# Fondamenti di Informatica ◇ 2019-20

## Lezioni, Esercitazioni e Laboratorio

Cristiana Bolchini

# Indice

# Lezione 0 ◇ introduzione al corso

16-09-2019

## Lezione 1 ◇ la rappresentazione dell'informazione – parte 1

17-09-2019

## Lezione 2 ◇ la rappresentazione dell'informazione – parte 2

19-09-2019

# Lezione 3 ◇ algoritmi

23-09-2019

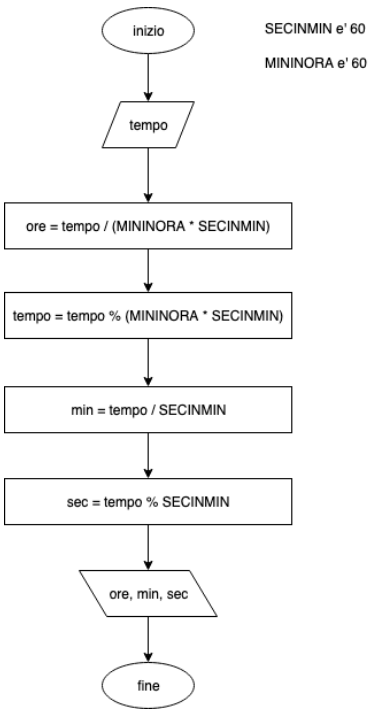
dal problema all'algoritmo
- ◇ proprietà dell'algoritmo
  - ○ non ambiguo
  - ○ deterministico
  - ○ termina in un numero finito di passi
- ◇ tipologia di istruzioni
  - ○ istruzioni effettive
  - ○ istruzioni di controllo

diagrammi di flusso
- ◇ blocchi elementari
  - ○ inizio
  - ○ fine
  - ○ istruzioni semplici
  - ○ visualizzazione risultati
  - ○ blocco di selezione

---

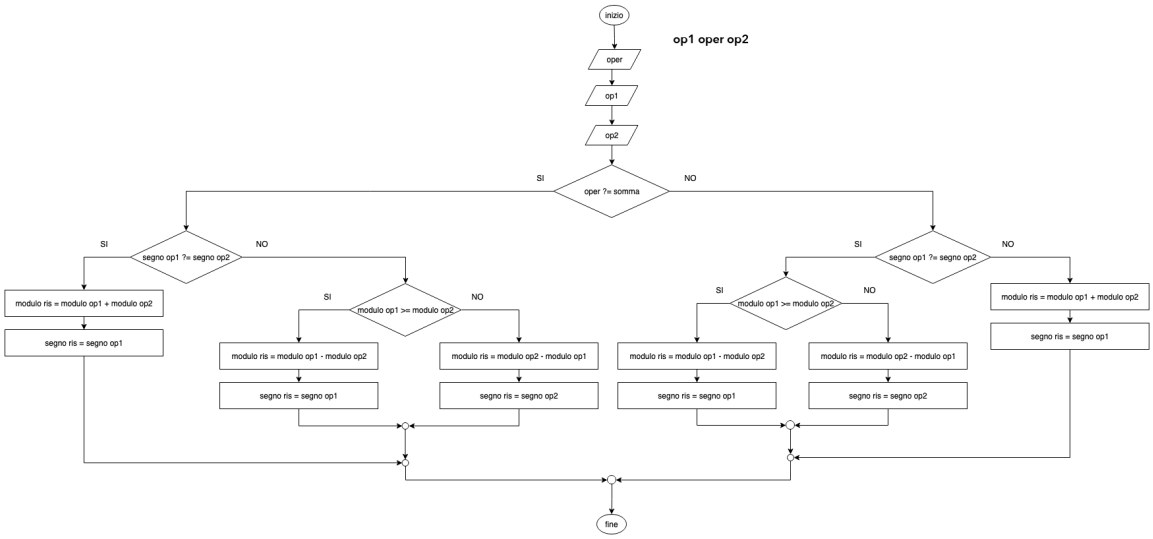
## Esercizio 4.1: Tempo espresso in ore, minuti e secondi

Si realizzi l'algoritmo che acquisito un valore intero (senz'altro positivo) che rappresenta una quantità di tempo espressa in secondi, di calcola e visualizza la stessa quantità di tempo espressa in ore, minuti e secondi. Per esempio, se si inserisce 3812, l'algoritmo calcola e visualizza 1 (ora) 3 (minuti) 32 (sec).



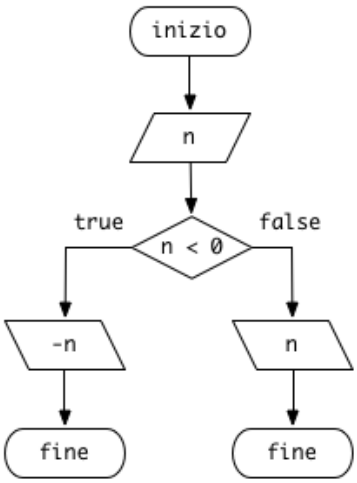
## Esercizio 4.2: Operazioni di somma e sottrazione in modulo e segno

Si realizzi l'algoritmo che acquisito un operatore (`'+'` o `'-'`) e due operandi effettua l'operazione e visualizza il risultato.

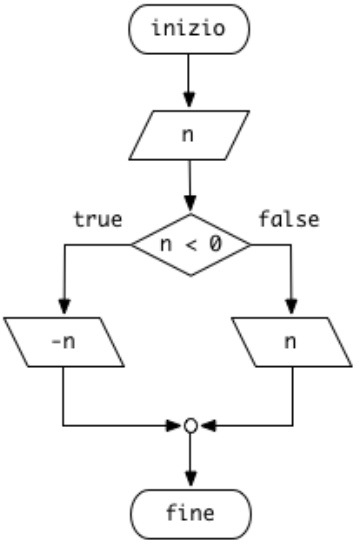



## Esercizio 4.3: Valore assoluto

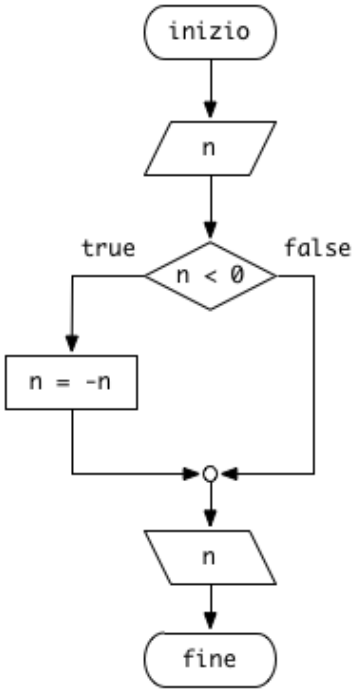
Si realizzi l'algoritmo che acquisito un valore intero calcola e visualizza il suo valore assoluto.



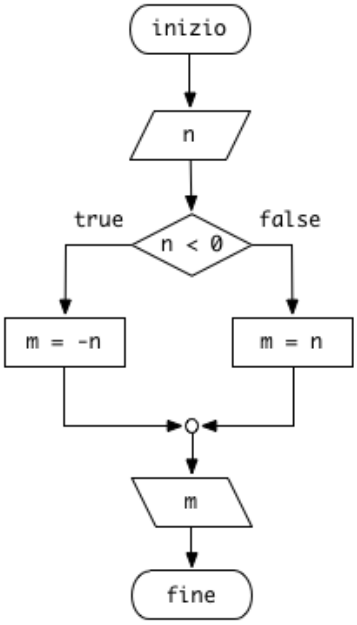

In questo caso l'algoritmo non calcola il valore assoluto, ma lo visualizza direttamente. Guardando la soluzione come una scatola nera, si ottiene il risultato richiesto, quindi soddisfa il comportamento atteso. Non rispetta la richiesta di calcolare il valore e poi visualizzarlo, ma questo aspetto è invisibile esternamente. Ci sono più `fine`, aspetto poco pulito perché diventa difficile riconciliare l'algoritmo.




Anche in questo caso non si calcola il valore assoluto, ma si visualizza il risultato. È più pulito l'utilizzo di un unico punto di `fine`.




Questo algoritmo effettivamente calcola il valore assoluto e poi lo visualizza. Di fatto sovrascrive il valore iniziale con il valore assoluto, quindi eventualmente si perde questa informazione. Dal punto di vista della scatola nera, anche questa soluzione fa ciò che è richiesto.




Questo algoritmo ha il comportamento richiesto e non perde il dato iniziale. L'approccio è migliore non tanto in relazione allo specifico problema (estremamente semplice) ma in termini di "buone" soluzioni, considerando che

- ◇ può essere necessario non tanto visualizzare il valore assoluto, ma utilizzarlo per ulteriori computazioni
- ◇ il "costo" di un elemento in più per memorizzare il valore assoluto è irrisorio.

## Lezione 4 ◇ introduzione al C: struttura, tipi, operatori

24-09-2019

- ◇ struttura di un programma
  - ○ dichiarazione di variabili
  - ○ elaborazione
  - ○ visualizzazione dei risultati
- ◇ istruzioni
- ◇ commenti `/* */`
- ◇ tipi di dati di base: `int`, `float`, `double`, `char`
- ◇ `return 0`
- ◇ `#define`
- ◇ input / output formattato
  - ○ acquisizione formattata `scanf`
  - ○ visualizzazione `printf`

---

### Esercizio 5.1: Tempo in ore, minuti e secondi

Scrivere un programma che acquisito un valore intero che rappresenta un lasso di tempo espresso in secondi calcola e visualizza lo stesso tempo in ore, minuti e secondi.

