Istruzioni

Programmazione "strutturata"

- Si parla di programmazione strutturata se si utilizzano solo le seguenti strutture per alterare il flusso di controllo [*Dijkstra, 1969*]:
  - **concatenazione** (o **composizione**)
    legata al concetto di enumerazione
  - **selezione** (o **istruzione condizionale**)
    ramifica il flusso di controllo in base al valore vero o falso di una espressione detta "*condizione di scelta*"
  - **iterazione**
    dà luogo all'esecuzione ripetuta di un' istruzione (semplice o composta) finché permane vera una espressione detta "*condizione di iterazione*"
- **OBIETTIVO: rendere i programmi più *leggibili*, *modificabili* e *mantenibili***

Teorema di Jacopini-Böem

Teorema Le strutture di ***concatenazione***, ***iterazione*** e ***condizione*** costituiscono un insieme **completo** in grado di esprimere ***tutte le funzioni calcolabili***

Quindi

L'uso di queste *sole* strutture di controllo *non limita* il potere espressivo.

Programmazione strutturata in C:

Istruzioni composte

- concatenazione di istruzioni "semplici" → **;**
- blocco → **{ }**

Istruzioni condizionali

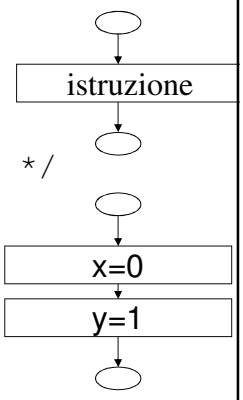
- alternativa → **if (), if () else**
- selezione → **switch ()**

Istruzioni di iterazione

- ciclo → **while (), do while (), for (; ;)**

Istruzioni "semplici"

Qualsiasi espressione (logica, condizionale, assegnamento) seguita da un **;**

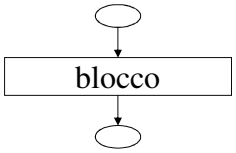


Esempi

```
/* concatenazione di due istruzioni */
     x = 0;  y = 1;
/* una sola istruzione */
     x = 0, y = 1;

     k++;
/*  non fa nulla */
     3;
/*  istruzione nulla */
     ;
```

Blocco

<blocco> ::=
{
[<dichiarazioni >]
{ <istruzioni> }
}



Note

- dopo un blocco non occorre il punto e virgola (esso *termina* le istruzioni semplici, non *separa* istruzioni)
- il campo d'azione delle <dichiarazioni > che compaiono entro il blocco è il blocco stesso
- i blocchi possono essere annidati

Esercizio

Calcola la somma di due numeri dati.

- Riprendiamo l'algoritmo specificato con i flowchart e lo codifichiamo in C

Ricordiamo la struttura di un programma C

La parte <main> è l'unica obbligatoria in un programma C, ed è espressa dalla produzione:

```
<main> ::=    main(){
                        [ <dichiarazioni> ]
                        [ <sequenza-istruzioni>
]
                    }
```

Dichiarazioni e definizioni: variabili, tipi, costanti, etc.

Sequenza istruzioni: sequenza di frasi (*istruzioni*) del linguaggio

In C il **`main()`** è una funzione come le altre, con un'unica caratteristica che la differenzia: **è la prima funzione che viene eseguita**!

Esercizio (*cont.*)

Programma (*in linguaggio C*)

```
main()
 {
  int a, b, somma;
   printf("Immettere due numeri: ");
   scanf("%d %d", &a, &b);
   somma=a+b;
   printf("Somma: %d\n", somma);
 }
```

Esercizio

IL PROBLEMA:

Data una temperatura in ingresso, espressa in gradi Celsius o Fahrenheit, calcolare il corrispondente valore espresso rispettivamente in gradi Fahrenheit o Celsius

... LA SPECIFICA ....

