | '?'
46 | IDENT(Lexing.lexeme lexbuf) }
47
48
49 (* Math *)
50 | '+' { BINARY_OP("++") }
51 | '-' { BINARY_OP("--") }
52 | '*' { BINARY_OP("*") }
53 | '/' { BINARY_OP("/") }
54 | '%' { BINARY_OP("%") }
55 | '>' { BINARY_OP(">") }
56 | '<' { BINARY_OP("<") }
57 | "and" { BINARY_OP("and") }
58 | "or" { BINARY_OP("or") }
59 | "not" { BINARY_OP("not") }
60 | "<>" { BINARY_OP("<>") }
61 | '(' { LEFT_PAREN }
62 | ')' { RIGHT_PAREN }
63
64
65 (* Constants *)
66 | ['0'-'9']+ ('.' | ['0'-'9']+) ?
67 | NUMBER(float_of_string(Lexing.lexeme lexbuf)) }
68 | ',' [^',']* ','
69
70
71 let s = Lexing.lexeme lexbuf in
72 let substr = String.sub s 1 ((String.length s) - 2) in
73 STRING(substr)
74
75

```

Nov 09, 01 6:56

parser.mly

Page 1/3

```

1  /*
2  * This is the yacc grammar for basic2c
3  *
4  * Copyright (c) 2001 Brock Wilcox
5  * Released under the terms of the GNU GPL
6  * See copying.txt for details
7  */
8
9
10 /* File structure */
11 %token EOL
12 %token EOF
13
14 /* Variable assignment */
15 %token LET EQUAL
16
17 /* Looping constructs */
18 %token DO
19 %token IF THEN ELSE END
20 %token FOR TO
21 %token WHILE
22 %token GOTO LABEL COLON
23
24 /* Commands (things like print) and identifiers */
25 %token COMMA /* to separate parameters */
26 %token <string> IDENT
27
28 /* Math */
29 %token <string> BINARY_OP
30 %token LEFT_PAREN RIGHT_PAREN
31
32 /* Constants */
33 %token <float> NUMBER
34 %token <string> STRING
35
36 /* Directive for where to start parsing */
37 %start block
38
39
40 %type <Bnode.block> block
41
42 %%
43
44 block:
45 | EOL block
46 | { $2 }
47 | statement EOL block
48 | statement EOL
49 | EOF
50 | { $1::[] }
51 | { [] };
52
53 statement:
54 | assignment
55 | { $1 }
56 | loop
57 | { $1 }
58 | command
59 | { $1 };
60
61 assignment:
62 | LET IDENT EQUAL expression
63 | {
64   | Bnode.add_var $2 Bnode.NumVar;
65   | Bnode.Assign(Bnode.Variable($2),$4)
66 }
67
68 | LET IDENT EQUAL STRING
69
70 | {
71   | Bnode.add_var $2 Bnode.StrVar;
72   | Bnode.Assign(Bnode.Variable($2),Bnode.Str($4))
73 };

```

Nov 09, 01 6:56

parser.mly

Page 2/3

```

74 expression:
75 | expression EQUAL expression
76 | { Bnode.Binary_Op("=", $1,$3) }
77 | expression BINARY_OP expression
78 | { Bnode.Binary_op($2,$1,$3) }
79 | LEFT_PAREN expression RIGHT_PAREN
80 | { $2 }
81 | NUMBER
82 | { Bnode.Number($1) }
83 | identifier
84 | { $1 };
85
86 loop:
87 | if_loop
88 | { $1 }
89 | for_loop
90 | { $1 }
91 | while_loop
92 | { $1 }
93 | goto_loop
94 | { $1 };
95
96 if_loop:
97 | IF expression THEN statement
98 | { Bnode.If_loop($2,$4::[],[]) }
99 | IF expression THEN EOL block END
100 | { Bnode.If_loop($2,$5,[]) }
101 | IF expression THEN EOL block ELSE EOL block END
102 | { Bnode.If_loop($2,$5,$8) };
103
104 for_loop:
105 | FOR IDENT EQUAL expression TO expression DO EOL block END
106 | {
107   | Bnode.add_var $2 Bnode.NumVar;
108   | Bnode.For_loop(Bnode.Variable($2),$4,$6,$9)
109 };
110
111 while_loop:
112 | WHILE expression DO EOL block END
113 | { Bnode.While_loop($2,$5) };
114
115 goto_loop:
116 | IDENT COLON
117 | { Bnode.Label($1) }
118 | GOTO identifier
119 | { Bnode.Command("goto", $2::[]) };
120
121 command:
122 | IDENT parameter_list
123 | { Bnode.Command($1,$2) };
124 | IDENT
125 | { Bnode.Command($1,[]) };
126
127 parameter_list:
128 | expression_or_string COMMA parameter_list
129 | { $1::$3 }
130 | expression_or_string
131 | { $1::[] };
132
133 expression_or_string:
134 | expression
135 | { $1 }
136 | STRING
137 | { Bnode.Str($1) };
138
139 /* constant:
140 | NUMBER
141 | { Bnode.Number($1) }
142 | STRING
143 | { Bnode.Str($1) };
144 */
145
146 identifier:

```

Nov 09, 01 6:56

parser.mly

Page 3/3

