NUMERO: 710 DATA: 07/10/2013

A P P U N T I

STUDENTE: Beghini

MATERIA: Informatica

Prof. Acquaviva

INFORMATICA – a.a. 2012/13
Esercitazione di Laboratorio 1

Obiettivi dell'esercitazione

- Disegnare flow-chart
- Prendere confidenza con l'ambiente di sviluppo, compilazione e debug.
- Eseguire programmi scritti in linguaggio C per acquisire da tastiera, manipolare e visualizzare a video valori numerici interi

Contenuti tecnici

- Definizione della funzione *main* in un progamma C
- Definizione di variabili intere (*int*) e loro utilizzo
- Uso di strutture elementari nei flow chart.

Da risolvere preferibilmente in laboratorio

Esercizio 1. Utilizzando l'ambiente di sviluppo, scrivere, compilare ed eseguire il seguente programma in linguaggio C, verificando che non ci siano *errori* né *warning* in fase di compilazione

```
#include <stdio.h>
int main(void)
{
        int x , y, z;

        printf("Introduci un numero intero: ");
        scanf("%d", &x);
        y = 3;
        z = x/y;

        printf("%d/%d=%d\n", x, y, z);
        return 0 ;
}
```

Dopo averlo eseguito, esercitarsi con l'esecuzione passo a passo osservando il valore delle variabili x, y e z tramite i 'watch', provare con diversi valori: 0,9,15,20.

Esercizio 2. Disegnare il flow-chart per il calcolo del modulo (valore assoluto) di un numero; in particolare il programma dovrà:

a) Acquisire da input un valore intero, positivo o negativo, e memorizzarlo in una variabile opportunamente definita.
b) Stabilire utilizzando la struttura elementare *if-then(-else)* se tale variabile contiene un valore negativo e, in questo caso, trasformarlo nel corrispondente valore positivo
c) Inviare in output il valore finale, ovvero il modulo del valore acquisito

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 3 di 159

DEBUG → eseguire passo-passo algoritmo vedendo come variano le variabili nel programma e in base a quello capire se sono presenti errori

COMPILAZIONE crea → file eseguibile → poi interpretato dal computer ed eseguito

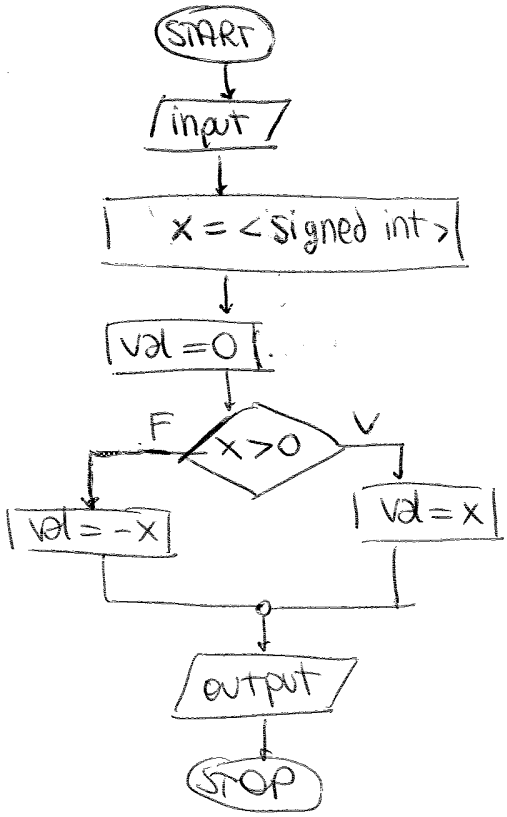
SCANF → blocca programma finché utente non inserisce valore

BREAKPOINT → compilatore esegue tutte le righe prima e poi si blocca aspettando istruzioni dal programmatore

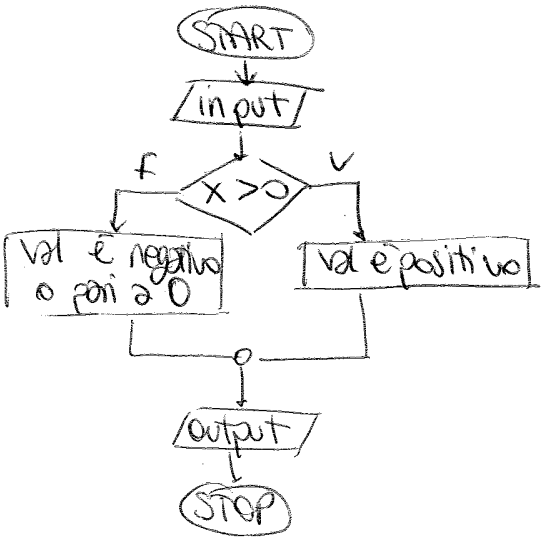
WATCHES → local variables
x = indirizzo cella della memoria RAM

* non inizializzando le variabili sono inizialmente = n° indirizzo
con debug si vedono i valori che assumono le variabili nel corso del programma.

ESERCIZIO 2



ESERCIZIO 3



© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 5 di 159

ESERCIZIO 6

```
#include <stdio.h>
Int main();
{ int a=0 , b=0;
Float x=0;
Printf("inserisci primo coefficiente");
Scanf("%d", &a);
Printf("inserisci secondo coefficiente");
Scanf("%d", &b);
if a=0 then printf("impossibile");
else x=-b/a;
printf("il valore dell'incognita è", x);
return 0;
}
```

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 7 di 159

Esercizio 3. Scrivere un programma che definisca 3 variabili reali (*float*) chiamate *length*, *width* e *perimeter*, corrispondenti a base, altezza e perimetro di 1 rettangolo:

a. Inizializzi le variabili *length* e *width* usando dei valori scelti dal programmatore
b. Partendo da questi dati, calcoli il perimetro del rettangolo e lo salvi nella variabile *perimeter*

Visualizzare il contenuto delle variabili mediante l'uso della opzione di debug: Debug→Debbuging windows →Watches

---

Da risolvere a casa

Esercizio 4. Scrivere un programma che definisca 2 variabili intere chiamate *var1*, *var2*. Il programma dovrà:

a. Assegnare i seguenti valori alle variabili definite:
   - *var1*← 25
   - *var2*← -53
b. Senza utilizzare una variabile di appoggio, scambiare i valori delle variabile *var1* e *var2*.

Suggerimento: è consigliabile cercare su internet quali metodi matematici permettono di scambiare due variabili senza l'aiuto di una terza

Visualizzare il contenuto delle variabili mediante l'uso della opzione di debug: Debug→Debbuging windows →Watches

Esercizio 5. Scrivere un programma che definisca 3 variabili reali (*float*) chiamate *price*, *tax* e *receipt*, e:

a. Assegnare valori scelti dal programmatore per *price* e *tax*
b. Partendo da questi dati, calcoli il prezzo comprensivo delle tasse (*price+price*tax/100*) e lo salvi nella variabile *receipt*

Visualizzare il contenuto delle variabili mediante l'uso della opzione di debug: Debug→Debbuging windows →Watches

Esercizio 6. Partendo dal flow-chart disegnato per l'esercizio 2 della settimana 1, si scriva un programma C per il calcolo del modulo (valore assoluto) di un numero; in particolare il programma dovrà:

a. Acquisire da tastiera un valore intero, positivo o negativo, e memorizzarlo in una variabile opportunamente definita
b. Stabilire utilizzando il costrutto condizionale *if* se tale variabile contiene un valore negativo e, in questo caso, trasformarlo nel corrispondente valore positivo
c. Stampare a video il valore finale, ovvero il modulo del valore acquisito

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 9 di 159

INFORMATICA, A.A. 2012/2013
Esercitazione di Laboratorio 3

Obiettivi
- Risolvere problemi gestendo input-output

Contenuti tecnici
- Uso di scanf e printf
- Uso della direttiva #define
- Uso base di espressioni aritmetiche
- Uso operatori relazionali
- Uso operatori logici

Da risolvere preferibilmente in laboratorio

Esercizio 1. Scrivere un programma che:

a. Definisca 2 variabili di tipo intero: int_1 e int_2
b. Definisca 2 variabili di tipo reale: float_1 e float_2.
c. Tramite la funzione scanf acquisisca da tastiera un valore reale ed uno intero.
d. Assegni alle 2 variabili reali il valore reale, ed alle due variabili intere il valore intero.
e. Visualizzi su schermo usando la funzione printf il valore assunto dalle 4 variabili con il seguente formato:
   i. *int_1* occupando almeno 5 spazi,
   ii. *int_2* occupando almeno 5 spazi e completando gli eventuali spazi liberi con zeri,
   iii. *float_1* occupando almeno 5 spazi e con una precisione di 2 posizioni dopo il punto decimale,
   iv. *float_2* occupando almeno 2 spazi e con una precisione di 3 posizioni dopo il punto decimale.

Esempio: valori acquisiti da tastiera 3 e 3.5

```
Variable  Value
int_1         3
int_2 00003
float_1   3.50
float_2 3.500
```

f. Si provi il programma con i seguenti valori: -3 e -3.5, 1000 e 1000.4567, 1 e 1.01

Approfondimento: si modifichi il programma in modo che acquisisca esclusivamente un valore reale da tastiera tramite la funzione scanf, e lo assegni a tutte le 4 variabili.