Le relazioni tra le grandezze:

c * 9/5 = f - 32

→ c = (f - 32) * 5/9

→ f = 32 + c * 9/5

Esercizio (*Programma*)

```
#include <stdio.h>
main()
{
  const float  F1=9.0, F2=5., SH=32;
  float  temp_inp, temp_out;
  char  scala;
  scanf("%f", &temp_inp);
  scanf("%c", &scala);
  temp_out = (scala == 'C') ?
  (SH+temp_inp*F1/F2) : (F2/F1*(temp_inp-SH));
  printf ("%f\n", temp_out);
}
```

Esempio (blocco annidato)

```
#include <stdio.h>
main()
{      /* INIZIO BLOCCO */
   const float  F1=9.0, F2=5., SH=32;
   float  temp_inp, temp_out;
   scanf("%f", &temp_inp);
   {/* INIZIO BLOCCO INTERNO */
       char  scala;
       scanf("%c", &scala);
       temp_out = (scala == 'C') ?
         (SH+temp_inp*F1/F2) : (F2/F1*(temp_inp-SH));
    }/* FINE BLOCCO INTERNO */
    printf ("%f\n", temp_out);
}            /* FINE BLOCCO */
```

Il campo d'azione della variabile **scala** è limitato al blocco più interno in cui **scala** è dichiarata.
La variabile **scala** non è più visibile (e neppure ***esistente!***) dopo la **'}'** che chiude il blocco interno

Esercizio (*Specifica*)

- Scrivere un programma che dato un intero positivo compreso fra 100 e 999, produce in uscita il numero ottenuto invertendo l'ordine delle cifre che formano il numero in ingresso.
- Per esempio: **234** → **432**

Esercizio (*Algoritmo*)

Idea!: *Utilizzare le operazioni di modulo e di divisione fra numeri interi.*

Dato un *numero*, valgono le seguenti relazioni:

- **Unita = (numero)%10;**
  - **Es., 234%10 = 4**
- **Decine = (numero/10)%10;**
  - **Es., (234/10)%10 = 3**
- **Centinaia = (numero/100)%10;**
  - **Es., (234/100)%10 = 2**

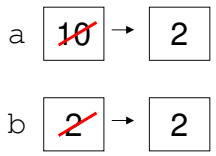
Esercizio (*Programma*)

```
int main()
{
  int numero;
  int unita, decine, centinaia, risultato;

  scanf("%d", &numero);

  unita = (numero)%10;
  numero = numero /10;
  decine = numero%10;
  numero = numero / 10;
  centinaia = numero%10;

  risultato =unità*100+decine*10+centinaia;
  printf("%d", risultato);
}
```

Esercizio

- Scrivere un programma che effettua lo scambio dei valori di due variabili

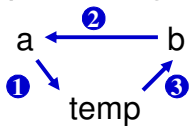
```
main(){
int a=10,b=2;
a=b;
b=a;
}
```



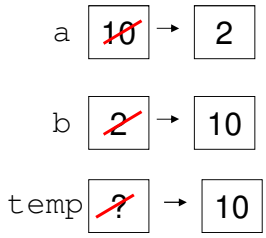
- Questo programma non può funzionare!!

Esercizio (cont.)

- Serve una *variabile ausiliaria* (**temp**) dove salvare il valore della variabile che viene sovrascritta per prima per poi assegnarlo alla seconda variabile




```
main(){
int a=10,b=2,temp;
temp=a;
a=b;
b=temp;}
```



Esercizio

Scrivere un programma che, dati hh:mm:ss di inizio viaggio e il tempo di percorrenza in secondi, calcoli l'arrivo nella forma hh:mm:ss.

```
#include <stdio.h>
void main ()
{ int ore, minuti, secondi, tempov;

  printf("ore: ");          scanf("%d",&ore);                10
  printf("minuti: ");             scanf("%d",&minuti);       21
  printf("secondi:");             scanf("%d",&secondi);      37
  printf("tempo in secondi:"); scanf("%d",&tempov);          23678
  secondi = secondi+tempov;  printf("%d\n",secondi);         23715
  minuti = minuti+secondi/60; printf("%d\n",minuti);         416
  secondi = secondi % 60;         printf("%d\n",secondi);    15
  ore = ore + minuti / 60;        printf("%d\n",ore);        16
  minuti = minuti % 60;    printf("%d\n",minuti);            56
  ore = ore % 24;          printf("%d\n",ore);               16
 }
```


Esercizio

Scrivere un programma per il calcolo del resto per macchine per pagamenti automatici.

