Strutture Dati

Lezione 8

La valutazione di una espressione

Oggi parleremo di …

- Un'applicazione dello ***stack***
  - la valutazione di una espressione
  - rappresentazione di una espressione
  - valutazione di una espressione postfissa
  - trasformazione dal formato infisso al formato postfisso.

L'ordine di valutazione

```
(!lab[riga_succ][col_succ] && !segna[riga_succ][col_succ])
*dietro = (*dietro + 1) % MAX_SIZE;
a = b + c / d;
```

- Quale è l'ordine con cui eseguire le operazioni?

```
a = b + c / d – e * f;     b = 4  c = 10  d = 2  e = 3  f = 5

a = (4 + (10 / 2)) – (3 * 5)
  = (4 + 5) – 15
  = 9 – 15
  = -6

a = (((4 + 10) / 2) – 3) * 5
  = ((14 / 2) – 3) * 5
  = (7 – 3) * 5
  = 4 * 5
  = 20
```

L'ordine di valutazione

- Ogni linguaggio di programmazione ha una gerarchia di operatori.
- Per modificare l'ordine degli operatori si introducono le parentesi
  - vengono eseguite per prime le espressioni nelle parentesi più interne.

```
a = b + c / d – e * f;          a = (b + c) / (d – e) * f;
```

La gerarchia degli operatori C

Simboli	Operatori	Priorità	Associatività
( )	Chiamata di funzione	17	da sin a des
[ ]	Elemento di array		
-> .	Membro di struttura o unione		
-- ++	Decremento, incremento (postfix)	16	da sin a des
-- ++	Decremento, incremento (infix)	15	da des a sin
!	Not logico		
~	Complemento a uno		
- +	Meno o più unario		
& *	Indirizzo o indirezione		
sizeof	Dimensione in byte		
(tipo)	Cast di tipo	14	da des a sin
* / %	Moltiplicativi	13	da sin a des
+ -	Somma e sottrazione binaria	12	da sin a des
<< >>	Traslazione	11	da sin a des
> >= < <=	Relazionali	10	da sin a des
== !=	Uguaglianza	9	da sin a des
&	AND su bit	8	da sin a des
^	XOR su bit	7	da sin a des
|	OR su bit	6	da sin a des
&&	AND logico	5	da sin a des
||	OR logico	4	da sin a des
?:	Condizionale	3	da des a sin
= += -= /= *= %= <<= >>= &= ^= |=	Assegnazione	2	da des a sin
,	Virgola	1	da sin a des

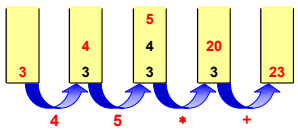
Valutazione di un'espressione postfissa

```
a + b   ⟶   a b +
a + b * c   ⟶   a b c * +
```

- Si esamina l'espressione da sin a des
- Si pongono gli operandi nello stack finchè non si incontra un operatore
- Si eliminano dallo stack gli operandi richiesti dall'operatore
- Si pone il risultato dell'operazione nello stack
- Si ripete sino ad incontrare la fine dell'espressione
  - il risultato dell'intera espressione si trova in cima allo stack

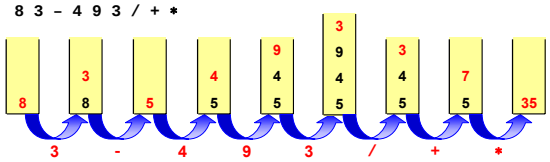
Valutazione di un'espressione postfissa

a + b * c ⟶ a b c * +



3 4 5 * +

8 3 – 4 9 3 / + *



Rappresentazione di un'espressione

- Consideriamo solo espressioni contenenti gli operatori binari +, -, /, *, %
- Gli operandi sono rappresentati da un solo carattere
- Rappresentiamo l'intera espressione come un array di caratteri.

```
#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*/
```

Valutazione di un'espressione postfissa

```
int valuta(void)
{
   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');   /* ins nello stack */
         else {
               /* elimina i 2 operandi dallo stack ponendovi il risultato */
                op2=del(&top);
                op1=del(&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 del(&top);      /* restituisce il risultato */
}
```

Valutazione di un'espressione postfissa

```
precedenza get_token(char *simbolo, int *n)
{
 /* acquisisce il token successivo;
 simbolo è la rappresentazione del carattere che viene fornito;
 il token è rappresentato dal suo valore enumerativo,
 che è fornito nel nome della funzione */

 *simbolo = espr[(*n)++];
 switch (*simbolo) {
        case '(' :    return parensx;
        case ')' :    return parendx;
        case '+' :    return plus;
        case '-' :    return meno;
        case '/' :    return dividi;
        case '*' :    return per;
        case '%' :    return mod;
        case ' ' :    return eos;
        default  :    return operando;
   }
}
```

Dal formato infisso al postfisso

1. Inserire tutte le parentesi nell'espressione infissa
2. Spostare tutti gli operatori binari in modo da sostituire le corrispondenti parentesi destre
3. Cancellare tutte le parentesi

a + b / c – d * e

((a + (b / c)) – (d * e))
((a + b c /) – d e *)
(a b c / + – d e *)
a b c / + d e * -

Non è conveniente!

