Un esempio di calcolatrice multifunzione realizzata con `flex` e `bison`

Slide di Daniele Paolo Scarpazza

Tema

- Realizzare una calcolatrice:
  - a notazione infissa;
  - con supporto per la definizione e l'uso di variabili utente;
  - con funzioni matematiche integrate;
- Utilizzando **`flex`** e **`bison`**

Lessico della calcolatrice

```
%{
...
symrec *sym_table = (symrec *)0;
symrec *s;
%}
DIGIT                    [0-9]
LETTER                   [A-Za-z]
ALPHANUM                 [0-9A-Za-z]
%%
{DIGIT}+("."{DIGIT}+)?   { yylval.val=atof(yytext); return NUM; }
{LETTER}{ALPHANUM}*      { s = getsym(yytext);
                           if (s == 0)
                             s = putsym(yytext, VAR);
                           yylval.tptr = s;
                             return s->type;
                         }
"=="                     { return EQUAL;}
"="|"*"|"("|")"|"+"|"/"|"-"|"^"|"\n" {return *yytext;}
[ \t]+
```

Sintassi della calcolatrice

```
%{
#include <math.h>
#include "mfcalc.h"
%}

%union {
  double  val;
  symrec  *tptr;
}

%token <val>  NUM
%token <tptr> VAR FNCT
%type  <val>  exp

%right  '='
%left   EQUAL
%left   '-' '+'
%left   '*' '/'
%left   NEG          /* negation-unary minus */
%right  '^'          /* exponentiation       */
```

Sintassi della calcolatrice

```
input:                          /* empty string */
    | input line
    ;

line:    '\n'
    | exp '\n'                  { printf ("\t%.10g\n", $1);          }
    | error '\n'                { yyerrok;                           }
    ;

exp: NUM                        { $$ = $1;                           }
    | VAR                       { $$ = $1->value.var;                }
    | VAR '=' exp               { $$ = $3; $1->value.var = $3;       }
    | FNCT '(' exp ')'          { $$ = (*($1->value.fnctptr))($3);   }
    | exp EQUAL exp             { $$ = ($1 == $3) ? -1 : 0;          }
    | exp '+' exp               { $$ = $1 + $3;                      }
    | exp '-' exp               { $$ = $1 - $3;                      }
    | exp '*' exp               { $$ = $1 * $3;                      }
    | exp '/' exp               { $$ = $1 / $3;                      }
    | '-' exp  %prec NEG        { $$ = -$2;                          }
    | exp '^' exp               { $$ = pow($1, $3);                  }
    | '(' exp ')'               { $$ = $2;                           }
    ;
```

Tipi per variabili e funzioni

```
struct symrec {
  char * name;              /* nome simbolico                */
  int type;                 /* vale FNCT oppure VAR          */
  union {
    double var;             /* VAR>  valore della variabile */
    double (*fnctptr)();    /* FNCT> puntatore alla funzione*/
  } value;
  struct symrec * next;     /* punta al prossimo elemento    */
};

typedef struct symrec symrec;
extern symrec *sym_table;
symrec *putsym();
symrec *getsym();
```

Gestione dei simboli

```
struct init {
  char *fname;
  double (*fct)();
};

struct init arith_fncts[] = {
  "sin",   sin,
  "cos",   cos,
  "atan",  atan,
  "exp",   exp,
  "sqrt",  sqrt,
  0,       0
};

init_table() {
  int i;
  symrec *ptr;
  for (i = 0; arith_fncts[i].fname != 0; i++) {
    ptr = putsym(arith_fncts[i].fname, FNCT);
    ptr->value.fctptr = arith_fncts[i].fnct;
  }
}
```

Gestione dei simboli

```
symrec * putsym(char *sym_name, int sym_type) {
  symrec *ptr;
  ptr = (symrec *) malloc(sizeof (symrec));
  ptr->name = (char *) malloc (strlen(sym_name) + 1);
  strcpy(ptr->name, sym_name);
  ptr->type = sym_type;
  ptr->value.var = 0;
  ptr->next = (struct symrec *) sym_table;
  sym_table = ptr;
  return ptr;
}

symrec * getsym(char *sym_name) {
  symrec *ptr;
  for (ptr = sym_table; ptr != (symrec *) 0;
       ptr = (symrec *) ptr->next)
    if (strcmp(ptr->name, sym_name) == 0)
      return ptr;
  return 0;
}
```