LE STRUTTURE DI CONTROLLO

Manuale linguaggio C

Istruzioni di controllo

Una istruzione di controllo può essere:

- una istruzione composta (**blocco**)
- una istruzione condizionale (**selezione**)
- una istruzione di iterazione (**ciclo**)

Istruzioni di controllo

Le istruzioni di controllo sono alla base della programmazione strutturata (Dijkstra, 1969): solo le seguenti strutture per alterare il flusso di controllo .

Concetti chiave:

- **concatenazione o composizione**
- **selezione o istruzione condizionale:** ramifica il flusso di controllo in base al valore vero o falso di una espressione ("condizione di scelta")
- **ripetizione o iterazione:** esegue ripetutamente un'istruzione finché rimane vera una espressione ("condizione di iterazione")

OBIETTIVO: rendere i programmi più leggibili, modificabili e manutenibili

Teorema di Jacopini - Böhm

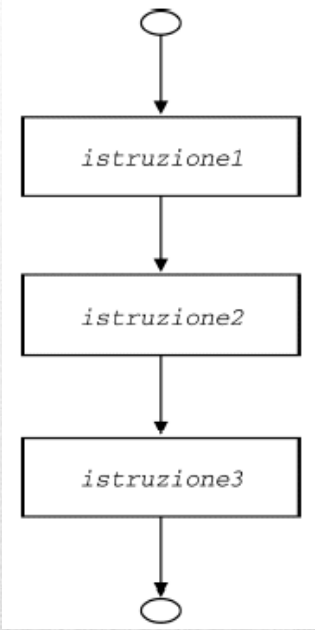
Le strutture di concatenazione, iterazione e selezione costituiscono un **insieme completo** in grado di esprimere tutte le funzioni calcolabili.

Dunque, l'uso di queste sole strutture di controllo non limita il potere espressivo.

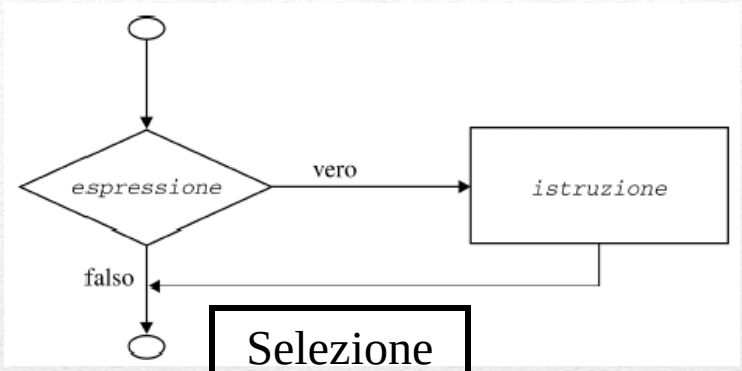
P.es., un linguaggio con i seguenti
Tipi di dato: Naturali con l'operazione di somma (+)
Istruzioni: assegnamento
istruzione composta
istruzione condizionale
istruzione di iterazione
è un linguaggio completo, cioè è un linguaggio in grado di esprimere tutte le funzioni calcolabili.

La dimostrazione del teorema è basata sulla Turing-equivalenza di un "mini-linguaggio" che fornisca solo tali strutture di controllo.

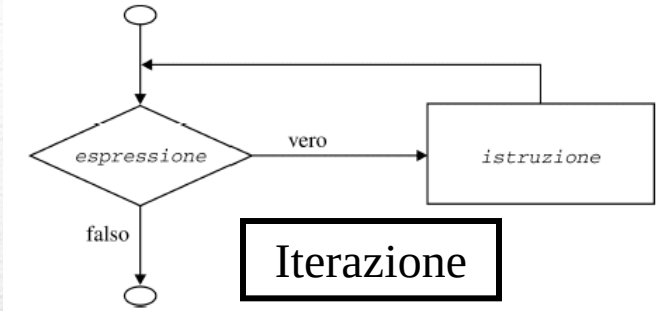
Teorema di Jacopini - Böhm



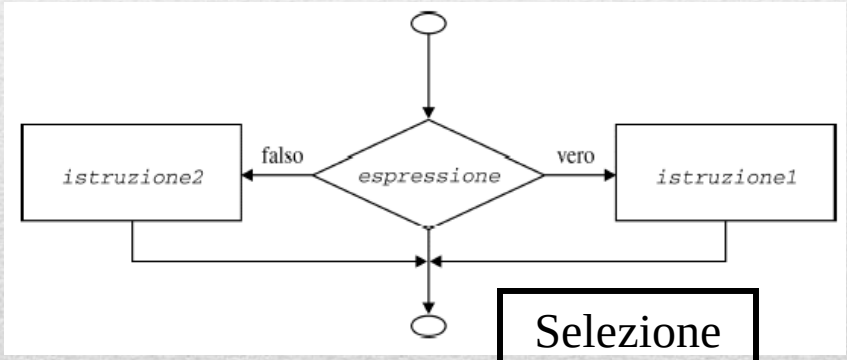
Sequenza



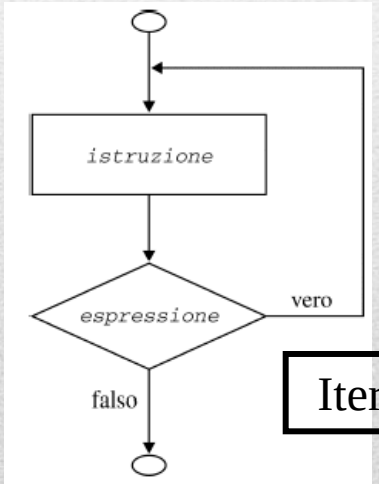
Selezione



Iterazione



Selezione



Iterazione

Strutture di controllo del flusso: istruzioni e blocchi di istruzioni

In un linguaggio le **strutture di controllo del flusso** specificano l'ordine secondo il quale le operazioni devono essere effettuate.

Ogni **istruzione** in C deve essere terminata da un **;**

Un'istruzione costituita dal solo punto e virgola è un'istruzione nulla che non ha effetto.

Un **blocco di istruzioni** è una sequenza di istruzioni C racchiuse da **{** e **}**

Il corpo stesso di una funzione è un blocco di istruzioni.