**Argomenti:** assegnamento. commenti `/* */`, operatori.

```c
#include <stdio.h>

#define MIN_IN_ORA 60
#define SEC_IN_MIN 60

int main(int argc, char * argv[])
{
  int tempo;  /* quantita' di tempo inserita dall'utente */
  int ore, min, sec; /* corrispondenti ore, minuti e secondi */
  int tmp; /* per non rovinare il dato iniziale */

  /* acquisizione */
  scanf("%d", &tempo);

  /* elaborazione */
  ore = tempo / (MIN_IN_ORA * SEC_IN_MIN);
  tmp = tempo % (MIN_IN_ORA * SEC_IN_MIN);
  min = tmp / SEC_IN_MIN;
  sec = tmp % SEC_IN_MIN;

  /* visualizzazione */
  printf("%d %d %d\n", ore, min, sec);

  return 0;
}
```

Versione alternativa. Il numero di operazioni eseguite è lo stesso e le variabili risparmiate non fanno la differenza.

```c
#include <stdio.h>
#define MIN_IN_ORA 60
#define SEC_IN_MIN 60

int main(int argc, char * argv[])
{
  int tempo;  /* quantita' di tempo inserita dall'utente */
  int ore, min, sec; /* corrispondenti ore, minuti e secondi */

  /* acquisizione */
  scanf("%d", &tempo);

  /* elaborazione */
  ore = tempo / (MIN_IN_ORA * SEC_IN_MIN);
  min = (tempo % (MIN_IN_ORA * SEC_IN_MIN)) / SEC_IN_MIN;
  sec = tempo % (MIN_IN_ORA * SEC_IN_MIN) % SEC_IN_MIN;

  /* visualizzazione */
  printf("%d\n%d\n%d\n", ore, min, sec);

  return 0;
}
```

## Lezione 5 ◇ costrutto di selezione `if`, `if-else`, `if-else-if`

26-09-2019

- ◇ cast implicito ed esplicito
- ◇ costrutto di selezione `if`, `if-else`, `if-else-if`
  - valutazione dell'espressione tra parentesi
  - `if (a)` equivale a `if(a != 0)` e `if(!a)` equivale `if(0 == a)`
  - attenzione al == (meglio scrivere `if(_costante_ == _variabile_)`)
  - semantica di `if(a = b)`
  - ordine di valutazione delle condizioni composte
  - relazione tra `else` e `if` in assenza di parentesi
- ◇ De Morgan

---

### Esercizio 6.1: Positivo, negativo o nullo

Scrivere un programma che acquisito un valore visualizza + se positivo, − se negativo, altrimenti, seguito da un carattere a-capo `'\n'`.

**Argomenti:** costrutto `if`

```c
#include <stdio.h>
int main(int argc, char * argv[])
{
  int val;
  char s;

  scanf("%d", &val);
  if (val > 0)
    s = '+';
  else
    if (val < 0)
      s = '-';
    else
      s = ' ';
  printf("%c\n", s);
  return 0;
}
```

Versione alternativa.

```c
#include <stdio.h>
int main(int argc, char * argv[])
{
  int val;
  char s;

  scanf("%d", &val);
  if (val > 0)
    s = '+';
  else if (val < 0)
    s = '-';
  else
```

```
    s = ' ';
  printf("%c\n", s);
  return 0;
}
```

Versione con `#define`.

```
#include <stdio.h>
#define POS '+'
#define NEG '-'
#define NUL ' '
int main(int argc, char * argv[])
{
  int val;
  char s;

  scanf("%d", &val);
  if (val > 0)
    s = POS;
  else if (val < 0)
    s = NEG;
  else
    s = NUL;
  printf("%c\n", s);
  return 0;
}
```

### Esercizio 6.2: Anno bisestile

Scrivere un programma che acquisito un valore intero positivo che rappresenta un anno, visualizza 1 se l'anno è bisestile, 0 altrimenti.

**Argomenti:** algoritmo. divisore/multiplo.

```
#include <stdio.h>
int main(int argc, char * argv[])
{
  int val;
  int ris;

  scanf("%d", &val);
  if ((val % 400 == 0) || (val % 4 == 0) && (val % 100 != 0))
    ris = 1;
  else
    ris = 0;
  printf("%d\n", ris);
  return 0;
}
```

### Esercizio 6.3: Terna pitagorica – Proposto

Scrivere un programma che acquisiti tre valori interi visualizzi 1 se costituiscono una terna pitagorica, 0 altrimenti.

### Esercizio 6.4: Ordinamento crescente/decrescente – Proposto

Scrivere un programma che acquisisca un carattere e tre valori interi. Se il carattere è + visualizza i tre valori in ordine crescente, se è – li visualizza in ordine decrescente.

**Argomenti:** costrutto `if`. algoritmo.

```c
#include <stdio.h>
#define UP '+'
#define DOWN '-'

int main(int argc, char * argv[])
{

  int n1, n2, n3;
  char dir;

  scanf("%c", &dir);
  scanf("%d%d%d", &n1, &n2, &n3);
  if(n1 >= n2)
    if(n2 >= n3){
      min = n3;
      med = n2;
      max = n1;
    } else if (n1 >= n3) {
      min = n2;
      med = n3;
      max = n1;
    } else {
      min = n2;
      med = n1;
      max = n3;
    } else /* n2 > n1 */
    if(n1 >= n3){
      min = n3;
      med = n1;
      max = n2;
    } else if (n2 >= n3) {
      min = n1;
      med = n3;
      max = n2;
    } else {
      min = n1;
      med = n2;
      max = n3;
    }
    if(dir == UP)
      printf("%d\t%d\t%d\n", min, med, max);
    else
      printf("%d\t%d\t%d\n", max, med, min);
    return 0;
}
```

# Laboratorio 0 ◇ introduzione all'ambiente di lavoro

26-09-2019

## Esercizio 7.1: Conversione del tempo in ore, minuti e secondi

Scrivere un programma che acquisito un valore intero positivo che rappresenta una durata temporale espressa in secondi, calcoli e visualizzi lo stesso intervallo di tempo espresso in ore, minuti e secondi.

**Argomenti:** algoritmo

**Ingresso/Uscita:**
**input:** `un numero intero`
**output:** `tre interi separati da uno spazio (seguito da un carattere 'a-capo')`

**Alcuni casi di test per il collaudo:**

**input:** `453`
**output:** `0 7 33`

**input:** `43268`
**output:** `12 1 8`

```c
#include <stdio.h>

#define MIN_IN_ORA 60
#define SEC_IN_MIN 60

int main(int argc, char * argv[])
{
  int tempo;  /* quantita' di tempo inserita dall'utente */
  int ore, min, sec; /* corrispondenti ore, minuti e secondi */
  int tmp; /* per non rovinare il dato iniziale */

  /* acquisizione */
  scanf("%d", &tempo);

  /* elaborazione */
  ore = tempo / (MIN_IN_ORA * SEC_IN_MIN);
  tmp = tempo % (MIN_IN_ORA * SEC_IN_MIN);
  min = tmp / SEC_IN_MIN;
  sec = tmp % SEC_IN_MIN;

  /* visualizzazione */
  printf("%d %d %d\n", ore, min, sec);

  return 0;
}
```

## Esercizio 7.2: Resto

Scrivere un programma che acquisisce un intero e calcola e visualizza il numero di monete da 2 euro, 1 euro, 50, 20, 10, 5, 2 e 1 centesimo.

**Argomenti:** algoritmo

**Ingresso/Uscita:**
**input:** un numero intero
**output:** otto numeri interi

**Alcuni casi di test per il collaudo:**

**input:** 453
**output:** 2 0 1 0 0 0 1 1

**input:** 188
**output:** 0 1 1 1 1 1 1 1

```c
#include <stdio.h>

#define EUR2 200
#define EUR1 100
#define CENT50 50
#define CENT20 20
#define CENT10 10
#define CENT5 5
#define CENT2 2

int main(int argc,  char * argv[])
{

  float eur;
  int cent, r;
  int e2, e1, c50, c20, c10, c5, c2, c1;

  scanf("%f", &eur);
  cent = eur * 100; /* essendo cent una variabile di tipo int, tutte le cifre
   decimali di eur dopo la seconda vengono "tagliate" */

  e2 = cent / EUR2;
  r = cent % EUR2;       /*r %= EUR2*/

  e1 = r / EUR1;
  r = r % EUR1;

  c50 = r / CENT50;
  r = r % CENT50;

  c20 = r / CENT20;
  r = r % CENT20;

  c10 = r / CENT10;
  r = r % CENT10;

  c5 = r / CENT5;
  r = r % CENT5;

  c2 = r / CENT2;

  c1 = r % CENT2;

```

```
43    printf("%d %d %d %d %d %d %d %d\n", e2, e1, c50, c20, c10, c5, c2, c1);
44
45    return 0;
46 }
```

## Lezione 6 ◇ costrutto cicliclo `while` e `do-while`

30-09-2019

- ◇ costrutto ciclico `while`
  - valutazione dell'espressione tra parentesi
  - esecuzione del corpo solo se l'espressione è vera
  - l'istruzione che aggiorna l'espressione è tipicamente l'ultima del corpo del ciclo
- ◇ costrutto ciclico `do-while`
  - valutazione dell'espressione tra parentesi
  - esecuzione del corpo almeno una volta

---

### Esercizio 8.1: Minimo, massimo e valor medio

Scrivere un programma che acquisisce una sequenza di 53 valori interi e calcola e visualizza valor minimo, valor massimo e media dei valori.

