STRUTTURE DATI e LABORATORIO II
Esercitazione n° 7

Valutazione delle espressioni aritmetiche

La valutazione di una espressione è un problema che gli esperti di informatica incontrano spesso. Esempi di espressioni sono

letter[count]==getchar()!=EOL;
leggi(&simbolo,&n) && token != eos;
a/b-c+d*e-a*c;

Valutare un'espressione significa calcolare un valore dipendente dagli operatori, dalle funzioni e dagli operandi che compaiono nell'espressione.

Scrivere un programma in C che valuti una espressione numerica con parentesi in cui gli operatori sono '+', '-', '/', '*' '%'.

Suggerimenti:
Le operazioni da fare sono:

- Leggere l'espressione da valutare in notazione infix e memorizzarla come una stringa;
- Trasformare l'espressione infix in espressione postfix mediante il seguente algoritmo

```
void postfix(void)
    /* trasforma un'espressione infissa in postfissa*/
 {
 precedenza get_token(char *simbolo, int *n);
 char print_token(precedenza token);
 void add(int *top, int item);
 int  delete(int *top);

 char simbolo;
 precedenza token;
 int n=0;
 int top=0;
 int counter=-1;
 char buf;              /*pone eos nello stac*/
 stack[0]=eos;
 for (token=get_token(&simbolo,&n); token != eos;
         token=get_token(&simbolo,&n)) {
    if(token == operando){
         espr_post[++counter]=simbolo;
          }
    else if (token ==parendx) {
                     /* svuota lo stack fino alla parentesi*/
      while (stack[top] != parensx) {
         buf=print_token(delete(&top));
         espr_post[++counter]= buf;
          }
      delete(&top);         /*elimina la parentesi sinistra*/
    }
```

```
      else  {
                        /*elimina e visualizza i simboli la cui prd
                         e' >= alla prf del token corrente */
        while (prd[stack[top]] >= prf[token]){
           buf=print_token(delete(&top));
           espr_post[++counter]=buf ;
          }
        add(&top,token);
      }
   }
   while ((buf=print_token(delete(&top))) != '\0') {
          espr_post[++counter]= buf;
        }
     espr_post[++counter]='\0';
     printf("espressione post_fix: ");
     for (n=0; n<=counter;n++)
         printf("%c ", espr_post[n]);
}
```

- Valutare l'espressione postfissa mediante l'algoritmo

```
int valuta(void)
  /* valuta un'espressione postfissa memorizzata in espr
   si suppone che gli operandi siano numeri ad una sola cifra*/
{
  precedenza get_token(char *simbolo, int *n);
  void add(int *top, int item);
  int  delete(int *top);

  precedenza token;
  char simbolo;
  int op1, op2;
  int n=0;                /*contatore per l'espressione*/
  int top=-1;

  token=get_token(&simbolo, &n);
  while (token != eos) {
    if(token == operando)
       add(&top, simbolo-'0');      /*inseriesce nello stack*/
    else  {
        /* elimina i 2 operandi e pone il risultato nello stack*/
       op2=delete(&top);        /*elimina dallo stack*/
       op1=delete(&top);
       switch (token) {
        case plus : add(&top, op1+op2);
                    break;
        case meno : add(&top, op1-op2);
                    break;
        case per:   add(&top, op1*op2);
                    break;
        case dividi:add(&top, op1/op2);
                    break;
        case mod:   add(&top, op1%op2);
                    break;
        }
    }
    token=get_token(&simbolo, &n);
  }
  return delete(&top);      /*restituisce il risultato*/
 }
```

Le funzioni add() e delete() riferite allo stack possono definirsi come

```
void add(int *top, int item)

           /* aggiunge un elemento allo stack globale */
    {
      if(*top >=MAX_STACK_SIZE-1) {
       printf("\nstack_full");
       return;
      }
      stack[++*top] = item;
    }


 int  delete(int *top)
   {
     if(*top == -1)
            return printf("\nstack_empty");           /*fornisce un
                                                      codice d'errore*/
     return stack[(*top)--];
 }
```

Le dichiarazione globali possono essere

```
#define MAX_STACK_SIZE 100    /*dimensione max dello stack*/
#define MAX_EXPR_SIZE 100     /*dimensione max espressione*/

typedef enum {parensx, parendx, plus, meno, per ,dividi,
           mod, eos, operando} precedenza;
int stack[MAX_STACK_SIZE];               /*stack globale*/
char espr[MAX_EXPR_SIZE];                /*stringa di input*/
char espr_post[MAX_EXPR_SIZE];           /*stringa post_fix */
static int prd[]= { 0, 19, 12, 12, 13, 13, 13, 0};
static int prf[]= {20, 19, 12, 12, 13, 13, 13, 0};
```

Buon lavoro!