Esercizio 2. Definire e assegnare dei valori iniziali alle variabili intere A, B e C. Se eseguo la seguente istruzione:

C = (A==B)

ESERCIZIO 1

```
#include <stdio.h>
#include <stdlib.h>

int main()

{

  int int_1 = 0, int_2 = 0 ;
  float float_1 = 0, float_2 = 0;

 Printf("inserire valore intero e valore reale");
  scanf("%d %f", &int_1, &float_1 );

  // mettendo gli spazi scanf capisce che non deve considerare spazi e invii, in alternativa posso scrivere
  scanf("%*d".. che indica di saltare la lettura del comando invio

  int_2=int_1;
  float_2=float_1;
  printf("int_1: %5d\n", int_1);          // 5 spazi disponibili in tutto
  printf("int_2: %.5d\n", int_2);         // scrive numero preceduto da zeri che occupano tutti posti rimanenti
  printf("float_1: %5.2f\n", float_1);  // minimo di 5 spazi disponibile con 2 di questi riservati alla parte decimale
  printf("float_2: %2.3f\n", float_2);
   return 0;
}
```

ESERCIZIO 2

```
int main()

{
   int A, B, C;

   printf ("A= ");
   scanf("%d",&A);
   printf ("B= ");
   scanf("%d",&B);

   C=(A==B);
   printf("C=(A==B)= %d\n",C);

   C=(A!=B);
   printf("C=(A!=B)= %d\n",C);

   C=(A>=B);
   printf("C=(A>=B)= %d\n",C);

   C=(A<=B);
   printf("C=(A<=B)= %d\n",C);

   A=0;
   B=0;
   C=0;
   for(A=0; A<=1; A++)
```

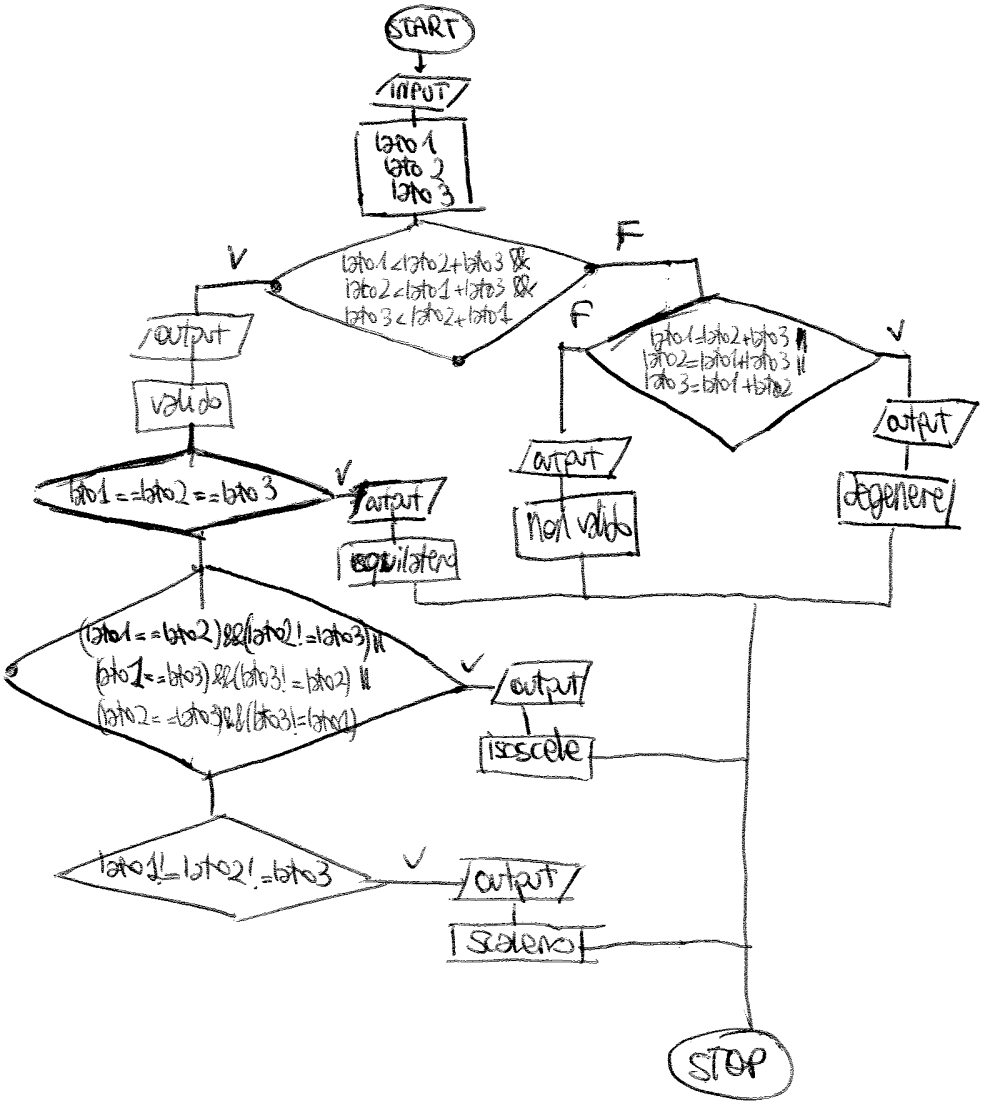
ESERCIZIO 6

```
int main()
{
   int a, somma=0, i=0;

   for (i=0; i<2; i++)
    {
       printf ("numero= ");
       scanf("%d", &a);
       somma+=a;
    }
   printf("la media : %f", somma/(float)i);

   return 0;
}
```

ESERCIZIO 5



12BHD INFORMATICA – 2012-2013

Esercitazione di Laboratorio 4

Obiettivi dell'esercitazione

- Risolvere problemi che implichino scelte logiche
- Sperimentare il concetto di iterazione

Contenuti tecnici

- Uso dei costrutti *if-then-else e switch*
- Uso dei costrutti iterativi *while, do-while* e *for*
- Introduzione all'uso degli operatori *cast* e *sizeof*

Da risolvere preferibilmente in laboratorio

Esercizio 1. Si scriva un programma in linguaggio C in grado di determinare se l'equazione di secondo grado ($ax^2 + bx + c = 0$) ha soluzioni reali. In particolare:

a. Si definiscano tre variabili chiamate *a, b* e *c*, corrispondenti ai parametri dell'equazione
b. Si acquisisca da tastiera il valore di *a, b* e *c*
c. Si calcoli il cosiddetto *discriminante* della formula risolutiva
    i. In caso il delta sia positivo, visualizzare il seguente messaggio "L'equazione ha due soluzione REALI distinte"
    ii. In caso il delta sia nullo, visualizzare il seguente messaggio "L'equazione ha due soluzione REALI coincidenti"
    iii. Altrimenti stampare a video un messaggio per segnalare che l'equazione non ha soluzioni reali

Esercizio 2. Si scriva un programma in linguaggio C che, dato un numero intero tra 1 e 12 che rappresenta il mese corrente, utilizzi il costrutto *switch* per stampare il nome del mese per esteso (1→"Gennaio", 2→"Febbraio", 3→"Marzo", …, 12 → "Dicembre").
Il programma gestisca anche le situazioni di inserimento di valori non compresi nell'intervallo 1-12.
Approfondimento: modificare il programma in modo che accetti come input una data nella forma gg/mese/anno (esempio: 23/3/2012) e stampi la stessa data con il mese per esteso (esempio: 23 marzo 2012). Si consiglia l'utilizzo del costrutto *switch*.

Esercizio 3. Si scriva un programma C che acquisisca numeri interi da tastiera finché non viene inserito il valore *0*.
Suggerimento: si utilizzi il costrutto iterativo *while* oppure *do-while*
Approfondimento: modificare il programma accumulando (ovvero continuando a sommare) in una variabile *i* valori inseriti prima dell'immissione del numero 0; al termine dell'acquisizione il programma stampi a video il valore calcolato.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 17 di 159

c) Realizzare a questo punto un algoritmo che, tenendo conto delle rappresentazioni binarie dei numeri senza segno e in complemento a 2, permetta di rilevare il valore max: per i numeri con segno, si può attribuire a *value* il valore iniziale di 0, poi si incrementa ripetutamente *value*. È noto che se si incrementa di 1 il valore massimo positivo, si ottiene overflow e il valore diventa negativo. Il valore cercato è dunque il precedente al primo valore negativo trovato. Tradurre l'algoritmo in programma e collaudarlo. Come si può modificare l'algoritmo (e il programma) perché operi con i numeri senza segno?

```
      case 7:
      printf("LUGLIO");
      break;
      case 8:
      printf("AGOSTO");
      break;
      case 9:
      printf("SETTEMBRE");
      break;
      case 10:
      printf("OTTOBRE");
      break;
      case 11:
      printf("NOVEMBRE");
      break;
      case 12:
      printf("DICEMBRE");
      break;
      default:
      printf("INSERISCI UN VALORE TRA 1 E 12");
      break;
   }
  return 0;
 }
```

APPROFONDIMENTO 2

```
int main()
{
     int MESE=0,GIORNO=0,ANNO=0;

   printf("INSERISCI DATA NELLA FORMA GG_MESE_ANNO\n");
   scanf("%d %d %d",&GIORNO, &MESE, &ANNO);

   switch(MESE)
   {
      case 1:
      printf("%d GENNAIO %d",GIORNO, ANNO);
      break;
      case 2:
      printf("%d FEBBRAIO %d",GIORNO, ANNO);
      break;
      case 3:
      printf("%d MARZO %d",GIORNO, ANNO);
      break;
      case 4:
      printf("%d APRILE %d",GIORNO, ANNO);
      break;
      case 5:
      printf("%d MAGGIO %d",GIORNO, ANNO);
      break;
      case 6:
      printf("%d GIUGNO %d",GIORNO, ANNO);
      break;
      case 7:
      printf("%d LUGLIO %d",GIORNO, ANNO);
      break;
      case 8:
```

```
   do
   {
      printf("INSERISCI NUMERO INTERO MINORE DI 40\n");
      scanf("%d",&DATO);
   }
   while (DATO>NMAX);

   for(I=0;I<=DATO;I++)
   {
      for(J=0;J<I;J++)
         printf("*");
      printf("\n");
   }
   return 0;
}
```

ESERCIZIO 5

```
int main()
{
   int X1=1, X2=0, XI,I;

   printf("0 1 ");
   for(I=2;I<20;I++)
   {
     XI=(X1)+(X2);
     printf("%d ", XI);
     X2=X1;
     X1=XI;

   }
   return 0;
}
```

ESERCIZIO 6

```
int main()
{
   float N;
   int X,I=0,SOMMA=0,MEDIA=0;

   printf("SCRIVI UN NUMERO REALE\n");
   scanf("%f",&N);

   do
   {
      printf("INSERISCI NUMERO INTERO\n");
      scanf("%d",&X);
      I++;
      SOMMA=SOMMA+X;
      MEDIA=SOMMA/I;

   }
   while ((I<10)&&(MEDIA<N));
   printf("MEDIA %d\n", MEDIA);

   return 0;
}
```

Esercizio 3. Scrivere un programma C che definisca due vettori *v1* e *v2* di N elementi di tipo intero e memorizzi nei vettori valori "accettabili" acquisiti da tastiera secondo quanto segue:

a. In *v1* siano memorizzati solo i valori positivi ed i valori negativi multipli di 3
b. In *v2* siano memorizzati solo i valori negativi non multipli di 3 e dispari
c. Tutti gli altri valori acquisiti siano ignorati
d. L'inserimento si conclude quando uno dei due vettori è pieno; a questo punto si stampi a video il contenuto dei vettori acquisiti.