**Argomenti:** costrutto `while`

```c
#include <stdio.h>

#define NDATI 53

int main(int argc, char * argv[])
{

  int num, cont;
  int min, max;
  float avg;

  scanf("%d", &num);
  min = num;
  max = num;
  tot = num;
  cont = 1;
  while(cont < NDATI){
    scanf("%d", &num);
    if(num < min)
      min = num;
    else if (num > max)
      max = num;
    tot = tot + num;
    cont++;
  }
  avg = (float)tot / NDATI;
  printf("%d %d %f\n", min, max, avg);
  return 0;
}
```

### Esercizio 8.2: Fattoriale

Chiedere all'utente un valore non negativo, e fino a quando non è tale ripetere la richiesta, quindi calcolare e visualizzare il fattoriale.

**Argomenti:** costrutto `while`, costrutto `do-while`.

```c
#include <stdio.h>

int main(int argc, char * argv[])
{
  int i, n;
  int fatt;

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

  fatt = 1;
  i = 2;
  while(i <= n){
    fatt = fatt * i;
    i++;
  }
  printf("%d\n", fatt);

  return 0;
}
```

versione alternativa che distingue il caso limite $num = 0$ e $num = 1$ inutilmente.

```c
#include <stdio.h>

int main(int argc, char * argv[])
{
  int i, n;
  int fatt;

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

  if(n == 0 || n == 1)
    fatt = 1;
  else {
    fatt = 1;
    i = 2;
    while(i <= n){
      fatt = fatt * i;
      i++;
    }
  }
  printf("%d\n", fatt);

  return 0;
}
```

versione alternativa che distrugge inutilmente il valore iniziale senza un reale risparmio

```c
#include <stdio.h>

int main(int argc, char * argv[])
{
```

```c
  int n;
  int fatt;

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

  if(n == 0 || n == 1)
    fatt = 1;
  else {
    fatt = 1;
    while(n > 1){
      fatt = fatt * n;
      n--;
    }
  }
  printf("%d\n", fatt);

  return 0;
}
```

# Esercitazione 1 ◇ costrutti elementari di controllo

01-10-2019

## Esercizio 9.1: Conversione in binario, allo specchio

Scrivere un programma che acquisito un valore intero positivo visualizza la sua rappresentazione in base 2 (con i bit in ordine inverso).

**Argomenti:** algoritmo. resto della divisione, `#define`.

```c
#include <stdio.h>

#define BASE 2

int main(int argc, char * argv[])
{

  int val, num;

  scanf("%d", &num);
  val = num;

  while(val > 0){
    printf("%d", val % BASE);
    val = val / BASE;
  }
  printf("\n");

  return 0;
}
```

## Esercizio 9.2: Potenza del 2

Scrivere un programma che acquisito un valore `n` intero positivo calcola e stampa la prima potenza del 2 superiore ad `n`.

**Argomenti:** algoritmo. cicli, `while`, `#define`.

```c
#include <stdio.h>

#define BASE 2

int main(int argc, char * argv[])
{
  int n,p;

    scanf("%d",&n);
    p = 1;
    while(p<n)
        p = BASE*p;

    printf("%d\n",p);
    return 0;
}
```

### Esercizio 9.3: Numero di cifre di un valore

Scrivere un programma che acquisito un valore intero calcola e visualizza il numero di cifre di cui è composto.

```c
#include <stdio.h>

#define BASE 10

int main(int argc, char * argv[])
{

  int num;
  int val, dim;

  do
    scanf("%d", &num);
  while(num < 0);

  if(num == 0)
    dim = 1;
  else {
    dim = 0;
    val = num;
    while(val > 0){
      val = val / BASE;
      dim++;
    }
  }
  printf("%d\n", dim);

    return 0;
}
```

### Esercizio 9.4: Einstein

Si dice che Einstein si divertisse molto a stupire gli amici con il gioco qui riportato.
Scrivete il numero 1089 su un pezzo di carta, piegatelo e datelo ad un amico che lo metta da parte. Ciò che avete scritto non deve essere letto fino a che non termina il gioco.
A questo punto chiedere ad una persona di scrivere 3 cifre qualsiasi (ABC), specificando che la prima e l'ultima cifra devono differire di almeno due ($|A - C| >= 2$). Per fare atmosfera, giratevi e chiudete gli occhi. Una volta scritto il numero di tre cifre, chiedete all'amico di scrivere il numero che si ottiene invertendo l'ordine delle cifre (CBA). A questo punto fate sottrarre dal numero più grande quello più piccolo: $XYZ = |ABC - CBA|$ e fate anche calcolare il numero che si ottiene invertendo le cifre del numero risultante: ZYX. Infine, fate sommare questi ultimi due numeri: XYZ + ZYX e constatate che il risultato è 1089.
Fate estrarre il foglio tenuto da parte fino a questo momento: 1089!
Realizzate un programma che fa i seguenti passi:

- ◇ Chiede all'utente di inserire un numero di 3 cifre, specificando il vincolo tra la prima e l'ultima cifra, e se i vincoli non sono rispettati chiede nuovamente il valore
- ◇ Acquisisce il valore
- ◇ Visualizza il numero inserito e il numero con le cifre in ordine inverso
- ◇ Visualizza la differenza tra i due valori (deve essere un numero positivo)
- ◇ Visualizza il numero con le cifre in ordine inverso

◇ Visualizza il risultato della somma tra questi due valori (dovrebbe essere 1089 per qualsiasi numero di 3 cifre che rispetta il vincolo tra la prima e l'ultima cifra)

```c
#include <stdio.h>

#define BASE 10
#define MAX 999

int main(int argc, char * argv[])
{

  int num, tmp;
  int sx, mid, dx;
  int diff, numinv, diffinv, ris;

  do {
    scanf("%d", &num);
    dx = num % BASE;
    sx = num / (BASE * BASE);
    if(dx - sx > 0)
      diff = dx - sx;
    else
      diff = sx - dx;
  } while(diff < 2 || num > MAX);
  mid = (num / BASE) % BASE;
  numinv = dx * BASE * BASE + mid * BASE + sx;
  if(num > numinv)
    diff = num - numinv;
  else
    diff = numinv - num;
  dx = diff % BASE;
  sx = diff / (BASE * BASE);
  mid = (diff / BASE) % BASE;
  diffinv = dx * BASE * BASE + mid * BASE + sx;
  ris = diff + diffinv;
  printf("%d %d %d\n", numinv, diff, diffinv);
  printf("%d\n", ris);
  return 0;
}
```

## Esercizio 9.5: Conversione in binario, senza array – Proposto e risolto

Acquisire un valore intero e calcolare e visualizzare la sua rappresentazione nel sistema binario, mostrando le cifre nell'ordine corretto. Se l'utente inserisce 6, il valore visualizzato è 110. Il suggerimento è che un numero binario può anche essere visto come un numero in base dieci, ossia 110 in binario può anche essere visto come centodieci in decimale ...

**Argomenti:** algoritmo. resto della divisione, `#define`. moltiplicazioni ripetute per la base.

```c
#include <stdio.h>

#define BASEDST 2
#define BASEOUT 10

int main(int argc, char * argv[])
```

```c
{

  int val, decimale;
  int binario, potenza;
  int bit;

  scanf("%d", &decimale);
  val = decimale;
  binario = 0;
  potenza = 1;

  while(val > 0){
    bit = val % BASEDST;
    binario = binario + bit * potenza;
    potenza = potenza * BASEOUT;
    val = val / BASEDST;
  }
  printf("%d\n", binario);

  return 0;
}
```

### Esercizio 9.6: Conversione in binario, allo specchio, con dimensione

Scrivere un programma che acquisisce un primo valore intero, il dato da convertire in binario, ed un secondo dato intero, il numero di bit su cui rappresentarlo. Il programma visualizza la rappresentazione in base 2 (con i bit in ordine inverso) del valore acquisito, eventualmente aggiungendo bit di padding.

```c
#include <stdio.h>

#define BASE 2

int main(int argc, char * argv[])
{

  int val, num;
  int dim, i;

  scanf("%d", &num);
  scanf("%d", &dim);

  val = num;
  while(dim > 0){
    printf("%d", val % BASE);
    val = val / BASE;
    dim--;
  }
  printf("\n");

  return 0;
}
```

03-10-2019

# Lezione 7 ◇ array monodimensionali e ciclo `for`

07-10-2019

- ◇ array monodimensionali
- ◇ dati di tipo omogeneo
- ◇ dimensione dell'array costante e nota a priori
- ◇ indice degli elementi da `0` a `dimensione` - 1
- ◇ costrutto ciclico a conteggio `for` (a condizione iniziale)
- ◇ le tre parti del costrutto
  - ○ istruzioni prima di iniziare il ciclo
  - ○ condizioni, semplici e composte
  - ○ istruzioni a chiusura del corpo del ciclo
- ◇ separatore per le istruzioni nella prima e terza parte del costrutto
- ◇ parti sono facoltative

---

### Esercizio 10.1: 5 dati in ingresso in ordine inverso

Scrivere un programma che acquisiti 5 numeri interi li visualizza in ordine inverso.

