INTERMEZZO:
getchar() e putchar()

Il C, fornisce due funzioni (in realtà due macro) che consentono di ricevere un carattere dall'input e di stampare un carattere sullo schermo. Un'istruzione come:

```
char c;
c = getchar();
```

consente di scrivere un carattere da tastiera e salvarlo nella variabile C.

```
char c = 'a';
putchar(c);
```

Consente di scrivere su schermo il carattere contenuto nella variabile c.

getchar() e putchar()

```
#include <stdio.h>
int main void
{
    int c;
    while ((c = getchar()) != EOF){
      putchar(c);
      putchar(c);
    }
    return 0;
}
```

Legge una successione di caratteri da tastiera fino a quando non occorre il simbolo speciale EOF (-1) e ristampa su schermo due volte i caratteri letti.

getchar() e putchar()

```
/* Stampa in maiuscolo */
#include <stdio.h>
int main void
{
      int c;
      while ((c = getchar()) != EOF){
         if (c >= 'a' && c <= 'z')
                putchar(c+'A'-'a');
         else putchar(c);
      }
      return 0;
}
```

Stampa in maiuscolo tutti i caratteri letti. L'istruzione `putchar(c+'A'-'a');` consente di stampare il carattere maiuscolo corrispondente al minuscolo letto.

Conta caratteri in un file

```
#include <stdio.h>
int main(void){
     int blank_cnt=0,digit_cnt=0,letter_cnt=0,
          nl_cnt=0,other_cnt=0,c;
     while (c=getchar()!= EOF){
        if (c==' ') ++blank_cnt;
        else if (c>='0' && c <= '9') ++digit_cnt;
        else if ((c>='a' && c<='z') ||
                  (c>='A' && c>='Z')) ++letter_cnt;
        else if (c=='\n') ++nl_cnt
        else ++other_cnt;
        printf("spazi = %d,lettere =%d,
                 numeri = %d,linee = %d, altri = %d\n",
                blank_cnt,letter_cnt,digit_cnt,
                nl_cnt,other_cnt);
}
```

Un quinto esempio: massimo comune divisore

Scrivere un programma iterativo per calcolare il massimo comune divisore fra due interi positivi X e Y, usando la seguente proprietà del mcd:

se X>Y allora MCD(X,Y)=MCD(X-Y,Y)
se Y>X allora MCD(X,Y) = MCD(X,Y-X)
se X=Y allora MCD(X,Y)=X=Y

```
int x,y; /* input*/
int mcd; /* output */

/* [Pre x=X,y=Y, X,Y>0] */
/* [Post mcd = MCD(X,Y)] */
```

massimo comune divisore: algoritmo

IDEA algoritmo:
Se ad ogni passo dell'iterazione sottraggo y da x se x>y oppure sottraggo x da y se y>x, arriverò ad un momento in cui x=y. !!
In entrambi i casi per la proprietà precedente non sto modificando il valore del massimo comune divisore di x ed y iniziali. Quando termino restituisco il valore trovato per cui x=y

Inizializzazione: x e y hanno i loro valori originali.
Iterazione: x= x-y se x>y oppure y =y-x se y>x
Terminazione: Non voglio continuare il ciclo non appena x=y. Condizione di terminazione:x=y

massimo comune divisore: implementazione

```
#include <stdio.h>
int main(void){
     int x,y; /* input */
     int mcd; /*output*/
     int i; /* var ausiliare */
     / * [Pre: x=x,y=Y, X,Y>0]     *
       * [Post mcd= MCD(X,Y)] */
     printf("Immetti x ed y");
     scanf("%d%d",&x,&y);
     while (x!=y)}
       if (x<y) x-=y;
       else y-=x;;
     }
     mcd =x;
     printf("MCD =%d\n",mcd);
}
```

Il ciclo for

La semantica può essere spiegata usando il ciclo while

```
for(expr1; expr2;expr3)
     I
```

Expr1: rappresenta le istruzioni di Inizializzazione

Expr2: e' la condizione di entrata nel ciclo

Expr3: sono le istruzioni che in un ciclo while si occupano di incrementare o decrementare la variabile contatore

```
for(i=1; i<=10; i++)
     sum +=i;
```

```
i=1
while(i<=10){
     sum +=i;
     ++i;
}
```

Esempi di cicli for

Calcola il fattoriale di n>=1

```
for(i=1; i<=n; i++)
     f*=i;
```

Domanda:
perchè funziona anche con l'incremento come ultima istruzione nel ciclo ??

