

Fondamenti di Informatica ◇ 2019-20

Lezioni, Esercitazioni e Laboratorio

Cristiana Bolchini

Indice

dal problema all'algoritmo

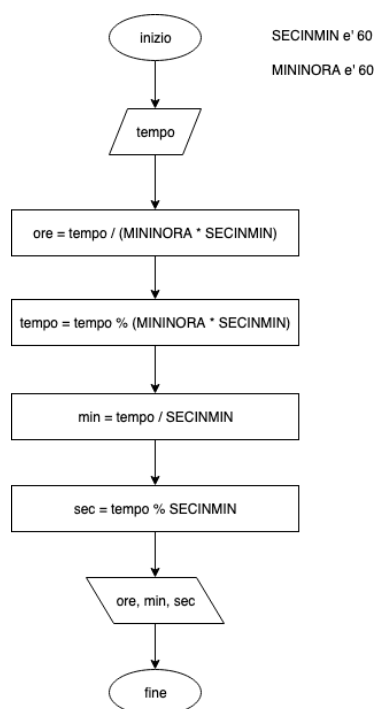
- ◇ proprietà dell'algoritmo
 - non ambiguo
 - deterministico
 - termina in un numero finito di passi
- ◇ tipologia di istruzioni
 - istruzioni effettive
 - istruzioni di controllo

diagrammi di flusso

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

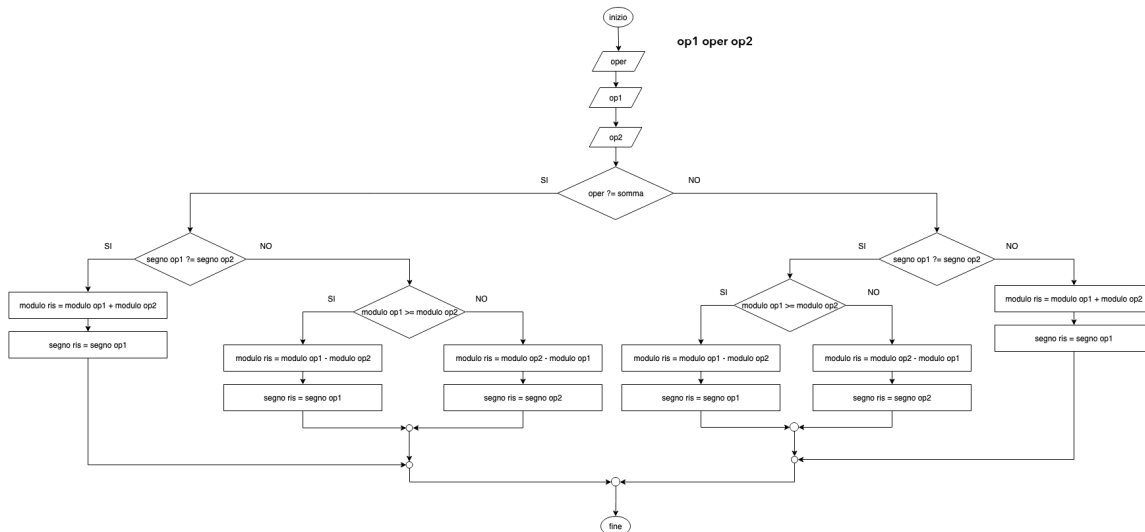
Esercizio 4.1: Tempo espresso in ore, minuti e secondi

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).



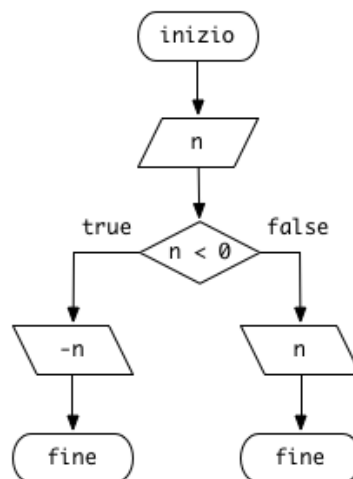
Esercizio 4.2: Operazioni di somma e sottrazione in modulo e segno

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

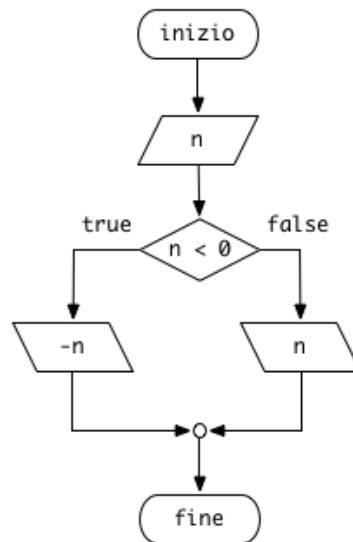


Esercizio 4.3: Valore assoluto

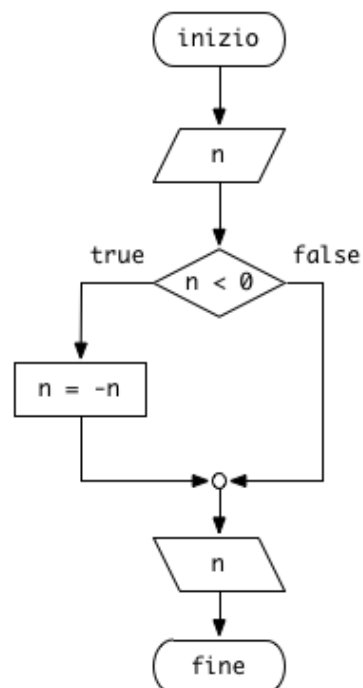
Si realizzi l'algoritmo che acquisito un valore intero calcola e visualizza il suo valore assoluto.



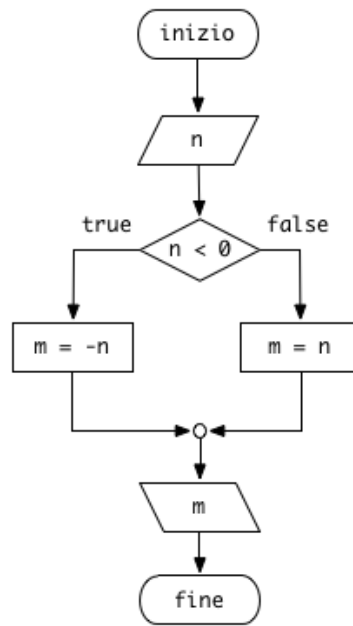
In questo caso l'algoritmo non calcola il valore assoluto, ma lo visualizza direttamente. Guardando la soluzione come una scatola nera, si ottiene il risultato richiesto, quindi soddisfa il comportamento atteso. Non rispetta la richiesta di calcolare il valore e poi visualizzarlo, ma questo aspetto è 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.

```
1 #include <stdio.h>
2
3 #define MIN_IN_ORA 60
4 #define SEC_IN_MIN 60
5
6 int main(int argc, char * argv[])
7 {
8     int tempo; /* quantita' di tempo inserita dall'utente */
9     int ore, min, sec; /* corrispondenti ore, minuti e secondi */
10    int tmp; /* per non rovinare il dato iniziale */
11
12    /* acquisizione */
13    scanf("%d", &tempo);
14
15    /* elaborazione */
16    ore = tempo / (MIN_IN_ORA * SEC_IN_MIN);
17    tmp = tempo % (MIN_IN_ORA * SEC_IN_MIN);
18    min = tmp / SEC_IN_MIN;
19    sec = tmp % SEC_IN_MIN;
20
21    /* visualizzazione */
22    printf("%d %d %d\n", ore, min, sec);
23
24    return 0;
25 }
```

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

```
1 #include <stdio.h>
2 #define MIN_IN_ORA 60
3 #define SEC_IN_MIN 60
4
5 int main(int argc, char * argv[])
6 {
7     int tempo; /* quantita' di tempo inserita dall'utente */
8     int ore, min, sec; /* corrispondenti ore, minuti e secondi */
9
10    /* acquisizione */
11    scanf("%d", &tempo);
12
13    /* elaborazione */
14    ore = tempo / (MIN_IN_ORA * SEC_IN_MIN);
15    min = (tempo % (MIN_IN_ORA * SEC_IN_MIN)) / SEC_IN_MIN;
16    sec = tempo % (MIN_IN_ORA * SEC_IN_MIN) % SEC_IN_MIN;
17
18    /* visualizzazione */
19    printf("%d\n%d\n%d\n", ore, min, sec);
20
21    return 0;
22 }
```

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

```
1 #include <stdio.h>
2 int main(int argc, char * argv[])
3 {
4     int val;
5     char s;
6
7     scanf("%d", &val);
8     if (val > 0)
9         s = '+';
10    else
11        if (val < 0)
12            s = '-';
13        else
14            s = ' ';
15    printf("%c\n", s);
16    return 0;
17 }
```

Versione alternativa.

```
1 #include <stdio.h>
2 int main(int argc, char * argv[])
3 {
4     int val;
5     char s;
6
7     scanf("%d", &val);
8     if (val > 0)
9         s = '+';
10    else if (val < 0)
11        s = '-';
12    else
```

```
13     s = ' ';
14     printf("%c\n", s);
15     return 0;
16 }
```

Versione con #define.

```
1 #include <stdio.h>
2 #define POS '+'
3 #define NEG '-'
4 #define NUL ' '
5 int main(int argc, char * argv[])
6 {
7     int val;
8     char s;
9
10    scanf("%d", &val);
11    if (val > 0)
12        s = POS;
13    else if (val < 0)
14        s = NEG;
15    else
16        s = NUL;
17    printf("%c\n", s);
18    return 0;
19 }
```

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.

```
1 #include <stdio.h>
2 int main(int argc, char * argv[])
3 {
4     int val;
5     int ris;
6
7     scanf("%d", &val);
8     if ((val % 400 == 0) || (val % 4 == 0) && (val % 100 != 0))
9         ris = 1;
10    else
11        ris = 0;
12    printf("%d\n", ris);
13    return 0;
14 }
```

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: costruito `if`. algoritmo.

```
1 #include <stdio.h>
2 #define UP '+'
3 #define DOWN '-'
4
5 int main(int argc, char * argv[])
6 {
7
8     int n1, n2, n3;
9     char dir;
10
11     scanf("%c", &dir);
12     scanf("%d%d%d", &n1, &n2, &n3);
13     if(n1 >= n2)
14         if(n2 >= n3){
15             min = n3;
16             med = n2;
17             max = n1;
18         } else if (n1 >= n3) {
19             min = n2;
20             med = n3;
21             max = n1;
22         } else {
23             min = n2;
24             med = n1;
25             max = n3;
26         } else /* n2 > n1 */
27         if(n1 >= n3){
28             min = n3;
29             med = n1;
30             max = n2;
31         } else if (n2 >= n3) {
32             min = n1;
33             med = n3;
34             max = n2;
35         } else {
36             min = n1;
37             med = n2;
38             max = n3;
39         }
40     if(dir == UP)
41         printf("%d\t%d\t%d\n", min, med, max);
42     else
43         printf("%d\t%d\t%d\n", max, med, min);
44     return 0;
45 }
```

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

```
1 #include <stdio.h>
2
3 #define MIN_IN_ORA 60
4 #define SEC_IN_MIN 60
5
6 int main(int argc, char * argv[])
7 {
8     int tempo; /* quantita' di tempo inserita dall'utente */
9     int ore, min, sec; /* corrispondenti ore, minuti e secondi */
10    int tmp; /* per non rovinare il dato iniziale */
11
12    /* acquisizione */
13    scanf("%d", &tempo);
14
15    /* elaborazione */
16    ore = tempo / (MIN_IN_ORA * SEC_IN_MIN);
17    tmp = tempo % (MIN_IN_ORA * SEC_IN_MIN);
18    min = tmp / SEC_IN_MIN;
19    sec = tmp % SEC_IN_MIN;
20
21    /* visualizzazione */
22    printf("%d %d %d\n", ore, min, sec);
23
24    return 0;
25 }
```

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

```
1 #include <stdio.h>
2
3 #define EUR2 200
4 #define EUR1 100
5 #define CENT50 50
6 #define CENT20 20
7 #define CENT10 10
8 #define CENT5 5
9 #define CENT2 2
10
11 int main(int argc, char * argv[])
12 {
13
14     float eur;
15     int cent, r;
16     int e2, e1, c50, c20, c10, c5, c2, c1;
17
18     scanf("%f", &eur);
19     cent = eur * 100; /* essendo cent una variabile di tipo int, tutte le cifre
20                        decimali di eur dopo la seconda vengono "tagliate" */
21
22     e2 = cent / EUR2;
23     r = cent % EUR2; /* r %= EUR2*/
24
25     e1 = r / EUR1;
26     r = r % EUR1;
27
28     c50 = r / CENT50;
29     r = r % CENT50;
30
31     c20 = r / CENT20;
32     r = r % CENT20;
33
34     c10 = r / CENT10;
35     r = r % CENT10;
36
37     c5 = r / CENT5;
38     r = r % CENT5;
39
40     c2 = r / CENT2;
41
42     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 ciclico 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

```
1 #include <stdio.h>
2
3 #define NDATI 53
4
5 int main(int argc, char * argv[])
6 {
7
8     int num, cont;
9     int min, max;
10    float avg;
11
12    scanf("%d", &num);
13    min = num;
14    max = num;
15    tot = num;
16    cont = 1;
17    while(cont < NDATI){
18        scanf("%d", &num);
19        if(num < min)
20            min = num;
21        else if (num > max)
22            max = num;
23        tot = tot + num;
24        cont++;
25    }
26    avg = (float)tot / NDATI;
27    printf("%d %d %f\n", min, max, avg);
28    return 0;
29 }
```

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.

```
1 #include <stdio.h>
2
3 int main(int argc, char * argv[])
4 {
5     int i, n;
6     int fatt;
7
8     do
9         scanf("%d", &n);
10    while(n < 0);
11
12    fatt = 1;
13    i = 2;
14    while(i <= n){
15        fatt = fatt * i;
16        i++;
17    }
18    printf("%d\n", fatt);
19
20    return 0;
21 }
```

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

```
1 #include <stdio.h>
2
3 int main(int argc, char * argv[])
4 {
5     int i, n;
6     int fatt;
7
8     do
9         scanf("%d", &n);
10    while(n < 0);
11
12    if(n == 0 || n == 1)
13        fatt = 1;
14    else {
15        fatt = 1;
16        i = 2;
17        while(i <= n){
18            fatt = fatt * i;
19            i++;
20        }
21    }
22    printf("%d\n", fatt);
23
24    return 0;
25 }
```

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

```
1 #include <stdio.h>
2
3 int main(int argc, char * argv[])
4 {
```

```
5  int n;
6  int fatt;
7
8  do
9      scanf("%d", &n);
10 while(n < 0);
11
12 if(n == 0 || n == 1)
13     fatt = 1;
14 else {
15     fatt = 1;
16     while(n > 1){
17         fatt = fatt * n;
18         n--;
19     }
20 }
21 printf("%d\n", fatt);
22
23 return 0;
24 }
```

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.

```
1 #include <stdio.h>
2
3 #define BASE 2
4
5 int main(int argc, char * argv[])
6 {
7
8     int val, num;
9
10    scanf("%d", &num);
11    val = num;
12
13    while(val > 0){
14        printf("%d", val % BASE);
15        val = val / BASE;
16    }
17    printf("\n");
18
19    return 0;
20 }
```

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.