Da risolvere a casa

Esercizio 4. Scrivere un programma C che acquisisca un massimo di *N* valori interi, con *N* costante definita a piacimento. L'acquisizione deve procedere finché la serie di numeri è monotona, ovvero costituita da numeri in ordine crescente o decrescente. Stampare il contenuto del vettore al termine dell'acquisizione

Esempi:
(N=10)

1 4 6 10 4 ← l'inserimento del valore 4 termina le iterazioni
9 7 6 7 ← l'inserimento del valore 7 termina le iterazioni
1 2 3 4 5 6 7 8 9 10 ← ho acquisito 10 numeri quindi mi fermo

Suggerimento: scrivere innanzitutto una versione semplificata scegliendo una singola direzione di monotonia (o crescente o decrescente), quindi passare alla soluzione completa.

Esercizio 5. Scrivere un programma C che scandisca un vettore di *N* valori interi, e determini se esiste una serie crescente di tre numeri consecutivi. In caso positivo, il programma deve stampare la serie di numeri e la posizione del primo valore.

ORDINA VETTORI

```
for(i=0; i<N;i++){
  {
    for(j=0; j<N ; j++)
    {
      if (v[i] > v[j])
        {
          tmp = v[i];
          v[i]=v[j]
          v[j]= tmp;
        }
```

ORDINE : CRESCENTE

```
  MAX=V[0];
  INDICE=0;
  for(I=1;I<N;I++)
  {
    if (MAX<V[I])
    {
      MAX=V[I];
      INDICE=I;
    }
  }

  printf("IL MASSIMO E' %d\n", MAX);
  printf("L'INDICE E' %d\n", INDICE);

  return 0;
}
```

APPROFONDIMENTO 2 :

```
#include <stdio.h>
#include <stdlib.h>
#define N 3

int main()
{
  int V[N], I, MAX, INDMAX[N]={0};


  for (I=0;I<N;I++)
  {
    printf("INSERISCI UN VALORE INTERO POSITIVO\n");
    scanf("%d", &V[I]);
  }

  for (I=0;I<N;I++)
  {
    printf("%d ", V[I]);
  }
  printf("\n");

  MAX=V[0];
  for(I=1;I<N;I++)
  {
    if (MAX<V[I])
      MAX=V[I];
  }

  printf("IL MASSIMO : %d\n", MAX);
  printf("IN POSIZIONE : \n");
```

```
  return 0;
}
```

ESERCIZIO 4

```
#include <stdio.h>
#include <stdlib.h>
#define N 5

int main()
{
  int V[N]={0},I=0,VALORE,MINORE,MAGGIORE,CRESCENTE=1,DECRESCENTE=1;

  printf("INSERISCI VALORE INTERO\n");
  scanf("%d", &V[0]);

  MINORE=V[0];
  MAGGIORE=V[0];
  I=1;

  do
  {
    printf("INSERISCI VALORE INTERO\n");
    scanf("%d", &VALORE);
    if (VALORE>MINORE)
    {
      V[I]=VALORE;
      MINORE=V[I];
      DECRESCENTE=0;
      I++;
    }
    else
    {
      if (VALORE<MAGGIORE)
      {
        V[I]=VALORE;
        MAGGIORE=V[I];
        CRESCENTE=0;
        I++;
      }

      else
      {
        I=N;
      }

  }
  while ((I<N)||((CRESCENTE=0)&&(DECRESCENTE=0)));

  for (I=0;I<N;I++)
  {
    printf("%d",V[I]);
  }
```

12BHD INFORMATICA, A.A. 2012/2013
Esercitazione di Laboratorio 6

Obiettivi dell'esercitazione

- Elaborare e manipolare il contenuto di un vettore precedentemente acquisito
- Scrivere programmi che includano semplici funzioni

Contenuti tecnici

- Uso avanzato dei vettori
- Uso dei cicli annidati per l'analisi dei vettori
- Uso preliminare di funzioni di calcolo con parametri passati *by value*

Da risolvere preferibilmente in laboratorio

Esercizio 1. Si scriva un programma C che:

a. legga un vettore di N elementi interi (con N costante predefinita)
b. determini se gli elementi di tale vettore costituiscono una successione palindroma.

Suggerimento: una successione si dice palindroma se e' identica letta da sinistra verso destra o da destra verso sinistra.
Esempio: le seguenti successioni di valori sono palindrome:

```
12  3  12
1  4  5  4  1
10 10 10
```

mentre la seguente non è palindroma:

```
1 3 4 3 2
```

Esercizio 2. Si scriva un programma C che:

a. legga 2 vettori di N elementi interi (con N costante predefinita)
b. stabilisca se i due vettori contengono gli stessi elementi, anche disposti in ordine differente

Esempio: siano dati i due vettori seguenti:

```
v1 → 15  3  12  13  29
v2 → 15  29  13  3  12
```

questi contengono gli stessi valori, anche se in posizioni differenti.
Invece, i due vettori seguenti:

```
v1 → 11  3  12  18  29
v2 → 12  29  13  4  12
```

non contengono gli stessi valori.
Approfondimento: considerare la possibilità che ci siano valori ripetuti tra quelli memorizzati nei vettori. Ad esempio

```
v1 → 12  3  12  13  29
v2 → 12  29  13  3  12
```

contengono gli stessi valori ed il 12 compare 2 volte per vettore.
Invece, i due vettori seguenti:

```
v1 → 12  3  13  13  29
v2 → 12  29  13  3  12
```

non contengono gli stessi valori.

```
#include <stdio.h>
#include <stdlib.h>
```

ESERCIZIO 1

```
#define N 5

int main()

{
   int V[N]={0},I,PALINDROMA=1;

   for (I=0;I<N;I++)
   {
      printf("INSERISCI UN VALORE INTERO\n");
      scanf("%d", &V[I]);
   }

   for (I=0;I<N;I++)
   {
      if(V[I]!= V[N-I-1])
      {
         PALINDROMA=0;
      }
   }

   if (PALINDROMA==0)
   {
      printf("NON PALINDROMA\n");
   }
   if (PALINDROMA==1)
   {
      printf("PALINDROMA");
   }
   return 0;
}
```

ESERCIZIO 2

```
#define N 3

int main()
{
   int V1[N]={0},V2[N]={0},I,J,TROVATI=0;

   for (I=0;I<N;I++)
   {
      printf("INSERISCI UN VALORE INTERO\n");
      scanf("%d", &V1[I]);
   }
   printf("\n");
```

```
  for (I=0;I<N;I++)
  {
    for(J=0;J<N;J++)
    {
      if ((V1[I]==V2[J])&&(V1_FLAG[I]==0)&&(V2_FLAG[I]==0));
      {
        TROVATI++;
        V1_FLAG[I]=1;
        V2_FLAG[J]=1;
      }
    }
  }

  if (TROVATI==N)
  {
    printf("UGUALI\n");
  }
  else
  {
    printf("DIVERSI");
  }
  return 0;
}
```

ESERCIZIO 3

tipo del return. ~~> void se non ritorna niente

```
int POWER (int BASE, int EXPONENT);                    /*PROTOTIPO*/

int main()                                             /*UTILIZZO*/
{
  int B, E, RISULTATO;

  printf("INSERISCI VALORE PER LA BASE\n");
  scanf("%d", &B);

  printf("INSERISCI VALORE PER L'ESPONENTE\n");
  scanf("%d", &E);

  RISULTATO=POWER(B,E);

  printf("RISULTATO: %d", RISULTATO);

  return 0;
}

int POWER(int PRIMO_VALORE, int SECONDO_VALORE)        /*DEFINIZIONE*/
{
  int RISULTATO=1, I;
```

12BHD INFORMATICA, A.A. 2012/2013
Esercitazione di Laboratorio 7

Obiettivi dell'esercitazione

- Scrivere programmi che utilizzino caratteri e stringhe

Contenuti tecnici

- Uso avanzato delle funzioni e dei vettori
- Uso del tipo *char*
- Uso delle funzionalità contenute in *ctype.h* e *math.h*

Da risolvere preferibilmente in laboratorio

Esercizio 1. Si scriva un programma C che, dati due vettori di uguale dimensione N (`vbase` e `vexponent`), elevi ciascun elemento del vettore `vbase` alla potenza indicata nell'elemento di `vexponent` avente lo stesso indice (ossia `vbase[i]` elevato a `vexponent[i]`). I risultati dovranno essere memorizzati nella corrispondente posizione di un terzo vettore denominato `vris`. Si utilizzi la funzione *power* definita nel corso del precedente laboratorio e avente il seguente prototipo:

```
int power(int base, int exponent);
```

Vengano inseriti prima i valori delle N basi e poi quelli degli N esponenti; vengano alla fine visualizzati i valori di `vris`.

Esempio

Siano inseriti dall'utente i valori seguenti (per N pari a 5):

```
vbase      → 5 2 7 4 9
vexponent  → 2 6 1 8 3
```

Il vettore risultato sarà il seguente:

```
vris       → 25 64 7 65536 729
```

Suggerimento: richiamare tante volte la funzione *power* quanti sono gli elementi dei vettori e ogni volta salvare il contenuto in una posizione opportuna di `vris`.

Esercizio 2. Si scriva un programma C che

a. **nel main** chieda all'utente di inserire N valori e li metta in un vettore `vett`, quindi chieda un ulteriore valore `x`
b. **passi** sia il vettore sia `x` **ad una funzione** che moltiplichi ciascuno degli elementi del vettore per `x` e il cui prototipo sia

```
void mult(int v[], int n, int x);
```

Il **main** poi visualizzi il vettore dopo la moltiplicazione.

Nota bene: la funzione riceve il vettore per riferimento e quindi può modificare i valori stessi del vettore.

Esercizio 3. Si scriva un programma C che acquisisca caratteri da tastiera fino alla ricezione di un "a capo". Dopo tale evento il programma deve fornire all'utente le seguenti statistiche:

a. il numero di caratteri introdotti;
b. il numero di caratteri alfabetici;
c. il numero di caratteri maiuscoli;
d. il numero di cifre decimali;

INFORMATICA
Esercitazione di Laboratorio 8

Obiettivi dell'esercitazione

- Scrivere programmi che leggano e manipolino caratteri e stringhe

Contenuti tecnici

- Uso dell'operatore di conversione *%s*
- Uso delle funzionalità contenute in *string.h* e *ctype.h*
- Uso delle funzioni per la formattazione delle stringhe

Da risolvere preferibilmente in laboratorio

Esercizio 1. Si scriva un programma che:

a. Definisca un vettore di caratteri e acquisisca una stringa al suo interno
b. Analizzi tale stringa rispondendo alle seguenti domande
   i. Quanto è lunga la stringa?
   ii. Quanti caratteri sono alfabetici e quanti numerici?