- Si supponga che la macchina disponga di un numero illimitato di monete da € 1 e € 2 e di banconote da € 5.
- Il programma riceve in input la cifra spesa e quella inserita dal cliente per il pagamento. Entrambi i valori sono interi.
- Il programma deve stampare la composizione del resto (ovvero quanti pezzi da €1,€2,€5) che si realizza col **minor numero** di pezzi.

Esercizio (Programma)

```
#include <stdio.h>
#define moneta1 1
#define moneta2 2
#define moneta3 5

int main(void) {
int spesa, contanti,resto;
int pezzi1,pezzi2,pezzi3;

printf("Cifra spesa: ");
scanf("%d",&spesa);
printf("Cifra inserita:");
scanf("%d",&contanti);
resto = contanti - spesa;
pezzi3 = resto/moneta3;
resto = resto % moneta3;
pezzi2 = resto/moneta2;
resto = resto%moneta2;
pezzi1 = resto/moneta1;
printf("Composizione del
   resto\n");
printf("pezzi da € %d:
   %d\n",moneta1,pezzi1);
printf("pezzi da € %d:
   %d\n",moneta2,pezzi2);
printf("pezzi da € %d:
   %d",moneta3,pezzi3);
}
```


Istruzioni condizionali

Solitamente vengono rese disponibili due tipi di istruzioni condizionali:

- Istruzione di **SCELTA** (semplice) o **ALTERNATIVA**
- Istruzione di **SCELTA MULTIPLA** (questa non è essenziale, ma migliora l'espressività del linguaggio)

In C, l'espressione ternaria condizionale

condizione ? espr1 : espr2

fornisce già un modo per esprimere il concetto di selezione, che però ha un'espressività limitata

- Il codice basato su essa diviene *praticamente illeggibile* appena la complessità del problema aumenta (anche di poco).
- L'istruzione di scelta fornisce un più alto livello di espressività, comprensibilità e leggibilità.

Istruzione di scelta semplice (o Alternativa)

Consente di scegliere fra ***due istruzioni alternative*** in base al verificarsi di una particolare ***condizione***

<istruzione-di-scelta> ::=
if (***condizione***) < istruzione1>**;**
[**else** < istruzione2>**;**]

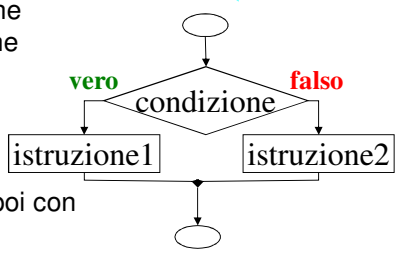
Notare il ;

Opzionale

Condizione **è un' espressione logica** che viene valutata al momento dell' esecuzione dell' istruzione **if**

- se *condizione* risulta vera si esegue <istruzione1>,
- altrimenti si esegue <istruzione2>

In entrambi i casi l' esecuzione continua poi con l' istruzione che segue l' istruzione **if**.



NOTA Se *condizione* è falsa e la parte **else** (opzionale) è omessa, si passa subito all' istruzione che segue l' istruzione **if**.

Osservazione

- Secondo la sintassi, <*istruzione1*> e <*istruzione2*> sono ciascuna una *singola istruzione*
- Per specificare più istruzioni in uno dei due rami, è necessario utilizzare un *blocco*

```
if (n > 0)
     {                    /* inizio blocco */
      a = b + 5;
      c = x + a - b;
     }                    /* fine  blocco */
else n = b*5;
```

Istruzioni *if* annidate

- Come caso particolare, <istruzione1> o <istruzione2> potrebbero essere un' altra <istruzione-di-scelta>
- In questo caso *occorre attenzione ad associare le parti* ***else*** *(che sono opzionali) all'* ***if*** *corretto*

In base alla sintassi del linguaggio C, l' ***else*** **è sempre associato all'** ***if*** **più interno**
Se questo comportamento non soddisfa o crea ambiguità, occorre **inserire esplicitamente un blocco**

?

```
if (n > 0)
   if (a>b) n = a;
else n = b*5;          /* else riferito a if (a>b) */

if (n > 0)
   { if (a>b) n = a; }
else n = b*5;          /* else riferito a if (n>0) */
```