```
1 #include <stdio.h>
2
3 #define BASE 2
4
5 int main(int argc, char * argv[])
6 {
7     int n, p;
8
9     scanf("%d", &n);
10    p = 1;
11    while(p < n)
12        p = BASE * p;
13
14    printf("%d\n", p);
15    return 0;
16 }
```

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.

```
1 #include <stdio.h>
2
3 #define BASE 10
4
5 int main(int argc, char * argv[])
6 {
7
8     int num;
9     int val, dim;
10
11     do
12         scanf("%d", &num);
13     while(num < 0);
14
15     if(num == 0)
16         dim = 1;
17     else {
18         dim = 0;
19         val = num;
20         while(val > 0){
21             val = val / BASE;
22             dim++;
23         }
24     }
25     printf("%d\n", dim);
26
27     return 0;
28 }
```

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 ($|A - C| \geq 2$). 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)

```
1 #include <stdio.h>
2
3 #define BASE 10
4 #define MAX 999
5
6 int main(int argc, char * argv[])
7 {
8
9     int num, tmp;
10    int sx, mid, dx;
11    int diff, numinv, diffinv, ris;
12
13    do {
14        scanf("%d", &num);
15        dx = num % BASE;
16        sx = num / (BASE * BASE);
17        if(dx - sx > 0)
18            diff = dx - sx;
19        else
20            diff = sx - dx;
21    } while(diff < 2 || num > MAX);
22    mid = (num / BASE) % BASE;
23    numinv = dx * BASE * BASE + mid * BASE + sx;
24    if(num > numinv)
25        diff = num - numinv;
26    else
27        diff = numinv - num;
28    dx = diff % BASE;
29    sx = diff / (BASE * BASE);
30    mid = (diff / BASE) % BASE;
31    diffinv = dx * BASE * BASE + mid * BASE + sx;
32    ris = diff + diffinv;
33    printf("%d %d %d\n", numinv, diff, diffinv);
34    printf("%d\n", ris);
35    return 0;
36 }
```

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.

```
1 #include <stdio.h>
2
3 #define BASEDST 2
4 #define BASEOUT 10
5
6 int main(int argc, char * argv[])
```

```

7 {
8
9     int val, decimale;
10    int binario, potenza;
11    int bit;
12
13    scanf("%d", &decimale);
14    val = decimale;
15    binario = 0;
16    potenza = 1;
17
18    while(val > 0){
19        bit = val % BASEDST;
20        binario = binario + bit * potenza;
21        potenza = potenza * BASEOUT;
22        val = val / BASEDST;
23    }
24    printf("%d\n", binario);
25
26    return 0;
27 }

```

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.

```

1 #include <stdio.h>
2
3 #define BASE 2
4
5 int main(int argc, char * argv[])
6 {
7
8     int val, num;
9     int dim, i;
10
11    scanf("%d", &num);
12    scanf("%d", &dim);
13
14    val = num;
15    while(dim > 0){
16        printf("%d", val % BASE);
17        val = val / BASE;
18        dim--;
19    }
20    printf("\n");
21
22    return 0;
23 }

```

03-10-2019

- ◇ 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.

```
1 #include <stdio.h>
2
3 int main(int argc, char * argv[])
4 {
5
6     int n1, n2, n3, n4, n5;
7
8     scanf("%d", &n1);
9     scanf("%d%d%d%d", &n2, &n3, &n4, &n5);
10
11     printf("%d%d%d%d%d\n", n5, n4, n3, n2, n1);
12
13     return 0;
14 }
```

Versione con array.

```
1 #include <stdio.h>
2
3 #define NELEM 5
4
5 int main(int argc, char * argv[])
6 {
7
8     int dati[NELEM], i;
9
10    i = 0;
11    while(i < NELEM){
12        scanf("%d", &dati[i]);
13        i++;
14    }
```

```

15  i--; /* i vale 5 */
16  while(i >= 0){
17      printf("%d", dati[i]);
18      i--;
19  }
20
21  return 0;
22 }

```

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.

```

1  #include <stdio.h>
2
3  #define NDATI 20
4
5  int main(int argc, char * argv[])
6  {
7
8      int val[BASE], i;
9      int thr, cont;
10
11     i = 0;
12     while(i < NDATI){
13         scanf("%d", &val[i]);
14         i++;
15     }
16     scanf("%d", &thr);
17     i = 0;
18     cont = 0;
19     while(i < NDATI){
20         if(val[i] > thr)
21             cont++;
22         i++;
23     }
24     printf("%d\n", cont);
25     return 0;
26 }

```

Versione con il costrutto `for`.

```

1  #include <stdio.h>
2
3  #define NDATI 20
4
5  int main(int argc, char * argv[])
6  {
7
8      int val[BASE], i;
9      int thr, cont;
10
11     for(i = 0; i < NDATI; i++)
12         scanf("%d", &val[i]);
13     scanf("%d", &thr);
14     cont = 0;
15     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

```

```

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     i = 0;
19     do{
20         bin[i] = n % BASE;
21         n = n / BASE;
22         i++;
23     } while(n > 0);
24
25     for(i--; i >= 0; i--)
26         printf("%d", bin[i]);
27     printf("\n");
28
29     return 0;
30 }

```

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

```

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;
11     int bin[SIZE], i;
12
13     do
14         scanf("%d", &dec);
15     while(dec < MIN || dec > MAX);
16
17     for(i = SIZE-1; i >= 0; i--){
18         bin[i] = dec % BASE;
19         dec = dec / BASE;
20     }
21
22     for(i=0; i < SIZE; i++)
23         printf("%d", bin[i]);
24     printf("\n");
25
26     return 0;
27 }

```

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)

```
1 #include <stdio.h>
2
3 #define BASE 10
4
5 int main(int argc, char * argv[])
6 {
7     int num, rev;
8     int tmp, digit;
9
10    scanf("%d", &num);
11
12    tmp = num;
13    rev = 0;
14    while(tmp > 0){
15        digit = tmp % BASE;
16        rev = rev * BASE + digit;
17        tmp = tmp / BASE;
18    }
19
20    printf("%d\n", rev);
21
22    return 0;
23 }
```

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. costruito `for`.

```
1 #include <stdio.h>
2
3 #define NELEM 16
4 #define LALPHA 26
5 #define INIT 'a'
6
7 int main(int argc, char * argv[])
8 {
9
10     char seq[NELEM];
11     int cont[LALPHA];
12     int i, pos;
13
14     for(i = 0; i < NELEM; i++)
15         scanf("%c", &seq[i]);
16
17     for(i = 0; i < LALPHA; i++)
18         cont[i] = 0;
19
20     for(i = 0; i < NELEM; i++){
21         pos = seq[i] - INIT;
22         cont[pos]++;
23     }
24
25     for(i = 0; i < LALPHA; i++)
26         if(cont[i] > 0)
27             printf("%c %d\n", INIT+i, cont[i]);
28
29     return 0;
30 }
```

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.

```
1 #include <stdio.h>
2
```

```
3 #define NMAX 50
4 #define LIMIT 0
5
6 int main(int argc, char * argv[])
7 {
8     int val[NMAX], i;
9     int num, dim;
10    int prime, isdiv, div, half;
11
12    dim = 0;
13    scanf("%d", &num);
14    while(num > LIMIT){
15        val[dim] = num;
16        dim++;
17        scanf("%d", &num);
18    }
19    prime = 0;
20    for(i = 0; i < dim && prime == 0; i++){
21        printf("%d\n", val[i]);
22        /* verifica se il numero e' primo */
23        half = val[i] / 2;
24        div = 2;
25        /*
26        do {
27            isdiv = val[i] % div;
28            div++;
29        }(div < half && isdiv != 0);
30        */
31        isdiv = 1;
32        while(isdiv && div < half){
33            isdiv = val[i] % div;
34            div++;
35        }
36        if(isdiv != 0)
37            prime = 1;
38    }
39    printf("%d\n", !prime);
40
41    return 0;
42 }
```

- ◇ 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.

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

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`.

```
1 typedef struct date_s {
2     int g, m, a;
3 } 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`.

```
1 #define NL 41
2 #define NESAMI 18
3 typedef struct student_s {
4     char last[NL+1], first[NL+1];
5     date_t dnascita, dlaurea;
6     int grades[NEX];
7     float avg;
8     char level;
9 } student_t;
```

- ◇ 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.

```
1 #include <stdio.h>
2
3 #define N 25
4
5 int main(int argc, char * argv[])
6 {
7     char seq[N+1], tmp;
8     int i, j;
9
10    scanf("%s", seq);
11
12    for(j = 0; seq[j] != '\0'; j++)
13        ;
14
15    /* oppure */
16    /*
17     j = 0;
18     while(seq[j] != '\0')
19         j++;
20     */
21
22    for(i = 0, j--; i < j; i++, j--){
23        tmp = seq[i];
24        seq[i] = seq[j];
25        seq[j] = tmp;
26    }
27
28    printf("%s\n", seq);
29
30    return 0;
31
32 }
```

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.

```
1 #include <stdio.h>
2 #define N 25
3
4 int main(int argc, char * argv[])
5 {
6     char seq[N+1], tmp;
7     int i, j, inv;
8
9     scanf("%s", seq);
10
11     for(j = 0; seq[j] != '\0'; j++)
12         ;
13
14     for(i = 0, j--; i < j; i++, j--){
15         tmp = seq[i];
16         seq[i] = seq[j];
17         seq[j] = tmp;
18     }
19
20     for(i = 0; seq[i] != '\0'; )
21         if(seq[i] == 'a' || seq[i] == 'e' || seq[i] == 'i' || seq[i] == 'o' || seq[i]
22            == 'u')
23             for(j = i; seq[j] != '\0'; j++) /* vengono spostati tutti di 1 */
24                 seq[j] = seq[j+1];
25         else
26             i++;
27     seq[i] = '\0';
28
29     scanf("%s", seq);
30     for(i = 0, inv = 0; seq[inv] != '\0'; inv++){
31         if(seq[inv] != 'a' && seq[inv] != 'e' && seq[inv] != 'i' && seq[inv] != 'o'
32            && seq[inv] != 'u'){
33             seq[i] = seq[inv];
34             i++;
35         }
36     }
37     seq[i] = '\0';
38
39     printf("%s\n", seq);
40     return 0;
41 }
```

versione alternativa

```
1 #include <stdio.h>
2 #define N 25
3
4 int main(int argc, char * argv[])
5 {
6     char seq[N+1], tmp;
7     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'
22          && seq[inv] != 'u'){
23          seq[i] = seq[inv];
24          i++;
25      }
26  seq[i] = '\0';
27
28  printf("%s\n", seq);
29  return 0;
30 }

```

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 /* se v1 e' piu' lunga di v2, senz'altro non trova cio' che cerca .. */
30 if(found && i < len2) /* si potrebbe scrivere anche if(i < len2) */
31     found = 0;
32
33 printf("%d\n", found);
34 return 0;
35 }
```

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

```
1 #include <stdio.h>
2 #define STOP 0
3
4 int main(int argc, char * argv[])
5 {
6     int prec, last;
7     int prod;
8     int i, isq;
9
10    do
11        scanf("%d", &prec);
12    while(prec < 0);
13
14    if(prec > 0){
15        scanf("%d", &last);
16        while(last != STOP){
17            if(last > 0 && prec > 0)
18                if(last != prec)
19                    if(last % 2 == 0 && prec % 2 == 0){
20                        prod = last * prec;
21                        i = 2;
22                        isq = i*i;
23                        while(isq < prod){
24                            i++;
25                            isq = i*i;
26                        }
27                        if(isq == prod)
28                            printf("%d %d\n", prec, last);
29                        /*
30                        for(i = 2; i*i < prod; i++)
31                            ;
32                        if(i*i == prod)
33                            printf("%d %d", prec, last);
34                        */
35                    }
36
37            prec = last;
38            scanf("%d", &last);
39        }
40    }
41    return 0;
42 }
```

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. costruito `for`.

```
1 #include <stdio.h>
2 #define NELEM 10
3
4 int main(int argc, char * argv[])
5 {
6
7     int s1[NELEM], s2[NELEM];
8     int ham[NELEM], i, dist;
9
10    for(i = 0; i < NELEM; i++)
11        scanf("%d", &s1[i]);
12
13    for(i = 0; i < NELEM; i++)
14        scanf("%d", &s2[i]);
15
16    dist = 0;
17    for(i = 0; i < NELEM; i++)
18        if(s1[i] == s2[i])
19            ham[i] = 0;
20        else {
21            ham[i] = 1;
22            dist++;
23        }
24
25    for(i = 0; i < NELEM; i++)
26        printf("%d ", ham[i]);
27
28    printf(" %d\n", dist);
29    return 0;
30 }
```

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 sì, 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 sì, 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 lo 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.

```
1 #include <stdio.h>
2
3 #define LEN 16
4 #define BASE 10
5
6 int main(int argc, char * argv[])
7 {
8
9     char numcc[LEN];
10    int i, tot, dim, digit;
11    int ris;
12
13    for(i = 0; i < LEN; i++)
14        scanf("%c", &numcc[i]);
15
16    tot = 0;
17    i = LEN - 2;
18    for(; i >= 0; i = i-2){
19        digit = (numcc[i] - '0')*2;
20        if(digit < BASE)
21            tot = tot + digit;
22        else
23            tot = tot + digit / BASE + digit % BASE;
24    }
25
26    for(i = LEN - 1; i >= 0; i = i-2)
27        tot = tot + (numcc[i] - '0');
28
29    /* alternativa */
30    for(i = LEN-1; i >= 0; i = i - 2){
31        tot = tot + (numcc[i] - '0');
32        digit = (numcc[i-1] - '0')*2;
33        if(digit < BASE)
```



```

34         tot = tot + digit;
35     else
36         tot = tot + digit / BASE + digit % BASE;
37 }
38
39 /* NON COSI'
40 for(i = LEN-1; i >= 0; i--){
41     if(i % 2)
42         tot = tot + (numcc[i] - '0');
43     else {
44         digit = (numcc[i] - '0')*2;
45         if(digit < BASE)
46             tot = tot + digit;
47         else
48             tot = tot + digit / BASE + digit % BASE;
49     }
50 }
51 */
52
53
54 if(tot % BASE)
55     ris = 0;
56 else
57     ris = 1;
58
59 printf("%d\n", ris);
60 return 0;
61 }

```

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)

```

1 #include <stdio.h>
2 #define N 10
3
4 int main(int argc, char * argv[])
5 {
6
7     int v[N], n;
8     int i, j, tot;
9     float m[N];
10
11     for(i = 0; i < N; i++)
12         scanf("%d", &v[i]);
13

```

```

14  scanf("%d", &n);
15
16  m[0] = v[0];
17  for(i = 1; i < n; i++){
18      for(j = 0, tot = 0; j <= i; j++)
19          tot = tot + v[j];
20      m[i] = (float) tot / j;
21  }
22
23  for(; i < N; i++){
24      for(j = 0, tot = 0; j < n; j++)
25          tot = tot + v[i-j];
26      m[i] = (float) tot / n;
27  }
28
29  for(i = 0; i < N; i++)
30      printf("%f ", m[i]);
31
32  printf("\n");
33
34  return 0;
35 }

```

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 `aabbbdbbbbhhhhhzzzz` il programma visualizza `z4h5b4d2b3a2`.

Argomenti: array monodimensionali. caratteri. algoritmo.

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