Approfondimento: acquisita una seconda stringa, stabilire se quest'ultima è inclusa nella prima (ad esempio: "importante" include "porta")

Esercizio 2. Si scriva un programma C che:

a. Acquisisca una stringa di massimo N caratteri (con N valore costante)
b. Ne manipoli il contenuto
   i. Trasformando tutte le lettere minuscole in maiuscole
   ii. Rimpiazzando tutti i caratteri non alfanumerici con il carattere '_'
   iii. Sostituendo i caratteri numerici con il carattere '*'
c. Scandisca la stringa manipolata per contare quante parole sono presenti al suo interno, considerando una o più occorrenze del carattere '_' come separatore tra parole.

Approfondimento: l'ordine in cui vengono eseguite le manipolazioni influenza il risultato? Verificare la risposta scrivendo due versioni del programma che manipolino la stringa in modi differenti.

Esercizio 3. Si scriva un programma che acquisisca 2 stringhe corrispondenti a 2 orari nel formato *hh:mm*. Il programma deve:

a. Controllare le stringhe, segnalando i casi in cui il formato non sia rispettato (ad esempio 10,30 non è valido)
b. Stabilire se l'orario contenuto nella prima stringa è precedente a quello contenuto nella seconda stringa
c. In caso affermativo, tradurre i 2 orari in valori interi corrispondenti all'orario espresso come distanza in minuti da 00:00 e calcolarne la differenza
d. Convertire il risultato (sarà un numero intero positivo) in una stringa così composta "<intervallo calcolato>_minuti" e la stampi a video.

andrea.acquaviva@polito.it
edoardo.patti@polito.it

DOMANDE ESAME

1. conversione
2. tabelle verità → (tabelle, proprietà associativa - commutativa - distributiva, de Morgan)
3. architettura calcolatore

Server Farm → insieme di elaboratori server collocati in un apposito locale (centro di calcolo) presso una media o grande azienda.

Supercomputer (2012) → potenza (16 PFlops (= Peta Flops))
IBM BlueGene/Q
Livermore Labs

floating Point operation per second → FLOPS
(operazioni con numeri reali al secondo)

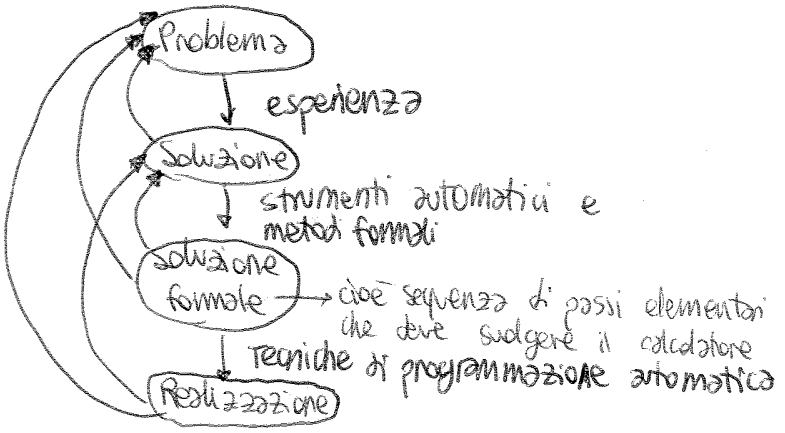
K M G T
kilo mega giga Tera
2^{10}

PROGRAMMA

Dalla specifica di un problema alla sua realizzazione come programma da eseguire su un elaboratore

Sequenza di operazioni/istruzioni da svolgere.

PROGETTARE PROGRAMMA



Difficoltà:
i punti critici sono: • sviluppo soluzione informale
soluzione del problema passa • formalizzazione della soluzione attraverso lo sviluppo di un algoritmo.

ALGORITMO: sequenza passaggi logici per risolvere un problema che il calcolatore può eseguire
(algoritmo ≠ programma)
Descrizione precisa (formale) di una sequenza finita d'azioni che devono essere eseguite per giungere alla soluzione finale del problema.

RIASSUNTO
- rappresentazione dati → digitale (ne garantisce robustezza e compattezza informazione)
- sequenza d'istruzioni → programma
- programma: passi di una soluzione di un problema
  → algoritmo

- PROBLEMA
- IDEA → soluzione
- ALGORITMO → soluzione formale
- PROGRAMMA → traduzione algoritmo in linguaggio comprensibile dall'elaboratore
- TEST
- DOCUMENTAZIONE → commenti nel programma per spiegare le funzioni delle varie parti

RAPPRESENTAZIONE SOLUZIONE

- differenza tra soluzione informale (parole) e formale (sequenza operazioni elementari) sta nel modo di rappresentare un algoritmo

① pseudo codice = linguaggio simile a quello di programmazione
② diagramma di flusso

} strumenti per rappresentare una soluzione in modo formale

PSEUDO CODICE

VANTAGGI
- immediato

SVANTAGGI
- descrizione poco astratta algoritmo
- interpretazione + complicata

DIAGRAMMA DI FLUSSO

VANTAGGI
- + intuitivi perché utilizzano formalismo grafico
- descrizione dell'algoritmo + astratta

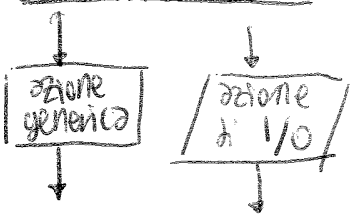
SVANTAGGI
- Richiedono apprendimento della funzione dei vari tipi di blocco

DIAGRAMMI DI FLUSSO (FLOW-CHART)

- Strumenti grafici che rappresentano l'evoluzione logica della risoluzione del problema
- Composti da
  - BLOCCHI ELEMENTARI → descrivono azioni e decisioni (solo tipo binario)
  - ARCHI ORIENTATI → per collegare blocchi e descrivere sequenza di svolgimento azioni

- **BLOCCHI INIZIO/FINE**

- **BLOCCHI DI AZIONE**



azione generica → es. max=0
azione input/output
- dati inseriti da utente
- calcolatore da all'utente soluzione del problema

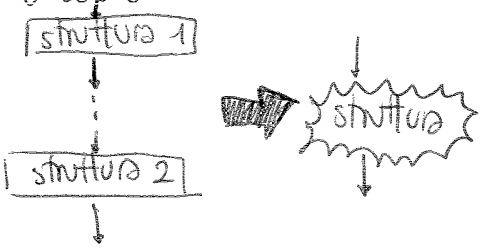
- **BLOCCHI DI DECISIONE**

STRUTTURE ELEMENTARI

Diagramma strutturato se contiene queste strutture elementari

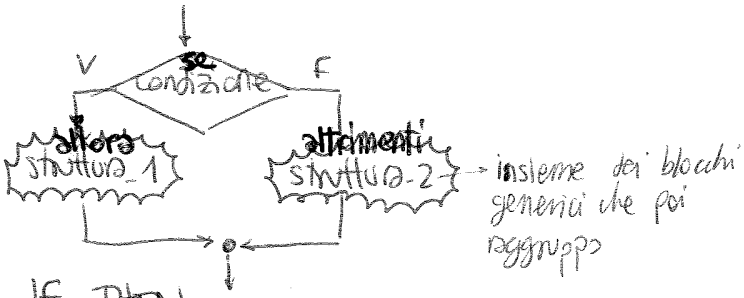
- un solo BLOCCO START
- un solo BLOCCO STOP
- sequenza di BLOCCHI AZIONE e INPUT/OUTPUT
- IF-THEN-(ELSE)
- WHILE-DO
- REPEAT-UNTIL

SEQUENZA

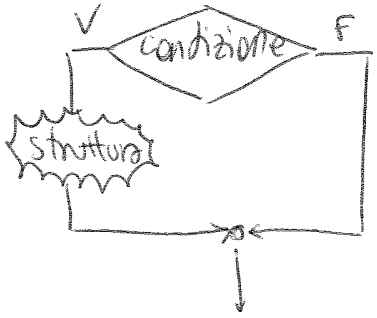


IF-THEN-ELSE

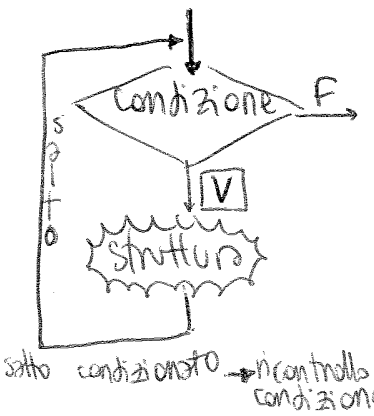
se - allora - altrimenti



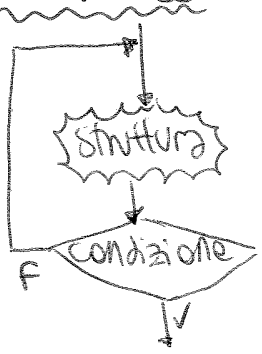
IF-THEN



WHILE-DO



DO-WHILE



* DO-WHILE → struttura si esegue almeno 1 volta
* WHILE-DO → struttura si esegue minimo numero di volte = 0

COMPILATORE C

- traduce programmi SORGENTI scritti in linguaggio C in programmi ESEGUIBILI
- programma eseguibile è a disposizione del programmatore
- controlla assenza di ERRORI DI SINTASSI del linguaggio
  - semantica ≠ sintassi!
- non serve all'utente finale
- ce ne sono ≠, sia gratuiti che commerciali

AMBIENTI INTEGRATI

- Applicazioni software integrate che contengono al loro interno
  - editor di testi per programmatori
  - compilatore C
  - ambiente di verifica dei programmi (debugger)
- IDE: Integrated Development Environment

PROGRAMMARE

- scrittura di un documento=file sorgente che descrive la soluzione del problema in oggetto.
- Non esistono soluzioni analitiche o universali
- È un'operazione complessa organizzata in stadi successivi