**Argomenti:** array monodimensionale. cicli a conteggio. costrutto `for`.

```c
#include <stdio.h>

int main(int argc, char * argv[])
{

  int n1, n2, n3, n4, n5;

  scanf("%d", &n1);
  scanf("%d%d%d%d", &n2, &n3, &n4, &n5);

  printf("%d%d%d%d%d\n", n5, n4, n3, n2, n1);

  return 0;
}
```

Versione con array.

```c
#include <stdio.h>

#define NELEM 5

int main(int argc, char * argv[])
{

  int dati[NELEM], i;

  i = 0;
  while(i < NELEM){
    scanf("%d", &dati[i]);
    i++;
  }
```

```
  i--;   /* i vale 5 */
  while(i >= 0){
    printf("%d", dati[i]);
    i--;
  }

  return 0;
}
```

### Esercizio 10.2: Sopra soglia

Scrivere un programma che acquisisce 20 valori interi, quindi un intero valore soglia e calcola e visualizza in numero di campioni strettamente superiori alla soglia.

```
#include <stdio.h>

#define NDATI 20

int main(int argc, char * argv[])
{

  int val[BASE], i;
  int thr, cont;

  i = 0;
  while(i < NDATI){
    scanf("%d", &val[i]);
    i++;
  }
  scanf("%d", &thr);
  i = 0;
  cont = 0;
  while(i < NDATI){
    if(val[i] > thr)
      cont++;
    i++;
  }
  printf("%d\n", cont);
  return 0;
}
```

Versione con il costrutto `for`.

```
#include <stdio.h>

#define NDATI 20

int main(int argc, char * argv[])
{

  int val[BASE], i;
  int thr, cont;

  for(i = 0; i < NDATI; i++)
    scanf("%d", &val[i]);
  scanf("%d", &thr);
  cont = 0;
  for(i = 0; i < NDATI; i++)
```

```
16      if(val[i] > thr)
17        cont++;
18    printf("%d\n", cont);
19    return 0;
20  }
```

### Esercizio 10.3: Conversione in binario – Proposto

Scrivere un programma che acquisisce un valore compreso tra 0 e 1023, estremi inclusi e finché non è tale lo richiede. Quindi effettua la conversione in base 2 e la visualizza.

**Argomenti:** algoritmo. validazione ingresso. dimensione dell'array.

```
1  #include <stdio.h>
2  #define MAX 1023
3  #define MIN 0
4  #define SIZE 10
5  #define BASE 2
6
7  int main(int argc, char * argv[])
8  {
9
10   int dec, n;
11   int bin[SIZE], i;
12
13   do
14     scanf("%d", &dec);
15   while(dec < MIN || dec > MAX);
16
17   n = dec;
18
19   i = SIZE - 1;
20   while(n > 0){
21     bin[i] = n % BASE;
22     n = n / BASE;
23     i--;
24   }
25   if(dec == 0){
26     bin[0] = 0;
27     i--;
28   }
29
30   for(i++; i < SIZE; i++)
31     printf("%d", bin[i]);
32   printf("\n");
33
34   return 0;
35 }
```

Versione alternativa con ciclo `do-while`.

```
1  #include <stdio.h>
2  #define MAX 1023
3  #define MIN 0
4  #define SIZE 10
```

```c
#define BASE 2

int main(int argc, char * argv[])
{

  int dec, n;
  int bin[SIZE], i;

  do
    scanf("%d", &dec);
  while(dec < MIN || dec > MAX);

  n = dec;
  i = 0;
  do{
    bin[i] = n % BASE;
    n = n / BASE;
    i++;
  } while(n > 0);

  for(i--; i >= 0; i--)
    printf("%d", bin[i]);
  printf("\n");

  return 0;
}
```

Versione alternativa in cui si visualizza il numero sul numero completo di bit massimo.

```c
#include <stdio.h>
#define MAX 1023
#define MIN 0
#define SIZE 10
#define BASE 2

int main(int argc, char * argv[])
{

  int dec;
  int bin[SIZE], i;

  do
    scanf("%d", &dec);
  while(dec < MIN || dec > MAX);

  for(i = SIZE-1; i >= 0; i--){
    bin[i] = dec % BASE;
    dec = dec / BASE;
  }

  for(i=0; i < SIZE; i++)
    printf("%d", bin[i]);
  printf("\n");

  return 0;
}
```

# Esercitazione 2 ◇ array monodimensionali

08-10-2019

## Esercizio 11.1: Numero allo specchio

Scrivere un programma in C che acquisito un valore intero calcola e visualizza il valore ottenuto invertendo l'ordine delle cifre che lo compongono. Ad esempio, se il programma acquisisce il valore 251, il programma visualizza 152. Si ipotizzi (non si devono far verifiche in merito) che il valore acquisito non dia problemi di overflow e sia senz'altro strettamente positivo. Se l'utente inserisce il valore 1000, il programma visualizza 1 (questo perchè CALCOLA e poi visualizza). Se l'utente inserisce 9001 il programma visualizza 1009.

**Argomenti:** algoritmo.

*Tratto dal tema d'esame del* 04/09/2015 (Variante)

```c
#include <stdio.h>

#define BASE 10

int main(int argc, char * argv[])
{
  int num, rev;
  int tmp, digit;

  scanf("%d", &num);

  tmp = num;
  rev = 0;
  while(tmp > 0){
    digit = tmp % BASE;
    rev = rev * BASE + digit;
    tmp = tmp / BASE;
  }

  printf("%d\n", rev);

  return 0;
}
```

## Esercizio 11.2: Conta lettere

Scrivere un programma che acquisisce una sequenza di 50 caratteri e per ogni carattere letto mostra quante volte compare nella sequenza, mostrandoli in ordine alfabetico. Si consideri che i caratteri inseriti siano tutti caratteri minuscoli. Per esempio, se l'utente inserisce `sequenzadiprova` (solo 16 caratteri in questo caso, per questioni di leggibilità), il programma visualizza

```
a 2
d 1
e 2
i 1
n 1
```

```
o 1
p 1
q 1
r 1
s 1
u 1
v 1
z 1
```

**Argomenti:** array monodimensionale. cicli a conteggio. contatori di occorrenze. codice ASCII di un carattere. costrutto `for`.

```c
#include <stdio.h>

#define NELEM 16
#define LALPHA 26
#define INIT 'a'

int main(int argc, char * argv[])
{

  char seq[NELEM];
  int cont[LALPHA];
  int i, pos;

  for(i = 0; i < NELEM; i++)
    scanf("%c", &seq[i]);

  for(i = 0; i < LALPHA; i++)
    cont[i] = 0;

  for(i = 0; i < NELEM; i++){
    pos = seq[i] - INIT;
    cont[pos]++;
  }

  for(i = 0; i < LALPHA; i++)
    if(cont[i] > 0)
      printf("%c %d\n", INIT+i, cont[i]);

  return 0;
}
```

### Esercizio 11.3: Niente primi

Scrivere un programma che acquisisce una sequenza di al più 50 valori interi strettamente maggiori di 1 e che si ritiene terminata quando l'utente inserisce un valore minore o uguale a 1. Il programma, una volta acquisiti i dati, visualizza 1 se tra di essi non c'è alcun numero primo, 0 altrimenti.

**Argomenti:** array monodimensionale. cicli annidati. numero primo.

```c
#include <stdio.h>

```

```c
#define NMAX 50
#define LIMIT 0

int main(int argc, char * argv[])
{
  int val[NMAX], i;
  int num, dim;
  int prime, isdiv, div, half;

  dim = 0;
  scanf("%d", &num);
  while(num > LIMIT){
    val[dim] = num;
    dim++;
    scanf("%d", &num);
  }
  prime = 0;
  for(i = 0; i < dim && prime == 0; i++){
    printf("%d\n", val[i]);
    /* verifica se il numero e' primo */
    half = val[i] / 2;
    div = 2;
    /*
    do {
      isdiv = val[i] % div;
      div++;
    }(div < half && isdiv != 0);
    */
    isdiv = 1;
    while(isdiv && div < half){
      isdiv = val[i] % div;
      div++;
    }
    if(isdiv != 0)
      prime = 1;
  }
  printf("%d\n", !prime);

  return 0;
}
```

## Lezione 8 ◇ array bidimensionali, `struct` e `typedef`

10-10-2019

- ◇ array pluri-dimensionali: dimensione 2
  - scansione mediante due cicli annidati
  - linearizzazione della memoria
- ◇ `struct`
  - nome facoltativo, ma sempre utilizzato
  - array di `struct`
  - `struct` con campo array
- ◇ `typedef`

### Esercizio 12.1: Matrice identità

Si scriva un programma che acquisisce i dati di una matrice di dimensione `5x5` di valori interi. Il programma visualizza 1 se si tratta di una matrice identità, 0 altrimenti.