```

1  #include <stdio.h>
2  #define N 20
3
4  int main(int argc, char * argv[])
5  {
6      char str[N], car;
7      int i, count;
8
9      for(i = 0; i < N; i++)
10         scanf("%c", &str[i]);
11     i--;
12
13     car = str[i];
14     count = 1;
15     for(i--; i >= 0; i--){
16         if(str[i] == car)
17             count++;
18         else {
19             printf("%c%d", car, count);

```

```

20     count = 1;
21     car = str[i];
22 }
23 }
24 printf("%c%d\n", car, count);
25 return 0;
26 }

```

Versione alternativa

```

1 #include <stdio.h>
2 #define N 20
3
4 int main(int argc, char * argv[])
5 {
6     char str[N], car;
7     int i, len, count;
8
9     for(i = 0; i < N; i++)
10         scanf("%c", &str[i]);
11
12     i--;
13     car = str[i];
14     while(i >= 0){
15         count = 0;
16         while(i >= 0 && str[i] == car){
17             count++;
18             i--;
19         }
20         printf("%c%d", car, count);
21         car = str[i];
22     }
23     printf("\n");
24     return 0;
25 }

```

Versione alternativa con il costrutto for

```

1 #include <stdio.h>
2 #define N 20
3
4 int main(int argc, char * argv[])
5 {
6     char str[N], car;
7     int i, len, count;
8
9     for(i = 0; i < N; i++)
10         scanf("%c", &str[i]);
11
12     i--;
13     car = str[i];
14     while(i >= 0){
15         for(count = 0; i >= 0 && str[i] == car; i--, count++)
16             ;
17         printf("%c%d", car, count);
18         car = str[i];
19     }
20     printf("\n");
21     return 0;

```


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 3×3 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

```
1 #include <stdio.h>
2
3 #define N 50
4 #define SEPU '/'
5 #define SEPW '\\\
6
7 int main(int argc, char * argv[])
8 {
9     char seq[N+1];
10    int i, s;
11
12    gets(seq);
13
14    s = -1;
15    for(i = 0; seq[i] != '\0'; i++)
16        if(seq[i] == SEPU || seq[i] == SEPW)
17            s = i;
18
19    /* s indica l'ultimo separatore o -1 se non ce ne sono */
20    printf("%s\n", &seq[s+1]);
21    /*
22    for(i = s+1; seq[i] != '\0'; i++)
23        printf("%c", seq[i]);
24    printf("\n");
25    */
26    return 0;
27 }
```

oppure:

```
1 #include <stdio.h>
2
3 #define N 50
4 #define SEPU '/'
5 #define SEPW '\\ '
6
7 int main(int argc, char * argv[])
8 {
9     char seq[N+1];
10    int i, s;
11
12    gets(seq);
13
14    for(i = 0; seq[i] != '\0'; i++)
15        ;
16
17    i--;
18    while(seq[i] != SEPU && seq[i] != SEPW && i >= 0)
19        i--;
20
21    /* i indica l'ultimo separatore o -1 se non ce ne sono */
22    printf("%s\n", &seq[i+1]);
23
24    return 0;
25 }
```

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.

```
1 #include <stdio.h>
2
3 #define N 9
4 #define BASE 10
5
6 int main(int argc, char * argv[])
7 {
8
9     char seqn[N+1];
10    int val;
11    int tmp, nc, i;
12
13    scanf("%d", &val);
14
15    /* inseriti in ordine inverso */
16    nc = 0;
17    while(val > 0){
18        seqn[nc] = val % BASE + '0';
19        val = val / BASE;
20        nc++;
21    }
```

```

22  seqn[nc] = '\0';
23
24  /* ribalto la stringa */
25  for(i = 0; i < nc / 2; i++){
26      tmp = seqn[i];
27      seqn[i] = seqn[nc-1-i];
28      seqn[nc-1-i] = tmp;
29  }
30
31  printf("%s\n", seqn);
32  return 0;
33 }

```

Versione alternativa.

```

1  #include <stdio.h>
2
3  #define N 9
4  #define BASE 10
5
6  int main(int argc, char * argv[])
7  {
8
9      char seqn[N+1];
10     int val;
11     int tmp, nc, i;
12
13     scanf("%d", &val);
14
15     /* conto il numero di cifre */
16     nc = 0;
17     tmp = val;
18     while(val > 0){
19         val = val / BASE;
20         nc++;
21     }
22     seqn[nc] = '\0';
23
24     /* scrivo le cifre da quella meno significativa */
25     for(i = nc-1; i >= 0; i--){
26         seqn[i] = tmp % BASE + '0';
27         tmp = tmp / BASE;
28     }
29
30     printf("%s\n", seqn);
31     return 0;
32 }

```

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 coordinate 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 dichiara un opportuno tipo di dato (`vertex_t`) per rappresentare i vertici del poligono.

```

1 #include <stdio.h>
2 #define VMIN 3
3 #define VMAX 10
4
5 typedef struct vertex_s {
6     int x, y;
7 } vertex_t;
8
9 int main(int argc, char * argv[])
10 {
11     vertex_t polig[VMAX];
12     int num, i;
13     int tot;
14     float area;
15
16     do
17         scanf("%d", &num);
18     while (num < VMIN || num > VMAX);
19
20     for(i = 0; i < num; i++)
21         scanf("%d%d", &polig[i].x, &polig[i].y);
22
23     /* calcolo dell'area */
24     tot = 0;
25     /* contributi positivi */
26     for(i = 0; i < num-1; i++)
27         tot = tot + polig[i].x * polig[i+1].y;
28     tot = tot + polig[i].x * polig[0].y;
29     /* contributi negativi */
30     for(i = 1; i < num; i++)
31         tot = tot - polig[i].x * polig[i-1].y;
32     tot = tot - polig[0].x * polig[i-1].y;
33     if(tot < 0)
34         tot = -tot;
35     area = tot / 2.0;
36
37     printf("%f\n", area);
38
39     return 0;
40 }

```

Esercizio 15.5: Quadrato magico

Si scriva un programma che acquisisce i dati di una matrice di dimensione 3×3 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 diagonalì dia sempre lo stesso valore.

Argomenti: array bidimensionali. algoritmo.

```

1 #include <stdio.h>
2
3 #define DIM 3
4

```



```

5 int main(int argc, char * argv[])
6 {
7
8     int m[DIM][DIM], i, j;
9     int sum, ism, tot;
10
11     for(i = 0; i < DIM; i++)
12         for(j = 0; j < DIM; j++)
13             scanf("%d", &m[i][j]);
14
15     /* prima somma - diagonale */
16     sum = 0;
17     for(i = 0; i < DIM; i++)
18         sum += m[i][i];
19
20     /* antidiagonale */
21     for(i = 0, tot = 0; ism && i < DIM; i++){
22         tot += m[i][DIM-1-i];
23         if(tot != sum)
24             ism = 0;
25     else
26         ism = 1;
27
28     /* righe */
29     for(i = 0; ism && i < DIM; i++){
30         for(j = 0, tot = 0; j < DIM; j++)
31             tot += m[i][j];
32         if(tot != sum)
33             ism = 0;
34     }
35
36     /* colonne */
37     for(i = 0; ism && i < DIM; i++){
38         for(j = 0, tot = 0; j < DIM; j++)
39             tot += m[j][i];
40         if(tot != sum)
41             ism = 0;
42     }
43
44     printf("%d", ism);
45     if(ism)
46         printf(" %d", sum);
47
48     printf("\n");
49     return 0;
50 }

```

Versione alternativa.

```

1 #include <stdio.h>
2
3 #define DIM 3
4
5 int main(int argc, char * argv[])
6 {
7
8     int m[DIM][DIM], i, j;
9     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 }

```

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

```
1 #include <stdio.h>
2
3 int main(int argc, char * argv[])
4 {
5     int n1, n2;
6     int ris;
7
8     scanf("%d", &n1);
9     scanf("%d", &n2);
10
11     if(n1 && n2)
12         if(n1 % n2 == 0 || n2 % n1 == 0)
13             ris = 1;
14         else
15             ris = 0;
16     else
17         ris = 0;
18
19     printf("%d\n", ris);
20     return 0;
21 }
```

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` è 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

```
1 #include <stdio.h>
2 #define BASE 10
3
4 int main(int argc, char * argv[])
5 {
6     int num, k;
7     int val, i, ncifre;
8
9     do
10         scanf("%d", &num);
11     while(num <= 0);
12
13     do
14         scanf("%d", &k);
15     while(k <= 0);
16
17     val = num;
18     i = 0;
19     while(val > 0){
20         val = val/BASE;
21         i++;
22     }
23     for(; k > i; i--)
24         printf("0");
25     printf("%d\n", num);
26
27     return 0;
28 }
```

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

```
1 #include <stdio.h>
2 #define BLOCK '#'
3 #define MIN 1
4 #define MAX 16
5
6 int main(void)
7 {
8     int h;    /* altezza */
9     int i, j;
10
11     do {
12         scanf("%d", &h);
13     } while(h < MIN || h > MAX);
14
15     i = 1;
16     while(i <= h)
17     {
18         /* spazi */
19         j = 0;
20         while(j < h - i){
21             printf(" ");
22             j++;
23         }
24         j = 0;
25         while(j < i){
26             printf("%c", BLOCK);
27             j++;
28         }
29         /* separatore */
30         printf(" ");
31         j = 0;
32         while(j < i){
33             printf("%c", BLOCK);
34             j++;
35         }
36         j = 0;
37         while(j < h - i){
38             printf(" ");
39             j++;
40         }
41         printf("\n");
42         i++;
43     }
44
45     return 0;
46 }
```

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`

```
1 #include <stdio.h>
2
3 #define N 10
4
5 int main(int argc, char * argv[])
6 {
7     char seq[N+1];
8     int tmp, i;
9
10    scanf("%s", seq);
11
12    for(i = 0; seq[i] != '\0'; i++)
13        ;
14
15    i--;
16    tmp = seq[i];
17    for(; i > 0; i--)
18        seq[i] = seq[i-1];
19    seq[0] = tmp;
20
21    printf("%s\n", seq);
22    return 0;
23 }
```

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

```
1 #include <stdio.h>
2
3 #define BASE 10
4
5 int main (int argc, char *argv[])
6 {
7     int n;
8     int i, isprime;
9
10    do
11        scanf("%d", &n);
12    while(n <= 0);
13
14    isprime = 1;
15
16    while (n > 1 && isprime)
17    {
18        /* e' un numero primo ? */
19        if (n % 2 == 0 && n != 2)
20            isprime = 0;
21        else {
22            half = n/2;
23            for(i = 3; i < half && isprime; i = i+2)
24                if(n % i == 0)
25                    isprime = 0;
26        }
27        n = n / BASE;
28    }
29
30    printf("%d\n", isprime);
31    return 0;
32 }
```

- ◇ intestazione
- ◇ tipo di dato restituito
- ◇ parametri formali e attuali
- ◇ prototipo
- ◇ return
- ◇ tipo void
- ◇ passaggio array monodimensionali e dimensione
- ◇ visibilità delle variabili

Esercizio 17.1: Combinazioni

Scrivere un programma che acquisisce due interi n e k strettamente positivi, con k non superiore a n e finchè non è tale li richiede (entrambi). Quindi calcola e visualizza il numero di combinazioni di n elementi, presi k per volta.

```
1 #include <stdio.h>
2
3
4 int main(int argc, char * argv[])
5 {
6
7     int n, k;
8     int ris;
9     int fn, fk, fnk, i;
10
11     do {
12         do
13             scanf("%d", &n);
14         while(n <= 0);
15
16         do
17             scanf("%d", &k);
18         while(k <= 0);
19
20     } while(k < n);
21
22     fn = 1;
23     for(i = 2; i <= n; i++)
24         fn = fn * i;
25
26     fk = 1;
27     for(i = 2; i <= k; i++)
28         fk = fk * i;
29
30     fnk = 1;
31     for(i = 2; i <= nk; i++)
32         fnk = fnk * i;
33
34     ris = fn / fk * fnk;
35     printf("%d\n", ris);
36
37     return 0;
38 }
```

versione con sottoprogrammi

```
1 #include <stdio.h>
2
3 int getintpos(void);
4 int fattoriale(int);
5
6 int main(int argc, char * argv[])
7 {
8
9     int n, k;
10    int ris;
11    int fn, fk, fnk, i;
12
13    do {
14        n = getintpos();
15        k = getintpos();
16
17    } while(k < n);
18
19    fn = fattoriale(n);
20    fk = fattoriale(k);
21    fnk = fattoriale(n-k);
22
23    ris = fn / fk * fnk;
24    printf("%d\n", ris);
25
26    return 0;
27 }
28
29 int getintpos(void)
30 {
31     int val;
32
33     do
34         scanf("%d", &val);
35     while(val <= 0);
36     return val;
37 }
38
39 int fattoriale(int val)
40 {
41     int i;
42     int f;
43
44     f = 1;
45     for(i = 2; i <= val; i++)
46         f = f * i;
47     return f;
48 }
49
50
51 int combinazioni(int n, int k)
52 {
53     int fn, fk, fnk;
54     int ris;
55
56     fn = fattoriale(n);
```

```

7   fk = fattoriale(n);
8   fnk = fattoriale(n);
9
10  ris = fn / fk * fnk;
11  return ris;
12 }

```

Esercizio 17.2: Massimo di un array

Scrivere un sottoprogramma che ricevuto in ingresso un array di numeri interi e *qualsiasi altro parametro ritenuto strettamente necessario* restituisce l'indice dell'elemento con valore massimo.

Argomenti: sottoprogrammi. passaggio array monodimensionale.

```

1  int imaxarrayint(int[], int);
2
3  int imaxarrayint(int v[], int dim)
4  {
5      int imax, i;
6
7      imax = 0;
8      for(i = 1; i < dim; i++)
9          if(v[i] > v[imax])
10             imax = i;
11     return imax;
12 }

```

Esercizio 17.3: Triangolo – Proposto e risolto

Scrivere un sottoprogramma che ricevuti in ingresso tre interi, restituisce 1 nel caso in cui si tratti delle dimensioni dei lati di un triangolo valido 0 altrimenti. Le condizioni che devono sussistere sono:

- ◊ le dimensioni sono positive
- ◊ la somma di due dimensioni non è mai superiore alla terza dimensione

```

1
2  int validtriangle(int len1, int len2, int len3)
3  {
4      int ris;
5
6      if(len1 > 0 && len2 > 0 && len3 > 0)
7          if(len1 + len2 > len3 && len1 + len3 > len2 && len3 + len2 > len1)
8              ris = 1;
9          else
10             ris = 0;
11     else
12         ris = 0;
13     return ris;
14 }
15
16 int validtriangle(int len1, int len2, int len3)
17 {
18
19     if(len1 > 0 && len2 > 0 && len3 > 0)
20         if(len1 + len2 > len3 && len1 + len3 > len2 && len3 + len2 > len1)
21             return 1;
22     return 0;
23 }

```

- ♦ passaggio parametri per copia valore, per copia indirizzo
- ♦ record di attivazione

Esercizio 18.1: Massimo, minimo e media dei valori di un array

Scrivere un sottoprogramma che ricevuto in ingresso un array di numeri interi e *qualsiasi altro parametro ritenuto strettamente necessario trasmette* al chiamante l'indice dell'elemento con valore massimo, quello con valore minimo e la media dei valori dell'array.

Argomenti: sottoprogrammi. passaggio array monodimensionale. passaggio parametri per indirizzo.