Esempio

Data una temperatura in ingresso, espressa in gradi Celsius o Fahrenheit, calcolare il corrispondente valore espresso nell' altro formato.

```
main()
{
  const float F1=9.0, F2=5.0, SH=32;
  int c, f, temper;
  char scala;

printf("Inserisci la temperatura e la scala (C o F): ");
scanf("%d", &temper);
scanf("%c", &scala);
if (scala != 'F')
{  c = temper;  f = SH + temper * F1/F2; }
else { f = temper;  c = F2/F1 * (temper - SH); }
}
```

Esercizio

Dato il seguente programma indicare il valore delle variabili a, b, c e f al termine di ogni istruzione:

(NB: compilare solo le risposte di fianco alle istruzioni effettivamente eseguite)

int a=3, b=1, c=2; float f=0.0;

a+=++b*c;	a=	b=	c=	f=
if(b>2) b=c-=a;	a=	b=	c=	f=
else b=c+=2;	a=	b=	c=	f=
f=a+b+c / 3;	a=	b=	c=	f=

Esercizio (cont.)

Dato il seguente programma indicare il valore delle variabili a, b, c e f al termine di ogni istruzione:

(NB: compilare solo le risposte di fianco alle istruzioni effettivamente eseguite)

int a=3, b=1, c=2; float f=0.0;

a+=++b*c;	a= 7	b= 2	c= 2	f= 0.0
if(b>2) b=c-=a;	a=	b=	c=	f=
else b=c+=2;	a= 7	b= 4	c= 4	f= 0.0
f=a+b+c / 3;	a= 7	b= 4	c= 4	f=12.0

Esercizio

Dato il seguente frammento di codice indicare il valore delle variabili al termine dell'esecuzione.

```
int a=3,b=2;
if (((b*b+1)%2)==(a-2)) {
a = (b-1)%3;
b -= a*a;}
else
a = (b+1)/2;
b -= a;
```

a = b =

Esercizio (cont.)

Dato il seguente frammento di codice indicare il valore delle variabili al termine dell'esecuzione.

```
int a=3,b=2;
if (((b*b+1)%2)==(a-2)) {
a = (b-1)%3;
b -= a*a;}
else
a = (b+1)/2;
b -= a;
```

a = 1 b = 0

Esercizio

Dato il seguente frammento di codice indicare il valore delle variabili al termine dell'esecuzione.

```
int a=2,b=-1;
if ( ((b*b)%2) != (a+3) ) {
if ((a = -(b-2)%3) != b+1)
b= (b-1)/2+a++; }
else {
if (a = b+1)
if(a%2==0)
b= (b-1)/2+a++;
else a += (b-1)/2;  }
```

a = b =

```
int a=-2,b=-1;
if ( ((b*b)%2) != (a+3) ) {
if ((a = (b-2)%3) != b+1)
b= (b-1)/2+a++; }
else {
if (a = b+1)
if(a%2==0)
b= (b-1)/2+a++;
else a += (b-1)/2;  }
```

a = b =

Esercizio (cont.)

Dato il seguente frammento di codice indicare il valore delle variabili al termine dell'esecuzione.

```
int a=2,b=-1;
if ( ((b*b)%2) != (a+3) ) {
if ((a = -(b-2)%3) != b+1)
b= (b-1)/2+a++; }
else {
if (a = b+1)
if(a%2==0)
b= (b-1)/2+a++;
else a += (b-1)/2;  }
```

a =0 b = -1

```
int a=2,b=-1;
if ( ((b*b)%2) != (a+3) ) {
if ((a = -(b-2)%3) != b+1)
b= (b-1)/2+a++; }
else {
if (a = b+1)
if(a%2==0)
b= (b-1)/2+a++;
else a += (b-1)/2;  }
```

a =0 b = -1

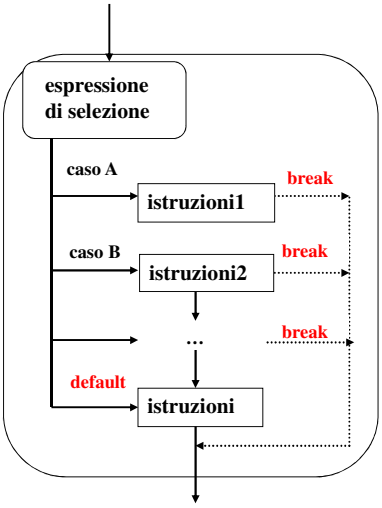
Esercizi

- Tradurre in C gli algoritmi già definiti con i flowchart che risolvono i seguenti problemi:
  - Trovare il massimo di due valori dati
  - Trovare il minimo di due valori dati
  - Calcolare il valore assoluto di un numero dato
  - Verificare se un numero è pari
- Scrivere un programma che calcola il massimo di tre valori dati