STADI DI SVILUPPO DI UN PROGRAMMA

- costruzione di un programma è operazione iterativa
- previsti passi a ritroso che rappresentano le reazioni a risultati non rispondenti alle esigenze nelle diverse fasi
- suddivisione in più fasi (- dispendiosi) permette di mantenere i passi a ritroso più brevi possibile
- È necessario effettuare dei test tra una fase e la successiva affinché i ricicli siano più corti possibili
- Una volta scritto e collaudato il programma, possono verificarsi le seguenti condizioni
  1. programma è stato scritto non correttamente → indietro di 1 livello
  2. programma descritto male in termini formali ma corretto concettualmente → indietro di 2 livelli
  3. programma errato concettualmente, necessita di una soluzione ≠ → torno all'inizio

EDITOR PER PROGRAMMATORI

- Colorazione ed evidenziazione della sintassi
- Indentazione automatica
- Attivazione automatica della compilazione
- Identificazione delle parentesi corrispondenti
- Molti disponibili, sia gratuiti che commerciali

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 49 di 159

IDENTIFICATORI

- indica il nome di un dato (e di altre entità) in un programma
- permette di dare nomi intuitivi ai dati
- nome unico all'interno di un preciso "ambiente di visibilità"

Dati ≠ ⊜ Nomi ≠

TIPO

- interpretazione dei dati in memoria
- legato a spazio occupato da un dato (es: intero, decimale, ...)
- permette di definire tipi "primitivi" (numeri, simboli), indipendentemente dal tipo di memorizzazione del sistema

TIPO DI ACCESSO

modalità di accesso ai dati

① variabili
- dati modificabili
- valore modificabile in un punto qualsiasi del programma

② costanti
- dati a sola lettura
- valore assegnato una volta per tutte

ESEMPIO

① chiedere dati → stampa messaggio: 'inserisci cateto'
② associa dato all'identificatore → leggi cateto1 (es. cateto = ③) → numero letto
③ stampa 'inserisci altro cateto'
④ leggi cateto2 → INPUT
⑤ ipotenusa ← $\sqrt{cateto1^2 + cateto_2^2}$ ← ASSEGNAZIONE
⑥ stampa valore ipotenusa ← OUTPUT

ISTRUZIONI

Istruzioni le operazioni che il linguaggio permette di eseguire (traducendole) a livello macchina

- Pseudo-istruzioni
  - ↳ direttive non eseguibili
- Istruzioni elementari
  - ↳ operazioni direttamente corrispondenti ad operazioni hardware
  - esempio: interazione con i dispositivi I/O, modifica/accesso a dati
- Istruzioni di controllo del flusso
  - ↳ permettono di eseguire delle combinazioni di operazioni complesse

LINGUAGGIO DI PROGRAMMAZIONE

- conoscere → parole chiave, tipi predefiniti, istruzioni e la loro sintassi
- l'estensione ad altri linguaggi è immediata

ARCHITETTURA DEGLI ELABORATORI

BLOCCHI FONDAMENTALI ELABORATORE

unità ingresso (tastiera) → memoria (computer) → unità uscita (monitor video)

unità elaborazione → unità uscita

* chip fondamentali

Memoria

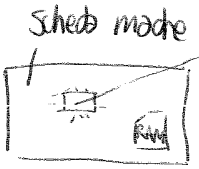
→ MEMORIA CENTRALE [INTERNA] RAM* (8-16 GB)

random access memory

- ruolo di processore
- nella scheda madre
- + veloce
- - capacità
- volatile: contenuto cancellato senza corrente elettrica perché formato da transistori

accesso casuale → distinto da accesso sequenziale

cioè posso accedere a qualsiasi cella di memoria in un determinato tempo senza dover accedere prima ad altre celle di memoria → efficiente



(1-2 T) HDD → hard disk

CD/DVD → memoria massa

MEMORIA DI MASSA (...1-2 T) (dischi) [ESTERNA]

- capacità > RAM
- + lenta
- + lontana processore
- non nella scheda madre
- non volatile perché formate da elementi ottici, magnetici o transistori modificati (USB) cioè tecnologie diverse consentono di conservare i dati anche senza corrente.

CAPACITÀ ↑	- VELOCITÀ ACCESSO ↓

Byte → 8 bit → bit = cifra binaria

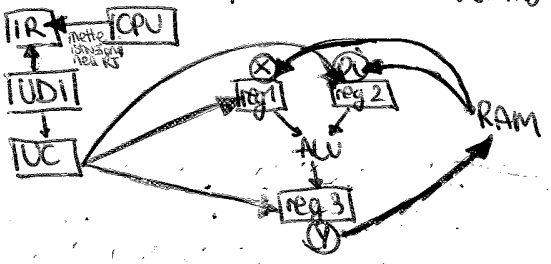
Microprocessore* (MP) → visto uso intensivo, necessità di tecnologie di packing diverse per disperdere calore.

chip che realizza le funzioni di un CPU → unit (unità centrale di elaborazione) central processing

unità integrate in un unico chip — più funzioni tutte contenute in un unico chip per rendere + rapido scambio informazioni

in un computer o in un sistema digitale.

CPU



Registri (memoria locale) ↔ unità aritmetica (ALU)

↕ unità di controllo

PROGRAM COUNTER indica istruzione successiva da eseguire

INSTRUCTION REGISTER contiene istruzione da eseguire, inviata in base al comando del CU

- **REGISTRO** → contengono informazioni elaborate da ALU → elementi di memoria locale usati per conservare temporaneamente dati (risultati parziali)
  - piccole memorie (ordine 4-8 Byte) → dimensione di 1 reg (8...64 bit)
  - pochi (8...128)
- **ALU** (Aritmetic Logic Unit) → legge dati dai registri e salva risultati su altri registri
  - svolge tutti i calcoli aritmetici (+, -, *, /) e logici (>, <, and, or, not → logica binaria)
  - composta da circuiti combinatori

① BUS DATI → trasmette dati tra memoria RAM e CPU

② BUS INDIRIZZI → indica numero che corrisponde alla cella di memoria dalla quale va letto il dato
Arriva prima bus indirizzo alla RAM, poi arriva bus dati che sa già quale è la cella di memoria dalla quale va prelevato il dato
La dimensione del bus indirizzo dipende dalle dimensioni della memoria, cioè dal numero massimo di celle della memoria

2^x indirizzi = 4GB

$x = \log_2 4GB = 32$.

quindi con 4GB di memoria ho 2^{32} indirizzi

MASSIMA MEMORIA INTERNA

- dimensione ABUS ↔ max n° celle memoria indirizzabili
- dimensione DBUS ↔ dimensione di una cella di memoria → posso trasferire in un'unica volta il contenuto della cella di memoria quindi non ha senso avere RAM di dimensioni maggiori di quelle dei DBUS, perché i bus dovrebbero fare + "giri" per trasmettere info presenti in una cella di memoria

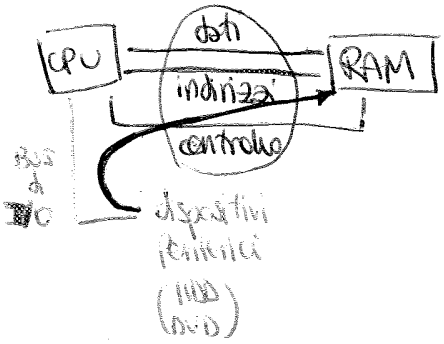
MAX MEMORIA = $2^{|ABUS|} \times |DBus|$ bit

esempio (ABus = 20 bit, DBus = 16 bit)
max mem = $2^{20} \times 2$ byte = 2 MB
ossia 1M celle di memoria, ognuna da 2 byte

③ BUS CONTROLLO → specifica ulteriore sul destinatario della comunicazione
→ mi dice tra tutte le memorie presenti, quali andare a prendere nella motherboard

Nella scheda madre c'è una memoria non volatile, il Bios, che contiene le informazioni necessarie per l'avvio della CPU.
Tutto quello che è contenuto nell'hard disk non può essere utilizzato dalla CPU deve prima essere copiato sulla RAM; questo invece è quello che avviene per il BIOS(Rom).
ROM (Read Only Memory), costruita con tecnologia FLASH

↓
una memoria sulla quale non si scrivono dati, ma si leggono solo le prime operazioni di base che deve eseguire il CPU appena acceso.



Bus di controllo
ha anche una linea che se
= 1 allora è coinvolta la periferica
= 0 se non sono coinvolti dispositivi periferici, ma la trasmissione dati avviene solo a livello della scheda madre

MASSIMA MEMORIA ESTERNA

→ non dipende da ABUS perché vista come dispositivo periferico (di input o di output)
→ max memoria esterna dipende dal bus I/O, al quale sono collegati i periferici

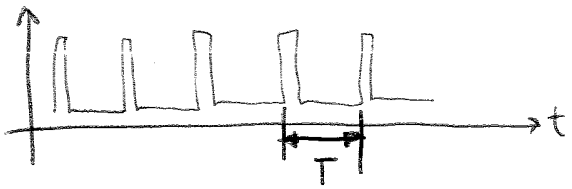
IL CLOCK

Generatore di riferimento temporale comune per tutti gli elementi costituenti il sistema di elaborazione (soprattutto CPU).
Le attività devono essere tutte sincronizzate

T = PERIODO DI clock (s) = distanza tra un evento ed un altro.

F = FREQUENZA DI clock ($s^{-1} = Hz$) = cicli al secondo = $1/T$

Più corto è il periodo di clock, maggiore è la frequenza con cui i segnali di attivazione vengono inviati al C.U.



(in un Intel Core i7-2700 f = 3,56 Hz)

TEMPISTICA ISTRUZIONI

- un ciclo-macchina è l'intervallo di tempo in cui viene svolta un'operazione elementare ed è un multiplo intero del periodo del clock.
- l'esecuzione di un'istruzione richiede un numero intero di cicli macchina, variabile a seconda del tipo di istruzione.