```
1 #include <stdio.h>
2
3 #define N 45
4
5 void minmaxavgarrint(int [], int, int *, int *, float *);
6
7 void fillarrint(int [], int);
8
9 int main(int argc, char * argv[])
10 {
11     int dati[N], num;
12     int pmax, pmin;
13     float avgval;
14
15     do
16         scanf("%d", &num);
17     while(num < 2 || num > N);
18
19     fillarrint(dati, num);
20
21     minmaxavgarrint(dati, num, &pmax, &pmin, &avgval);
22
23     printf("%d\t%d\t%f", dati[pmax], dati[pmin], avgval);
24     return 0;
25 }
26
27
28 void minmaxavgarrint(int v[], int dim, int * posimax, int * posimin, float * avg
29 )
30 {
31     int imax, imin, i;
32     int tot;
33     float m;
34
35     imax = 0;
36     imin = 0;
37     tot = v[0];
38     for(i = 1; i < dim; i++){
39         if(v[i] > v[imax])
40             imax = i;
```

```
40     else if (v[i] < v[imin])
41         imin = i;
42     tot = tot + v[i];
43 }
44 m = (float) tot / dim;
45
46 /* aggiorno la memoria del chiamante */
47 *posimin = imin;
48 *posimax = imax;
49 *avg = m;
50 }
51
52 void fillarrint(int v[], int nval)
53 {
54     int i;
55     for(i = 0; i < nval; i++)
56         scanf("%d", &v[i]);
57 }
```

- ◇ passaggio di array bidimensionali
- ◇ passaggio di struct

Esercizio 19.1: Lunghezza di una stringa

Scrivere il sottoprogramma che riceve in ingresso una stringa, calcola e restituisce la sua lunghezza.

```
1 int mystrlen(char []);
2
3 int mystrlen(char s[])
4 {
5     for(i = 0; s[i] != '\0'; i++)
6         ;
7     return i;
8 }
```

Esercizio 19.2: Matrice identità

Scrivere un sottoprogramma che ricevuto in ingresso un array bidimensionale e qualsiasi altro parametro ritenuto strettamente necessario restituisce 1 se si tratta di una matrice identità, 0 altrimenti. La dimensione dell'array dichiarata è N.

```
1 #define N 10
2
3 int identita(int m[][N], int dim)
4 {
5     int i, j;
6     int isi;
7
8     isi = 1;
9     for(i = 0; i < dim && isi; i++)
10         for(j = 0; j < dim && isi; j++)
11             if((i == j && m[i][j] != 1) || i != j && m[i][j] != 0)
12                 isi = 0;
13
14     return isi;
15 }
16
17 int identita(int m[][N], int dim)
18 {
19     int i, j;
20
21     for(i = 0; i < dim; i++)
22         for(j = 0; j < dim; j++)
23             if((i == j && m[i][j] != 1) || i != j && m[i][j] != 0)
24                 return 0;
25
26     return 1;
27 }
```

Esercizio 19.3: Vertici di un poligono

Si consideri il seguente tipo di dato per rappresentare punti nello spazio piano.

```

1 typedef struct p {
2     int x, y;
3 } punto_t;

```

Si scriva un sottoprogramma che ricevuto in ingresso un array di tipo `punto_t` e qualsiasi altro parametro ritenuto strettamente necessario acquisisca i dati relativi ai vertici di un poligono.

```

1 #define NMAX 10
2 #define NMIN 3
3
4 typedef struct _point {
5     int x, y;
6 } point_t;
7
8 void getpoints(point_t [], int);
9 float areagauss(point_t [], int);
10
11 int main(int argc, char * argv[])
12 {
13
14     point_t vertice[NMAX];
15     int num;
16     float area;
17
18     do
19         scanf("%d", &num);
20     while(num < NMIN || num > NMAX); /* while(!(num >= NMIN && num <= NMAX)) */
21
22     getpoints(vertice, num);
23     area = areagauss(vertice, num);
24     printf("%f\n", area);
25     return 0;
26 }
27
28
29 void getpoints(point_t v[], int dim)
30 {
31     int i;
32
33     for(i = 0; i < dim; i++)
34         scanf("%d%d", &v[i].x, &v[i].y);
35
36 }

```

Esercizio 19.4: Vertice?

Con riferimento al tipo prima definito, scrivere un sottoprogramma che ricevuto in ingresso un array di tipo `punto_t` e qualsiasi altro parametro ritenuto strettamente necessario, ed un punto, determini se il punto è un vertice del poligono o meno, restituendo rispettivamente 1 o 0.

```

1 int findpoint(point_t v[], int dim, point_t p)
2 {
3     int i;
4     int isv;
5
6     isv = 0;
7     for(i = 0; i < dim && !isv; i++) /* && isv == 0 */
8         if(v[i].x == p.x && v[i].y == p.y)

```

```

9     isv = 1;
10    return isv;
11 }
12
13 int findpoint(point_t v[], int dim, point_t p)
14 {
15     int i;
16
17     for(i = 0; i < dim; i++)
18         if(v[i].x == p.x && v[i].y == p.y)
19             return 1;
20     return 0;
21 }

```

Versione con il passaggio per indirizzo

```

1 int findpoint(point_t v[], int dim, point_t * p)
2 {
3     int i;
4     int isv;
5
6     isv = 0;
7     for(i = 0; i < dim && !isv; i++) /* && isv == 0 */
8         if(v[i].x == *p.x && v[i].y == *p.y)
9             isv = 1;
10    return isv;
11 }

```

Esercizio 19.5: Combinazione stringhe – Proposto

Programma che acquisisce due stringhe di al più 20 caratteri ciascuna e consente all'utente di selezionare una tra le seguenti operazioni:

- ◇ 1. verificare se la prima stringa è contenuta nella seconda (in caso positivo visualizza 1, 0 altrimenti)
- ◇ 2. verificare se la seconda stringa è contenuta nella prima (in caso positivo visualizza 1, 0 altrimenti)
- ◇ 3. concatena la seconda stringa alla prima e visualizza il risultato
- ◇ 4. concatena la prima stringa alla seconda e visualizza il risultato
- ◇ 5. concatena le stringhe in ordine alfabetico crescente e visualizza il risultato
- ◇ 6. termine programma

Il programma chiede ripetutamente all'utente quale operazione svolgere, acquisisce le stringhe, svolge l'operazione richiesta e ripresenta il menù, fino a quando l'utente decide di terminare il programma, selezionando l'opportuna voce del menù.

```

1 #include <stdio.h>
2
3 #define MAXSTR 20
4 #define MENUMIN 1
5 #define MENUMAX 6
6
7 int main(int argc, char * argv[])
8 {
9
10     char s1[MAXSTR+1], s2[MAXSTR+1]; /* input */
11     char conc[MAXSTR*2+1];

```

```

12  int isin;
13  int sel;
14  int stop;
15  int i, j, start, ord;
16
17  stop = 0;
18  do {
19      do {
20          printf("1: prima sequenza contenuta nella seconda\n");
21          printf("2: seconda sequenza contenuta nella prima\n");
22          printf("3: concatena seconda sequenza a prima\n");
23          printf("4: concatena prima sequenza a seconda\n");
24          printf("5: concatena le sequenze in ordine alfabetico crescente\n");
25          printf("6: termine\n");
26          scanf("%d", &sel);
27      } while (sel < MENUMIN || sel > MENU MAX);
28      if (sel == 6)
29          stop = 1;
30      else {
31          printf("inserisci due stringhe di al piu' %d caratteri\n", MAXSTR);
32          scanf("%s%s", s1, s2);
33          if (sel == 1){
34              isin = 0;
35              i = 0;
36              j = 0;
37              while (!isin && s1[i] != '\0' && s2[j] != '\0'){
38                  while (s1[i] != s2[j] && s2[j] != '\0')
39                      j++;
40                  start = j;
41                  while (s1[i] == s2[j] && s1[i] != '\0' && s2[j] != '\0'){
42                      i++;
43                      j++;
44                  }
45                  if (s1[i] == '\0' && s1[i-1] == s2[j-1])
46                      isin = 1;
47                  else if (s2[j] != '\0'){
48                      j = start + 1;
49                      i = 0;
50                  }
51              }
52              printf("\n%d\n\n", isin);
53          } else if (sel == 2){
54              isin = 0;
55              i = 0;
56              j = 0;
57              while (!isin && s2[i] != '\0' && s1[j] != '\0'){
58                  while (s2[i] != s1[j] && s1[j] != '\0')
59                      j++;
60                  start = j;
61                  while (s2[i] == s1[j] && s2[i] != '\0' && s1[j] != '\0'){
62                      i++;
63                      j++;
64                  }
65                  if (s2[i] == '\0' && s2[i-1] == s1[j-1])
66                      isin = 1;
67                  else if (s1[j] != '\0'){
68                      j = start + 1;

```



```

69         i = 0;
70     }
71 }
72 printf("\n%d\n\n", isin);
73 } else if (sel == 3) { /* prima seconda */
74     for(i = 0; s1[i] != '\0'; i++)
75         conc[i] = s1[i];
76     for(j = 0; s2[j] != '\0'; j++, i++)
77         conc[i] = s2[j];
78     conc[i] = '\0';
79     printf("\n%s\n\n", conc);
80 } else if (sel == 4) { /* seconda prima */
81     for(i = 0; s2[i] != '\0'; i++)
82         conc[i] = s2[i];
83     for(j = 0; s1[j] != '\0'; j++, i++)
84         conc[i] = s1[j];
85     conc[i] = '\0';
86     printf("\n%s\n\n", conc);
87 } else if (sel == 5) { /* alfabetico */
88     ord = 0;
89     for(i = 0; s1[i] != '\0' && !ord; i++){
90         if(s1[i] < s2[i])
91             ord = 1;
92         else if (s1[i] > s2[i])
93             ord = -1;
94     }
95     if(s1[i] == '\0') {
96         if(s1[i] < s2[i])
97             ord = 1;
98         else if (s1[i] > s2[i])
99             ord = -1;
100     }
101     if(ord == 1){
102         for(i = 0; s1[i] != '\0'; i++)
103             conc[i] = s1[i];
104         for(j = 0; s2[j] != '\0'; j++, i++)
105             conc[i] = s2[j];
106     } else {
107         for(i = 0; s2[i] != '\0'; i++)
108             conc[i] = s2[i];
109         for(j = 0; s1[j] != '\0'; j++, i++)
110             conc[i] = s1[j];
111     }
112     conc[i] = '\0';
113     printf("\n%s\n\n", conc);
114 }
115 }
116 } while (!stop);
117 return 0;
118 }

```

Versione con sottoprogrammi

```

1 #include <stdio.h>
2
3 #define MAXSTR 20
4 #define MENUMIN 1
5 #define MENUMAX 6

```

```

6
7 void menu();
8 int getoptmenu(int, int);
9 int strinstr(char [], char []);
10 void concat(char [], char [], char []);
11
12 int main(int argc, char * argv[])
13 {
14
15     char s1[MAXSTR+1], s2[MAXSTR+1]; /* input */
16     char conc[MAXSTR*2+1];
17     int isin;
18     int sel;
19     int stop;
20     int i, j, restart, ord;
21
22     stop = 0;
23     do {
24         sel = getoptmenu(MENUMIN, MENUMAX);
25         if (sel == 6)
26             stop = 1;
27         else {
28             printf("inserisci due stringhe di al piu' %d caratteri\n", MAXSTR);
29             scanf("%s%s", s1, s2);
30             if (sel == 1){
31                 isin = strinstr(s1, s2);
32                 printf("\n%d\n\n", isin);
33             } else if (sel == 2){
34                 isin = strinstr(s2, s1);
35                 printf("\n%d\n\n", isin);
36             } else if (sel == 3) { /* prima seconda */
37                 concat(s1, s2, conc);
38                 printf("\n%s\n\n", conc);
39             } else if (sel == 4) { /* seconda prima */
40                 concat(s2, s1, conc);
41                 printf("\n%s\n\n", conc);
42             } else if (sel == 5) { /* alfabetico */
43                 ord = 0;
44                 for(i = 0; s1[i] != '\0' && !ord; i++){
45                     if(s1[i] < s2[i])
46                         ord = 1;
47                     else if (s1[i] > s2[i])
48                         ord = -1;
49                 }
50                 if(s1[i] == '\0') {
51                     if(s1[i] < s2[i])
52                         ord = 1;
53                     else if (s1[i] > s2[i])
54                         ord = -1;
55                 }
56                 if(ord == 1)
57                     concat(s1, s2, conc);
58                 else
59                     concat(s2, s1, conc);
60                 printf("\n%s\n\n", conc);
61             }
62         }
63     }

```

```

63     } while (!stop);
64     return 0;
65 }
66
67
68
69 void menu()
70 {
71     printf("1: prima sequenza contenuta nella seconda\n");
72     printf("2: seconda sequenza contenuta nella prima\n");
73     printf("3: concatena seconda sequenza a prima\n");
74     printf("4: concatena prima sequenza a seconda\n");
75     printf("5: concatena le sequenze in ordine alfabetico crescente\n");
76     printf("6: termine\n");
77     return ; /* opt */
78 }
79
80 int getoptmenu(int smin, int smax)
81 {
82     int sel;
83     do {
84         menu();
85         scanf("%d", &sel);
86     } while (sel < smin || sel > smax);
87     return sel;
88 }
89
90 int strinstr(char s1[], char s2[])
91 {
92
93     int isin;
94     int i, j, start;
95
96     isin = 0;
97     i = 0;
98     j = 0;
99     while (!isin && s1[i] != '\0' && s2[j] != '\0'){
100         while (s1[i] != s2[j] && s2[j] != '\0')
101             j++;
102         start = j;
103         while (s1[i] == s2[j] && s1[i] != '\0' && s2[j] != '\0'){
104             i++;
105             j++;
106         }
107         if (s1[i] == '\0' && s1[i-1] == s2[j-1])
108             isin = 1;
109         else if (s2[j] != '\0'){
110             j = start + 1;
111             i = 0;
112         }
113     }
114     return isin;
115 }
116
117 void concat(char s1[], char s2[], char ris[])
118 {
119

```

```

120  int i, j;
121
122  for(i = 0; s1[i] != '\0'; i++)
123      ris[i] = s1[i];
124  for(j = 0; s2[j] != '\0'; j++, i++)
125      ris[i] = s2[j];
126  ris[i] = '\0';
127 }

```

Esercizio 19.6: Cifra del numero – Proposto

Scrivere un sottoprogramma `cifra` che riceve come parametri due interi `num` e `k`. Se `k` è strettamente positivo, il sottoprogramma calcola e restituisce la `k`-esima cifra del numero `num` a partire da destra. Nel caso in cui `k` non sia strettamente positivo o `k` sia maggiore del numero effettivo di cifre di `num`, il sottoprogramma restituisce `-1`.

Scrivere un programma che chiede all'utente i due valori `num` e `k` ed invoca il sottoprogramma `cifra` visualizzando poi il risultato, seguito da un carattere a-capo `'\n'`.

Argomenti: sottoprogrammi. algoritmo.