```

147 | IDENT
148 | { Bnode.Variable($1) };
149

```

Nov 09, 01 6:38

bnode.ml

Page 1/2

```

1 (* bnode.ml -- define structures used in basic2c
2 *
3 * Copyright (c) 2001 Brock Wilcox
4 * Released under the terms of the GNU GPL
5 * See copying.txt for details
6 *)
7
8 exception Error of string
9
10 type bnode =
11   | Assign of bnode * bnode (* var = exp *)
12   | For_loop of bnode * bnode * bnode (* block (* var, start, stop, body *)
13   | While_loop of bnode * block (* condition, body *)
14   | If_loop of bnode * block (* condition, iftrue, iffalse *)
15   | Binary_op of string * bnode * bnode (* op, left_param, right_param *)
16   | Command of string * bnode list (* name, parameters *)
17   | Label of string
18   | Variable of string
19   | Number of float
20   | Str of string
21
22 and block = bnode list
23
24 type vartype =
25   | StrVar
26   | NumVar
27
28 let var_env : (string * vartype) list ref = ref []
29
30 let add_var name t =
31 if ~ (List.exists (fun (n,_) -> n = name) !var_env) then
32   var_env := (name,t)::!var_env
33
34 let rec pp_string_block pad = function
35   [] -> ""
36   | h::t -> (pp_string h pad) ^ (pp_string_block pad t)
37
38 and pp_string_binop op left right pad =
39   (pp_string left pad) ^
40   begin match op with
41     "+" -> "+"
42     "-" -> "-"
43     "*" -> "*"
44     "/" -> "/"
45     "=" -> "=="
46     "%" -> "%"
47     ">" -> ">"
48     "<" -> "<"
49     "and" -> "&&"
50     "or" -> "||"
51     "not" -> "!"
52     "<>" -> "!="
53     _ -> raise (Error ("Unknown binary operator " ^ op))
54   end ^ (pp_string right pad)
55
56 and pp_string_print pad = function
57   [] -> ""
58   | h::t -> begin
59     match h with
60     | Str(s) -> pad ^ "printf(\"" ^ s ^ "\\");\n"
61     | Variable(name) ->
62       let t = List.assoc name !var_env in
63       pad ^ "printf(\"" ^ begin
64         match t with
65         | StrVar -> "%s"
66         | NumVar -> "%f"
67         end ^ "\\," ^ name ^ ");\n"
68     | _ -> raise (Error "Error in print")
69   end ^ (pp_string_print pad t)
70
71
72
73 and pp_string_command name params pad =

```

```

74 match name with
75 | "n" | "print" → (pp_string_print pad params) ^ pad ^ "print(\"\n\");\n";
76 (* pad ^ "printf(\"\n\");\n" *)
77 | "goto" → pad ^ "goto " ^ (pp_string (List.hd params) pad) ^ "\n";
78 | _ → raise (Error ("Unknown command " ^ name))
79
80 and pp_string_node pad =
81 let npad = pad ^ " " in
82 match node with
83 | Assign(var, exp) →
84   pad ^ (pp_string var pad) ^ " = " ^ (pp_string exp pad) ^ "\n";
85 | For_loop(var, start, stop, body) →
86   pad ^ "for( " ^ (pp_string var pad) ^ " = " ^ (pp_string start npad) ^ " ; "
87   ^ (pp_string var pad) ^ " <= " ^ (pp_string stop npad)
88   ^ " ; ++ " ^ (pp_string var pad) ^ " )\n";
89   ^ pad ^ "{\n"
90   ^ (pp_string_block npad body)
91   ^ pad ^ "}"
92 | While_loop(condition, body) →
93   pad ^ "while( " ^ (pp_string condition pad) ^ " )\n";
94   ^ pad ^ "{\n"
95   ^ (pp_string_block npad body)
96   ^ pad ^ "}"
97 | If_loop(condition, iftrue, iffalse) →
98   pad ^ "if( " ^ (pp_string condition pad) ^ " )\n" ^ pad ^ "{\n";
99   ^ (pp_string_block npad iftrue)
100  ^ pad ^ "}" else {\n";
101  ^ (pp_string_block npad iffalse)
102  ^ pad ^ "}"
103 | Binary_op(op, left, right) → pp_string_binop op left right pad
104 | Command(name, params) → pp_string_command name params pad
105 | Label(name) → pad ^ name ^ "\n";
106 | Variable(name) → name { * "VARIABLE" * }
107 | Number(n) → string_of_float n
108 | Str(s) → "\"" ^ s ^ "\""
109
110 let pp_string_declare str (name, t) =
111 match t with
112 | NumVar → str ^ " float " ^ name ^ "\n";
113 | StrVar → str ^ " char* " ^ name ^ "\n";
114
115 let pp_string_vars var_env =
116 List.fold_left pp_string_declare "" (lvar_env)
117
118 let pp_string_program prog vars =
119 let varstr = pp_string_vars vars in
120 "\n/* Converted with basic2c */\n\n";
121 ^ "#include <stdio.h>\n\n";
122 ^ "int main(int argc, char** argv)\n";
123 ^ "{\n";
124 ^ varstr
125 ^ (pp_string_block " " prog)
126 ^ " return 0;\n";
127 ^ "}"
128
129 (* End of basic2c.h *)
130

```