MIPS (=milioni di istruzione per secondo)

* BRANCH = salto < condizionato / incondizionato

PAROLE CHIAVE → riservate

Sono 32

auto	double	long	switch
break	else	register	typedef
case	enum	return	union
char	extern	short	unsigned
const	float	signed	void
continue	for	sizeof	volatile
default	go to	static	while
do	int	struct	
if			

STRUTTURA DI UN PROGRAMMA C

PARTE DICHIARATIVA GLOBALE (dichiarazione variabili) → elenco oggetti che compongono programma e specifica delle loro caratteristiche
- categoria oggetti → tipicamente dati
- tipo degli oggetti → numerici, non numerici

```
main()
{ parte dichiarativa locale
  parte esecutiva
}
```

main() → parola chiave che indica punto di inizio del programma quando viene eseguito dal sistema operativo. Contenuto è delimitato da parentesi graffe {...}

parte dichiarativa locale → elenco oggetti che compongono il main e specifica delle loro caratteristiche

parte esecutiva → sequenza di istruzioni descritte nel diagramma di flusso

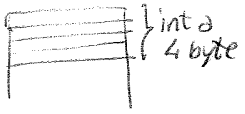
Esempio

```
#include <stdio.h>
main()
{
  int a;
  float b;

  printf("inserire numero intero(A):");
  scanf("%d", &a);
  printf("inserire numero reale(B):");
  scanf("%f", &b) != 1)
  printf("A = %d\n", a);
  printf("B = %f\n", b);
}
```

`#include <stdio.h>` → libreria in cui sto printf

`int a; float b;` → dichiarazione: serve per riservare spazio in memoria per le variabili →



VARIABILI

locazioni di memoria destinate alla memorizzazione di dati il cui valore è modificabile

<tipo> <variabile>;

identificatore che indica il nome della variabile

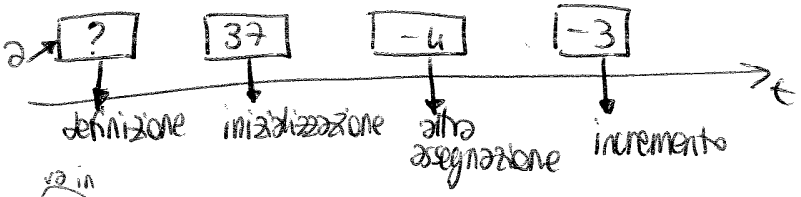
<tipo> <lista variabili> → per dichiarazioni multiple
lista di identificatori separati da virgole.

Esempio

```
int i, j;
int N;
int x;
i=0;
j=2;
N=100;
x=-3124;
```

i 0
j 2
N 100
x -3124

Ogni variabile in ogni istante possiede un certo valore.
le variabili appena definite hanno un valore ignoto (non inizializzate), ma in momenti diversi il valore può cambiare.



va in
a=0 → assegnazione
non viceversa

- while (1) {a=a+1}
  1=V
  0=F
  while vero = sempre incremento

- while (a==0) {a=a+1; 1=0+1}
  al secondo giro a=1 quindi non riesegue il while, lo esegue solo la prima volta

- while (a<10) {a=a+1}
  a
  CONTATORE
  sempre inizializzato

STRUTTURA A BLOCCHI

Si può raccogliere istruzioni in blocchi racchiudendole tra graffe, per delimitare un ambiente di visibilità di 'oggetti' (variabili, costanti) ed è corrispondente ad una sequenza di istruzioni.

Esempio

```
{
  int a=2;
  int b;
  b=2*a;
}
```

→ a e b sono definite solo all'interno del blocco

CONVERSIONE DECIMALE-BINARIO

Divisioni successive per 2
Si prendono i resti in ordine inverso

13 : 2	
6	1
3	0
1	1
0	1

$13_{10} = 1101_2$

TERMINOLOGIA

Bit = cifra nella numerazione binaria

Byte = 8 bit

Word = quantità di bit che calcolatore utilizza per comunicare con la memoria (stesso calcolatori sono a 64 bit = 8 byte)

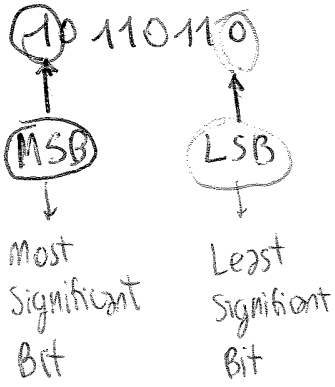
è una potenza di 2 dei Byte : 1, 2, 4, 8, 16, ...

Per utilizzo efficiente word = n° bit istruzioni dei programmi

(es. calcolatore a 64 bit esegue istruzioni trasmettendo 64 bit per volta)

Double word/long word = è possibile trasmettere n° bit > word eseguendo più transazioni/comunicazioni con la memoria

(es. per informazione = 128 bit → 2 comunicazioni con word a 64 bit)



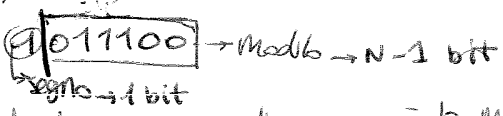
Bit più a destra hanno peso minore, cioè hanno SIGNIFICATIVITÀ minore.

OVERFLOW

errore che si verifica in un sistema di calcolo automatico quando risultato di un'operazione non è rappresentabile con la medesima codifica e numero di bit degli operandi.
Nella somma in binario l'overflow si ha quando si lavora con numero fisso di bit o quando si ha carry su MSB.

NUMERI CON SEGNO

Dato che segno può essere solo di 2 tipi, posso usare il primo bit per indicare se numero è positivo = 1
negativo = 0



È facile rappresentarlo in binario, ma la soluzione + semplice non è la migliore

a. modulo e segno
b. complemento a uno
c. complemento a due
d. eccesso X

a) CODIFICA MODULO e SEGNO

Svantaggi

① doppio zero +0, -0
② operazioni complesse } scarsa efficienza operazioni → complessità dei circuiti che lo realizzano
es. somma A+B

	A > 0	A < 0
A>0	A+B	B-|A|
B<0	A-|B|	-(|A|+|B|)

ogni volta che si deve fare un'operazione, prima è necessario fare dei controlli e questo richiede tempo e complessità di circuiti.

La rappresentazione modulo e segno su N bit:

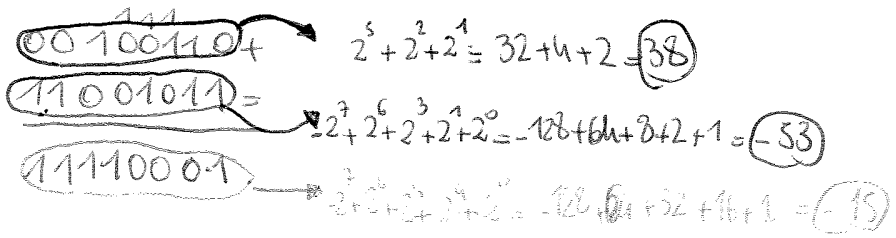
$$-(2^{N-1}-1) \leq x \leq +(2^{N-1}-1)$$

8 bit → [-127 ... +127]
16 bit → [-32767 ... +32767]

Somma e sottrazione in CA2 si effettuano direttamente senza badare ai segni degli operandi.

SOMMA

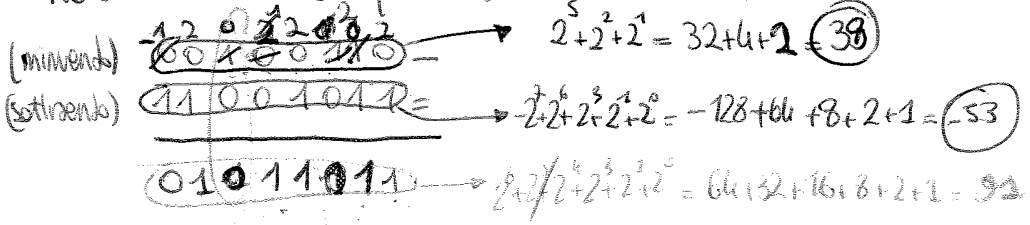
00100110 + 11001011



verifica = 38 − 53 = −15 ✓

SOTTRAZIONE

- Si può fare sommando al minuendo il CA2 del sottraendo.
- Nella sottrazione non importa segno



verifica = 38 − (−53) = 91

oppure:

CA2 sottraendo si ottiene negando sottraendo:

sottraendo: 11001011

CA2: 00110100 + 1 } 00110101

minuendo +
CA2 sottraendo =
00100110 +
00110100
01011011

> 1. sottrazione binaria
> = 2. minuendo + CA2 sottraendo
>
> CA2 sottraendo = $\overline{\text{sottraendo}} + 1$
> cioè nego sottraendo e aggiungo 1

38 − (−53) = 91 (=)

38 + CA2 = risultato differenza = 01011011 = 64+16+8+4+2+1 = 91

OVERFLOW

SOMMA → operandi con segno discorde → NO OVERFLOW

→ operandi con segno concorde → possibile overflow quando risultato ha segno discorde.

↓

Se somma 2 numeri positivi me ne aspetto uno positivo, quindi se risultato è negativo ho overflow.
(non importa carry).

DIFFERENZA → stesse regole somma

RAPPRESENTAZIONI NUMERICHE

Dati N bit, si possono codificare 2^N oggetti distinti utilizzabili per varie rappresentazioni numeriche.

Esempio (3 bit):

"oggetti" binari	000	001	010	011	100	101	110	111
num. naturali	0	1	2	3	4	5	6	7
numeri relativi (modulo)	+0	+1	+2	+3	-0	-1	-2	-3
numeri relativi (CA2)	+0	+1	+2	+3	-4	-3	-2	-1

CARATTERI

occorre codifica standard perché è tipo di operazione più utilizzato.

- codice ASCII (American Standard Code for Information Interchange)
- codice EBCDIC (Extended BCD Interchange Code)

CODICE ASCII

Usato anche nelle telecomunicazioni, associa carattere a un numero e decide n° bit con cui rappresentare numero.

Usa 8 bit (→ [0..255] range n° codifica ASCII)
- 52 caratteri alfabetici (a..z, A..Z)
- 10 cifre (0..9)
- segni interpunzione
- caratteri di controllo (invio, esc, spazio...)