Dopo un blocco non deve essere inserito il **;**

ISTRUZIONI DI SELEZIONE

Manuale linguaggio C

Istruzione di selezione: if

if (expr1) { ... }
else {... }

Se *expr1* è vera viene eseguito il blocco *if*. La clausola *else* è opzionale e si riferisce sempre alla condizione *if* immediatamente precedente.

```
if(a > b)
   printf("a è maggiore di b\n");
if(0 <= i && i < n)
        printf("il valore della variabile i ricade all'interno dell'intervallo 0 - %d\n", n);
```

Operatori relazionali

<	minore di
>	maggiore di
<=	minore o uguale a
>=	maggiore o uguale a

Operatori di uguaglianza

==	uguale a
!=	diverso da

Operatori logici

!	negazione logica
&&	and logico
||	or logico

```c
/*
numero pari o dispari,
il programma utilizza esclusivamente
istruzioni if
*/
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int numero;

    printf("inserisci un numero: ");
    scanf("%d",&numero);
    if ((numero % 2) == 0)
        printf("\nIl numero inserito e' PARI");
    else
        printf("\nIl numero inserito e' DISPARI");

    printf("\n\n");
    system("PAUSE");
    return 0;
}
```

I blocchi di istruzioni

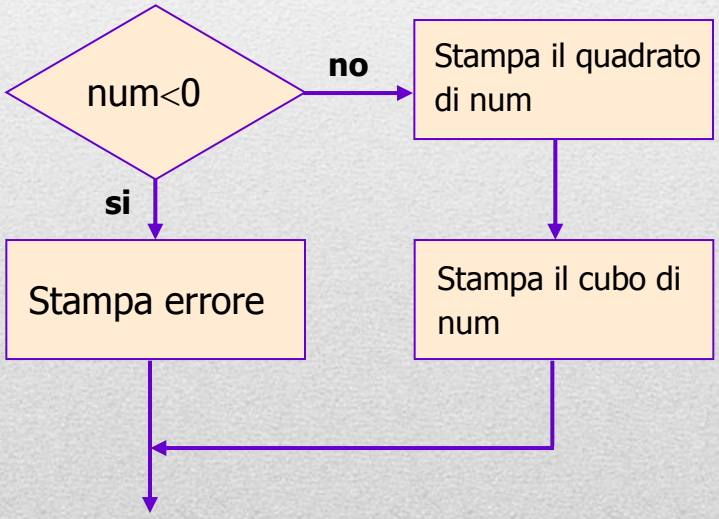
- Un **blocco di istruzioni** deve essere contenuto all'interno di **parentesi graffe**: il corpo di una funzione è un caso particolare di blocco di istruzioni

  ⇒ Per eseguire in modo condizionale più di una singola istruzione serve racchiudere l'insieme di istruzioni in un blocco

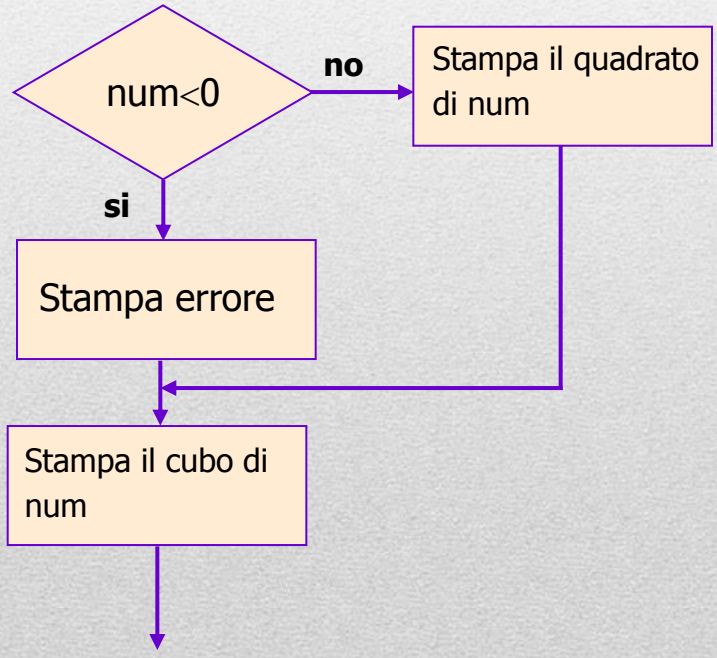
```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int num;

    printf("Introdurre un numero non negativo: ");
    scanf("%d", &num);
    if (num<0)
        printf("Il numero immesso è negativo.\n");
    else
    {
        printf("Il quadrato di %lf è: %lf \n", num, num*num);
        printf("Il cubo di %lf è: %lf \n", num, num*num*num);
    }
    return 0;
}
```

I blocchi di istruzioni

```
if (num<0)
    print errore
else
{
    print quadrato
    print cubo
}
```




```
if (num<0)
    print errore
else
    print quadrato
    print cubo
```



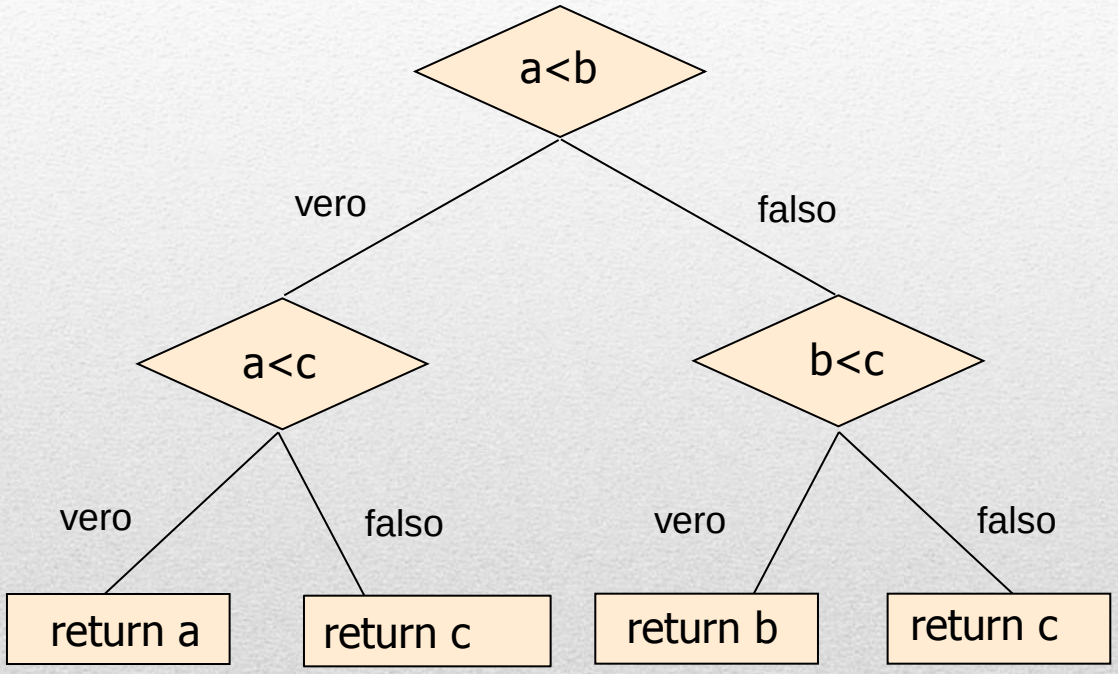
Le parentesi graffe garantiscono la correttezza del flusso di controllo