```

1 (* main.ml - Main entry point for basic2c
2 *
3 * This just takes an input from stdin and dumps to stdout
4 *
5 * Copyright (c) 2001 Brock Wilcox
6 * Released under the terms of the GNU GPL
7 * See copying.txt for details
8 * *)
9
10 let from_stdin () =
11 let lexbuf = Lexing.from_channel stdin in
12 let result = Parser.block_lexer.token_lexbuf in
13 print_string (Bnode.pp_string_program result Bnode.var_env);
14 print_newline();
15
16 let _ = from_stdin ()
17
18

```

Nov 09, 01 6:51

## testing.bas

Page 1/2

```

1  REM there are two types of comments.
2  ' That was the first, and this is the second.
3  ' I like the second better.
4
5  ' Print some newlines, note the allowance of C's backslashed chars
6  print "\n\n" ' prints 3 newlines in all
7
8  ' Test a string assignment, print result
9  let aaarg = "Hello!"
10 print aaarg
11
12 ' Now lets test a numerical assignment, print result
13 let b = 5
14 print b
15
16 ' Finally lets do an expression assignment, print result
17 let c = b * 3 + 4
18 print c
19
20 ' Note that there is the same precedence as C
21 let c = 4 + b * 3
22 print c
23
24 ' Lets test out multiple arguments for print
25 print "C = ", c, " and b = ", b
26
27 ' Good. Now lets do something a bit more... dare I say... wild?
28 for i = 1 to 10 do
29   print "i = ", i
30 ' Note these spiffy one-line if statements
31 if i = 6 then print "Why was six afraid of seven?"
32 if i = 9 then print "Because seven ate nine! HAHahaha"
33
34 end
35
36 ' While writing this example I couldn't help but remember that when
37 ' I used to really program in basic I would use '?' instead of print
38 ' So I had to add a special rule in the lexer and then make '?' a
39 ' synonym for the print command. As a side effect you can now have a
40 ' variable named '?
41 ? "I like to print using '?'s, don't you?"
42
43 let counter = 0
44
45 ' Now lets try some looping.
46 infinity:
47
48 let counter = counter + 1
49
50 if counter > 5 then
51   goto end_of_time
52 else
53   print "counter = ", counter
54 end
55
56 goto infinity
57
58 end_of_time:
59
60 ? "Now that we are at the end of time, counter = ", counter
61
62 ' Perhaps we're beginning to remember why Dijkstra considered GOTO harmful?
63
64 ' The Right Way(TM) to try that
65 let counter = 0
66 let keepgoing = 1
67 while keepgoing = 1
68   let counter = counter + 1
69   if counter > 5 then
70     let keepgoing = 0
71   else
72     ? "counter = ", counter
73

```

Friday November 09, 2001

testing.bas

5/6

Nov 09, 01 6:51

## testing.bas

Page 2/2

```

74 end
75 end
76 ? "Now counter is ", counter
77
78 ' I think I've tested just about everything. As you can see
79 ' even though this hack is such a hack it still works. It is quite
80 ' fragile though... I could go on and on about the things that will
81 ' break it in its current state. However, I will digress, since
82 ' I don't expect much more from less than 5 hours of coding.
83
84 ? "Thats all folks!"
85
86

```

1	<b>Table of Contents</b>		
2	1 <i>Makefile</i> ..... sheets	1 to 1 ( 1) pages	1- 1 18 lines
3	2 <i>lexer.ml</i> ..... sheets	1 to 1 ( 1) pages	2- 2 73 lines
4	3 <i>parser.mly</i> ..... sheets	2 to 3 ( 2) pages	3- 5 150 lines
5	4 <i>bnode.ml</i> ..... sheets	3 to 4 ( 2) pages	6- 7 131 lines
6	5 <i>main.ml</i> ..... sheets	4 to 4 ( 1) pages	8- 8 19 lines
7	6 <i>testing.bas</i> ..... sheets	5 to 5 ( 1) pages	9- 10 87 lines