**Argomenti:** array bidimensionali. algoritmo.

```c
#include <stdio.h>

#define N 5

int main(int argc, char * argv[])
{
  int mat[N][N];
  int i, j;
  int isid;

  for(i = 0; i < N; i++)
    for(j = 0; j < N; j++)
      scanf("%d", &mat[i][j]);

  isid = 1;
  for(i = 0; i < N && isid; i++)
    for(j = 0; j < N && isid; j++)
      if(i == j && mat[i][j] != 1 || i != j && mat[i][j] != 0)
        isid = 0;

      /*
      if(i == j) {
        if(mat[i][j] != 1)
          isid = 0;
      } else
        if(mat[i][j] != 0)
          isid = 0;
      */

  printf("%d\n", isid);

  return 0;
}
```

### Esercizio 12.2: Date: definizione di tipo

Definire un nuovo tipo di dato, cui dare nome `date_t` , per rappresentare date in termini di giorno, mese ed anno.

**Argomenti:** `typedef`.

```
typedef struct date_s {
   int g, m, a;
} date_t;
```

### Esercizio 12.3: Studente: definizione di tipo

Definire un nuovo tipo di dato, cui dare nome `student_t` per rappresentare le informazioni seguenti relative ad uno studente:

- ◇ cognome e nome: due array di caratteri di 41 caratteri ciascuno
- ◇ data di nascita e data di immatricolazione: due data
- ◇ voti esami: un array di NESAMI interi (dovete saperlo voi)
- ◇ media: voto medio (relativo agli esami superati)
- ◇ livello: 'T' o 'M' per distinguere se si è immatricolati alla triennale o alla magistrale.

**Argomenti:** `typedef`.

```
#define NL 41
#define NESAMI 18
typedef struct student_s {
   char last[NL+1], first[NL]+1;
   date_t dnascita, dlaurea;
   int grades[NEX];
   float avg;
   char level;
} student_t;
```

# Lezione 9 ◇ stringhe

14-10-2019

- ◇ sequenza di caratteri delimitata dal terminatore '\0'
- ◇ allocazione dell'array con `N+1` elementi
- ◇ acquisizione
  - `scanf`: segnaposto `%s` e niente `&` davanti al nome dell'array
  - `gets`: serve poi solo il nome dell'array, trattandosi *sempre* di caratteri
- ◇ scansione con condizione `s[i] != '\0'`

---

### Esercizio 13.1: Stringa allo specchio

Scrivere un programma che acquisisce una stringa di al più 25 caratteri, inverte l'ordine di tutti i caratteri in essa contenuta e la visualizza.

**Argomenti:** stringa. algoritmo.

```c
#include <stdio.h>

#define N 25

int main(int argc, char * argv[])
{
  char seq[N+1], tmp;
  int i, j;

  scanf("%s", seq);

  for(j = 0; seq[j] != '\0'; j++)
    ;

  /* oppure */
  /*
  j = 0;
  while(seq[j] != '\0')
    j++;
  */

  for(i = 0, j--; i < j; i++, j--){
    tmp = seq[i];
    seq[i] = seq[j];
    seq[j] = tmp;
  }

  printf("%s\n", seq);

  return 0;

}
```

### Esercizio 13.2: Stringa allo specchio senza vocali

Scrivere un programma che acquisisce una stringa di al più 25 caratteri, inverte l'ordine di tutti i caratteri ed elimina tutte le vocali in essa contenute. Ci sono solo caratteri minuscoli.

**Argomenti:** stringa. algoritmo.

```c
#include <stdio.h>
#define N 25

int main(int argc, char * argv[])
{
  char seq[N+1], tmp;
  int i, j, inv;

  scanf("%s", seq);

  for(j = 0; seq[j] != '\0'; j++)
    ;

  for(i = 0, j--; i < j; i++, j--){
    tmp = seq[i];
    seq[i] = seq[j];
    seq[j] = tmp;
  }

  for(i = 0; seq[i] != '\0'; )
    if(seq[i] == 'a' || seq[i] == 'e' || seq[i] == 'i' || seq[i] == 'o' || seq[i
    ] == 'u')
      for(j = i; seq[j] != '\0'; j++) /* vengono spostati tutti di 1 */
        seq[j] = seq[j+1];
    else
      i++;
  seq[i] = '\0';

  scanf("%s", seq);
  for(i = 0, inv = 0; seq[inv] != '\0'; inv++)
    if(seq[inv] != 'a' && seq[inv] != 'e' && seq[inv] != 'i' && seq[inv] != 'o'
    && seq[inv] != 'u'){
      seq[i] = seq[inv];
      i++;
    }
  seq[i] = '\0';

  printf("%s\n", seq);
  return 0;
}
```

versione alternativa

```c
#include <stdio.h>
#define N 25

int main(int argc, char * argv[])
{
  char seq[N+1], tmp;
  int i, inv;
```

```
8
9   scanf("%s", seq);
10
11  for(inv = 0; seq[inv] != '\0'; inv++)
12    ;
13
14  for(i = 0, inv--; i < inv; i++, inv--){
15    tmp = seq[i];
16    seq[i] = seq[inv];
17    seq[inv] = tmp;
18  }
19
20  for(i = 0, inv = 0; seq[inv] != '\0'; inv++)
21    if(seq[inv] != 'a' && seq[inv] != 'e' && seq[inv] != 'i' && seq[inv] != 'o'
     && seq[inv] != 'u'){
22      seq[i] = seq[inv];
23      i++;
24    }
25  seq[i] = '\0';
26
27  printf("%s\n", seq);
28  return 0;
29 }
```

### Esercizio 13.3: Anagrammi – Proposto e risolto

Scrivere un programma che acquisisce due stringhe di al più 20 caratteri e determina se una sia l'anagramma dell'altra, visualizzando 1 in caso affermativo, 0 altrimenti.

**Argomenti:** algoritmo. stringhe. dimensione dell'array.

```
1  #include <stdio.h>
2
3  #define N 20
4  #define USED 1
5  #define UNUSED 0
6
7  int main(int argc, char * argv[])
8  {
9
10   char v1[N+1], v2[N+2];
11   int used[N], i, j, len2;
12   int found;
13
14   scanf("%s%s", v1, v2);
15
16   for(i = 0; v2[i] != '\0'; i++)
17     used[i] = UNUSED;
18   len2 = i;
19
20   found = 1;
21   for(i = 0; v1[i] != '\0' && found; i++){
22     found = 0;
23     for(j = 0; v2[j] != '\0' && !found; j++)
24       if(v1[i] == v2[j] && used[j] == UNUSED){
```

```
25          used[j] = USED;
26          found = 1;
27        }
28    }
29    if(i < len2)
30      found = 0;
31    /*
32    for(i = 0; v2[i] != '\0' && found; i++)
33      if(used[i] == UNUSED)
34        found = 0;
35    */
36    printf("%d\n", found);
37    return 0;
38  }
```

# Analisi esercizi 1 ◇ array

15-10-2019

## Esercizio 14.1: Numeri vicini

Scrivere un programma in C che chiede all'utente di inserire una sequenza di numeri interi terminata dallo 0 (lo 0 non fa parte della sequenza). Il programma ignora i valori negativi e valuta e stampa a video le coppie di numeri consecutivi che soddisfano tutte le condizioni che seguono:

- ◇ sono diversi tra di loro,
- ◇ sono entrambi numeri pari,
- ◇ il loro prodotto è un quadrato perfetto.

**Argomenti:** algoritmo. numeri pari. quadrato perfetto

```c
#include <stdio.h>
#define STOP 0

int main(int argc, char * argv[])
{
  int prec, last;
  int prod;
  int i, isq;

  do
    scanf("%d", &prec);
  while(prec < 0);

  if(prec > 0){
    scanf("%d", &last);
    while(last != STOP){
      if(last > 0 && prec > 0)
        if(last != prec)
          if(last % 2 == 0 && prec % 2 == 0){
            prod = last * prec;
            i = 2;
            isq = i*i;
            while(isq < prod){
              i++;
              isq = i*i;
            }
            if(isq == prod)
              printf("%d %d\n", prec, last);
            /*
            for(i = 2; i*i < prod; i++)
              ;
            if(i*i == prod)
              printf("%d %d", prec, last);
            */
          }

      prec = last;
      scanf("%d", &last);
    }
  }
  return 0;
}
```

## Esercizio 14.2: Distanza di Hamming

Scrivere un programma che acquisiti due sequenze di 10 bit ciascuna (forniti una cifra alla volta), calcola e visualizza il vettore della distanza di Hamming, ed infine la distanza. Un esempio di esecuzione è il seguente:

```
1 0 0 1 0 0 1 0 0 1
1 1 1 1 0 0 0 1 0 1
0 1 1 0 0 0 1 1 0 0    4
```

**Argomenti:** array monodimensionale. cicli a conteggio. costrutto `for`.

```c
#include <stdio.h>
#define NELEM 10

int main(int argc, char * argv[])
{

  int s1[NELEM], s2[NELEM];
  int ham[NELEM], i, dist;

  for(i = 0; i < NELEM; i++)
    scanf("%d", &s1[i]);

  for(i = 0; i < NELEM; i++)
    scanf("%d", &s2[i]);

  dist = 0;
  for(i = 0; i < NELEM; i++)
    if(s1[i] == s2[i])
      ham[i] = 0;
    else {
      ham[i] = 1;
      dist++;
    }

  for(i = 0; i < NELEM; i++)
    printf("%d ", ham[i]);

  printf("  %d\n", dist);
  return 0;
}
```

## Esercizio 14.3: Carta di credito

Scrivere un programma che acquisisce un numero di carta di credito Visa costituito da 13 o 16 caratteri numerici e verifica che si tratti di un numero di carta valido oppure no. Nel primo caso visualizza 1, altrimenti 0. L'algoritmo che verifica la correttezza "sintattica" di un numero (inventato da Hans Peter Luhn di IBM) è il seguente:

- ◇ moltiplicare una cifra si, una no, per 2 partendo dalla penultima a destra, quindi sommare tali cifre,
- ◇ sommare a tale valore, le cifre che non sono state moltiplicate per 2,
- ◇ se il valore ottenuto è un multiplo di 10, il numero è valido.