Domanda:
Cosa calcola il seguente programma?

```
for(j=2; k % j == 0; j++)
     sum +=j;
```

Domanda:
È corretto il seguente programma?

```
for(j=0, j<n, j+=3)
     sum +=i;
```

Cicli for "troncati"

Le varie expr nel ciclo for possono essere omesse, una o anche tutte. Comunque devono SEMPRE essere presenti i ";"

```
i=1;
sum=0;
for( ; i>=10; ++i)
     sum +=i;
```

è CORRETTO e somma i numeri da 1 a 10. Cosi come:

```
i=1;
sum=0;
for(;i>=10 ;  )
     sum += i++;
```

Cicli for "troncati"

Se expr2 manca nel ciclo for viene assunta una condizione sempre differente da 0 (cioè vera) per cui un ciclo for come il seguente non terminerebbe mai.

```
i=1;
sum=0;
for( ; ; )
     sum +=i;
```

I cicli for come quelli while sono istruzioni che al pari delle altri possono essere istruzioni innestate dentro altri cicli while o for.

Uno dei vantaggi del ciclo for è che sia il controllo che la gestione della variabile indice vengono scritti all'inizio del ciclo e sono più leggibili specialmente quando ci sono più cicli innestati uno nell'altro.

Verifica del prodotto

(1) Dopo l'inzializzazione l'invariante si verifica

[Pre: x=X,y=Y]

prod=0;

[Post: XY = prod +xy]

(2) se l'invariante è vera prima di entrare nel ciclo, allora dopo essere entrato ed aver eseguito le istruzioni I del ciclo, l'invariante continua ad essere vera

[Pre: XY = prod +xy, (y != 0)]

prod = prod +x;

y = y-1;

[Post: XY = prod +xy]

(3) Non appena esco dal ciclo l'invariante implica la postcondizione Q del ciclo

[Pre: XY= prod +xy , (y=0)] → [Post: prod =XY]

Costrutto do... while

Il C come altri linguaggi mette a disposizione un costrutto while in cui a differenza del caso normale, il ciclo viene ripetuto almeno una volta.

Questo è il costrutto do.... while, la cui sintassi è data dalla seguente regole

<do_statement> ::= do <statement> while (<espressione>)

La semantica del costrutto è la seguente

```
do {
     I
} while (expr)
```

Esegui le istruzioni in I, poi valuta `expr`. Se `expr` è differente da 0, allora torna all'inzio delle istruzioni in I.

Costrutto do... while: Esempi

Leggere una sequenza di interi dall' input fin quando non viene inserito un intero positivo

```
do{
printf("Inserisci:");
     scanf("%d",&x);
     if (x<=0){
       printf("ERRORE \n");
      error = 1;
     }
     else error = 0;
} while (!error)
```

Semplificato in C

```
do{
printf("Inserisci:");
     scanf("%d",&x);
     if (error=(x<=0))
       printf("ERRORE \n");
} while (!error)
```

Istruzione goto

L'istruzioni goto, presente in molti linguaggi di basso livello, permette di effettuare un salto incondizionato da una linea ad un'altra di un programma. Tale linea viene segnalata da un etichetta.

Sintassi:

<istruzione etichettata>::= <identificatore> : <istruzione>;

<istruzione> ::= <istruzione composta> | <istruzione di selezione>
<istruzione iterativa> | <istruzione etichettata> | ...
.....

Esempio

```
goto jump;
if (a == b)
    jump: x= y*n;
else  x= y/n;
```

```
goto bre23;
.....
.....
bre23: jump : x=y;
```

Istruzioni break e continue

Ci sono due istruzioni che interrompono il normale flusso del controllo.