```

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

```

```
36 }
37
38 int cifrav2(int num, int k)
39 {
40     int i;
41
42     i = 0;
43     while(i < k-1 && num > 0){
44         num = num / BASE;
45         i++;
46     }
47     if(i == k)
48         return num % BASE;
49     return -1;
50 }
```

Esercizio 20.1: Somma a k

Scrivere un programma che acquisisce una sequenza di al più 100 valori interi e un intero strettamente positivo k . L'acquisizione della sequenza termina quando l'utente inserisce un numero negativo o nullo, oppure quando vengono acquisiti 100 valori. Il programma visualizza 1 se la sequenza contiene due valori tali che la loro somma sia k , 0 altrimenti. Dopo il valore visualizzato, mettere un 'a-capo'. Per realizzare la soluzione si sviluppi un sottoprogramma `cercasomma` che ricevuto in ingresso k , l'array contenente i dati e qualsiasi altro parametro ritenuto strettamente necessario, restituisce 1 o 0 nel caso trovi i due valori la cui somma è k .

Ingresso/Uscita:

input: una sequenza di al più 101 valori interi

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

Alcuni casi di test per il collaudo:

input: 10 15 3 7 -4 17

output: 1

input: 2 5 6 55 10 -11 100

output: 0

input: 2 2 3 -8 7

output: 0

```
1 #include <stdio.h>
2 #define MAX_ELEM 100
3
4 int cercasomma(int, int[], int);
5
6 int main(int argc, char * argv[]) {
7     int v[MAX_ELEM], dim;
8     int ris;
9
10    dim = 0;
11    scanf("%d", &val);
12    while(val > 0 && dim < MAX_ELEM){
13        v[dim] = val;
14        dim++;
15        scanf("%d", &val);
16    }
17
18    do
19        scanf("%d",&k);
20    while(k <= 0);
21
22    ris = cercasomma(k, v, dim);
23
24    printf("%d\n", ris);
25    return 0;
26 }
27
28 int cercasomma(int somma, int dati[], int num)
29 {
```

```

30  int i, j, found;
31
32  found = 0;
33  for(i = 0; i < dim-1 && !found; i++)
34      for(j = i+1; j < dim && !found; j++)
35          if(dati[i] + dati[j] == somma)
36              found = 1;
37
38  return found;
39 }
40
41 /* versione alternativa
42
43 int cercasomma(int somma, int dati[], int num)
44 {
45     int i, j;
46
47     for(i = 0; i < dim-1; i++)
48         for(j = i+1; j < dim; j++)
49             if(dati[i] + dati[j] == somma)
50                 return 1;
51
52     return 0;
53 }
54
55 */

```

Esercizio 20.2: Solo in ordine

Scrivere un programma che acquisisce una sequenza di al più 20 valori interi, chiedendo all'utente inizialmente quanti valori vorrà fornire, `num`. Il programma acquisisce `num` valori e memorizza in una opportuna struttura dati la sequenza di valori i cui elementi sono strettamente crescenti, trascurando i valori che risultano non essere ordinati. Al termine dell'acquisizione il programma visualizza la lunghezza della sequenza, seguita, su una nuova riga, dalla sequenza stessa. L'utente inserirà sempre un numero di valori coerente con la richiesta. Avvalersi di due sottoprogrammi: `fillarrord` e `viewarr`: il primo memorizza i dati ritenuti validi, il secondo visualizza il contenuto di un array.

Ingresso/Uscita:

input: sequenza di interi

output: sequenza di interi

Alcuni casi di test per il collaudo:

```

input:  10
        3 1 4 5 -1 3 1 4 5 -1
output: 3

        3 4 5
input:  6
        -1 3 -1 3 -1 3
output: 2

        -1 3
input:  8
        9 8 7 6 5 4 3 2
output: 1

        9

```

```
1 #include <stdio.h>
2
3 #define N 20
4
5 int fillarrord(int[], int);
6 void viewarr(int[], int);
7
8 int main(int argc, char * argv[])
9 {
10     int seq[N], num;
11     int nord;
12
13     do
14         scanf("%d", &num);
15     while(num < 2 || num > N);
16
17     nord = fillarrord(seq, num);
18
19     printf("%d\n", nord);
20     viewarr(seq, nord);
21
22     return 0;
23 }
24
25 int fillarrord(int v[], int dim)
26 {
27     int nletti, i;
28     int val;
29
30     /* primo elemento */
31     scanf("%d", &v[0]);
32     nletti = 1;
33     i = 1;
34     /* gli altri */
35     do{
36         scanf("%d", &val);
37         if(val > v[i-1]){
38             v[i] = val;
39             i++;
40         }
41         nletti++;
42     }while(nletti < dim);
43     return i;
44 }
45
46 void viewarr(int v[], int dim)
47 {
48     int i;
49
50     for(i = 0; i < dim; i++)
51         printf("%d ", v[i]);
52     printf("\n");
53 }
```

Esercizio 20.3: Quadro di parole

Scrivere un programma che acquisisce un valore intero strettamente positivo `num`, che rappresenta il numero di parole (ciascuna di al più 25 caratteri) che verranno poi fornite, e che comunque non saranno mai più di 20. Il programma acquisisce le `num` parole e le visualizza, una per riga, all'interno di un rettangolo creato dal carattere `*`.

Per esempio, se l'utente fornisce:

```
5
Hello
world
in
un
rettangolo
```

il programma visualizza:

```
*****
*Hello      *
*world      *
*in         *
*un         *
*rettangolo*
*****
```

Ingresso/Uscita:

input: un intero e una sequenza di stringhe

output: una sequenza di caratteri

```
1 #include <stdio.h>
2
3 #define MAX_STRINGS 20
4 #define MAX_CHAR 25
5 #define BORDER '*'
6
7 int mystrlen(char[]); /* oppure la strlen di libreria */
8
9 int main(int argc, char * argv[]) {
10     char parole[MAX_STRINGS][MAX_CHAR+1];
11     int num, max_len, word_len;
12     int i, j, c;
13
14     do
15         scanf("%d", &num);
16     while(num <= 0 || num > MAX_STRINGS);
17
18     for (i = 0; i < num; i++)
19         scanf("%s", parole[i]);
20
21     /*calcola massima lunghezza tra le stringhe*/
22     max_len = mystrlen(parole[0]);
23     for (i = 1; i < num; i++){
24         word_len = mystrlen(parole[i]);
```

```

25     if(word_len > max_len)
26         max_len = word_len;
27 }
28
29 /* spazio interno al rettangolo pari alla parola piu' lunga */
30 word_len = max_len;
31 /* spazio del bordo include i caratteri prima e dopo */
32 max_len = max_len + 2;
33
34 /* bordo superiore */
35 for(i = 0; i < max_len; i++)
36     printf("%c", BORDER);
37 printf("\n");
38
39 /* per tutte le parole */
40 for(i = 0; i < num; i++){
41     printf("%c", BORDER);
42     /* visualizzo un carattere per volta perche' servono poi degli spazi */
43     /* non fare %s perche' poi si deve comunque ripassare */
44     for(j = 0; parole[i][j] != '\0'; j++)
45         printf("%c", parole[i][j]);
46     /* aggiungo eventuali spazi */
47     for(; j < word_len; j++)
48         printf(" ");
49     printf("%c\n", BORDER);
50 }
51
52 /* bordo inferiore */
53 for(i = 0; i < max_len; i++)
54     printf("%c", BORDER);
55 printf("\n");
56
57 return 0;
58 }
59
60 int mystrlen(char s[])
61 {
62     int i;
63     for(i = 0; s[i] != '\0'; i++)
64         ;
65     return i;
66 }

```

Esercizio 20.4: Mix di due array ordinati

Scrivere un sottoprogramma che acquisisce due sequenze di valori interi, ciascuna di 20 elementi. Il programma ordina le due sequenze in senso crescente, quindi visualizza la sequenza dei valori acquisiti, in senso crescente e senza ripetizioni. Al termine dell'esecuzione, le due sequenze sono ordinate. Nel realizzare la soluzione, scrivere un sottoprogramma `sortarr` che ricevuto in ingresso un array, un intero `updown` e qualsiasi altro parametro ritenuto strettamente necessario, ordina il contenuto dell'array in senso crescente se `updown` vale 1, in senso decrescente se vale -1

Ingresso/Uscita:

input: quaranta sequenze di numeri positivi

output: una sequenza di numeri positivi

Alcuni casi di test per il collaudo:

input: -1 2 4 2 5 6 8 1 0 7 3 4 9 -9 9 9 9 9 9 9
 7 3 4 5 6 7 8 9 2 1 0 5 6 8 1 0 1 2 4 2
output: -9 -1 0 1 2 3 4 5 6 7 8 9

input: 9 8 7 6 5 4 3 2 1 0 0 0 0 0 0 0 0 0 0 0
input: 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
output: 0 1 2 3 4 5 6 7 8 9

```
1 #include <stdio.h>
2
3 #define N 20
4 #define UP 1
5 #define DOWN -1
6
7 void sortarr(int[], int, int);
8 void fillarr(int[], int);
9
10 int main(int argc, char * argv[])
11 {
12     int seq1[N], seq2[N];
13     int i, j;
14     int val;
15
16     fillarr(seq1, N);
17     fillarr(seq2, N);
18
19     sortarr(seq1, N, UP);
20     sortarr(seq2, N, UP);
21
22     i = 0;
23     j = 0;
24
25     if(seq1[i] <= seq2[j]){
26         val = seq1[i];
27         printf("%d ", seq1[i]);
28         i++;
29     } else {
30         val = seq2[j];
31         printf("%d ", seq2[j]);
32         j++;
33     }
34     while(i < N && j < N){
35         if(seq1[i] < seq2[j]){
36             if(seq1[i] != val){
37                 printf("%d ", seq1[i]);
38                 val = seq1[i];
39             }
40             i++;
41         } else {
42             if(seq2[j] != val){
43                 printf("%d ", seq2[j]);
44                 val = seq2[j];
45             }
46             j++;
47         }
48     }
49     for(; i < N; i++){
```

```

50     if(seq1[i] != val){
51         printf("%d ", seq1[i]);
52         val = seq1[i];
53     }
54 }
55 for(;j < N;j++)
56     if(seq2[j] != val){
57         printf("%d ", seq2[j]);
58         val = seq2[j];
59     }
60 printf("\n");
61 return 0;
62 }
63
64
65 void sortarr(int v[], int dim, int updown)
66 {
67     int i, j, move;
68     int tmp;
69
70     move = 1;
71     if(updown == UP)
72         for(i = 0; i < dim && move; i++){
73             move = 0;
74             for(j = i+1; j < dim; j++)
75                 if(v[i] > v[j]){
76                     tmp = v[i];
77                     v[i] = v[j];
78                     v[j] = tmp;
79                     move = 1;
80                 }
81         }
82     else
83         for(i = 0; i < dim && move; i++){
84             move = 0;
85             for(j = i+1; j < dim; j++)
86                 if(v[i] < v[j]){
87                     tmp = v[i];
88                     v[i] = v[j];
89                     v[j] = tmp;
90                     move = 1;
91                 }
92         }
93 }
94
95
96 void fillarr(int v[], int dim)
97 {
98     int i;
99
100    for(i = 0; i < dim; i++)
101        scanf("%d", &v[i]);
102 }

```

Esercizio 20.5: Sottostringa piú lunga senza ripetizioni

Scrivere un programma che acquisita una stringa di al piú 30 caratteri, individui la sottostringa piú lunga in essa contenuta, senza caratteri ripetuti. Il programma visualizza la lunghezza di tale sottostringa, seguita da un carattere 'a-capo'.

Ingresso/Uscita:

input: una stringa

output: un intero

Alcuni casi di test per il collaudo:

input: abcabcbcb

output: 3

input: alfabeto

output: 7

input: bbbbbb

output: 1

```
1 #include <stdio.h>
2 #define MAX_L 30
3
4 int main(int argc, char *argv[]) {
5     char seq[MAX_L+1];
6     int maxlen, start;
7     int posdup, len;
8     int i, j, k;
9
10    scanf("%s", seq);
11    /* calcolo la lunghezza */
12    for(len = 0; seq[len] != '\0'; len++)
13        ;
14
15    maxlen = 0;
16    start = 0;
17    /* mi fermo quando la lunghezza della stringa rimanente e' insufficiente */
18    while(seq[start] != '\0' && (len - start > maxlen)){
19        posdup = len; /* dove trovo un duplicato di qualcosa nella stringa */
20        for(i = start; i < posdup; i++)
21            for(j = i+1; j < posdup; j++)
22                if(seq[j] == seq[i])
23                    posdup = j;
24
25        /* ho trovato una sequenza senza ripetizioni */
26        /* la visualizzo — extra */
27        for(k = start; k < posdup; k++)
28            printf("%c", seq[k]);
29        printf(" lunghezza: %d attuale massimo: %d\n", posdup - start, maxlen);
30        if(posdup - start > maxlen){
31            maxlen = posdup - start;
32            printf("nuova migliore soluzione\n");
33        }
34        start++;
35    }
36    printf("\n\n%d\n", maxlen);
37    return 0;
38 }
```

Esercizio 21.1: Confronta lunghezza delle stringhe

Scrivere un sottoprogramma che riceve due stringhe in ingresso e:

- ◇ restituisce -1 se la prima stringa è più corta della seconda
- ◇ restituisce 0 se le due stringhe hanno la stessa lunghezza
- ◇ restituisce 1 se la prima stringa è più lunga della seconda

Scrivere quindi un programma che chiede all'utente due stringhe, che si assume abbiano lunghezza massima di 50 caratteri, e stampa:

- ◇ "piu' corta" se la prima stringa è più corta della seconda
- ◇ "uguale" se le due stringhe hanno la stessa lunghezza
- ◇ "piu' lunga" se la prima stringa è più lunga della seconda

Argomenti: Stringhe. Lunghezza stringhe. Sottoprogrammi.

```
1 #include <stdio.h>
2
3 #define N 50
4
5
6 int strlencmp(char [], char []);
7
8
9 int main(int argc, char* argv[])
10 {
11     char str1[N+1], str2[N+1];
12     int result;
13
14     gets(str1);
15     gets(str2);
16
17     result = strlencmp(str1, str2);
18
19     if(result == -1)
20         printf("piu' corta\n");
21     else if(result == 0)
22         printf("uguale\n");
23     else
24         printf("piu' lunga\n");
25
26     return 0;
27 }
28
29
30 int strlencmp(char str1[], char str2[])
31 {
32     int i;
33
34     for(i=0; str1[i] != '\0' && str2[i] != '\0'; i++)
35         ;
36
37     if(str1[i] == '\0' && str2[i] == '\0')
```

```

38     return 0;
39     if(str1[i] == '\0')
40         return -1;
41
42     return 1;
43
44 }

```

Esercizio 21.2: Selection sort

Il selection sort è un algoritmo di ordinamento che, dato un array di lunghezza n , ripete i seguenti passi (partendo da $i = 0$):

- ◊ seleziona il minimo elemento nella sezione di array che va da i a $n - 1$ (compreso)
- ◊ scambia l'elemento trovato con quello in posizione i
- ◊ incrementa l'indice i

Scrivere un sottoprogramma che riceve in ingresso un array di interi e tutti i parametri aggiuntivi che si ritengono necessari ed esegue un selection sort su di esso.

Scrivere quindi un programma che riceve in ingresso 10 numeri interi e li stampa ordinati in modo crescente.

Argomenti: Array monodimensionali. Algoritmi di ordinamento. Sottoprogrammi.