Per esempio, si consideri il numero di carta di credito seguente: $4003600000000014$.

- ◇ moltiplicare una cifra si, una no, per 2 partendo dalla penultima a destra, quindi sommare tali cifre (sono sottolineate le cifre da moltiplicare per 2):
  4003600000000014
  si moltiplica ogni cifra per 2:
  $1 \times 2 + 0 \times 2 + 0 \times 2 + 0 \times 2 + 0 \times 2 + 6 \times 2 + 0 \times 2 + 4 \times 2$
  si ottiene:
  $2 + 0 + 0 + 0 + 0 + 12 + 0 + 8$
  si sommano le cifre:
  $2 + 0 + 0 + 0 + 0 + 1 + 2 + 0 + 8 = 13$
- ◇ sommare a tale valore, le cifre che non sono state moltiplicate per 2,
  $13 + 4 + 0 + 0 + 0 + 0 + 0 + 3 + 0 = 20$
- ◇ se il valore ottenuto è un multiplo di 10, il numero è valido: in questo caso le è.

Potete provare con alcuni numeri suggeriti da PayPal: 4111111111111111, 4012888888881881, 4222222111212222.

**Argomenti:** array di caratteri. corrispondenza carattere numerico valore. cicli a conteggio. cifre di un numero.

```c
#include <stdio.h>

#define LEN 16
#define BASE 10

int main(int argc, char * argv[])
{

    char numcc[LEN];
    int i, tot, dim, digit;
    int ris;

  for(i = 0; i < LEN; i++)
    scanf("%c", &numcc[i]);

    tot = 0;
    i = LEN - 2;
    for(; i >= 0; i = i-2){
        digit = (numcc[i] - '0')*2;
        if(digit < BASE)
            tot = tot + digit;
        else
            tot = tot + digit / BASE + digit % BASE;
    }

    for(i = LEN - 1; i >= 0; i = i-2)
        tot = tot + (numcc[i] - '0');

    /* alternativa */
    for(i = LEN-1; i >= 0; i = i - 2){
        tot = tot + (numcc[i] - '0');
        digit = (numcc[i-1] - '0')*2;
        if(digit < BASE)
```

```
            tot = tot + digit;
        else
            tot = tot + digit / BASE + digit % BASE;
    }

    /* NON COSI'
    for(i = LEN-1; i >= 0; i--){
        if(i % 2)
            tot = tot + (numcc[i] - '0');
        else {
            digit = (numcc[i] - '0')*2;
            if(digit < BASE)
                tot = tot + digit;
            else
                tot = tot + digit / BASE + digit % BASE;
        }
    }
    */


    if(tot % BASE)
        ris = 0;
    else
        ris = 1;

    printf("%d\n", ris);
    return 0;
}
```

### Esercizio 14.4: Media mobile

Si scriva un programma che acquisiti 100 valori interi ed un valore intero `n` calcola e visualizza l'array di valori che costituiscono la media mobile dei dati in ingresso di finestra `n`. L'elemento `i`-esimo della media mobile viene calcolato come media degli `n` valori del vettore in ingresso che precedono e includono l'elemento `i`. Se l'elemento `i` è preceduto da meno di `n`-1 valori, la media si calcola su quelli.

**Argomenti:** array monodimensionali. calcolo della media.

*Tratto dal tema d'esame del* 03/07/2017 (variante)

```
#include <stdio.h>
#define N 10

int main(int argc, char * argv[])
{

  int v[N], n;
  int i, j, tot;
  float m[N];

  for(i = 0; i < N; i++)
    scanf("%d", &v[i]);

```

```c
  scanf("%d", &n);

  m[0] = v[0];
  for(i = 1; i < n; i++){
    for(j = 0, tot = 0; j <= i; j++)
      tot = tot + v[j];
    m[i] = (float) tot / j;
  }

  for(; i < N; i++){
    for(j = 0, tot = 0; j < n; j++)
      tot = tot + v[i-j];
    m[i] = (float) tot / n;
  }

  for(i = 0; i < N; i++)
    printf("%f ", m[i]);

  printf("\n");

  return 0;
}
```

### Esercizio 14.5: Conta caratteri

Scrivere un programma in C che acquisisca una sequenza `str` di 20 caratteri. Per ogni carattere `car` contenuto nella stringa `str`, a partire dall'ultimo fino ad arrivare al primo, il programma visualizza (senza lasciare spazi) il carattere `car`, seguito dal numero di volte in cui compare consecutivamente in quel punto della stringa. Per esempio, se l'utente inserisce `aabbbddbbbbhhhhhzzzz` il programma visualizza `z4h5b4d2b3a2`.

**Argomenti:** array monodimensionali. caratteri. algoritmo.

*Tratto dal tema d'esame del* 03/07/2017 (variante)

```c
#include <stdio.h>
#define N 20

int main(int argc, char * argv[])
{
  char str[N], car;
  int i, count;

  for(i = 0; i < N; i++)
    scanf("%c", &str[i]);
  i--;

  car = str[i];
  count = 1;
  for(i--; i >= 0; i--){
    if(str[i] == car)
      count++;
    else {
      printf("%c%d", car, count);
```

```
      count = 1;
      car = str[i];
    }
  }
  printf("%c%d\n", car, count);
  return 0;
}
```

Versione alternativa

```
#include <stdio.h>
#define N 20

int main(int argc, char * argv[])
{
  char str[N], car;
  int i, len, count;

  for(i = 0; i < N; i++)
    scanf("%c", &str[i]);

  i--;
  car = str[i];
  while(i >= 0){
    count = 0;
    while(i >= 0 && str[i] == car){
      count++;
      i--;
    }
    printf("%c%d", car, count);
    car = str[i];
  }
  printf("\n");
  return 0;
}
```

Versione alternativa con il costrutto `for`

```
#include <stdio.h>
#define N 20

int main(int argc, char * argv[])
{
  char str[N], car;
  int i, len, count;

  for(i = 0; i < N; i++)
    scanf("%c", &str[i]);

  i--;
  car = str[i];
  while(i >= 0){
    for(count = 0; i >= 0 && str[i] == car; i--, count++)
      ;
    printf("%c%d", car, count);
    car = str[i];
  }
  printf("\n");
  return 0;
```

```
22 }
```

# Esercitazione 3 ◇ array e stringhe

17-10-2019

## Esercizio 15.1: Determinante di una matrice dimensione 3

Scrivere un programma che acquisisce i dati di una matrice di dimensione `3x3` di numeri interi, quindi calcola e visualizzare il determinante.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$det(A) = a_{11} \times a_{22} \times a_{33} + a_{12} \times a_{23} \times a_{31} + a_{13} \times a_{21} \times a_{32} - a_{13} \times a_{22} \times a_{31} - a_{12} \times a_{21} \times a_{33} - a_{11} \times a_{23} \times a_{32}$$

**Argomenti:** array bidimensionale.

## Esercizio 15.2: Nome di file da percorso, nome ed estensione

Scrivere un programma che acquisisce una stringa di al più 50 caratteri che contenga il nome di un file completo di percorso e visualizza il solo nome.

**Argomenti:** stringa. algoritmo.

*Tratto dal tema d'esame del*

```c
#include <stdio.h>

#define N 50
#define SEPU '/'
#define SEPW '\\'

int main(int argc, char * argv[])
{
  char seq[N+1];
  int i, s;

  gets(seq);

  s = -1;
  for(i = 0; seq[i] != '\0'; i++)
      if(seq[i] == SEPU || seq[i] == SEPW)
        s = i;

  /* s indica l'ultimo separatore o -1 se non ce ne sono */
  printf("%s\n", &seq[s+1]);
  /*
  for(i = s+1; seq[i] != '\0'; i++)
    prinf("%c", seq[i]);
  printf("\n");
  */
  return 0;
}
```

oppure:

```c
#include <stdio.h>

#define N 50
#define SEPU '/'
#define SEPW '\\'

int main(int argc, char * argv[])
{
  char seq[N+1];
  int i, s;

  gets(seq);

  for(i = 0; seq[i] != '\0'; i++)
    ;

  i--;
  while(seq[i] != SEPU && seq[i] != SEPW && i>=0)
    i--;

  /* i indica l'ultimo separatore o -1 se non ce ne sono */
  printf("%s\n", &seq[i+1]);

  return 0;
}
```

### Esercizio 15.3: Da intero a stringa

Scrivere un programma che acquisito un intero crea una stringa che contiene le cifre dell'intero. L'intero ha al più 6 cifre.