Istruzioni if annidate

- Una singola istruzione *if* permette al programma di scegliere fra due alternative
- Talvolta è necessario specificare alternative successive: dopo aver preso la prima decisione, è necessario valutare la seconda, la terza, etc.
- Questa tipologia di controllo del flusso richiede un costrutto **if innestato** (o **annidato**)
- Esempio: Realizzare una funzione che, dati tre interi, ne determini il minimo

Istruzioni if annidate

```
...
int a, b, c;
{
      if  (a<b)
         if  (a<c)
             return a;
         else
             return c;
      else if  (b<c)
             return b;
         else
             return c;
}
...
```



Istruzioni if annidate

- Nelle istruzioni if annidate sorge il problema di far corrispondere ad ogni clausola else l'opportuna istruzione if
- Regola: *Una clausola else viene sempre associata all'istruzione if più vicina fra quelle precedenti*

  $\Rightarrow$ ad ogni istruzione if può corrispondere una sola clausola else
- Per facilitare la programmazione, è opportuno indentare correttamente i vari if:
  - Una clausola else dovrebbe sempre essere posta allo stesso livello di indentazione dell'if associato

Esempio

Presi in ingresso 3 numeri stamparli in ordine crescente

```
#include <stdio.h>

int main()
{
      int num1, num2, num3;
      int a, b, b;

      printf("\n Inserire primo valore: ");
      scanf("%d",&a);
      printf("\n Inserire secondo valore: ");
      scanf("%d",&b);
      printf("\n Inserire terzo valore: ");
      scanf("%d",&c);
```

```
if (a < b)
{
      if (a < c)
      {
            num1 = a;
            if (c < b)
            {
                  num2 = c;
                  num3 = b;
            }
            else
            {
                  num2 = b;
                  num3 = c;
            }
      }
      else
      {
            num1 = c;
            num2 = a;
            num3 = b;
      }
}
```

```
        }
        else
        {
                if (b < c)
                {
                        num1 = b;
                        if (c < a)
                        {
                                num2 = c;
                                num3 = a;
                        }
                        else
                        {
                                num2 = a;
                                num3 = c;
                        }
                }
                else
                {
                        num1 = c;
                        num2 = b;
                        num3 = a;
                }
        }

        printf("numeri in ordine crescente: %d, %d, %d",num1,num2,num3);

        return 0;
}
```

```c
/*
Indovina numero,
il programma utilizza esclusivamente
istruzioni if

vengono dati 3 tentativi all'utente
*/
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int numero, numeroDaIndovinare;

    numeroDaIndovinare = 1 + (rand()+time(NULL)) % 100;

    printf("INDOVINA NUMERO\n\n");
    printf("inserisci un numero da 1 a 100 (tentativo 1): ");
    scanf("%d",&numero);
    if (numero == numeroDaIndovinare)
        printf("\nINDOVINATO!");
```

Ricordarsi di usare la funzione
srand()
per inizializzare il generatore. Ad esempio:
srand(time(NULL));

```
    else
    {
        if (numero < numeroDaIndovinare)
            printf("\nIl numero inserito e' piu' piccolo di quello da indovinare!");
        else
            printf("\nIl numero inserito e' piu' grande di quello da indovinare!");
        printf("\n\ninserisci un numero da 1 a 100 (tentativo 2): ");
        scanf("%d",&numero);
        if (numero == numeroDaIndovinare)
            printf("\nINDOVINATO!");
        else
        {
            if (numero < numeroDaIndovinare)
                printf("\nIl numero inserito e' piu' piccolo di quello da indovinare!");
            else
                printf("\nIl numero inserito e' piu' grande di quello da indovinare!");
            printf("\n\ninserisci un numero da 1 a 100 (tentativo 3, L'ULTIMO): ");
            scanf("%d",&numero);
            if (numero == numeroDaIndovinare)
                printf("\nINDOVINATO!");
            else
                printf("\nTENTATIVI ESAURITI!");
        }
    }
    printf("\n\n");
    system("PAUSE");
    return 0;
}
```

Progetto di programmazione – orario

Scrivere un programma che chieda all'utente un orario nel formato a 24 ore e visualizzi lo stesso orario nel formato a 12 ore. Esempio:

input: Enter a 24-hour time: 21:11

output: Equivalent 12-hour time: 9:11 PM

Nota: non visualizzare 12:00 come 0:00

Progetto di programmazione - orario

```
#include <stdio.h>

int main(void)
{
  int hours, minutes;

  printf("Enter a 24-hour time: ");
  scanf("%d:%d", &hours, &minutes);

  printf("Equivalent 12-hour time: ");
  if (hours == 0)
    printf("12:%.2d AM\n", minutes);
  else if (hours < 12)
          printf("%d:%.2d AM\n", hours, minutes);
       else if (hours == 12)
               printf("%d:%.2d PM\n", hours, minutes);
            else
               printf("%d:%.2d PM\n", hours - 12, minutes);

  return 0;
}
```

Progetto di programmazione – scala Beafourt

Scrivere un programma che chieda all'utente di immettere un valore di velocità del vento (in nodi) e visualizzi la corrispondente descrizione, usando la seguente tabella.

