Aritmetica postfissa

Le espressioni aritmetiche sono normalmente scritte in notazione infissa, cioe' in modo tale che i simboli delle operazioni binarie compaiono tra gli oggetti su cui operano.

ESEMPIO: 20 + 15 * (100 – 5) – 21

Nella notazione postfissa gli operatori seguono gli oggetti su cui operano

ESEMPIO : 20 15 100 5 – * + 21 –

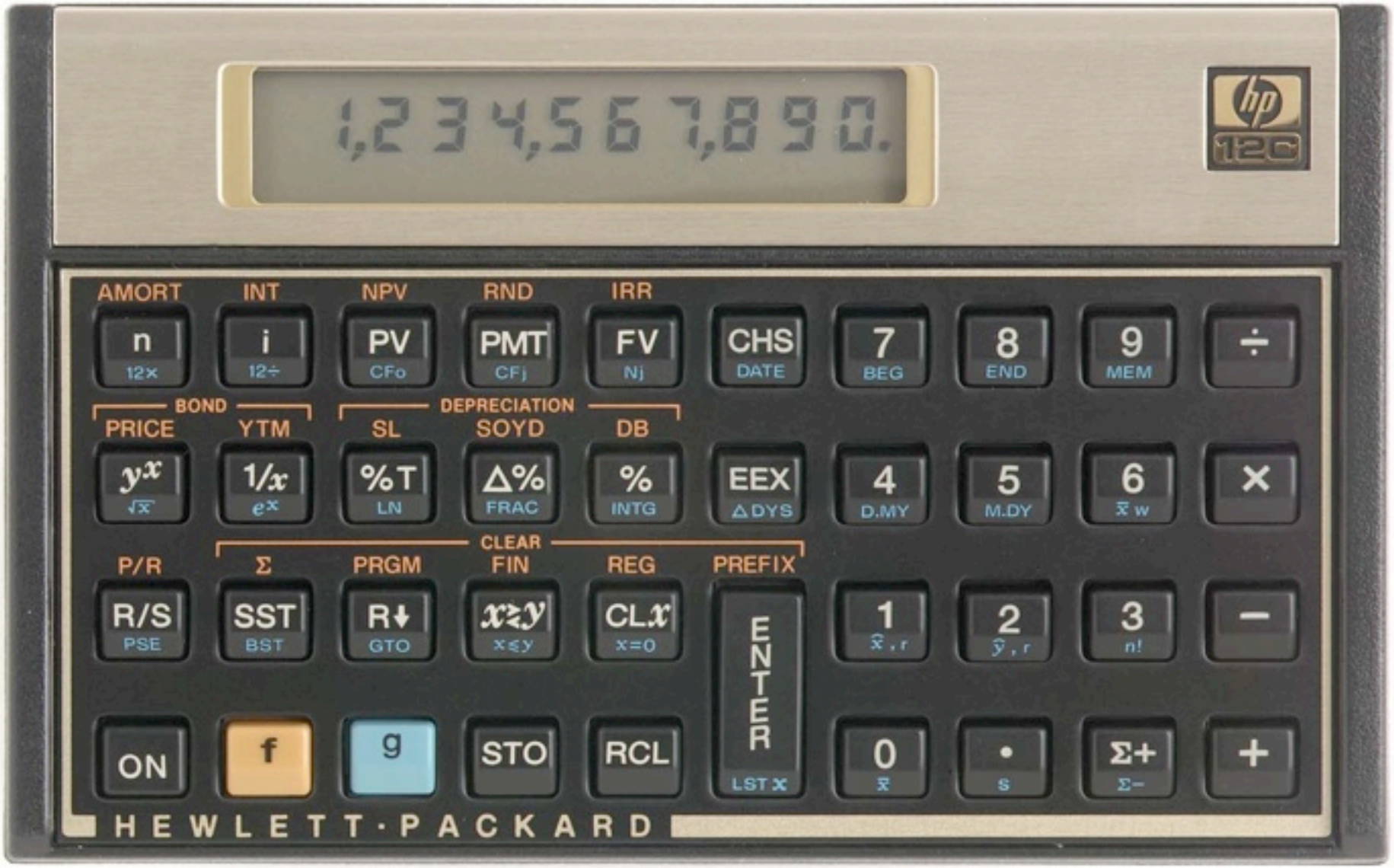
La notazione postfissa è detta polacca inversa (RPN) in omaggio al matematico polacco LUKASIEWICZ che l'ha introdotta.