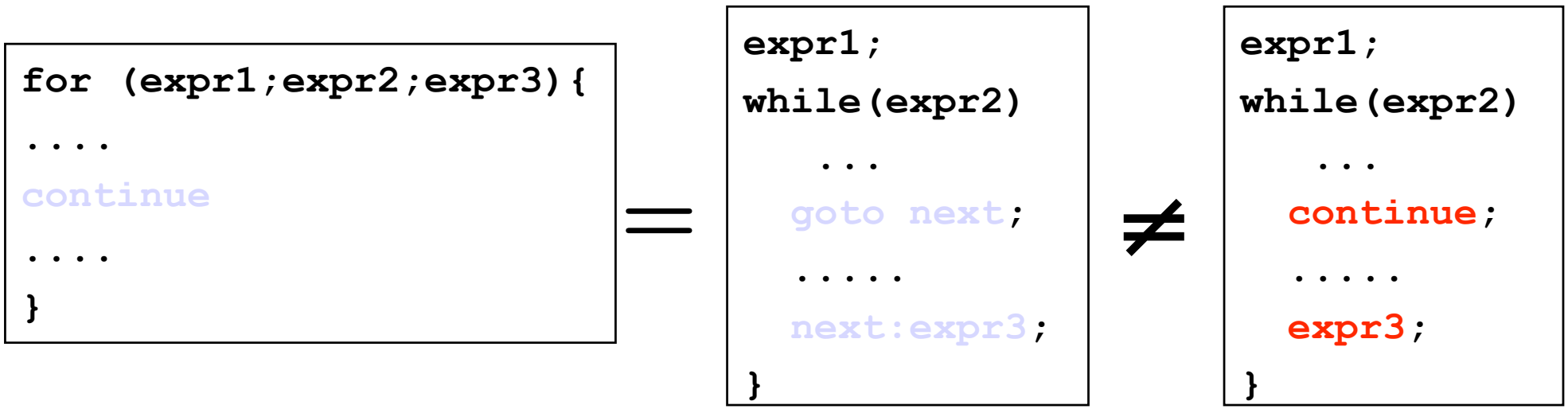
- **break** causa l'uscita dal ciclo più interno che la contiene o da uno switch
- **continue** interrompe l'esecuzione dell'iterazione e riporta il flusso di controllo all'inizio del ciclo, cioè all'iterazione immediatamente successiva.

Esempi

```
while (1){
    scanf("%lf",&x);
    if (x<0.0) break; /* se x è negativa esce dal ciclo */
    printf("Radice = %f\n",x);
}
/* il break salta a questo punto */
```

Istruzioni break e continue

```
Esempio
for(i=0;i<TOTAL;++i){
      c= getchar();
      if (c>='a' && c <= 'z') continue;
      ..... /* elabora gli altri caratteri */
      /* continue trasferisce il controllo a questo punto */
}
```



Selezione Multipla: switch

L'operatore switch è una scelta condizionale multipla.

```
char c;
switch(c){
case 'a':
     scanf(%lf",&x);
     break;
case 'b':
case 'B':
     ++i;
     break;
default:
     ++j;
}
```

solo espressioni intere

necessario break

tutti gli altri casi

Operatore condizionale

L'operatore condizionale e' un operatore ternario che permette di implementare semplici if-then-esle con l'uso di una semplice espressione.

Sintassi

<espressione condizionale>::= (<espressione>) ? <espressione> : <espressione>

Semantica: (expr1) ? expr : expr3

Viene valutata expr1 e se vera (differente da 0) si valuta expr2 e in questa caso il valore dell'espressione è quello assunto da expr2, altrimenti viene valutata expr3 e il valore dell'espressione è quello di expr3.

```
Minimo con if-then-else
if(x<y)
     min = x;
else min = y;

minimo con operatore condizionale
min = (x<y) ?x: y;
```

Operatore virgola

L'operatore virgola consente di scrivere più assegnamenti in un'unica espressione. È l'operatore a più bassa priorità. L'espressione con operatore virgola assume il valore del suo operando più a destra

Sintassi

<espressione virgole>::= <espressione>, <espressione>

Semantica: expr1, expr2

Viene valutata expr1 e poi expr2 e l'intera espressione assume il valore di expr2.