↓ Esempio

EOT (4) End-of-Transmission (usato anche nelle telecomunicazioni)

Sufficienti 8 bit per rappresentare tutti i numeri, in realtà ne bastano 7 come era originariamente

UNICODE e UTF-8 → cioè include ASCII

UNICODE:
- esprime caratteri in tutte le lingue del mondo (+ di un milione)
- sistema di codifica che assegna numero univoco ad ogni carattere usato per scrittura di testi indipendentemente da lingua, piattaforma informatica e programma

UTF-8:
- codifica di Unicode + usata
- 1 byte per caratteri US-ASCII (MSB=0)
- 2 byte per caratteri latini con simboli diacritici, Greco, Cirillico, Armeno, Ebraico, Arabo, Siriano e Maldiviano.
- 3 byte per altre lingue di uso comune
- 4 byte per caratteri rarissimi
- raccomandato da IETF per e-mail

ISTRUZIONI ELEMENTARI

corrispondono a blocchi di azione dei diagrammi di flusso:

- Assegnazione 
- input/output (I/O) 

ASSEGNAZIONE

<variabile> = <valore>

- Il valore viene assegnato alla variabile (non è un'uguaglianza)
- <variabile> e <valore> devono essere tipi compatibili
- <variabile> deve essere dichiarata precedentemente
- assegnazione può essere inclusa nella dichiarazione delle variabili.

ISTRUZIONI I/O

Divise in categorie in base al tipo di informazione letta o scritta:

- I/O formattato
- I/O a caratteri
- I/O "per righe" w stringhe

In C I/O sono gestite non come istruzioni ma come funzioni.

OUTPUT → printf()
INPUT → scanf()

} per essere utilizzate deve essere inserita la direttiva:

#include <stdio.h>

all'inizio del file sorgente, indica di includere il file stdio.h che è una libreria di C che contiene alcune funzioni come queste.

PRINTF

printf(<formato>, <arg-1>, ..., <arg n>);

→ quantità/espressioni che si vogliono stampare associati alle direttive di formato nello stesso ordine

Sequenza di caratteri che determina formato di stampa degli argomenti.
Può contenere:
- caratteri (stampati come appaiono)
- direttive di formato nella forma: %<carattere>
  - %d intero
  - %u unsigned
  - %s stringa = insieme di caratteri (parole/frase)
  - %c carattere
  - %x esadecimale
  - %o ottale
  - %f float
  - %g double
  - %(#d) → per mettere segno

base = 16 (H: Hexadecimal)
cifre: {0,1,...,9,A,B,C,D,E,F}
per scrivere in modo compatto i binari (4:1)
1011 1001 → 2
B 9 → 16

base = 8 (indicato con 0 per Octal)
cifre = {0,1,2,3,4,5,6,7}
usato per scrivere i binari in modo compatto (3:1)
10 111 001 → 2
2 7 1 → 8

Esempio

```
#<stdio.h>
main()
{
    int a;
    float b;
    printf("Dammi un numero intero (A): ");
    scanf("%d", &a);
    printf("Dammi numero reale (B):");
    scanf("%f", &b);
    printf("A=%d\n",a);
    printf("B=%f\n",b);
}
```

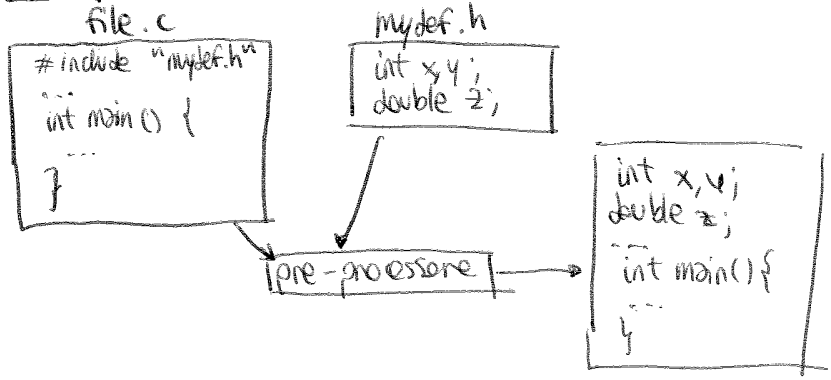
PRE-PROCESSORE C

La compilazione C passa attraverso un passo preliminare che precede la vera e propria traduzione in linguaggio macchina, il programma che realizza questa fase è il pre-processore.
Funzione principale: espansione delle direttive che iniziano con #; le principali sono #include e #define

DIRETTIVA #INCLUDE

#include <file> → <**<nome file>**> per includere un file di sistema
(es. #include <stdio.h>)
"<nome file" per includere file definito dal programmatore
(es. #include "miofile.h")
espanso ed incluso per intero nel file sorgente

Esempio



```
q = x / y;   // (q=2, troncamento)
r = x % y;   // (r=1)
```

Esempio: DIVISIONE TRA INTERI

```
#include <stdio.h>
main()
{
   int a,b;
   printf("Scrivi intero (A):");
   scanf("%d", &a);
   printf("Scrivi intero (B):");
   scanf("%d, &b);
   printf("A div B = %d\n", a/b);
   printf(" A mod B = %d\n", a%b);
}
```

Quesito

```
a=10;
b=25;
a=b;   } a=25
b=a;   }
```

? : :

b=25

OPERATORI DI CONFRONTO

uguaglianza		ordine	
a==b	uguale	a > b	maggiore
a!=b	diverso	a < b	minore
		a >= b	maggiore o uguale
		a <= b	minore o uguale

⚠ a==b confronto
a=b assegnazione

OPERATORI DI INCREMENTO

- CONTATORI → dato che nei cicli si usano molto spesso, in C c'è funzione apposita.

operatore ++ → +1 (es i=i+1)
operatore -- → -1 (es i=i-1)

Possono essere utilizzati in 2 notazioni:

- PREFISSA: variabile modificata prima di essere utilizzata nell'espressione

```
Y = ++X
X=4 → X=5
    → Y=5
```

- POST FISSA: variabile modificata dopo averla utilizzata nell'espressione

```
Y = X++
X=4 → Y=4
    → X=5
```

(1° assegnazione 2° incremento)

VARIABILI BOOLEANE (1847 - Boole)

Assumono solo 2 valori, FALSO o VERO.
In ogni problema è importante distinguere tra variabili dipendenti e indipendenti.

OPERATORI logici → UNARI: NOT → B → B
per collegare espressioni di confronto. → BINARI: AND, OR → B^2 → B
cioè lavora con 2 condizioni/input in entrata.

descritti da TAVOLE DI VERITÀ (N operandi → 2^N righe) dove sono elencate tutte le possibili combinazioni di valori delle variabili indipendenti ed il valore assunto dalla variabile dipendente.

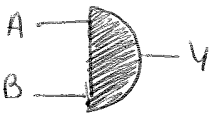
Vero = 1
Falso = 0

AND

condizione 1, condizione 2, risultato

A	B	A x B
0	0	0
0	1	0
1	0	0
1	1	1

risultato vero solo se entrambe le condizioni sono vere.



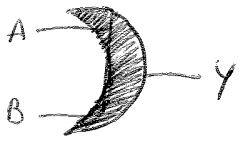
$Y = A \times B$

porta AND

OR

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

risultato vero basta che almeno una sia vera.



$Y = A + B$

porta OR

NOT

A	$\bar{A}$ oppure A'
0	1
1	0

$Y = A'$

all'interno ci sono transistor che invertono valore del bit.

porta NOT/INV

ha lo scopo di invertire il valore del bit

CIRCUITO = combinazione di porte logiche che, lavorando con 0,1, è in grado di lavorare con bit.

Esempio

```
if (!(a==b)) {*...}
    1 se a=b
con NOT davanti
0 se a=b
1 se a≠b → esegue 1° blocco di istruzioni ({*...}) se a≠b.
```

ESPRESSIONI BOOLEANE

combinazione di variabili ed operatori booleani.

FUNZIONI BOOLEANE

Applicazione molti-a-uno → $f: B^N \to N$
(es. f(AB) = A e (non B))

Dato che le variabili booleane possono assumere solo 2 valori, si possono dimostrare proprietà e teoremi considerando TUTTI i casi possibili.

n° variabili	n° combinazioni
2	4
3	8
4	16
...	...

è l'importante è scriverle come se fossero numeri binari progressivi

ESEMPIO: DIMOSTRAZIONE PROPRIETÀ DISTRIBUTIVA

$$A+(B\times C)=A\times B+A\times C$$

A	B	C	A+B×C	(A+B)×(A+C)
0	0	0	0+0×0=0	(0+0)×(0+0)=0
0	0	1	0+0×1=0	(0+0)×(0+1)=0
0	1	0	0+1×0=0	(0+1)×(0+0)=0
0	1	1	0+1×1=1	(0+1)×(0+1)=1
1	0	0	1+0×0=1	(1+0)×(1+0)=1
1	0	1	1+0×1=1	(1+0)×(1+1)=1
1	1	0	1+1×0=1	(1+1)×(1+0)=1
1	1	1	1+1×1=1	(1+1)×(1+1)=1

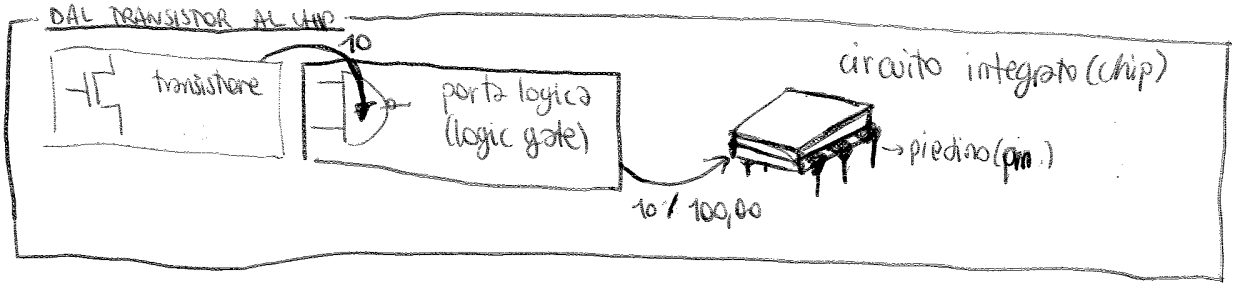
con OR se c'è 0 allora risultato dipende dal 2° termine.

con OR se c'è 1 allora sicuramente risultato finale è 1

(=)

$A+A' =$ sempre 1

$A \cdot A' =$ sempre 0



CAST → OPERATORE

Può essere necessario convertire esplicitamente un'espressione di uno specifico tipo.
- Quando regole di conversione automatica non si applicano

Esempio
```
int i;
double d;
i=d ⟶ fa perdere informazione
```

(<tipo>) <espressione>; ⟶ forza <espressione> ad essere interpretata come se fosse di tipo <tipo>

Esempio
```
#include <stdio.h>
main()
{
  int a,b;
  printf("scrivi intero A");
  scanf("%d", &a);
  printf("scrivi intero B");
  scanf("%d", &b);
  if (b==0)
    printf("errore: divisione per zero\n");
  else
    printf("A/B=%f\n", ((float)a)/b);
}
```

SIZEOF() → OPERATORE

Per calcolare nº byte usato dai tipi di dato di base; è applica ad un tipo e il risultato dice in byte qual è dimensione occupata in memoria da tale tipo.