Con la notazione polacca otteniamo espressioni aritmetiche che possono essere calcolate mano mano che vengono lette, pur non contenendo parentesi e senza far riferimento alle regole di precedenza degli operatori.

Infatti per esempio abc*+ corrisponde a a+b*c

mentre ab+c* corrisponde a (a+b)*c

La Hewlett-Packard produce certi modelli di calcolatrici, come quello nell'immagine, che usa esclusivamente la RPN, per la velocità di calcolo che si ottiene. Si vada alla pagina http://www.hp.com/calculators/news/rpn.html per verificare la semplicità d'uso della RPN.



Notazione polacca: definizione

Un'espressione aritmetica in notazione polacca e' una qualunque serie di operandi aritmetici x, y, … e operatori binari op (+,-,*,/) che si può formare mediante le regole seguenti:

1. ***ogni operando x è un'espressione aritmetica in notazione polacca***

2. ***se p1 e p2 sono espressioni aritmetiche in notazione polacca e op è un operatore aritmetico allora anche***

   p1p2 op

è un'espressione aritmetica in notazione polacca

ESEMPI:

p1 = 48 63 + (che corrisponde a 48+63)

p2 = 52 4 * (che corrisponde a 52 * 4)

p3 = p1 p2 - (che corrisponde a p1-p2)

Notazione polacca: regola di colcolo

1. **scandisci l'espressione da sinistra a destra fino a che raggiungi il primo operatore**
2. **applica l'operatore ai due operandi alla sua sinistra, sostituisci il risultato nell'espressione al posto di operandi e operatore**

ESEMPIO:

20 15 100 5 - * + 21 –

20 15 95 * + 21 –

20 1425 + 21 –

1445 21 –

1424

algoritmo per valutare un'espressione in notazione polacca

1. scandisci l'espressione da sinistra a destra.

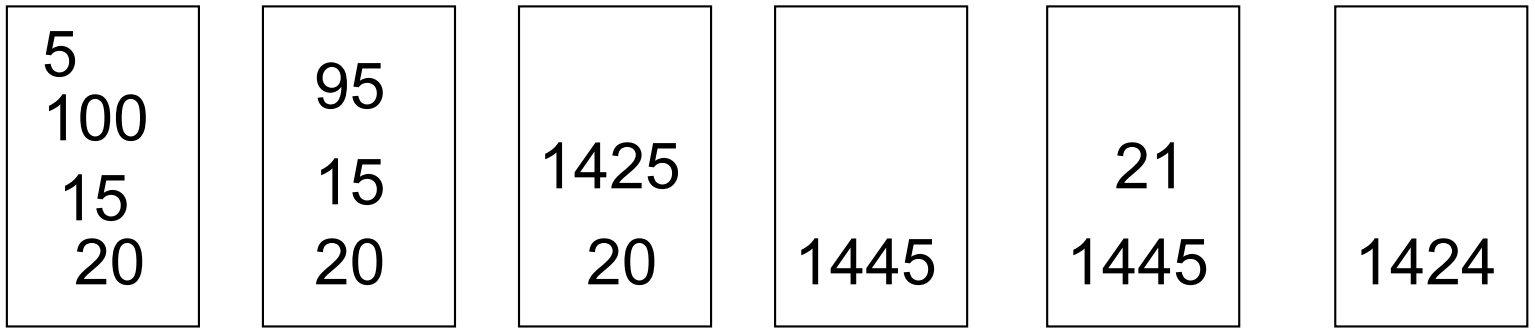
appena leggi un operando compi il passo 2.