Esempio

```
int a,b;
a=0, b=1;   /* Assume il valore 1. */
```

Es ciclo for

```
for(sum =0,i=1; i<=n;++i)
        sum +=1;

for(sum =0,i=1; i<=n; sum +=i, ++i)
      ;
```

Problema

Riscrivere il seguente codice senza break.

```
while (c= getchar()){
        if  (c== 'E')
            break;
        ++cnt;
        if (c>='0' && c<= '9')
            ++digit_cnt;
}
```

```
while ((c= getchar()) != 'E'){
        ++cnt;
        if (c>='0' && c<= '9')
            ++digit_cnt;
}
```

Problema

```
while ((c= getchar()) != EOF){
        if  (c== 'E')
            break;
        ++cnt;
        if (c>='0' && c<= '9')
            ++digit_cnt;
}
```

È corretto ????

```
while (((c= getchar()) != EOF) && ((c = getchar())  != 'E')){
        ++cnt;
        if (c>='0' && c<= '9')
            ++digit_cnt;
}
```

È corretto

```
while (((c= getchar()) != EOF) && ((c =  != 'E')){
        ++cnt;
        if (c>='0' && c<= '9')
            ++digit_cnt;
}
```

Problema

Scrivere un espressione che usi l'operatore condizionale per trovare il minimo tra tre interi

```
int x,y,z;
(((x<y) ? x : y) < z) ? ((x<y) ?x:y):z);
```

Problema

Riscrivere il seguente codice senza continue.

```
i= -5;
n=50;
while (i<n){
        ++i;
        if  (i==0)
            continue;
        total+=i;
        printf("i=%d, total = %d\n", i, total);
}
```

```
 i= -5;
n=50;
while (i<n){
        ++i;
        if  (i!=0){
            total+=i;
            printf("i=%d, total = %d\n", i, total);
        }
}
```

Problema

Scrivere un programma per approssimare la radice quadrata di un reale a, considerando che è possibile dimostrare che la successione

$$x_0 = 1$$
$$x_{i+1} = 1/2(x_i + a/x_i)$$

converge alla radice quadrata di a, per i che tende all'infinito.

double a; /* input */

double radice; /*output*/

Strategia: Calcoliamo i valori successivi della successione xi.

Domanda: Fino a quando ?

Risposta. i nostri reali hanno una precisione finita, perciò ci sarà un momento nel calcolo degli xi in cui non riusciremo a calcolare nuovi valori. Da un momento in poi inizieremo a calcolare valori sempre uguali. Vogliamo fermarci la prima volta che ciò accade.

Problema

Strategia. Usiamo un variabile x0 per salvare un valore della sequenza e un variabile x1 per calcolarci il successivo. Continuiamo a iterare mentre x0 e x1 non sono uguali.
double x0,x1;

Inizializzazione: x0=1 e x1=1/2*(x0+ a/x0)

Iterazione: Per avanzare di un passo nell'iterazione, settiamo x0 al vecchio valore di x1 e calcoliamo il nuovo x1, in funzione di x0
x0=x1;
x1=0.5*(x0+ a x0)

Terminazione: Termino la prima volta che x0=x1.

Dimostrazione di Terminazione: Prima o poi x0=x1 perchè la successione converge a un valore e perchè la precisione dei double è comunque finita.

Problema

```
double x0,x1;
int i /* conta il numero di iterazioni */

x0 = 1;                          /* inizializzo x0 */
x1 = 0.5*(x0+(a/x0));            /* inizializzo x1*/
i=0;                             /* inizializzo il contatore */
printf("a= %lf, val = %lf, Iter. = %d\n",a,x0*x0,i);
while (x0 != x1){
     x0 = x1;
     x1 = 0.5*(x0+(a/x0));
     ++i;
     printf("a= %lf, val = %lf, Iter. = %d\n", a,x0*x0,i);
               /* stampo ogni iterazione per verificare
                  quanto sono vicino al valore cercato */
}
printf("La radice quadrata è¨ = %lf\n",x0);
```

Problema 1

Problema:
si consideri la seguente funzione

```
int distanza(int a[], int b[] int dim)
/* pre: dim è la lunghezza di a e b;
         a è ordinata decrescentemente
         b è  ordinata crescntemente
        a[0]>= b[0]
        A[dim-1]<b[dim-1] */
/* Post: se distanza(a,b,dim)=i allora a[i]>=b[i] e a[i+1]<b[i+1] */
```

Esempio:

Descrivere l'idea e le variabili usate, l'inizializzazione, il corpo dell'iterazione, la terminazione, la verifica di terminazione e l'invariante.