Istruzione di scelta multipla

- Consente di scegliere fra ***molte istruzioni*** in base al valore di un' *espressione di selezione*
- L' espressione di selezione deve denotare un **valore numerabile** (*intero*, *carattere*,…)
- Il valore di tale espressione viene confrontato con le costanti che etichettano i vari casi: l' esecuzione prosegue dal ramo dell' etichetta corrispondente (se esiste)
- Se nessuna etichetta corrisponde al valore dell' espressione, si segue il ramo **default** (se specificato)
- Se neanche questo esiste, l' esecuzione prosegue con l' istruzione successiva all' istruzione **switch**

Istruzione di scelta multipla (*cont.*)



Istruzione di scelta multipla (sintassi)

<istruzione-di-scelta-multipla> ::=

switch (*espressione di selezione*)

{

case <*etichetta1*> **:** < istruzioni> [**break;**]

case <*etichetta2*> **:** < istruzioni> [**break;**]

…

[**default :** < istruzione>]

}

***espressione di selezione* è un' espressione che denota un valore numerabile** che viene valutata al momento dell' esecuzione dell' istruzione **switch**

Tutte le costanti <*etichetta*> devono essere dello stesso tipo del *selettore*

Osservazioni

- <istruzioni> denota una *sequenza* di istruzioni per cui *non occorre un blocco* per specificare più istruzioni
- I vari rami ***non* sono mutuamente esclusivi:** una volta imboccato un ramo, l'esecuzione *prosegue in generale con le istruzioni dei rami successivi*
- Per avere rami mutuamente esclusivi occorre **forzare esplicitamente l'uscita** mediante l'istruzione **break**

Esempio

```
main() {
  const float F1=9.0, F2=5., SH=32;
  int c, f, temp;
  char scala;

   printf("Inserisci la temperatura e la scala (C o F): ");
   scanf("%d", &temp);
   scanf("%c", &scala);
   switch (scala)
    {
      case 'C':  c = temp;  f = SH+temp*F1/F2;  break;
      case 'F':  f = temp;  c = F2/F1*(temp-SH);  break;
      default: printf("Errore\n");
    }
}
```

Necessità di *break* per rendere le due alternative mutuamente esclusive

Inutilità di **break** nell'ultimo ramo

Utilità di un ramo di ***default***, perché le due etichette catturano tutti i casi possibili, tranne l'errore in input

Una piccola calcolatrice

- Scrivere un programma che riceve in input una espressione del tipo

  <op1> op <op2>

  dove *<op1>* e *<op2>* sono operandi interi e *op* è un operatore algebrico (+, -,*, /) (ad esempio 2+3, 3/4) e restituisce in output il risultato.

PROBLEMA

Gli operandi sono interi e quindi riesco a leggerli e associarli direttamente a 2 variabili di tipo int.

Ma come faccio a gestire l'operatore?

L'operatore non può che essere di tipo char, ma poi devo essere in grado di interpretarlo all'interno del programma e quindi di eseguire l'operazione corretta.

Esercizio

```
# include <stdio.h>

main(){
    int op1, op2, ris;
    char op;
  printf("Digita l'espressione: ");
  scanf("%d %c %d", &op1, &op, &op2);
  switch (op)
    {
      case '+': ris = op1+op2;  break;
      case '-': ris = op1-op2;  break;
      case '*': ris = op1*op2;  break;
      case '/': ris = op1/op2;  break;
      default: printf("Errore\n");
    }
  printf("%d %c %d = %d", op1, op, op2, ris);
}
```

Pro e contro della scelta multipla

- L'istruzione **switch** evita una (lunga) serie di **if**
- Tuttavia:
  - è utilizzabile solo con espressioni ed etichette di tipo **numerabile** (int, char)
  - **non è utilizzabile** con numeri **reali** (float, double) o con tipi **strutturati** (stringhe, vettori, strutture...)