```

1  #include <stdio.h>
2
3  #define N 10
4
5
6  void selectionsort(int [], int);
7
8
9  int main(int argc, char* argv[])
10 {
11     int arr[N];
12     int i, n;
13
14     for(i=0; i<N; i++)
15         scanf("%d", &arr[i]);
16
17     selectionsort(arr, N);
18
19     for(i=0; i<N; i++)
20         printf("%d ", arr[i]);
21     printf("\n");
22
23     return 0;
24 }
25
26
27 void selectionsort(int arr[], int arr_size)
28 {
29     int i, j;
30     int min_i;
31     int tmp;
32

```

```

33     for(i=0; i<arr_size; i++){
34
35         min_i = i;
36
37         for(j=i+1; j<arr_size; j++)
38             if(arr[j] < arr[min_i])
39                 min_i = j;
40
41         tmp = arr[min_i];
42         arr[min_i] = arr[i];
43         arr[i] = tmp;
44     }
45
46 }

```

Esercizio 21.3: Matrice trasposta

Scrivere un sottoprogramma che data in ingresso una matrice di interi e tutti i parametri strettamente necessari, calcola e restituisce la sua trasposta.

Scrivere quindi un programma che dati in ingresso due interi, rispettivamente il numero di righe e il numero di colonne (che si assumono validi e inseriti correttamente), e una matrice di interi di quelle dimensioni (quindi numero_righe * numero_colonne interi) ne calcola e stampa la trasposta. Si assuma che la matrice abbia al massimo 15 righe e 20 colonne.

Argomenti: Array bidimensionali. Sottoprogrammi.

```

1  #include <stdio.h>
2
3  #define NRIG 15
4  #define NCOL 20
5
6
7  void trasponi(int[][NCOL], int[][NRIG], int, int);
8
9
10 int main(int argc, char* argv[])
11 {
12
13     int matrice[NRIG][NCOL];
14     int matrice_trasposta[NCOL][NRIG];
15     int nrig, ncol;
16     int i, j;
17
18     scanf("%d%d", &nrig, &ncol);
19     for(i=0; i<nrig; i++)
20         for(j=0; j<ncol; j++)
21             scanf("%d", &matrice[i][j]);
22
23     trasponi(matrice, matrice_trasposta, nrig, ncol);
24
25     for(i=0; i<ncol; i++){
26         for(j=0; j<nrig; j++)
27             printf("%d ", matrice_trasposta[i][j]);
28         printf("\n");
29     }

```

```
30
31     return 0;
32 }
33
34
35
36 void trasponi(int matrice[][NCOL], int matrice_trasposta[][NRIG], int nrig, int
    ncol)
37 {
38
39     int i, j;
40
41     for(i=0; i<nrig; i++)
42         for(j=0; j<ncol; j++)
43             matrice_trasposta[j][i] = matrice[i][j];
44
45 }
```

Esercizio 22.1: Prepara un cocktail

Si vuole scrivere un programma che, dati gli ingredienti di un cocktail, ne definisca la ricetta e la gradazione alcolica. Si assume di avere solamente ingredienti liquidi nella ricetta e che tali ingredienti non siano mai più di 10. Si ricorda, inoltre, che nelle ricette dei cocktail le quantità degli ingredienti sono definite in parti (che non necessariamente sono intere).

Definire una struttura dati che possa memorizzare le informazioni riguardanti un ingrediente di un cocktail, che sono:

- ♦ nome dell'ingrediente (dalla lunghezza massima di 50 caratteri)
- ♦ gradazione alcolica dell'ingrediente in percentuale (che si assume intera)
- ♦ numero di parti nel cocktail

Scrivere un programma che:

- ♦ prende in ingresso un intero n che corrisponde al numero di ingredienti nel cocktail, se il numero è minore di 1 o maggiore di 10, l'inserimento deve essere ripetuto;
- ♦ chiede in input, per ogni ingrediente, il nome, la gradazione alcolica e il numero di parti;
- ♦ richiede quanti litri l di cocktail si vogliono preparare;
- ♦ stampa, per ogni ingrediente, la quantità necessaria per la preparazione di l litri di cocktail;
- ♦ stampa la gradazione alcolica del cocktail

Argomenti: struct. typedef. Array monodimensionali.

```
1 #include <stdio.h>
2
3 #define MININGR 1
4 #define MAXINGR 10
5 #define MAXNOME 50
6
7
8 typedef struct ingrediente_s {
9
10     char nome[MAXNOME+1];
11     int gradazione;
12     float parti;
13
14 } ingrediente_t;
15
16
17 int main(int argc, char* argv[])
18 {
19     int n;
20     float l;
21     ingrediente_t ingredienti[MAXINGR];
22
23     int i;
24     float tot_parti;
25     float gradazione_finale;
26
27     do{
28         scanf("%d", &n);
```

```

29     } while (n < MININGR || n > MAXINGR);
30
31     for (i=0; i<n; i++){
32         scanf("%s", ingredienti[i].nome);
33         scanf("%d", &ingredienti[i].gradazione);
34         scanf("%f", &ingredienti[i].parti);
35     }
36
37     scanf("%f", &l);
38
39     tot_parti = 0;
40     for (i=0; i<n; i++)
41         tot_parti += ingredienti[i].parti;
42
43     gradazione_finale = 0;
44     for (i=0; i<n; i++)
45         gradazione_finale += ingredienti[i].gradazione * ingredienti[i].parti;
46     gradazione_finale /= tot_parti;
47
48     printf(" Ricetta:\n");
49     for (i=0; i<n; i++)
50         printf("%f litri di %s\n", ingredienti[i].parti / tot_parti * l,
51             ingredienti[i].nome);
52
53     printf(" Gradazione cocktail: %.2f %% \n", gradazione_finale);
54
55     return 0;
56 }

```

Esercizio 22.2: Controlla Sudoku

Il Sudoku è un gioco di logica in cui lo scopo è quello di completare una matrice 9×9 inserendo numeri tra 1 e 9. Il gioco termina quando la matrice è piena e rispetta i seguenti vincoli:

- ◊ ogni riga contiene tutti i numeri da 1 a 9
- ◊ ogni colonna contiene tutti i numeri da 1 a 9
- ◊ ogni matrice 3×3 ottenuta raggruppando righe e colonne contigue contiene tutti i numeri da 1 a 9. In questo gruppo di matrici si includono:
 - la matrice ottenuta prendendo le righe 0-2 delle colonne 0-2;
 - la matrice ottenuta prendendo le righe 0-2 delle colonne 3-5;
 - la matrice ottenuta prendendo le righe 0-2 delle colonne 6-8;
 - la matrice ottenuta prendendo le righe 3-5 delle colonne 0-2;
 - ...

Scrivere un sottoprogramma per ogni vincolo che, data una matrice di interi 9×9 e tutti i parametri strettamente necessari, restituisce 1 se il vincolo è rispettato, 0 altrimenti. Scrivere infine un sottoprogramma che, data una matrice di interi 9×9 e tutti i parametri strettamente necessari, restituisce 1 se tale matrice rappresenta una soluzione valida per il gioco del Sudoku, 0 altrimenti.

Argomenti: Sottoprogrammi. Array bidimensionali.

```

1 #include <stdio.h>
2
3 #define NRIG 9
4 #define NCOL 9

```

```

5 #define NRIG_SOTTOM 3
6 #define NCOL_SOTTOM 3
7 #define MINVAL 1
8 #define MAXVAL 9
9
10
11
12 int rispetta_righe(int[][NCOL]);
13 int rispetta_colonne(int[][NCOL]);
14 int rispetta_sottomatrici(int[][NCOL]);
15
16 int is_soluzione(int[][NCOL]);
17
18
19 int main(int argc, char* argv[])
20 {
21
22     int griglia[NRIG][NCOL];
23     int i, j;
24
25     for(i=0; i<NRIG; i++)
26         for(j=0; j<NCOL; j++)
27             scanf("%d", &griglia[i][j]);
28
29     if(is_soluzione(griglia))
30         printf("ESATTA!\n");
31     else
32         printf("SBAGLIATA!\n");
33
34     return 0;
35 }
36
37
38
39 int rispetta_righe(int griglia[][NCOL])
40 {
41
42     int i, j;
43     int usato[MAXVAL];
44     int valore_cell;
45
46     for(i=0; i<NRIG; i++){
47         for(j=0; j<MAXVAL; j++)
48             usato[j] = 0;
49         for(j=0; j<NCOL; j++){
50             valore_cell = griglia[i][j];
51             if(valore_cell < MINVAL || valore_cell > MAXVAL)
52                 return 0;
53             if(usato[valore_cell-1] == 1)
54                 return 0;
55             usato[valore_cell-1] = 1;
56         }
57     }
58
59     return 1;
60 }
61

```

```

62
63 int rispetta_colonne(int griglia[][NCOL])
64 {
65
66     int i, j;
67     int usato[MAXVAL];
68     int valore_cell;
69
70     for(i=0; i<NCOL; i++){
71         for(j=0; j<MAXVAL; j++)
72             usato[j] = 0;
73         for(j=0; j<NRIG; j++){
74             valore_cell = griglia[j][i];
75             if(valore_cell < MINVAL || valore_cell > MAXVAL)
76                 return 0;
77             if(usato[valore_cell-1] == 1)
78                 return 0;
79             usato[valore_cell-1] = 1;
80         }
81     }
82
83     return 1;
84 }
85
86
87 int rispetta_sottomatrici(int griglia[][NCOL])
88 {
89
90     int i, j;
91     int base_rig, base_col;
92     int usato[MAXVAL];
93     int valore_cell;
94
95     for(base_rig=0; base_rig<NRIG; base_rig+=NRIG_SOTTOM)
96         for(base_col=0; base_col<NCOL; base_col+=NCOL_SOTTOM){
97             for(i=0; i<MAXVAL; i++)
98                 usato[i] = 0;
99             for(i=0; i<NRIG_SOTTOM; i++)
100                 for(j=0; j<NCOL_SOTTOM; j++){
101                     valore_cell = griglia[base_rig+i][base_col+j];
102                     if(valore_cell < MINVAL || valore_cell > MAXVAL)
103                         return 0;
104                     if(usato[valore_cell-1] == 1)
105                         return 0;
106                     usato[valore_cell-1] = 1;
107                 }
108         }
109
110     return 1;
111 }
112
113
114 int is_soluzione(int griglia[][NCOL])
115 {
116
117     return rispetta_righe(griglia) && rispetta_colonne(griglia) &&
    rispetta_sottomatrici(griglia);

```

118
119 }

Esercizio 22.3: Zoo

Definire una tipo di dato che contenga informazioni sugli animali di uno zoo. Il nuovo tipo deve contenere:

- ◇ nome dell'animale, es: "Leone" (lunghezza massima 50 caratteri)
- ◇ specie dell'animale, es: "Panthera Leo" (lunghezza massima 50 caratteri)
- ◇ numero di esemplari adulti nello zoo
- ◇ numero di cuccioli nello zoo

Scrivere i seguenti sottoprogrammi:

- ◇ dato in ingresso l'array di animali dello zoo e tutti gli altri parametri strettamente necessari, restituisce il numero totale di animali nello zoo;
- ◇ dato in ingresso l'array di animali dello zoo e tutti gli altri parametri strettamente necessari, restituisce la percentuale di cuccioli rispetto al totale degli animali nello zoo;
- ◇ dato in ingresso l'array di animali dello zoo, una stringa contenente il nome di una specie e tutti gli altri parametri strettamente necessari, restituisce il numero totale di animali di quella specie all'interno dello zoo. Si consiglia di definire un sottoprogramma di supporto che, date in ingresso due stringhe, restituisce 1 se le due stringhe sono identiche, 0 altrimenti.

Argomenti: struct. typedef. Array monodimensionali. Stringhe. Sottoprogrammi.

```
1 #include <stdio.h>
2
3 #define MAXNOME 50
4 #define MAXSPECIE 50
5
6 typedef struct animale_s{
7
8     char nome[MAXNOME+1];
9     char specie[MAXSPECIE+1];
10    int adulti;
11    int cuccioli;
12
13 } animale_t;
14
15
16 int conta_animali(animale_t [], int);
17
18 float percentuale_cuccioli(animale_t [], int);
19
20 int conta_animali_specie(animale_t [], int, char []);
21 int mystrcmp(char [], char []);
22
23
24 int conta_animali(animale_t zoo[], int n)
25 {
26
27     int i, counter;
28
29     counter = 0;
30     for(i=0; i<n; i++)
```

```

31         counter += zoo[i].adulti + zoo[i].cuccioli;
32
33     return counter;
34 }
35
36
37 float percentuale_cuccioli(animale_t zoo[], int n)
38 {
39
40     int i, counter_cuccioli, counter_adulti;
41
42     counter_cuccioli = 0;
43     counter_adulti = 0;
44
45     for(i=0; i<n; i++)
46         counter_cuccioli += zoo[i].cuccioli;
47         counter_adulti += zoo[i].adulti;
48
49     return (float)(counter_cuccioli) / (counter_cuccioli + counter_adulti);
50 }
51
52
53 int conta_animali_specie(animale_t zoo[], int n, char specie[])
54 {
55
56     int i, counter;
57
58     counter = 0;
59     for(i=0; i<n; i++)
60         if(mystrcmp(zoo[i].specie, specie) == 1)
61             counter += zoo[i].cuccioli + zoo[i].adulti;
62
63     return counter;
64 }
65
66
67 int mystrcmp(char str1[], char str2[])
68 {
69
70     int i;
71     int uguali;
72
73     uguali = 1;
74     for(i=0; str1[i] != '\0' && str2[i] != '\0' && uguali == 1; i++)
75         if(str1[i] != str2[i])
76             uguali = 0;
77
78     if(str1[i] != '\0' || str2[i] != '\0')
79         uguali = 0;
80
81     return uguali;
82 }

```

Esercizio 23.1: Lunghezza di una stringa

Scrivere un sottoprogramma che ricevette in ingresso due stringhe restituisce 1 se non hanno caratteri in comune, 0 altrimenti.

```
1 #define N 30
2
3 int nocharsincommon(char [], char []);
4
5 int main(int argc, char * argv[])
6 {
7     char parola1[N+1], parola2[N+1];
8     int ris;
9
10    gets(parola1);
11    scanf("%s", parola2);
12    ris = nocharsincommon(parola1, parola2);
13    printf("%d\n", ris);
14    return 0;
15 }
16
17 int nocharsincommon(char s1[], char s2[])
18 {
19     int i, j;
20     int ris;
21
22     ris = 1;
23     for(i = 0; s1[i] != '\0' && ris; i++)
24         for(j = 0; s2[j] != '\0' && ris; j++)
25             if(s1[i] == s2[j])
26                 ris = 0;
27
28     return ris;
29 }
30
31 int nocharsincommon(char s1[], char s2[])
32 {
33     int i, j;
34
35     for(i = 0; s1[i] != '\0'; i++)
36         for(j = 0; s2[j] != '\0'; j++)
37             if(s1[i] == s2[j])
38                 return 0;
39     return 1;
40 }
```

Esercizio 23.2: Stringa nella stringa

Scrivere un sottoprogramma che ricevette in ingresso due stringhe restituisce 1 se la seconda è una sottostringa della prima (è contenuta nella prima).