Velocità (nodi)	**descrizione**
Minore di 1	Calmo
1 – 3	Bava di vento
4 – 27	Brezza
28 – 47	Burrasca
48 – 63	Tempesta
Oltre 63	Uragano

Progetto di programmazione - scala Beafourt

```
#include <stdio.h>
int main(void)
{
 int speed;

 printf("Enter a wind speed in knots: ");
 scanf("%d", &speed);

 if (speed < 1)
    printf("Calm\n");
 else if (speed <= 3)
         printf("Light air\n");
       else if (speed <= 27)
               printf("Breeze\n");
             else if (speed <= 47)
                     printf("Gale\n");
                   else if (speed <= 63)
                           printf("Storm\n");
                         else
                           printf("Hurricane\n");
 return 0;
}
```

Progetto di programmazione – codici a barre

Modificare il programma relativo alla stampa del carattere di controllo dei codici a barre in modo da controllare se un codice è valido. Dopo l'immissione del codice da parte dell'utente, il programma dovrà stampare VALID oppure NOT VALID. Esempio:

CONTROLLA ESEMPIO
XCHE CODICE
CONTROLLO FORSE
SBAGLIATO

VALID

NOT VALID

Progetto di programmazione – codici a barre

```
#include <stdio.h>
int main(void)
{
  int check_digit, d, i1, i2, i3, i4, i5, j1, j2, j3, j4, j5,
      first_sum, second_sum, total;

  printf("Enter the first (single) digit: ");
  scanf("%1d", &d);
  printf("Enter first group of five digits: ");
  scanf("%1d%1d%1d%1d%1d", &i1, &i2, &i3, &i4, &i5);
  printf("Enter second group of five digits: ");
  scanf("%1d%1d%1d%1d%1d", &j1, &j2, &j3, &j4, &j5);
  printf("Enter the last (single) digit: ");
  scanf("%1d", &check_digit);

  first_sum = d + i2 + i4 + j1 + j3 + j5;
  second_sum = i1 + i3 + i5 + j2 + j4;
  total = 3 * first_sum + second_sum;

  if (check_digit == 9 - ((total - 1) % 10))
    printf("VALID\n");
  else
    printf("NOT VALID\n");
  return 0;
}
```

Espressioni condizionali

La sintassi dell'operatore condizionale è:

expr1 ? expr2 : expr3

L'espressione viene valutata in vari stadi: *expr1* viene valutata per prima, se il suo valore è diverso da zero allora viene calcolata *expr2* il cui valore sarà quello dell'intera espressione condizionale. Se il valore di *expr1* è uguale a zero allora l'espressione condizionale assumerà il valore di *expr3*. Esempio:

```
int i,j,k;

i=1;
j=2;
k= i>j ? i : j;                 //adesso k è uguale a 2
k= (i>=0  ? i : 0)+j;           //adesso k è uguale a 3
```

L'ordine di precedenza dell'operatore condizionale è minore di quello degli altri operatori studiati, fatta eccezione per l'operatore di assegnamento

Espressioni condizionali: esempi

```
if  (i>j)
   return i;
else
   return j;
```

```
return i > j ? i : j;
```

```
if  (i>j)
   printf("%d\n",i);
else
 printf("%d\n",j);
```

```
printf("%d\n", i > j ? i : j);
```

Valori booleani nel C89

Un modo per aggirare la mancanza di un tipo di dato che possa assumere esclusivamente valori *vero* e *falso* è quello di dichiarare una variabile *int* e di assegnarle i valori 0 (falso) e 1 (vero).

```
int flag;

flag=0;
…
flag=1;
```

Non contribuisce alla leggibilità del programma: non è ovvio che alla variabile flag debbano essere assegnati solo valori booleani. Possiamo allora usare *macro*:

```
#define TRUE 1
#define FALSE 0

int flag;
flag=FALSE;
…
flag=TRUE;
…
if (flag==TRUE)          //oppure if (flag)
if (flag==FALSE)         //oppure if (!flag)
```

Valori booleani nel C89

Possiamo definire una macro che possa essere utilizzata come tipo:

```
#define BOOL int

BOOL flag;

flag=FALSE;
…
flag=TRUE;
…
if (flag==TRUE)          //oppure if (flag)
if (flag==FALSE)         //oppure if (!flag)
```

Il compilatore continuerà a trattare flag come una variabile *int*.
Vedremo anche come utilizzare la definizione di tipo e le enumerazioni per dichiare un tipo booleano.

Istruzione di selezione: switch

```
switch(expr) {
   case expr_const1: … break;
   case expr_const2: … break;
   case expr_const3: … break;
   default: …
}
```

Controlla se un espressione assume un valore intero in un insieme di **costanti** intere e fa eseguire una serie di istruzioni in corrispondenza del valore intero verificato.

Se il caso **default** è presente assume il significato di: "in tutti gli altri casi", cioè viene eseguito se l'espressione valutata nello *switch* non ha assunto nessuno dei valori indicati nei *case* precedenti.

La keyword *break* non è strettamente indispensabile: se non è presente viene eseguita sequenzialmente ogni istruzione a partire dal *case* raggiunto.

I possibili casi previsti possono essere solo delle costanti (scelta multipla più veloce di *if-else-if*).

L'esecuzione procede fino a che non si incontra *break* o si oltrepassa la fine dello switch (}).

Istruzione di selezione: switch

```
…
char input_arg;
{
      if  (input_arg == 'A')
          return 1;
      if  (input_arg == 'B')
          return 2;
      if  (input_arg == 'C')
          return 3;
      if  (input_arg == 'D')
          return 4;
      else
          return –1;
}
```

```
…
char input_arg;
{
      switch(input_arg)
      {
         case 'A': return 1;
         case 'B': return 2;
         case 'C': return 3;
         case 'D': return 4;
         default : return –1;
      }
}
```

- In presenza di cammini multipli all'interno di un programma, le diramazioni **if…else** possono complicare la comprensione del codice
- L'istruzione **switch** consente di specificare un numero illimitato di cammini di esecuzione in dipendenza dal valore di un'espressione

Il ruolo dell'istruzione break

L'esecuzione dell'istruzione *break* causa l'uscita del programma dal costrutto *switch* per passare all'istruzione successiva; essa è in sostanza una forma di **salto precalcolato**.