Esercizio (*Specifica*)

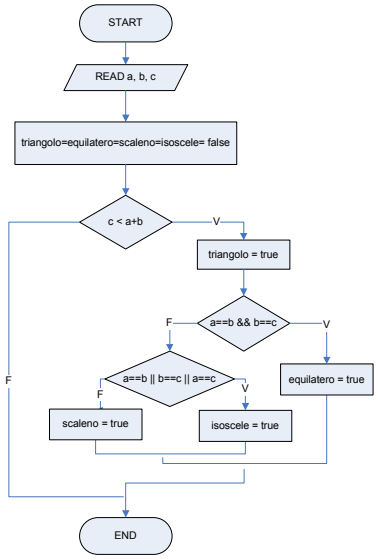
- Dati tre valori $a \leq b \leq c$ che rappresentano le lunghezze di tre segmenti, valutare se possono essere i tre lati di un triangolo.

  Se sì, stabilire il tipo di triangolo: *scaleno*, *isoscele*, *equilatero*.

OSSERVAZIONE

- perché tre segmenti possano costituire i lati di un triangolo, deve valere la relazione $c < (a + b)$

Esercizio (*Algoritmo*)



Esercizio (*Rappresentazione informazioni*)

- Servono 3 variabili (*float*) per rappresentare i valori dei 3 segmenti: **a**, **b**, **c**
- Serve, poi, una variabile logica (in C, di tipo *int*) per rappresentare il fatto se i 3 segmenti possono costituire o meno un triangolo: **triangolo**
- Servono, infine, 3 variabili logiche (in C, di tipo *int*) per rappresentare quale caratteristica del triangolo è vera: **scaleno**, **isoscele**, **equilatero**

Esercizio (*Programma*)

```
#include <stdio.h>
main()
 {
  float a, b, c;
  int triangolo, scaleno, isoscele, equilatero;

  printf("Inserisci le lunghezze dei tre segmenti:");
  scanf("%f %f %f",&a,&b,&c);
  triangolo=scaleno=isoscele=equilatero=0;
  if (a+b>c)
    { triangolo = 1;
      if ((a==b) && (b==c))
           equilatero=1;
      else if ((a==b) || (b==c) || (a==c))
                    isoscele=1;
            else
                    scaleno=1; }
 }
```


Esercizio (*nota*)

Sarebbe stato possibile utilizzare un' istruzione di scelta multipla ***switch*** invece di tanti ***if*** annidati?

No! Perché si tratta di verificare condizioni complesse (uguaglianze multiple), non di semplici scelte da fare in base al valore di una espressione di selezione.

Esercizio (*da completare e far eseguire*)

- Completare l'esercizio precedente mediante:
  - Stampa del risultato: "è un triangolo …" ovvero "non è un triangolo"

Esercizio (*Specifica*)

- Dati tre valori *a*, *b*, *c*, rappresentanti i coefficienti di un'equazione di secondo grado

$$a\,x^2 + b\,x + c = 0,$$

  calcolare le due radici (*se reali*)
- L'algoritmo è già stato descritto con i flowchart, si tratta solo di codificarlo in C

Esercizio (*Rappresentazione informazioni*)

- Servono 3 variabili (*float*) per rappresentare i valori dei 3 coefficienti dell'equazione: **a**, **b**, **c**
- Serve, poi, una variabile *float* per rappresentare **delta** (potrebbe essere utile, per chiarezza, utilizzare un'altra variabile per rappresentare la radice quadrata di delta: **d**)
- Servono, infine, 2 variabili *float* per rappresentare le due radici dell'equazione: **r1**, **r2**

Esercizio (*Programma*)

```
#include <math.h>
main ()
 {
   float a, b, c, delta, d, r1, r2;

   scanf("%f %f %f", &a, &b, &c);
   delta = b*b-4*a*c;
   if (delta>=0)
    {
      d  = sqrt(delta);
       x1 = -(b+d)/(2*a);
       x2 = -(b-d)/(2*a);
    }
  else
   printf("Nessuna soluzione reale!");
}
```

La *direttiva al pre-processore* #include<math.h> *non è un'istruzione C*
Ha lo scopo di rendere disponibili le funzioni matematiche, come sqrt(), fornite nella libreria *math.h* del linguaggio C