```
1 /* scrivere un sottoprogramma che ricevette in ingresso due stringhe restituisce
2 1 se la seconda e' una sottostringa della prima (e' contenuta nella prima) */
3
```



```

4 int strinstr(char s[], char cerca[])
5 {
6     int i, j, k;
7     int trovato;
8
9     trovato = 0;
10    for(i = 0; s[i] != '\0' && !trovato; i++){
11        j = 0;
12        while(s[i+j] == cerca[j] && cerca[j] != '\0' && s[i+j] != '\0')
13            j++;
14        if(cerca[j] == '\0')
15            trovato = 1;
16    }
17    return trovato;
18 }
19
20 int strinstr_v2(char s[], char cerca[])
21 {
22     int i;
23     int trovato;
24     trovato = 0;
25     for(i = 0; s[i] != '\0' && !trovato; i++)
26         trovato = secondapiualtro(&s[i], cerca)
27     return trovato;
28 }
29
30 int strinstr_v2(char s[], char cerca[])
31 {
32     int i;
33
34     for(i = 0; s[i] != '\0'; i++)
35         if(secondapiualtro(&s[i], cerca))
36             return 1;
37     return 0;
38 }
39
40
41
42 int secondapiualtro(char lunga[], char corta[])
43 {
44     int i;
45     i = 0;
46     while(lunga[i] == corta[i] && corta[i] != '\0' && lunga[i] != '\0')
47         i++;
48     if(corta[i] == '\0')
49         return 1;
50     return 0;
51 }

```

Esercizio 23.3: Cifre divisori

Scrivere un sottoprogramma che ricevuto in ingresso un valore intero calcola e restituisce il numero di cifre del numero che sono divisori del numero stesso.

```

1 #define BASE 10
2
3 int cifredivisori(int);
4

```

```

5 int cifredivisori(int num)
6 {
7     int val, cifra;
8     int cont;
9
10    cont = 0;
11    val = num;
12    while(num > 0){
13        cifra = num % BASE;
14        if(cifra && val % cifra == 0)
15            cont++;
16        num = num / BASE;
17    }
18 }

```

Esercizio 23.4: Insieme unione ordinato

Scrivere un sottoprogramma che ricevuti in ingresso due array di numeri interi e qualsiasi altro parametro ritenuto strettamente necessario, ordina entrambi gli array in senso crescente e *visualizza* l'insieme unione ordinato.

```

1 #define N 30
2 #define UP 1
3 #define DOWN 0
4
5 void sort(int [], int, int);
6
7 void visualizzaunioneordinata(int set1[], int set2[], int dim1, int dim2)
8 {
9     int i, j;
10
11    sort(set1, dim1, UP);
12    sort(set2, dim2, UP);
13
14    if(set1[0] < set2[0]){
15        printf("%d ", set1[0]);
16        i = 1;
17        j = 0;
18        pre = set1[0];
19    } else {
20        printf("%d ", set2[0]);
21        j = 1;
22        i = 0;
23        pre = set2[0];
24    }
25    while(i < dim1 && j < dim2){
26        if(set1[i] < set2[j]){
27            if(set1[i] != pre){
28                printf("%d ", set1[i]);
29                pre = set1[i];
30            }
31            i++;
32        } else {
33            if(set2[j] != pre){
34                printf("%d ", set2[j]);
35                pre = set2[j];
36            }
37        }
38    }
39 }

```

```
37     j++;
38 }
39 }
40 while(i < dim1){
41     if(set1[i] != pre){
42         printf("%d ", set1[i]);
43         pre = set1[i];
44     }
45     i++;
46 }
47 while(j < dim2){
48     if(set2[j] != pre){
49         printf("%d ", set2[j]);
50         pre = set2[j];
51     }
52     j++;
53 }
54 }
```

04-11-2017

05-11-2017

♦ utilità dei file, differenze tra file ASCII e file binari.

♦ accesso a file ASCII

- `FILE * fopen(char [], char []);`
- `int fclose(FILE *);`
- `int fscanf(FILE *, char [], ...);`
- `int fprintf(FILE *, char [], ...);`
- `char * fgets(char [], int, FILE *);`
- `int feof(FILE *);`

♦ lettura fino alla fine del file:

```
1 fscanf(fp, "%c", &car);
2 while(!feof(fp)){
3     /* utilizzo del valore letto e memorizzato in car */
4     /* ... */
5     fscanf(fp, "%c", &car);
6 }
```

♦ accesso a file binari

- `size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);`
- `size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);`

Esercizio 24.1: Numeri pari e dispari

Scrivere un programma che legge 100 valori interi dal file di testo ASCII `./dati.txt` e scrive in numeri pari nel file `./pari.txt` e quelli dispari nel file `./dispari.txt`.

```
1 #include <stdio.h>
2
3 #define N 100
4 #define NOMEFSRC "dati.txt"
5 #define NOMEFP "pari.txt"
6 #define NOMEFD "dispari.txt"
7
8 int main(int argc, char * argv[])
9 {
10     int n, i;
11     FILE * fin, * fd, * fp;
12
13     if(fin = fopen(NOMEFSRC, "r")){
14         if(fp = fopen(NOMEFP, "w")){
15             if(fd = fopen(NOMEFD, "w")){
16                 for(i = 0; i < N; i++){
17                     fscanf(fin, "%d", &n);
18                     if(n % 2)
19                         fprintf(fd, "%d\n", n);
20                     else
21                         fprintf(fp, "%d\n", n);
22                 }
23                 fclose(fd);
24             } else
25                 printf("Errore file %s\n", NOMEFD);
26 }
```

```

27     fclose(fp);
28 } else
29     printf("Errore file %s\n", NOMEFP);
30
31     fclose(fin);
32 } else
33     printf("Errore file %s\n", NOMEFSRC);
34
35 return 0;
36 }

```

Esercizio 24.2: Numeri pari e dispari binari

Scrivere un programma che legge 100 valori interi dal file binario `./dati.bin` e scrive in numeri pari nel file `./pari.txt` e quelli dispari nel file `./dispari.txt`.

```

1 #include <stdio.h>
2
3 #define N 100
4 #define NOMEFSRC "dati.bin"
5 #define NOMEFP "pari.txt"
6 #define NOMEFD "dispari.txt"
7
8 int main(int argc, char * argv[])
9 {
10     int n, i;
11     int dati[N];
12     FILE * fin, * fd, * fp;
13
14
15     if(fin = fopen(NOMEFSRC, "rb")){
16         if(fp = fopen(NOMEFP, "w")){
17             if(fd = fopen(NOMEFD, "w")){
18                 fread(dati, N, sizeof(int), fin);
19                 fclose(fd);
20                 for(i = 0; i < N; i++){
21                     if(dati[i] % 2)
22                         fprintf(fd, "%d\n", dati[i]);
23                     else
24                         fprintf(fp, "%d\n", dati[i]);
25                 }
26             } else
27                 printf("Errore file %s\n", NOMEFD);
28
29             fclose(fp);
30         } else
31             printf("Errore file %s\n", NOMEFP);
32
33         fclose(fin);
34     } else
35         printf("Errore file %s\n", NOMEFSRC);
36
37     return 0;
38 }

```

Esercizio 24.3: Numeri primi .. chissà quanti

Scrivere un programma che legge dal file di testo ASCII `./dati.txt` numeri interi (non si sa quanti dati il file contiene) e scrive i numeri primi nel file `primi.txt`.

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define NOMEFSRC "dati.txt"
5 #define NOMEFDST "primi.txt"
6
7 int isprime(int);
8
9 int main(int argc, char * argv[])
10 {
11     int n, i;
12     FILE * fin, * fout;
13
14     if(fin = fopen(NOMEFSRC, "r")){
15         if(fout = fopen(NOMEFDST, "w")){
16             fscanf(fin, "%d", &n);
17             while(!feof(fin)){
18                 if(n > 1 && isprime(n))
19                     fprintf(fout, "%d\n", n);
20                 fscanf(fin, "%d", &n);
21             }
22             fclose(fout);
23         } else
24             printf("Errore file %s\n", NOMEFDST);
25
26         fclose(fin);
27     } else
28         printf("Errore file %s\n", NOMEFSRC);
29
30     return 0;
31 }
32
33 int isprime(int n)
34 {
35     int i, sq;
36
37     if(n == 2)
38         return 1;
39     if(n <= 1 || n % 2 == 0)
40         return 0;
41     sq = sqrt(n);
42     for(i = 3; i < sq; i = i+2)
43         if(n % i == 0)
44             return 0;
45
46     return 1;
47 }
```

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.

```
1 #include <stdio.h>
2
3 #define LENNOME 40
4 #define LENVOC 32
5
6 int main(int argc, char * argv[])
7 {
8     char nome[LENNOME+1], voc[LENVOC+1];
9     FILE * fp;
10    int num;
11
12    gets(nome);
13    if(fp = fopen(nome, "r")){
14        num = 0;
15        fscanf(fp, "%s", voc);
16        while(!feof(fp)){
17            num++;
18            fscanf(fp, "%s", voc);
19        }
20        fclose(fp);
21        printf("%d\n", num);
22    } else
23        printf("problemi nell'accesso al file %s\n", nome);
24
25    return 0;
26 }
```

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.

```
1 #include <stdio.h>
2
3 #define LENNOME 40
4 #define LENVOC 32
5
6 int main(int argc, char * argv[])
7 {
8     char nome[LENNOME+1], voc[LENVOC+1];
9     FILE * fp;
10    int num;
11
12    gets(nome);
13    if(fp = fopen(nome, "r")){
14        num = 0;
```

```

15     while(fscanf(fp, "%s", voc)){
16         num++;
17         printf("%s\n", voc);
18     }
19     fclose(fp);
20     printf("%d\n", num);
21 } else
22     printf("problemi nell'accesso al file %s\n", nome);
23
24 return 0;
25 }

```

Esercizio 25.2: Copia file lowercase

Scrivere un programma che chiede all'utente il nome di un file (al più 40 caratteri) di testo e scrive un nuovo file di testo, il cui nome è uguale a quello inserito con l'aggiunta del suffisso "_lowercase". Il contenuto del nuovo file deve essere identico a quello del file originale, ma con le lettere maiuscole trasformate in minuscole.

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

```

1 #include <stdio.h>
2 #include <string.h>
3
4 #define LENNOME 40
5 #define LENNOME_LOWER LENNOME+10
6 #define TOLOWER 'a'-'A'
7 #define LOWERCASE "_lowercase"
8
9
10 int main(int argc, char* argv[])
11 {
12     char nomefile_sorgente[LENNOME+1];
13     char nomefile_destinazione[LENNOME_LOWER+1];
14     char carattere;
15     int i;
16     FILE *fp_sorgente;
17     FILE *fp_destinazione;
18
19     gets(nomefile_sorgente);
20     strcpy(nomefile_destinazione, nomefile_sorgente);
21     strcat(nomefile_destinazione, LOWERCASE);
22
23     if(fp_sorgente = fopen(nomefile_sorgente, "r")){
24         if(fp_destinazione = fopen(nomefile_destinazione, "w")){
25             fscanf(fp_sorgente, "%c", &carattere);
26             while(!feof(fp_sorgente)){
27                 if(carattere >= 'A' && carattere <= 'Z')
28                     carattere += TOLOWER;
29                 fprintf(fp_destinazione, "%c", carattere);
30                 fscanf(fp_sorgente, "%c", &carattere);
31             }
32             fclose(fp_destinazione);
33         } else
34             printf("Errore durante l'accesso al file %s\n",
35                 nomefile_destinazione);

```



```

35         fclose(fp_sorgente);
36     }else
37         printf("Errore durante l'accesso al file %s\n", nomefile_sorgente);
38
39     return 0;
40 }

```

Esercizio 25.3: Record di un Videogame

Abbiamo quasi terminato di implementare il nostro nuovo videogame, manca solo la possibilità di memorizzare i migliori record. Il file del salvataggio, che sarà in formato binario, conterrà (in successione) il numero di record che sono salvati all'interno del file e l'elenco dei record.

Come ogni videogioco che si rispetti, a ogni record è associato il nome del suo autore, di esattamente 3 lettere, e il rispettivo punteggio. Inoltre è possibile memorizzare al più i 10 record migliori.

Definire un nuovo tipo di dato che possa contenere le informazioni riguardanti un record. Scrivere quindi i sottoprogrammi `load` e `save` che, dati il nome del file, l'array dei migliori record e tutti i parametri strettamente necessari:

- ◊ legga dal file indicato i record (`load`)
- ◊ scriva i record nel file indicato (`save`)

Facoltativo: Scrivere un sottoprogramma `insert` che, dato l'array dei migliori record, un nuovo record e tutti i parametri strettamente necessari, inserisca il nuovo record tra i migliori, se necessario.

Argomenti: file binario. `typedef`. `sizeof`.

```

1  #include <stdio.h>
2  #include <string.h>
3
4  #define MAXNOME 3
5  #define MAXRECORDS 10
6
7
8  typedef struct record_s{
9
10     char nome[MAXNOME+1];
11     int punteggio;
12
13 } record_t;
14
15
16 void load(char[], record_t[], int*);
17 void save(char[], record_t[], int);
18
19 /* Facoltativo */
20 void insert(record_t[], int*, record_t);
21
22
23 int main(int argc, char* argv[])
24 {
25     return 0;
26 }
27
28
29 void load(char nomefile[], record_t records[], int* n_records)
30 {

```

```

31
32 FILE* fp;
33
34 if(fp = fopen(nomefile, "rb")){
35     fread(&n_records, sizeof(int), 1, fp);
36     fread(records, sizeof(record_t), *n_records, fp);
37     fclose(fp);
38 }else
39     printf("Errore durante l'accesso al file %s", nomefile);
40
41 }
42
43
44 void save(char nomefile[], record_t records[], int n_records)
45 {
46
47     FILE* fp;
48
49     if(fp = fopen(nomefile, "wb")){
50         fwrite(&n_records, sizeof(int), 1, fp);
51         fwrite(records, sizeof(record_t), n_records, fp);
52         fclose(fp);
53     }else
54         printf("Errore durante l'accesso al file %s", nomefile);
55
56 }
57
58
59 /* Facoltativo */
60 void insert(record_t records[], int* n_records, record_t nuovo_record)
61 {
62
63     int i, da_sostituire, i_min_punteggio;
64
65     da_sostituire = -1;
66
67     if(*n_records < MAXRECORDS){
68
69         da_sostituire = *n_records;
70         (*n_records)++;
71
72     }else{
73
74         i_min_punteggio = 0;
75         for(i=1; i<*n_records; i++){
76             if(records[i].punteggio < records[i_min_punteggio].punteggio)
77                 i_min_punteggio = i;
78
79             if(records[i_min_punteggio].punteggio < nuovo_record.punteggio)
80                 da_sostituire = i_min_punteggio;
81
82         }
83
84         if(da_sostituire >= 0){
85             strcpy(records[da_sostituire].nome, nuovo_record.nome);
86             records[da_sostituire].punteggio = nuovo_record.punteggio;
87         }
88     }

```

88
89 }

- ◇ allocazione dinamica: utilità
- ◇ malloc
- ◇ free
- ◇ tipo void *

Esercizio 26.1: Visualizza sequenza di numeri interi in ordine inverso

Scrivere un programma che chiede all'utente la quantità n di valori interi che intende inserire, quindi acquisisce n e li visualizza in ordine inverso.