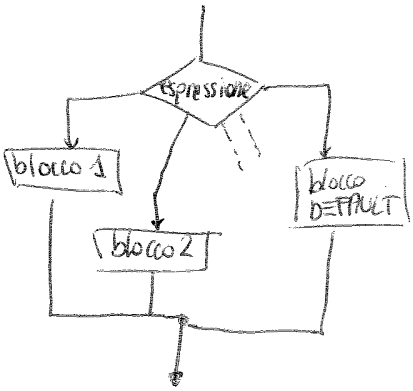
sizeof(<tipo>) ⟶ ritorna nº byte occupati da <tipo>

Può essere esteso al calcolo dello spazio occupato da espressioni, vettori e strutture.

Esempio
```
• unsigned int size;
  size = sizeof(float); /* size=4 */
• printf(" tipo: double , nº byte: %d \n", sizeof(double));
```

SWITCH

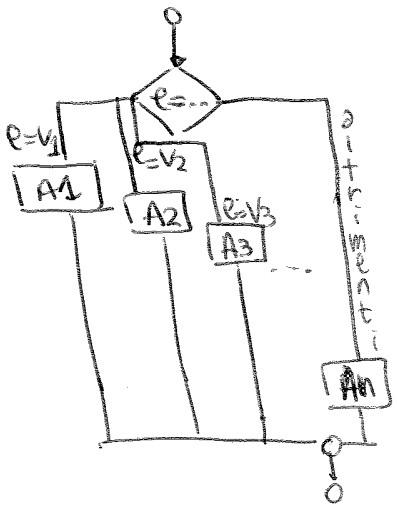
```
switch (<espressione>)        → espressione a valore numerico
{
  case <costante1>;           → devono essere 2 interi costanti
    <blocco1>                 → sequenze di istruzioni
    break;                      NO GRAFE!
  case <costante2>;
    <blocco2>
    break;
  ...
  default:
    <blocco default>
}
```



SWITCH → + da if che dà solo scelta binaria
→ scelta tra + condizioni → risultato di una espressione che dà risultato non 0 o 1 ma n
DEFAULT → via d'uscita se risultato non è tra le scelte precedenti, cioè se non si è verificata nessuna delle altre condizioni.

- In base al valore di <espressione> esegue istruzioni del case corrispondenti, nel caso nessun case venga intercettato, esegui istruzioni corrispondenti al caso default
- CASE → devono rappresentare condizioni mutualmente esclusive!
  → eseguiti in sequenza → per evitarlo si usa istruzione **break** all'interno di ogni blocco.

```
switch (e)
{
  case v1:
      A1;
    break
  case v2:
      A2;
    break;
  ...
  default:
    An;
}
```



Esempio

```
int x;
...
switch(x){
 case 1:
  printf("Solo nel caso 1\n");
   break;
 case 2:
  printf("Solo nel caso 2\n");
```

FOR

```
for(i=0, i<N, i++){      (ciclo viene eseguito N-i volte)
  corpo; D
}
```

(A, B, C)

vengono eseguite con lo stesso ordine di while:

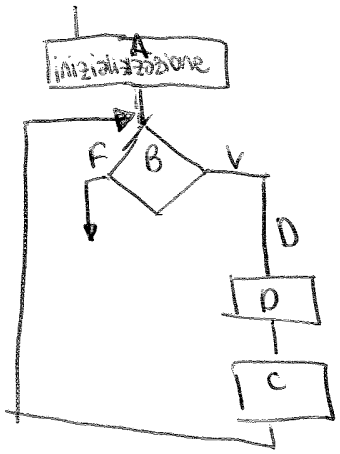
1. inizializzazione
2. (controllo) CONDIZIONE DI RIPETIZIONE
3. corpo
4. aggiornamento

cioè while=For come concetto.

posso anche scrivere

```
for( ; i<N; i++);
```

↳ se si è già inizializzato prima si può lasciare spazio vuoto
se ne possono lasciare vuoti uno o più. → tutti i campi possono essere vuoti!



si usa ciclo quando ↗ cose sono tante
↘ le istruzioni da eseguire dipendono da un input=condizione

Esercizio

```
main () {
  int N;
  char ch;
  scanf("%d %c", &N, &ch);
  for(i=0; i<N; i++){
    printf("%c", ch);
    printf("\n");
```

i<N → non <= perché si parte da 0, non da 1.

i=0	i<N	corretti perché ciclo viene fatto stesso numero di volte
i=1	i<=N	
i=2	i<=N+1	

```
• for(i=0 ) {
    scanf("%d", &z);
    s=s+z;
    i++;
  }
```

Esempio

- Leggi carattere ch ed un intero N, e stampa una riga di N caratteri ch.
  esempio: N=3 ch='*' output: ***
- formulazione iterativa:
  ripeti N volte l'istruzione 'stampa ch'

```
#include <stdio.h>
main()
{
 int N, i;
 char ch;
    scanf("%d %c", &N, &ch);
    for (i=0; i<N; i++)
       printf("%c", ch);
    printf("\n"); → va a capo solo alla fine della prima riga di asterischi.
}
```

CONFRONTO FOR – WHILE

Ciclo FOR è un caso particolare del ciclo WHILE

FOR:
- per cicli di conteggio
- numero iterazioni note a priori
- condizione fine ciclo tipo "conteggio"

WHILE:
- per cicli GENERALI
- numero iterazioni non note a priori
- condizione finale tipo "evento"

CICLI FOR CON ITERAZIONI NOTE

- ```
  int i;
  for (i=N; i>0; i--);   i: DECREMENTO
  {
  ...
  }
  ```

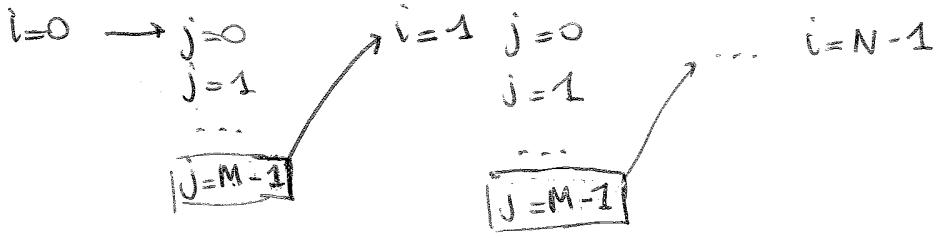
- ```
  int i;
  for (i=1; i<=N ,i++);
  {
  ...
  }
  ```

- ```
  int i;
  for (i=0; i<N; i++);
  {
  ...
  }
  ```

- ```
  int i;
  for (i=N-1; i>=0; i--);
  {
  ...
  }
  ```

CICLI ANNIDATI

```
For (i=0, i<N, i++) {        ciclo + esterno con i=0
   for (j=0, j<M, j++) {     ciclo interno con j=0
      ...
   }
}
```



ad ogni esecuzione del ciclo esterno, quello interno viene azzerato l'indice.

M x N volte eseguito ciclo

Alcuni problemi presentano una struttura bidimensionale, cioè l'operazione iterativa stessa può essere espressa come un'altra iterazione; cioè con un ciclo che contiene un altro ciclo.

- for ( )
  {for ( )
  ...
- while (..)
  {while (..)

Esempio

leggere N controllando che valore sia positivo; in caso contrario, ripetere lettura.

```
...
do
 scanf("%od", &a);
while (a<=0)
}
```

prima faccio lettura e poi controllo.

È sempre possibile trasformare un ciclo do in un ciclo while semplice anticipando e/o duplicando parte delle istruzioni

Esempio

```
#include <stdio.h>
main()
{
 int n;
  scanf("%od", &n);
 while (n<=0)
     scanf ("%od", &n);
}
```

intatti il giro successivo continua ha valore =0
e con l'and basta uno zero per uscire.

CONTINUE → • Termina l'iterazione corrente
• L'esecuzione continua con la prossima iterazione del ciclo.

Esempio

Acquisire da tastiera sequenza di numeri; ignorare numeri=0

```
int valore;
while(scanf("%d",&valore))
 {
  if (valore==0);
  {
  printf("valore non consentito\n");
  continue;
  }
  ...
}
```

→ se legge valore non consentito va direttamente a leggere nuovo valore.

- Un altro modo per eseguire il continue in MODO STRUTTURATO

```
int valore;
while(scanf("%d", &valore))
 {
  if (valore==0)
  {
  printf("valore non consentito\n");
  }
  else
  {
  ...
  }
}
```

Esempio:

```
int main()
{
  int dato[10];
  ...
  for (i=0; i<10; i++)
    scanf ("%d", &dato[i]);
  for (i=9; i>=10; i--)
    printf ("%d\n", dato[i]);
}
```

Insiemi di variabili dello STESSO TIPO aggregate in un' UNICA ENTITÀ
- Identificate globalmente da un nome
- Singole variabili (elementi) individuate da un indice, corrispondente alla loro posizione rispetto al primo elemento
- L'indice degli elementi parte da zero.
- Gli elementi di un vettore sono memorizzati in celle di memoria contigue

a [ ] ⟹ a [a_0 | a_1 | a_2 | a_3]

DICHIARAZIONE
<tipo> <nome vettore> [<dimensione>];

ACCESSO AD UN ELEMENTO
<nome vettore> [<posizione>];

Esempio
int V[10]; → definisce un insieme di 10 variabili intere
V[0], V[1], V[2], V[3], V[4], V[5], V[6], V[7], V[8], V[9].

È possibile assegnare un valore iniziale ad un vettore (solo) nella DICHIARAZIONE ed è equivalente ad assegnare OGNI elemento del vettore:
(vettore N elementi)
{<valore 0>, <valore 1>, ..., <valore N-1>}

Esempio
int lista[4] = {0, 2, 3, 6}

Se vengono assegnati valori a meno di N elementi, l'inizializzazione assegna a partire dal primo valore e gli altri sono posti uguali a zero.

L'indice che definisce posizione dell'elemento in un vettore DEVE essere intero, non numero costante, può anche essere risultato di un'espressione purché intera.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 97 di 159