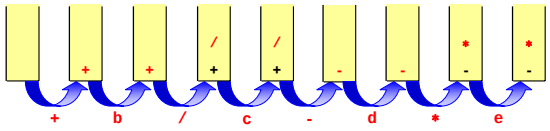
Dal formato infisso al postfisso

- Si esamina l'espressione infissa da sin a des.
- Gli operandi vanno mandati subito in output.
- Gli operatori vanno mandati in output tenendo conto delle loro precedenze.
- Gli operatori con priorità più elevata sono inviati all'output prima di quelli con priorità più bassa.
- Gli operatori vanno memorizzati nello stack finchè la priorità dell'operatore in cima allo stack è di livello più basso rispetto a quella dell'operatore in arrivo.
- Lo stack verrà svuotato quando si giunge alla fine dell'espressione.

Dal formato infisso al postfisso

- Caso senza parentesi

Esempio 1: `a + b / c – d * e`

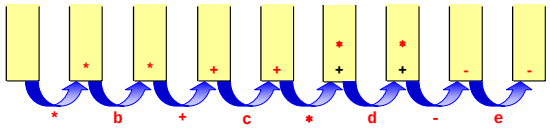


`a b c / + d e * -`

Dal formato infisso al postfisso

- Caso senza parentesi

Esempio 2: `a * b + c * d - e`



`a b * c d * + e -`

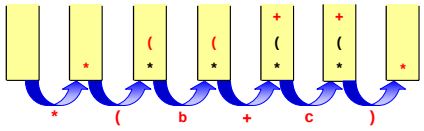
Dal formato infisso al postfisso

- In presenza di parentesi gli operatori vanno inseriti nello stack finchè non si incontra la parentesi destra.
- Lo stack viene svuotato finchè non si incontra la corrispondente parentesi sinistra, che viene quindi rimossa.

Dal formato infisso al postfisso

- Caso con parentesi

Esempio 1: `a * (b + c)`

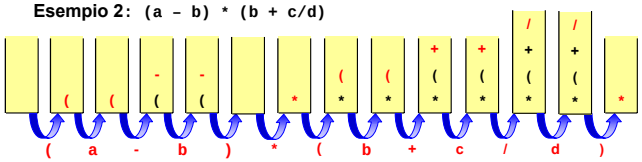


`a b c + *`

Dal formato infisso al postfisso

- Caso con parentesi

Esempio 2: `(a – b) * (b + c/d)`



`a b - b c d / + *`

Dal formato infisso al postfisso

- Ci sono due tipi di priorità
  - una priorità dentro lo stack, `prd`
  - una priorità fuori dallo stack, `prf`
- La parentesi di sinistra ha
  - bassa priorità quando si trova nello stack
  - alta priorità quando è fuori dallo stack

```
static int prd[]= {0, 19, 12, 12, 13, 13, 13, 0};
static int prf[]= {20, 19, 12, 12, 13, 13, 13, 0};
```

- Un operatore verrà estratto dallo stack se la sua priorità **dentro** lo stack è maggiore o uguale alla priorità **fuori** dallo stack del nuovo operatore.

Algoritmo di trasformazione

$O(n)$

```
void postfix(void)
{
   char simbolo;
   precedenza token;
   int n = 0;
   int top = 0;           /* pone eos nello stack */
   int counter = -1;

   stack[0] = eos;
   for (token=get_token(&simbolo,&n); token != eos;
         token=get_token(&simbolo,&n)) {
      if(token == operando)  printf("%c", simbolo);
      else if (token == parendx) {
                        /* svuota lo stack fino alla parentesi sx */
               while (stack[top] != parensx)     print_token(del(&top));

               del(&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])  print_token(del(&top));
                 add(&top,token);
               }
    }
    while ( (token = del(&top)) != eos)    print_token(token);
}
```

Laboratorio: Esercitazione 7

- Scrivere un programma in C che valuti una espressione aritmetica con parentesi in cui gli operatori sono '+', '-', '/', '*' '%'.
- **Suggerimenti:**
  - leggere l'espressione da valutare in notazione infix e memorizzarla come una stringa;
  - trasformare l'espressione infix in espressione postfix mediante l'algoritmo `postfix;`
  - valutare l'espressione postfissa mediante l'algoritmo `valuta.`