Quando l'ultima istruzione del caso è stata eseguita, il controllo passa alla prima istruzione del caso successivo; senza *break* il controllo passerebbe da un caso a quello successivo. Esempio:

```
switch(grade)
    {
      case 4: printf("Excellent");
      case 3: printf("Good");
      case 2: printf("Average");
      case 1: printf("Poor");
      case 1: printf("Failing");
      default : printf("Illegal grade");
    }
```

Se grade ha valore 3 cosa viene stampato?

Istruzioni condizionali: esempi

```
menu(void){
   printf("1. Controlla ortografia\n");
   printf("2. Corregge errori di ortografia\n");
   printf("3. Stampa gli errori di ortografia\n");
   printf("Un altro tasto per nessuna operazione\n");
   printf("Inserire la scelta: \n");

   car=getchar();     //accetta la selezione da tastiera
   switch(car) {
      case '1': contr_ortografia(); break;
      case '2': correggi_errori(); break;
      case '3': stampa_errori(); break;
      default: printf("Nessuna opzione selezionata");
   }
}
```

Pro e contro 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...)

Progetto di programmazione – conversione voti

Utilizzare l'istruzione switch per scrivere un programma che converta un voto numerico in un voto espresso tramite una lettera. Esempio:

Input: Enter numerical grade: 84
Output: Letter grade: B

Utilizzare la seguente scala:

A=90-100
B=80-89
C=70-79
D=60-69
F=0-59

Stampare un messaggio di errore nel caso in cui il voto inserito fosse maggiore di 100 o minore di 0

Progetto di programmazione – conversione voti

```
#include <stdio.h>

int main(void)
{
  int grade;

  printf("Enter numerical grade: ");
  scanf("%d", &grade);

  if (grade < 0 || grade > 100) {
    printf("Illegal grade\n");
    return 0;
  }

  // CONTINUA
```

Progetto di programmazione – conversione voti

```
  // CONTINUA

  switch (grade / 10) {
    case 10:
    case 9:  printf("Letter grade: A\n");
             break;
    case 8:  printf("Letter grade: B\n");
             break;
    case 7:  printf("Letter grade: C\n");
             break;
    case 6:  printf("Letter grade: D\n");
             break;
    case 5:
    case 4:
    case 3:
    case 2:
    case 1:
    case 0:  printf("Letter grade: F\n");
             break;
  }
  return 0;
}
```

Progetto di programmazione – stampa data

Scrivere un programma C che stampi la data secondo la modalità:

Dated this _____ day of _____, 20__

Partendo da una data immessa dall'utente in formato anglosassone mese/giorno/anno. Esempio:

Input: Enter date (mm/dd/yy) : 7/19/14
Output: Dated this 19th day of July, 2014.

Nota: al giorno va aggiunto il corretto suffisso «*th*», «*st*», «*nd*» o «*rd*»; il mese va indicato con il nome e non con il numero

Progetto di programmazione – stampa data

```
#include <stdio.h>

int main(void)
{
  int month, day, year;

  printf("Enter date (mm/dd/yy): ");
  scanf("%d /%d /%d", &month, &day, &year);

  printf("Dated this %d", day);
  switch (day) {
    case 1: case 21: case 31:
      printf("st"); break;
    case 2: case 22:
      printf("nd"); break;
    case 3: case 23:
      printf("rd"); break;
    default: printf("th"); break;
  }
  // CONTINUA
```

Progetto di programmazione – stampa data

```
// CONTINUA
printf(" day of ");

  switch (month) {
    case 1:  printf("January"); break;
    case 2:  printf("February"); break;
    case 3:  printf("March"); break;
    case 4:  printf("April"); break;
    case 5:  printf("May"); break;
    case 6:  printf("June"); break;
    case 7:  printf("July"); break;
    case 8:  printf("August"); break;
    case 9:  printf("September"); break;
    case 10: printf("October"); break;
    case 11: printf("November"); break;
    case 12: printf("December"); break;
  }

  printf(", 20%.2d.\n", year);          //visualizza le ultime due cifre dell'anno con lo 0 e non staccate
  return 0;
}
```

Progetto di programmazione – scarabeo

Scrivere un programma C che calcoli il valore di una parola sommando il valore associato alle sue lettere, come avviene nel gioco dello SCARABEO. Nella sua versione inglese i valori sono:

1: AEILNORSTU
2: DG
3: BCMP
4: FHVWY
5: K
8: JX
10: QZ

Input: Enter a word: pitfall
Output: Scrabble value: 12

Nota: il programma deve permettere di elaborare parole al cui interno siano presenti sia lettere maiuscole che minuscole (suggerimento: trasformare tutte le lettere in maiuscolo)

Progetto di programmazione – scarabeo

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

int main(void)
{
  int sum = 0;
  char ch;

  printf("Enter a word: ");

  // CONTINUA
```

La gestione dei caratteri tramite la libreria *ctype.h* consente di classificare i caratteri in alfanumerici, binari, esadecimali, spazi bianchi, caratteri di stampa, caratteri minuscoli, caratteri maiuscoli, caratteri di punteggiatura, e di passare dall'alfabeto maiuscolo a quello minuscolo e viceversa.
Le funzioni sono implementate in maniera indipendente dal sistema in uso (ad es., il set di caratteri ASCII), evitando quindi problemi di portabilità su altre piattaforme.

Progetto di programmazione – scarabeo

```
// CONTINUA

 while ((ch = getchar()) != '\n')      //legge un singolo carattere, restituendolo
   switch (toupper(ch)) {
     case 'D': case 'G':
       sum += 2; break;
     case 'B': case 'C': case 'M': case 'P':
       sum += 3; break;
     case 'F': case 'H': case 'V': case 'W': case 'Y':
       sum += 4; break;
     case 'K':
       sum += 5; break;
     case 'J': case 'X':
       sum += 8; break;
     case 'Q': case 'Z':
       sum += 10; break;
     default:
       sum++; break;
   }
 printf("Scrabble value: %d\n", sum);
 return 0;
}
```

getchar() è più veloce di *scanf*() perché più semplice rispetto alla seconda, progettata per leggere molti tipi di dato (così come la *putchar*() è più veloce della corrispondente *printf*())

Inoltre, *getchar*() è più flessibile, es.:

```
while (getchar() != '\n')
    ;            //salta il resto della riga

while (getchar() == ' ')
    ;            //salta gli spazi
```

Alcune funzioni della libreria ctype.h

int isalpha(int c);

restituisce il valore vero se il carattere appartiene all'insieme A-Z o a-z

int isalnum(int c);

restituisce il valore vero se il carattere appartiene all'insieme A-Z, a-z o 0-9.

int isdigit(int c);

restituisce il valore vero se il carattere appartiene all'insieme 0-9

int islower(int c);

restituisce il valore vero se il carattere appartiene all'insieme a-z.

int isupper(int c);

restituisce il valore vero se il carattere appartiene all'insieme A-Z.

tolower: se isupper è vero restituisce il carattere in minuscolo, altrimenti restituisce il carattere stesso.
toupper: se islower è vero restituisce il carattere in maiuscolo, altrimenti restituisce il carattere stesso.

Esercizi proposti

1. 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*)

2. Dato uno stipendio lordo, calcolare lo stipendio netto mensile (ipotizzando solo la tassazione IRPEF). Algoritmo:

Ipotesi semplificativa delle aliquote:
- Da 0 a 10000 euro è esente
- Da 10001 a 30000 euro è 20%
- Oltre 30000 è 30%

• Se (stipendio_lordo <= 10000) allora stipendio_netto= stipendio_lordo

• Se (30000 >= stipendio_lordo > 10000) bisogna togliere 10000 esentasse e al residuo applicare l'aliquota del 20%

• Se (stipendio_lordo > 30000) bisogna togliere 10000 esentasse, tra 10000 e 30000 applicare l'aliquota del 20%, oltre l'aliquota del 30%

• In tutti i casi: stipendio_mensile=stipendio_netto/12

I CICLI

Manuale linguaggio C

Istruzioni di iterazione

Per il Teorema di Jacopini-Böhm, una struttura di controllo iterativa sarebbe sufficiente: averne di più migliora l'espressività del linguaggio.

Le istruzioni di iterazione:

- hanno un solo punto di ingresso e un solo punto di uscita nel flusso del programma
- perciò possono essere interpretate come una singola azione in una computazione sequenziale.

Cicli: while

```
while (expr) {
   ...
}
```

expr è valutata prima di ogni iterazione; se è **vera** viene eseguito il blocco { … }

Poiché la valutazione della condizione è effettuata prima delle istruzioni che costituiscono il ciclo, il *loop* può essere eseguito zero volte oppure un numero finito o anche infinito di volte.

```
a=3;
b=5;
while(a < b) {
   printf("a = %d\n",a);
   a++;
}
```

Le due istruzioni comprese tra le graffe sono eseguite finché la variabile *a*, incremento dopo incremento, diventa uguale a *b*. A questo punto l'esecuzione prosegue con la prima istruzione che segue la graffa chiusa.

Esempi

```
int i;
i = 0;
while (i < 100) {
        printf("*");
        i++;
}
```

Stampa 100 asterischi

```
int conta, dato, somma;
printf("Immetti 10 interi: ");
somma = 0;
conta = 0;
while (conta < 10) {
        scanf("%d", &dato);
        somma += dato;
        conta ++;
}
printf("La somma e‘ %d\n", somma);
```

Leggere 10 interi, calcolarne la somma e stamparla.

Esempi

```
int lung, conta, dato, somma;
printf("Immetti la lunghezza della sequenza ");
printf("seguita dagli elementi della stessa: \n");
scanf("%d", &lung);
somma = 0;
conta = 0;
while (conta < lung) {
        scanf("%d", &dato);
        somma += dato;
        conta ++;
}
printf("La somma e‘ %d\n", somma);
```

Leggere un intero N seguito da N interi e calcolare la somma di questi ultimi.
Simile al precedente: il numero di ripetizioni necessarie non è noto al momento della scrittura del programma ma lo è al momento dell'esecuzione del ciclo.

Esempi

```
int conta, dato, massimo;

printf("Immetti 10 interi: \n");
massimo = 0;
conta = 0;
while (conta < 10) {
        scanf("%d", &dato);
        if (dato > massimo)
                massimo = dato;
        conta ++;
}
printf("Il massimo e‘ %d\n", massimo);
```

Leggere 10 interi positivi e stamparne il massimo.

Progetto di programmazione – tavola quadrati

Scrivere un programma C che stampi la tavola dei quadrati, chiedendo all'utente di immettere un valore *n* e stampando *n* righe ognuna contenente un numero e il suo quadrato. Esempio:

Input: Enter number of entries in the table: 5

Output:

1	1
2	4
3	9
4	16
5	24

Progetto di programmazione – tavola quadrati

```
#include <stdio.h>

int main(void)
{
  int i, n;

  printf("Enter number of entries in table: ");
  scanf("%d", &n);

  i = 1;
  while (i <= n) {
    printf("%10d%10d\n", i, i * i);          //visualizzo i numeri allineandoli alle colonne
    i++;
  }

  return 0;
}
```

Cicli: while

Se al primo test la condizione non è vera, il ciclo non viene eseguito neppure una volta.

E' indispensabile che all'interno delle graffe accada qualcosa che determini le condizioni necessarie per l'uscita dal ciclo: se l'espressione di controllo avrà sempre un valore diverso da zero il ciclo sarà infinito. In questo caso i successivi incrementi di *a* rendono falsa, prima o poi, la condizione da cui tutto il ciclo *while* dipende.

Esiste però un altro metodo per abbandonare un ciclo al verificarsi di una certa condizione: si tratta dell'istruzione **break**. Es.:

```
a=3;
b=10;
while(a < b) {
   printf("a = %d\n",a);
   if(++a == 5)
      break;
}
```

In questo caso *a* è incrementata e *poi* confrontata con il valore 5: se uguale, il ciclo è interrotto, altrimenti esso prosegue con le eventuali istruzioni successive.

Cicli: while

E' anche possibile escludere dall'esecuzione una parte del ciclo e forzare il ritorno al test. Es.:

```
while(a < b) {
  if(a++ < c)
    continue;
  printf("a = %d\n",a);
  if(++a == 100)
    break;
  --c;
}
```

a viene confrontata con *c* e *poi* incrementata. Se, prima dell'incremento essa è minore di *c* il flusso elaborativo ritorna al test dell'istruzione *while*; la responsabile del salto forzato è l'istruzione **continue**, che consente di iniziare da capo una nuova iterazione. In caso contrario viene chiamata printf() e, successivamente, viene effettuato il nuovo test con eventuale uscita dal ciclo.

Esercizio: implementare il calcolo del fattoriale

Cicli: while

I cicli while possono essere annidati a qualunque livello di profondità. Es.:

```
while(a < b) {
  if(a++ < c)
    continue;
  printf("a = %d\n",a);
  while(c < x)
    ++c;
  if(++a == 100)
    break;
  --c;
}
```

All'interno del ciclo per (a < b) ve n'è un secondo, per (c < x).
Già nella prima iterazione del ciclo "esterno", se la condizione (c < x) è vera si entra in quello "interno", che viene interamente elaborato (cioè *c* è incrementata finché assume valore pari ad x) prima che venga eseguita la successiva istruzione del ciclo esterno.
In pratica, ad ogni iterazione del ciclo esterno avviene una serie completa di iterazioni nel ciclo interno.

Eventuali istruzioni *break* o *continue* presenti nel ciclo interno sono relative esclusivamente a quest'ultimo

Cicli: do ... while

```
do {
   ...
} while (expr);
```

expr è valutata al termine di ogni iterazione; se è **vera** si riesegue il blocco { … }

```
do {
   if(a++ < c)
      continue;
   printf("a = %d\n",a);
   while(c < x)
      ++c;
   if(++a == 100)
      break;
   --c;
} while(a < b);
```

Un ciclo *do...while* è eseguito sempre almeno una volta, infatti il flusso elaborativo deve percorrere tutto il blocco di codice del ciclo prima di giungere a valutare per la prima volta la condizione.

L'istruzione *continue* non determina un salto a ritroso, bensì in avanti. Essa infatti forza in ogni tipo di ciclo un nuovo controllo della condizione; nei cicli *do...while* il test è a fine codice.

do ...while: controllo valori in input

Esempio 1: *n deve essere positivo*

```
do
        scanf("%d", &n);
while (n<=0);
```

Esempio 2: *n deve essere compreso fra 3 e 15 (inclusi)*

```
do
        scanf("%d", &n);
while ((n<3) || (n>15));
```

Esempio 3: *n deve essere negativo o compreso fra 3 e 15*

```
do
        scanf("%d", &n);
while ((n>=0) && ((n<3) || (n>15)));
```

do ...while: esempio

Somma un certo numero di valori inseriti dall'utente; 0 per uscire

```
int a, somma;

somma=0;
do {
    scanf("%d",&a);
    somma+=a;
} while(a!=0);
printf("La somma è %d\n",somma);
```

Progetto di programmazione – numero cifre

Scrivere un programma C che calcoli il numero di cifre presenti in un intero immesso dall'utente. Esempio:

Input: Enter a non-negative integer: 60
Output: The number has 2 digit(s)

Nota: dividiamo ripetutamente per 10 il numero immesso dall'utente, fino a quando questo non diventa uguale a zero. Il numero di divisioni effettuate corrisponde al numero di cifre. Siccome tutti gli interi (anche lo zero) hanno almeno una cifra usiamo il costrutto do … while

Progetto di programmazione – numero cifre

```
#include <stdio.h>

int main(void)
{
  int digits = 0, n;

  printf("Enter a non-negative integer: ");
  scanf("%d", &n);

  do {
    n /= 10;
    digits++;
  } while (n > 0);

  printf("The number has %d digit(s)\n", digits);

  return 0;
}
```

e se sostituissimo il do … while con il costrutto while?

```
  while (n > 0) {
      n /= 10;
      digits++;
   }
```

Cicli: for

```
for (expr1;expr2;expr3) {
    ...
}
```

expr1 è valutata prima della prima iterazione
expr2 è valutata prima di ogni iterazione
se è vera viene eseguito il blocco { … }
expr3 è valutata al termine di ogni iterazione

Il ciclo *for*, grazie alla sua logica

"*punto di partenza; limite; passo d'incremento*",

si presta ai casi in cui si conosce in partenza il numero di ripetizioni da compiere.

Cicli: for

Nei cicli *for* possiamo utilizzare le istruzioni *break* e *continue*: la prima per "saltar fuori" dal ciclo; la seconda per tornare immediatamente alla valutazione del test.
I cicli *for* possono essere annidati, e va tenuto presente che il ciclo più interno compie una serie completa di iterazioni ad ogni iterazione di quello che immediatamente lo contiene. Es.:

```
for(i = 1; i < k; i++) {
....
}
```

Prima di effettuare la prima iterazione, la variabile *i* è inizializzata a 1. Se essa risulta minore della variabile *k* il ciclo è eseguito una prima volta.

Al termine di ogni iterazione essa è incrementata e successivamente confrontata con *k*; se risulta minore di quest'ultima il ciclo è ripetuto.

Ciclo for: alcuni esempi

```
#include "stdio.h"
void main()
{
   int i;
   for(i = 100; i>0; i--)
            printf("%d ",i);
}
```

```
#include "stdio.h"
void main()
{
   int i;
   for(i = 0; i<=100; i+=5)
            printf("%d ",i);
}
```

```
#include "stdio.h"
void main()
{
   int i;
   for(i = 0; i<100; i++) {
            printf("Questo è il valore di i: %d ",i);
            printf(" e il quadrato di i è: %d\n", i*i);
}
```

```
x=10;
for(y = 10; y!=x; y++)
     printf("%d ",y);
printf("%d",y);
```

For con più di una variabile di controllo

Stampare i numeri da 0 a 98 con incremento di 2

```
#include "stdio.h"
void main()
{
   int i,j;
   for(i = 0, j=0; i+j<100; ++i, ++j)
            printf("%d \n",i+j);
}
```

For con alcune espressioni mancanti

```
for(i = 0; i!=123; )
scanf("%d ",&i);
```

Il loop viene eseguito fino a quando l'utente non inserisce il numero 123