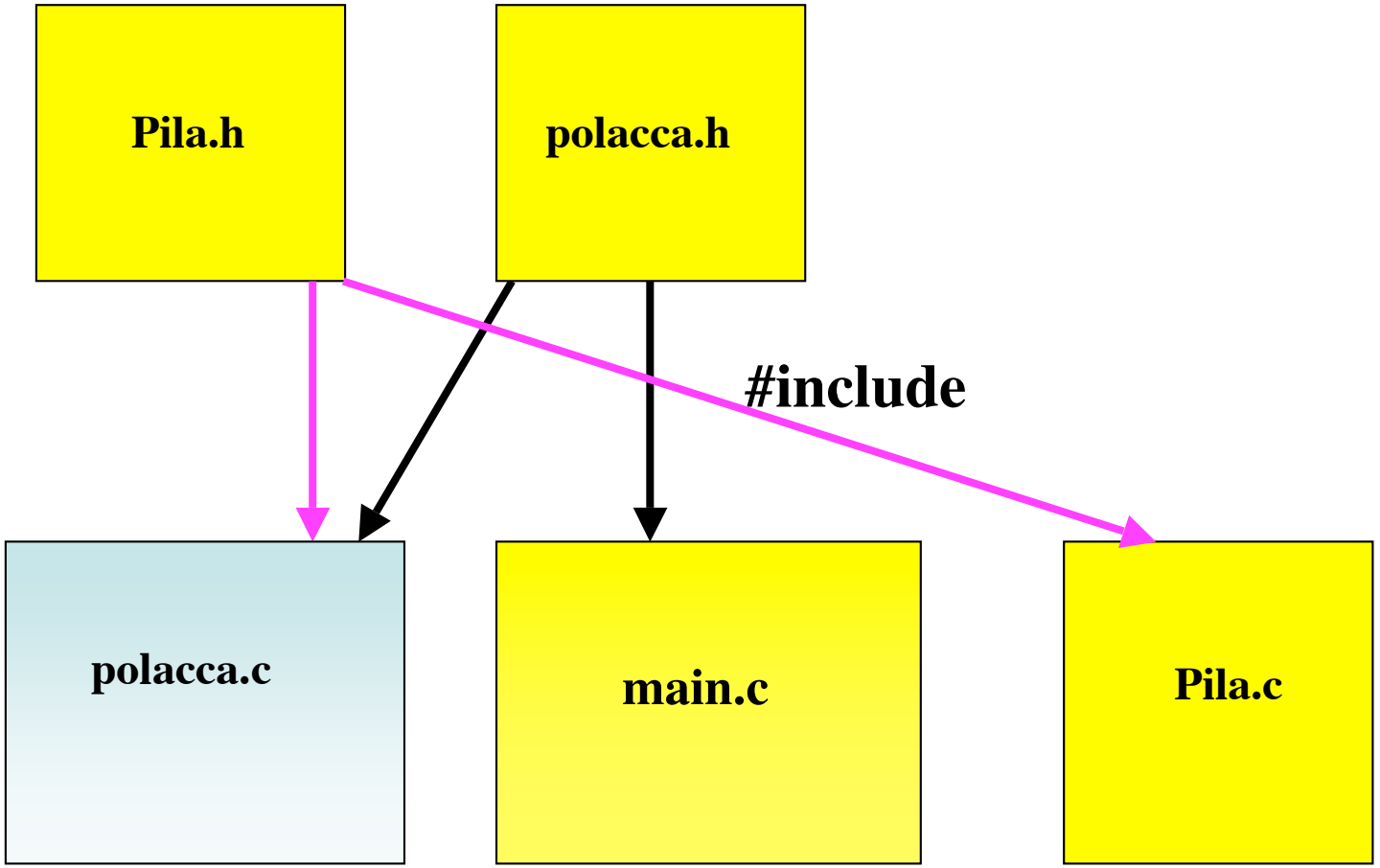
appena leggi un operatore compi il passo 3.