**Argomenti:** stringa. algoritmo. carattere numerico.

```c
#include <stdio.h>

#define N 9
#define BASE 10

int main(int argc, char * argv[])
{

  char seqn[N+1];
  int val;
  int tmp, nc, i;

  scanf("%d", &val);

  /* inseriti in ordine inverso */
  nc = 0;
  while(val > 0){
    seqn[nc] = val % BASE + '0';
    val = val / BASE;
    nc++;
  }
```

```
  seqn[nc] = '\0';

  /* ribalto la stringa */
  for(i = 0; i < nc / 2; i++){
    tmp = seqn[i];
    seqn[i] = seqn[nc-1-i];
    seqn[nc-1-i] = tmp;
  }

  printf("%s\n", seqn);
  return 0;
}
```

Versione alternativa.

```
#include <stdio.h>

#define N 9
#define BASE 10

int main(int argc, char * argv[])
{

  char seqn[N+1];
  int val;
  int tmp, nc, i;

  scanf("%d", &val);

  /* conto il numero di cifre */
  nc = 0;
  tmp = val;
  while(val > 0){
    val = val / BASE;
    nc++;
  }
  seqn[nc] = '\0';

  /* scrivo le cifre da quella meno significativa */
  for(i = nc-1; i >= 0; i--){
    seqn[i] = tmp % BASE + '0';
    tmp = tmp / BASE;
  }

  printf("%s\n", seqn);
  return 0;
}
```

### Esercizio 15.4: Area di un poligono

Scrivere un programma che calcola e visualizza l'area di un poligono, a partire dalle coordinate dei suoi vertici e usando la formula di Gauss. Il poligono ha al più 10 vertici, ciascuno rappresentato dalle coordinata `x` e `y` (valori interi). Il programma acquisisce prima il numero `num` di vertici del poligono, verificando che sia non superiore a 10, quindi acquisisce le coordinate dei `num` vertici, quindi procede al calcolo dell'area e la visualizza. Si dichiari un opportuno tipo di dato (`vertex_t`) per rappresentare i vertici del poligono.

```c
#include <stdio.h>
#define VMIN 3
#define VMAX 10

typedef struct vertex_s {
  int x, y;
} vertex_t;

int main(int argc, char * argv[])
{
  vertex_t polig[VMAX];
  int num, i;
  int tot;
  float area;

  do
    scanf("%d", &num);
  while (num < VMIN || num > VMAX);

  for(i = 0; i < num; i++)
    scanf("%d%d", &polig[i].x, &polig[i].y);

  /* calcolo dell'area */
  tot = 0;
  /* contributi positivi */
  for(i = 0; i < num-1; i++)
    tot = tot + polig[i].x * polig[i+1].y;
  tot = tot + polig[i].x * polig[0].y;
  /* contributi negativi */
  for(i = 1; i < num; i++)
    tot = tot - polig[i].x * polig[i-1].y;
  tot = tot - polig[0].x * polig[i-1].y;
  if(tot < 0)
    tot = -tot;
  area = tot / 2.0;

  printf("%f\n", area);

  return 0;
}
```

### Esercizio 15.5: Quadrato magico

Si scriva un programma che acquisisce i dati di una matrice di dimensione `3x3` di valori interi. Il programma visualizza 1 se si tratta di un quadrato magico seguito dal valore della somma, 0 altrimenti. Un quadrato magico è un insieme di numeri interi distinti in una tabella quadrata tale che la somma dei numeri presenti in ogni riga, in ogni colonna e in entrambe le diagonali dia sempre lo stesso valore.

**Argomenti:** array bidimensionali. algoritmo.

```c
#include <stdio.h>

#define DIM 3

```

```c
int main(int argc, char * argv[])
{

  int m[DIM][DIM], i, j;
  int sum, ism, tot;

  for(i = 0; i < DIM; i++)
    for(j = 0; j < DIM; j++)
      scanf("%d", &m[i][j]);

  /* prima somma - diagonale */
  sum = 0;
  for(i = 0; i < DIM; i++)
    sum += m[i][i];

  /* antidiagonale */
  for(i = 0, tot = 0; ism && i < DIM; i++)
    tot += m[i][DIM-1-i];
  if(tot != sum)
    ism = 0;
  else
    ism = 1;

  /* righe */
  for(i = 0; ism && i < DIM; i++){
    for(j = 0, tot = 0; j < DIM; j++)
      tot += m[i][j];
    if(tot != sum)
      ism = 0;
  }

    /* colonne */
  for(i = 0; ism && i < DIM; i++){
    for(j = 0, tot = 0; j < DIM; j++)
      tot += m[j][i];
    if(tot != sum)
    ism = 0;
  }

  printf("%d", ism);
    if(ism)
      printf(" %d", sum);

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

Versione alternativa.

```c
#include <stdio.h>

#define DIM 3

int main(int argc, char * argv[])
{

  int m[DIM][DIM], i, j;
  int sum, ism, tot, totcol;
```

```
10
11    for(i = 0; i < DIM; i++)
12      for(j = 0; j < DIM; j++)
13        scanf("%d", &m[i][j]);
14
15    /* diagonale e antidiagonale */
16    sum = 0;
17    for(i = 0; i < DIM; i++){
18      sum += m[i][i];
19      tot += m[i][DIM-1-i];
20    }
21    if(tot != sum)
22      ism = 0;
23    else
24      ism = 1;
25
26    /* righe e colonne */
27    for(i = 0; ism && i < DIM; i++){
28      for(j = 0, tot = 0, totcol; j < DIM; j++){
29        tot += m[i][j];
30        totcol += m[j][i];
31      }
32      if(tot != sum || totcol != sum)
33        ism = 0;
34    }
35
36    printf("%d", ism);
37    if(ism)
38      printf(" %d", sum);
39
40    printf("\n");
41    return 0;
42  }
```

# Laboratorio 1 ◇ algoritmi e array

17-10-2019

## Esercizio 16.1: Fattori

Scrivere un programma che acquisisce due numeri interi relativi e visualizza 1 se uno è un divisore dell'altro o viceversa, 0 altrimenti. Dopo il valore visualizzato, mettere un `'a-capo'`.

**Ingresso/Uscita:**
**input:** `due numeri interi`
**output:** `un intero (seguito da un carattere 'a-capo')`

**Alcuni casi di test per il collaudo:**

**input:** `5542 18`
**output:** `0`

**input:** `5542 17`
**output:** `1`

**input:** `13 1950`
**output:** `1`

```c
#include <stdio.h>

int main(int argc, char * argv[])
{
    int n1, n2;
    int ris;

    scanf("%d", &n1);
    scanf("%d", &n2);

    if(n1 && n2)
        if(n1 % n2 == 0 || n2 % n1 == 0)
            ris = 1;
        else
            ris = 0;
    else
        ris = 0;

    printf("%d\n", ris);
    return 0;
}
```

## Esercizio 16.2: Padding

Si vuole rappresentare a video un valore naturale `num` utilizzando un numero a scelta di cifre `k` inserendo 0 nelle posizioni più significative, fino a raggiungere la dimensione desiderata. Per esempio, volendo rappresentare `842` su `5` cifre, si ottiene `00842`.
Scrivere un programma che acquisisce due valori interi entrambi strettamente positivi (e finché non è così richiede il valore che non rispetta il vincolo) num e `k`, quindi rappresenta `num` su `k` cifre. Se `k` e' minore del numero di cifre presenti in `num`, il programma visualizza il valore `num` come è. Dopo il valore visualizzato, mettere un `'a-capo'`.

**Ingresso/Uscita:**
**input:** `due numeri interi (da verificare)`

**output:** `un intero (seguito da un carattere 'a-capo')`

**Alcuni casi di test per il collaudo:**

**input:** `11304 9`
**output:** `000011304`

**input:** `-4 9000 -5 -2 2`
**output:** `9000`

**input:** `1 1`
**output:** `1`

```c
#include <stdio.h>
#define BASE 10

int main(int argc, char * argv[])
{
    int num, k;
    int val, i, ncifre;

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

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

    val = num;
    i = 0;
    while(val > 0){
        val = val/BASE;
        i++;
    }
    for(; k > i; i--)
        printf("0");
    printf("%d\n", num);

    return 0;
}
```

## Esercizio 16.3: Super Mario


Nella preistoria dei videogiochi in Super Mario della Nintendo, Mario deve saltare da una piramide di blocchi a quella adiacente. Proviamo a ricreare le stesse piramidi in C, in testo, utilizzando il carattere cancelletto (#) come blocco, come riportato di seguito. In realtà il carattere # è più alto che largo, quindi le piramidi saranno un po' più alte.