```
gets(s);              //legge una stringa in s
if (*s)
   x=strlen(s);       //lunghezza della stringa

for ( ;x<10; ){
   printf("%d",x);
   ++x;
}
```

L'inizializzazione è vuota e il valore di *x* viene definito prima di entrare nel loop.

Esercizio: implementare il calcolo del fattoriale

Cicli annidati

Il ciclo di livello n–esimo deve terminare la propria esecuzione prima che il ciclo al livello n–1 possa riprendere l'iterazione

Lo specificatore di formato %5d forza la funzione *printf()* a stampare 5 caratteri per ogni intero: se il numero richiede meno di 5 caratteri, viene preceduto da un numero appropriato di spazi bianchi

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

/* Stampa una tavola pitagorica mediante cicli annidati*/

int main()
{
        int j, k;

        printf("    1    2    3    4    5    6    7    8    9   10\n");
        printf("   ----------------------------------------------------\n");

        for (j=1; j<=10; j++)
        {
                printf("%5d|", j);
                for (k=1; k<=10; k++)
                        printf("%5d", j*k);
                printf("\n");
        }
        return 0;
}
```

Maggior controllo sui cicli: istruzione di salto break

L'istruzione **break** serve ad imporre l'uscita da un loop di tipo *for*, *while*, *do…while*, oppure l'uscita dallo *switch*, e a far riprendere il controllo di flusso immediatamente dopo la fine del blocco da cui si esce.

Nel caso di cicli annidati, con l'istruzione break si esce solo dal ciclo più interno entro il quale si trova la chiamata alla istruzione (es.: in caso di *switch* dentro un altro ciclo, il break fa uscire solo dallo *switch*).

Maggior controllo sui cicli: istruzione di salto continue

L'istruzione **continue** si applica ai cicli ma non allo *switch*.
Essa interrompe l'esecuzione di un ciclo *for*, *while* o *do…while*, ma anziché uscire definitivamente dal ciclo fa eseguire la successiva iterazione.
In caso di cicli annidati, si interrompe solo il ciclo più interno in cui è avvenuta la chiamata alla *continue*.

Causa l'esecuzione immediata dell'istruzione di chiusura del ciclo, cui segue l'iterazione successiva del ciclo.

Esempi

```
int conta, dato, massimo;

printf("Immetti 10 interi: \n");
massimo = 0;
for (conta=0; conta<10; conta++)
{
        scanf("%d", &dato);
        if (dato > massimo)
                massimo = dato;
}
printf("Il massimo e‘ %d\n", massimo);
```

Leggere 10 interi positivi e stamparne il massimo.

Esempi

```
int riga, colonna;
const int Nmax = 10; /* indica il numero di righe e di colonne */

for (riga = 1; riga <= Nmax; riga++) {
        for (colonna = 1; colonna <= Nmax; colonna++)
                printf("%d ", riga * colonna);
        printf("\n");
}
```

Stampa della tavola pitagorica.

Progetto di programmazione – calcolo MCD

Scrivere un programma C che calcoli il massimo comune divisore di due interi immessi dall'utente. Esempio:

Input: Enter two integers: 12 28
Output: Greatest common divisor: 4

Nota: algoritmo classico per il calcolo del MCD (algoritmo di Euclide)

Progetto di programmazione – calcolo MCD

```
#include <stdio.h>

int main(void)
{
  int m, n, remainder;

  printf("Enter two integers: ");
  scanf("%d%d", &m, &n);

  while (n != 0) {
    remainder = m % n;
    m = n;
    n = remainder;
  }

  printf("Greatest common divisor: %d\n", m);

  return 0;
}
```

Progetto di programmazione – calendario

Scrivere un programma C che stampi il calendario di un mese: l'utente dovrà specificare il numero di giorni del mese e il giorno della settimana in cui questo comincia. Esempio:

Input: Enter number of days in month: 31
Enter starting day of the week (1=Sun, 7=Sat): 3

Output:

		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Nota: per ogni giorno stampato verificare se è l'ultimo della settimana, nel qual caso stampare un carattere new-line

Progetto di programmazione – calendario

```
#include <stdio.h>

int main(void)
{
  int i, n, start_day;

  printf("Enter number of days in month: ");
  scanf("%d", &n);
  printf("Enter starting day of the week (1=Sun, 7=Sat): ");
  scanf("%d", &start_day);

  /* print any leading "blank dates" */
  for (i = 1; i < start_day; i++)
    printf("   ");

  /* now print the calendar */
  for (i = 1; i <= n; i++) {
    printf("%3d", i);
    if ((start_day + i - 1) % 7 == 0)
      printf("\n");
  }

  return 0;
}
```

```c
/*
   Class average program with
   counter-controlled repetition */
#include <stdio.h>

int main()
{
   int counter, grade, total; float average;

   /* initialization phase */
   total = 0;
   counter = 1;

   /* processing phase */
   while ( counter <= 10 ) {
      printf( "Enter grade: " );
      scanf( "%d", &grade );
      total = total + grade;
      counter = counter + 1;
   }

   /* termination phase */
   average = total / 10;
   printf( "Class average is %f\n", average );

   return 0;   /* indicate program ended successfully */
}
```

```
Enter grade: 98
Enter grade: 76
Enter grade: 71
Enter grade: 87
Enter grade: 83
Enter grade: 90
Enter grade: 57
Enter grade: 79
Enter grade: 82
Enter grade: 94
Class average is 81
```

Program Output

```c
/*
   Class average program with
   sentinel-controlled repetition */
#include <stdio.h>

int main()
{
   float average;
   int counter, grade, total;

   /* initialization phase */
   total = 0;
   counter = 0;

   /* processing phase */
   printf( "Enter grade, -1 to end: " );
   scanf( "%d", &grade );

   while ( grade != -1 ) {
      total = total + grade;
      counter ++;
      printf( "Enter grade, -1 to end: " );
      scanf( "%d", &grade );
   }
```

```
25
26    /* termination phase */
27    if ( counter != 0 ) {
28       average = ( float ) total / counter;
29       printf( "Class average is %.2f", average );
30    }
31    else
32       printf( "No grades were entered\n" );
33
34    return 0;   /* indicate program ended successfully */
35 }
```

```
Enter grade, -1 to end: 75
Enter grade, -1 to end: 94
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 83
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.50
```