2. impila l'operando nella pila degli operandi

3. rimuovi dalla pila l' operando in cima e salvalo in op1, rimuovi dalla pila l'operando in cima e salvalo in op2, applica l'operatore a op2 e op1 e impila il risultato in cima alla pila.

Esempio: **20 15 100 5 – * + 21 –**




Pila.h
polacca.h
#include
polacca.c
main.c
Pila.c

Aritmetica postfissa, implementazione: file polacca.h

```
enum tokenT {add,subtract,multiply,divide,integer,ill};
/* i tokens in un espressione */

typedef enum tokenT tokenType;

int valutaPolacca (char* espr);
/*prec: espr!= NULL &&  espr != '\0' && espr è in RPN.
postc: restituisce il valore di espr.
*/

tokenType leggi (char ** espr,int *tokenvalue);
/* individua operandi e operatori nell'espressione aritmetica
   postfissa in input, restituendone il tipo e  predisponendo espr
   alla lettura del successivo token.
prec: espr != NULL && *espr != NULL && tokenValue != NULL &&
   *tokenValue ==0.
postc: restituisce add,subtract,multiply,divide se legge l' operatore
   +,-,*, / rispettivamente,integer,ill se legge un intero o un
   valore illegale inoltre il puntatore in espr è portato alla fine
   della porzione di espressione letta.*/
```

Le istruzioni di controllo sulle precondizioni sono omesse nel codice per non appesantire le definizione. Lo studente le aggiunga!

Aritmetica postfissa, implementazione: file polacca.c - le inclusioni

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <assert.h>
#include "polacca.h"
#include "pila.h"
```

Aritmetica postfissa, implementazione: file polacca.c

```
int valutaPolacca (char* espr)
{PilaP  PILA;
int tokenvalue, NumTop, NumSuc;
tokenType t;
PILA = costrPila(strlen(espr));          /*inizializza la pila*/
while  (*espr != '\0')
   {tokenvalue = 0;
    t = leggi(&espr,&tokenvalue);
    if (t == ill) return 0;
    if (t == integer) push(tokenvalue, PILA);
    else
        {NumTop = pop(PILA);
         NumSuc = pop(PILA);
         if (t == add)          NumSuc += NumTop;
         if (t == subtract)     NumSuc -= NumTop;
         if (t == multiply)     NumSuc *= NumTop;
         if (t == divide)       NumSuc /= NumTop;
         push(NumSuc,PILA);}
   }
distrPila(PILA);
return NumSuc;}
```

```
/*pone  in t il tipo
del token letto,
mette in tokenvalue
il suo valore
e predispone la
lettura del prossino
token*/
```

Aritmetica postfissa: la funzione “leggi”

```
tokenType leggi (char ** espr,int *tokenValue)
{tokenType token;
char *s = *espr;
while (isspace(*s)) s++; /* saltiamo eventuali spazi */
if ( isdigit(*s))
   {token = integer;
   while (isdigit(*s)) {*tokenValue = *tokenValue*10 + *s -'0';s++;}
   }
else
switch(*s)
   {case '+' : {token = add; s++; break;}
    case '-' : {token = subtract; s++; break;}
    case '*' : {token = multiply; s++; break;}
    case '/' : {token = divide; s++; break;}
    default  : {token = ill;
              printf("%c  è un carattere illegale.\n",*s);break;
                }
   }
*espr = s;
return token;}
```

Il main

```
#include "polacca.h"

int main (void)
{char * str;
/*si memorizzi in str una RPN*/
if (strlen(str))
printf("Il valore di str = %s è %d\n",str,valutaPolacca(str));
return 0;}
```