```
   #  #
  ##  ##
 ###  ###
####  ####
```

Notate che lo spazio tra le due piramidi è sempre costituito da **2** spazi, indipendentemente dall'altezza delle piramidi. Inoltre, alla fine delle piramidi **non ci devono essere spazi**. L'utente inserisce

l'altezza delle piramidi, che deve essere un valore strettamente positivo e non superiore a 16. In caso l'utente inserisca un valore che non rispetta questi vincoli, la richiesta viene ripetuta.

**Ingresso/Uscita:**
**input:** `un numero intero (da verificare)`
**output:** `una sequenza di caratteri`

```c
#include <stdio.h>
#define BLOCK '#'
#define MIN 1
#define MAX 16

int main(void)
{
    int h;    /* altezza */
    int i, j;

    do {
        scanf("%d", &h);
    } while(h < MIN || h > MAX);

    i = 1;
    while(i <= h)
    {
        /* spazi */
        j = 0;
        while(j < h - i){
            printf(" ");
            j++;
        }
        j = 0;
        while(j < i){
            printf("%c", BLOCK);
            j++;
        }
        /* separatore */
        printf("  ");
        j = 0;
        while(j < i){
            printf("%c", BLOCK);
            j++;
        }
        j = 0;
        while(j < h - i){
            printf(" ");
            j++;
        }
        printf("\n");
        i++;
    }

    return 0;
}
```

### Esercizio 16.4: Scorrimento a destra – rightshift

Scrivere un programma che acquisita una stringa di al più 10 caratteri, modifica la stringa in modo tale che la stringa finale sia quella iniziale, fatta scorrere a destra di una posizione, con l'ultimo carattere riportato in testa. Se per esempio la sequenza iniziale è `attraverso`, la stringa finale sarà `oattravers`. Una volta modificata la stringa memorizzata, visualizzarla e farla seguire da un carattere `'a-capo'`.

**Ingresso/Uscita:**
**input:** `al più 10 caratteri`
**output:** `al più 10 caratteri (seguiti da un carattere 'a-capo')`

**Alcuni casi di test per il collaudo:**

| | |
|---|---|
| **input:** | `attraverso` |
| **output:** | `oattravers` |
| **input:** | `ananas` |
| **output:** | `sanana` |
| **input:** | `trampolini` |
| **output:** | `itrampolin` |

```c
#include <stdio.h>

#define N 10

int main(int argc, char * argv[])
{
  char seq[N+1];
  int tmp, i;

  scanf("%s", seq);

  for(i = 0; seq[i] != '\0'; i++)
    ;

  i--;
  tmp = seq[i];
  for(; i > 0; i--)
    seq[i] = seq[i-1];
  seq[0] = tmp;

  printf("%s\n", seq);
  return 0;
}
```

### Esercizio 16.5: Troncabile primo a destra

Scrivere un programma che acquisisce un valore intero strettamente positivo, e finché non è tale lo richiede. Il programma analizza il valore intero e visualizza 1 nel caso sia un troncabile primo a destra, 0 altrimenti. Un numero si dice troncabile primo a destra se il numero stesso e tutti i numeri che si ottengono eliminando una alla volta la cifra meno significativa del numero analizzato al passo precedente, sono numeri primi. Per esempio, se il numero iniziale è 719, i numeri che si ottengono "eliminando una alla volta la cifra meno significativa del numero analizzato al passo precedente .." sono 71 e 7. Dopo il valore visualizzato, mettere un `'a-capo'`.

**Ingresso/Uscita:**
**input:** `un intero (da verificare)`
**output:** `un intero (seguito da un carattere 'a-capo')`

**Alcuni casi di test per il collaudo:**

**input:** `719`
**output:** `1`

**input:** `473`
**output:** `0`

**input:** `-42 -18 311111`
**output:** `0`

**input:** `3137`
**output:** `1`

```c
#include<stdio.h>

#define BASE 10

int main (int argc, char *argv[])
{
  int n;
  int i, isprime;

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

  isprime = 1;

  while (n > 1 && isprime)
  {
    /* e' un numero primo ? */
    if (n % 2 == 0 && n != 2)
      isprime = 0;
    else {
      half = n/2;
      for(i = 3; i < half && isprime; i = i+2)
        if(n % i == 0)
          isprime = 0;
    }
    n = n / BASE;
  }

  printf("%d\n", isprime);
  return 0;
}
```

# Lezione 10 ◇ sottoprogrammi

21-10-2019

# Lezione 11 ◇ sottoprogrammi

22-10-2019

# Esercitazione 4 ◇ sottoprogrammi

24-10-2017

# Laboratorio 2 ◇ sottoprogrammi

24-10-2019

# Esercitazione 5 ◇ sottoprogrammi

28-10-2019 (CB)

# Esercitazione 6 ◇ sottoprogrammi

29-10-2019 (CB)

## Analisi esercizi 2 ◇ array, stringhe

31-10-2019

04-11-2017

05-11-2017

# Lezione 12 ◇ file di testo e binari

07-11-2019

# Esercitazione 7 ◇ file

11-11-2019 (CB)

## Esercizio 25.1: Conta vocaboli

Scrivere un programma che chiede all'utente il nome di un file (al più 40 caratteri) di testo e conta e visualizza il numero di parole presenti nel file. Le parole sono separate esclusivamente da spazi e sono al più di 32 caratteri.

**Argomenti:** file ASCII. Accesso a file con contenuto di lunghezza non nota a priori.

```c
#include <stdio.h>

#define LENNOME 40
#define LENVOC 32

int main(int argc, char * argv[])
{
  char nome[LENNOME+1], voc[LENVOC+1];
  FILE * fp;
  int num;

  gets(nome);
  if(fp = fopen(nome, "r")){
    num = 0;
    fscanf(fp, "%s", voc);
    while(!feof(fp)){
      num++;
      fscanf(fp, "%s", voc);
    }
    fclose(fp);
    printf("%d\n", num);
  } else
    printf("problemi nell'accesso al file %s\n", nome);

  return 0;
}
```

Quest'altra soluzione non funziona: si finisce in un ciclo infinito. Per quanto il file sia finito, l'istruzione `fscanf` non cessa di essere eseguita e legge sempre l'ultimo vocabolo, restituendo 1.

```c
#include <stdio.h>

#define LENNOME 40
#define LENVOC 32

int main(int argc, char * argv[])
{
  char nome[LENNOME+1], voc[LENVOC+1];
  FILE * fp;
  int num;

  gets(nome);
  if(fp = fopen(nome, "r")){
    num = 0;
```

```
    while(fscanf(fp, "%s", voc)){
      num++;
      printf("%s\n", voc);
    }
    fclose(fp);
    printf("%d\n", num);
  } else
    printf("problemi nell'accesso al file %s\n", nome);

  return 0;
}
```

### Esercizio 25.2: Quale lettera usare per un lipogramma

Scrivere un programma che chiede all'utente il nome di un file (al più 40 caratteri) di testo e conta e visualizza il numero di parole che iniziano con ciascuna lettera dell'alfabeto. I caratteri contenuti nel file sono solo minuscoli. Le parole sono separate esclusivamente da spazi e sono al più di 32 caratteri. Visualizzare il messaggio:

```
parole che iniziano con a: 10
parole che iniziano con b: 3
parole che iniziano con e: 14
```

Non visualizzare alcun messaggio per le lettere che non hanno parole nel testo.

**Argomenti:** file ASCII. Accesso a file con contenuto di lunghezza non nota a priori. Contatori

```
#include <stdio.h>

#define LENNOME 40
#define LENVOC 32
#define LENALF 26
#define OFFSET 'a'

int main(int argc, char * argv[])
{
  char nome[LENNOME+1], voc[LENVOC+1];
  FILE * fp;
  int contatori[LENALF], i; /* contatori di vocaboli per iniziale */
  char iniz; /* iniziale del vocabolo */

  gets(nome);
  if(fp = fopen(nome, "r")){
    for(i = 0; i < LENALF; i++)
      contatori[i] = 0;
    fscanf(fp, "%s", voc);
    while(!feof(fp)){
      iniz = voc[0];
      i = iniz - OFFSET;
      contatori[i]++;
      fscanf(fp, "%s", voc);
    }
    fclose(fp);
    for(i = 0; i < LENALF; i++)
      /* eventualmente filtrare quelle che hanno 0 vocaboli */
```

```
29        if(contatori[i])
30          printf("parole che iniziano con %c: %d\n", i+OFFSET, contatori[i]);
31    } else
32      printf("problemi nell'accesso al file %s\n", nome);
33
34    return 0;
35  }
```

# Lezione 13 ◇ allocazione dinamica

12-11-2019

# Esercitazione 8 ◇ allocazione dinamica

14-11-2019

# Laboratorio 3 ◇ sottoprogrammi

14-11-2019

# Lezione 14 ◇ liste concatenate semplici

18-11-2019

# Lezione 15 ◇ liste concatenate semplici

19-11-2019

# Esercitazione 9 ◇ liste concatenate semplici

21-11-2019

## Laboratorio 4 ◇ lavoro di gruppo

21-11-2019

# Lezione 16 ◇ ricorsione

25-11-2019

# Esercitazione 10 ◇ ricorsione

26-11-2019

## Esercitazione 11 ◇ riepilogo

28-11-2019

# Esercitazione 12 ◇ riepilogo

02-12-2019

# Analisi esercizi 3 ◇ riepilogo

03-12-2019

## Lezione 17 ◇ parametri da riga di comando `argv`, `argc` e `getopt`

05-12-2019

## Laboratorio 5 ◇ lavoro di gruppo

05-12-2019

# Esercitazione 13 ◇ liste ripasso

09-12-2019 (CB)

## Lezione 18 ◇ architettura del calcolatore (hw & so) – parte 1

10-12-2019

# Lezione 19 ◇ architettura del calcolatore (hw & so) – parte 2

12-12-2019

## Analisi esercizi: partecipazione

Numero esercizi svolti: **032**
Numero esercizi proposti: **005**

# Indice analitico