Argomenti: allocazione dinamica. malloc. free. sizeof.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char * argv[])
5 {
6     int n;
7     int * v, i;
8
9     scanf("%d", &n);
10    v = malloc(n * sizeof(int));
11    if(v != NULL){ /* if(v) oppure if (v = malloc(n * sizeof(int))) */
12        for(i = 0; i < n; i++)
13            scanf("%d", (v + i)); /* scanf("%d", &v[i]); */
14
15        for(i--; i >= 0; i--)
16            printf("%d\n", *(v + i)); /* printf("%d", v[i]); */
17
18        free(v);
19    } else
20        printf("errore allocazione memoria per %d interi\n", n);
21
22    return 0;
23 }
24
```

Esercizio 26.2: Stringa di vocali

Scrivere un sottoprogramma che ricevuta in ingresso una stringa, crea una nuova stringa contenente tutte e sole le vocali contenute nella stringa di ingresso – di dimensioni strettamente idonee a contenere tale stringa – e la restituisce al chiamante.

```
1 char * strvoc(char []);
2
3 int isv(char);
4
5 char * strvoc(char s[])
6 {
7     int n, i, j;
8     char * v;
```

```

9
10 n = 0;
11 for(i = 0; s[i] != '\0'; i++)
12     if(isv(s[i]))
13         n++;
14
15 v = malloc((n+1)*sizeof(char));
16 if(v){
17     j = 0;
18     for(i = 0; s[i] != '\0'; i++)
19         if(isv(s[i])){
20             *(v + j) = s[i];
21             j++;
22         }
23 }
24
25 }

```

Esercizio 26.3: Primi nel file

Scrivere un sottoprogramma che ricevuto in ingresso il nome di un file restituisce al chiamante tutti i numeri primi in esso contenuti.

```

1 int isprime(int);
2
3 int * getprimes(char nome[], int * dim)
4 {
5     FILE * fp;
6     int n, val;
7     int * dati, i;
8
9     if(fp = fopen(nome, "r")){
10         n = 0;
11         fscanf(fp, "%d", &val);
12         while(!feof(fp)){
13             if(isprime(val))
14                 n++;
15             fscanf(fp, "%d", &val);
16         }
17         fclose(fp);
18         if(dati = malloc(n * sizeof(int))){
19             if(fp = fopen(nome, "r")){
20                 i = 0;
21                 fscanf(fp, "%d", &val);
22                 while(!feof(fp)){
23                     if(isprime(val)){
24                         *(dati + i) = val;
25                         i++;
26                     }
27                     fscanf(fp, "%d", &val);
28                 }
29                 fclose(fp);
30             } else {
31                 printf("errore apertura file %s\n", nome);
32                 free(dati); /* non mi serve */
33                 dati = NULL;
34                 n = 0;
35             }
36         }
37     }
38 }

```

```

36     } else {
37         printf("errore allocazione memoria di %d interi", n);
38         n = 0; /* mi dimentico dei valori primi che ho contato , dati e' NULL */
39     }
40 } else {
41     printf("errore apertura file %s\n", nome);
42     dati = NULL;
43     n = 0;
44 }
45
46 *dim = n;
47 return dati;
48
49 }

```

versione in cui si usano due return

```

1 int isprime(int);
2
3 int * getprimes(char nome[], int * dim)
4 {
5     FILE * fp;
6     int n, val;
7     int * dati, i;
8
9     if(fp = fopen(nome, "r")){
10         n = 0;
11         fscanf(fp, "%d", &val);
12         while(!feof(fp)){
13             if(isprime(val))
14                 n++;
15             fscanf(fp, "%d", &val);
16         }
17         fclose(fp);
18         if(dati = malloc(n * sizeof(int))){
19             if(fp = fopen(nome, "r")){
20                 i = 0;
21                 fscanf(fp, "%d", &val);
22                 while(!feof(fp)){
23                     if(isprime(val)){
24                         *(dati + i) = val;
25                         i++;
26                     }
27                     fscanf(fp, "%d", &val);
28                 }
29                 fclose(fp);
30                 *dim = n;
31                 return dati;
32             } else {
33                 printf("errore apertura file %s\n", nome);
34                 free(dati); /* non mi serve */
35             }
36         } else {
37             printf("errore allocazione memoria di %d interi", n);
38         }
39     } else {
40         printf("errore apertura file %s\n", nome);
41     }
42 }

```

```

42  /* se siamo qua e' perche' qualcosa e' andato storto */
43
44  *dim = 0;
45  return NULL;
46
47  }

```

versione in cui si passano entrambi i parametri per indirizzo

```

1  int isprime(int);
2
3  void getprimes(char nome[], int ** dinarray, int * dim)
4  {
5      FILE * fp;
6      int n, val;
7      int * dati, i;
8
9      if(fp = fopen(nome, "r")){
10         n = 0;
11         fscanf(fp, "%d", &val);
12         while(!feof(fp)){
13             if(isprime(val))
14                 n++;
15             fscanf(fp, "%d", &val);
16         }
17         fclose(fp);
18         if(dati = malloc(n * sizeof(int))){
19             if(fp = fopen(nome, "r")){
20                 i = 0;
21                 fscanf(fp, "%d", &val);
22                 while(!feof(fp)){
23                     if(isprime(val)){
24                         *(dati + i) = val;
25                         i++;
26                     }
27                     fscanf(fp, "%d", &val);
28                 }
29                 fclose(fp);
30                 *dim = n;
31                 *dinarray = dati;
32             } else {
33                 printf("errore apertura file %s\n", nome);
34                 free(dati); /* non mi serve */
35             }
36         } else {
37             printf("errore allocazione memoria di %d interi", n);
38         }
39     } else {
40         printf("errore apertura file %s\n", nome);
41     }
42     /* se siamo qua e' perche' qualcosa e' andato storto */
43
44     *dim = 0;
45     *dinarray = NULL;
46
47 }

```

versione in cui si crea una struttura dati e la si restituisce per valore

```

1 typedef struct _info {
2     int * dati;
3     int numdati;
4 } info_t;
5
6 int isprime(int);
7
8 info_t getprimes(char nome[])
9 {
10     FILE * fp;
11     int val, i;
12     info_t primi;
13
14     if(fp = fopen(nome, "r")){
15         primi.numdati = 0;
16         fscanf(fp, "%d", &val);
17         while(!feof(fp)){
18             if(isprime(val))
19                 (primi.numdati)++;
20             fscanf(fp, "%d", &val);
21         }
22         fclose(fp);
23         if(primi.dati = malloc(primi.numdati * sizeof(int))){
24             if(fp = fopen(nome, "r")){
25                 i = 0;
26                 fscanf(fp, "%d", &val);
27                 while(!feof(fp)){
28                     if(isprime(val)){
29                         *(primi.dati + i) = val;
30                         i++;
31                     }
32                     fscanf(fp, "%d", &val);
33                 }
34                 fclose(fp);
35                 return primi;
36             } else {
37                 printf("errore apertura file %s\n", nome);
38                 free(primi.dati); /* non mi serve */
39             }
40         } else {
41             printf("errore allocazione memoria di %d interi", n);
42         }
43     } else {
44         printf("errore apertura file %s\n", nome);
45     }
46     /* se siamo qua e' perche' qualcosa e' andato storto */
47
48     primi.numdati = 0;
49     primi.dati = NULL;
50     return primi;
51 }
52
```

Esercizio 26.4: Anagramma – Proposto

Scrivere un sottoprogramma che riceve in ingresso due stringhe (senz'altro di lunghezza uguale) e restituisce 1 se le stringhe sono una l'anagramma dell'altra, 0 altrimenti.

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 #define MAXLEN 100
6 #define USED 1
7 #define UNUSED 0
8
9 int anagramma(char *s1, char *s2);
10
11 int main(int argc, char * argv[]) {
12     char str1[MAXLEN + 1], str2[MAXLEN + 1];
13
14     scanf("%s", str1):
15     do
16         scanf("%s", str2);
17     while(strlen(str1) != strlen(str2));
18
19     printf("%d\n", anagramma(str1, str2));
20 }
21
22 int anagramma(char *s1, char *s2) {
23     int n, i, j, found;
24     int * tmp;
25     int count;
26
27     n = strlen(s1);
28     if(tmp = malloc(n * sizeof(int))){
29         for (i = 0; i < n; i++){
30             *(tmp + i) = UNUSED;
31         }
32         count = 0;
33
34         for (i = 0; i < n; i++){
35             found = 0;
36             for (j = 0; j < n && !found; j++){
37                 if (*(tmp + j) == UNUSED && s1[i] == s2[j]){
38                     found = 1;
39                     *(tmp + j) = USED;
40                     count++;
41                 }
42             }
43         }
44         free(tmp);
45     }
46     else
47         printf("anagramma: intersezione: errore nell'allocazione di %d char\n",
48             n);
49
50     if (count == n)
51         return 1;
52     return 0;
53 }
```

- ◇ allocazione dinamica: liste concatenate semplici
- ◇ tipo di dato per elementi della lista
 - testa della lista
 - elementi generici
 - ultimo elemento della lista

Esercizio 27.1: Duplica stringa

Scrivere un sottoprogramma che ricevuta in ingresso una stringa ne crea e restituisce una con lo stesso contenuto.

Argomenti: allocazione dinamica. malloc. sizeof. strdup

```
1 char * strdup(char []);
2
3 char * strdup(char s[])
4 {
5     int dim, i;
6     char * dup;
7
8     dim = strlen(s);
9     /* versione 1 */
10    dim++;
11    if(dup = malloc(dim*sizeof(char))) { /* if(dup = malloc((dim+1)*sizeof(char))
12        /*
13        for(i = 0; i < dim; i++) /* for(i = 0; i < dim; i++) */
14            *(dup + i) = s[i]; /* *(dup + i) = s[i]; */
15    } else /* *(dup + i) = s[i]; */
16        printf("errore allocazione %d char\n", dim); /* } */
17    return dup;
18 }
19 void strdupref(char s[], char ** strdup)
20 {
21     int dim, i;
22     char * dup;
23     dim = strlen(s);
24     dim++;
25     if(dup = malloc(dim*sizeof(char))) {
26         for(i = 0; i < dim; i++)
27             *(dup + i) = s[i];
28     } else
29         printf("errore allocazione %d char\n", dim);
30     *strdup = dup;
31     return;
32 }
33
34 void strdupref(char s[], char ** strdup)
35 {
36     int dim, i;
37     dim = strlen(s);
```

```

38 dim++;
39 if(*strdup = malloc(dim*sizeof(char))){
40     for(i = 0; i < dim; i++)
41         *(*strdup + i) = s[i];
42 } else
43     printf("errore allocazione %d char\n", dim);
44 return;
45 }
46
47 char * strdupcpy(char s[])
48 {
49     int dim;
50     char * dup;
51
52     dim = strlen(s);
53     if(dup = malloc((dim + 1)*sizeof(char)))
54         strcpy(dup, s);
55     else
56         printf("errore di allocazione %d char\n", dim);
57     return dup;
58 }
59 }

```

Esercizio 27.2: itos

Scrivere un sottoprogramma che ricevuto in ingresso un valore intero positivo restituisce la stringa i cui caratteri sono le cifre del valore in ingresso.

```

1 #define BASE 10
2 #define N 20
3 int contacifre(int);
4 void reverse(char []);
5
6 int main(int argc, char * argv[])
7 {
8     int n;
9     char * numero;
10
11     scanf("%d", &n);
12     numero = itos(n);
13     if(numero)
14         printf("%d == %s\n", n, numero);
15 }
16
17
18 char * itos(int n)
19 {
20     char * num;
21     int dim, i, p;
22
23     dim = contacifre(n);
24     if(num = malloc((dim + 1)*sizeof(char))){
25         p = pow(BASE, dim-1);
26         for(i = 0; i < dim; i++){
27             *(num + i) = (n / p) + '0';
28             n = n % p;
29             p = p / BASE;
30         }

```

```

31     *(num + i) = '\0';
32 } else
33     printf("errore allocazione %d caratteri\n", dim+1);
34 return num;
35 }
36
37
38 char * itos2(int n)
39 {
40     char * num;
41     int dim;
42
43     dim = contacifre(n);
44     if(num = malloc((dim+1)*sizeof(char))){
45         *(num + dim) = '\0';
46         for(dim--; dim >= 0; dim--){
47             *(num + dim) = (n % BASE) + '0';
48             n = n / BASE;
49         }
50     } else
51         printf("errore allocazione %d caratteri\n", dim+1);
52     return num;
53 }
54
55 char * itos3(int n)
56 {
57     char * num;
58     int i, dim;
59
60     dim = contacifre(n);
61     if(num = malloc((dim + 1)*sizeof(char))){
62         for(i = 0; i < dim; i++){
63             *(num + i) = (n % BASE) + '0';
64             n = n / BASE;
65         }
66         *(num + i) = '\0';
67         reverse(num);
68     } else
69         printf("errore allocazione %d caratteri\n", dim+1);
70     return num;
71 }

```

Esercizio 27.3: Tipo da lista

Definire un tipo di dato opportuno per la realizzazione di una lista concatenata semplice per la gestione dei valori interi.

```

1 typedef struct listi_s {
2     int dato;
3     struct listi_s * next;
4 } listi_t;

```

Esercizio 28.1: Operazioni da file

Scrivere un programma che legga dal file `operazioni.txt` un elenco a priori di lunghezza ignota di operazioni **tra valori interi** da eseguire, una per riga. L'operazione tra due operandi interi è presentata nel formato:

`<operando1> <operazione> <operando2>`

Tra il primo operando, l'operatore e il secondo operando c'è esattamente uno spazio, e gli operatori utilizzati sono solamente `+`, `-`, `*` o `/` (non effettuare controlli, è senz'altro così).

Il programma visualizza i risultati delle operazioni presenti nel file. Nella soluzione avvalersi di un sottoprogramma (che dovrete sviluppare) che riceve in ingresso i due operandi e l'operatore e restituisce al chiamante il risultato dell'operazione. Dopo la sequenza di risultati visualizzata, mettere un 'a-capo'.

Per esempio, se il contenuto del file `operazioni.txt` è il seguente

```
5 + 4
10 / 9
4 * 7
1520 - 5567
184 * 2
```

il programma visualizza: 9 1 28 -4047 368

Ingresso/Uscita:

input: file di testo contenente stringhe

output: una sequenza di interi

```
1 #include <stdio.h>
2
3 int calcola(int op1, char op, int op2);
4
5 int main(int argc, char const *argv[])
6 {
7     FILE *fp;
8     int out, res;
9     int op1, op2;
10    char op;
11
12    fp = fopen("operazioni.txt", "r");
13
14    out = fscanf(fp, "%d %c %d", &op1, &op, &op2);
15    while(out!=EOF) {
16        res = calcola(op1, op, op2);
17        printf("%d ", res);
18        out = fscanf(fp, "%d %c %d", &op1, &op, &op2);
19    }
20
21    printf("\n");
22
```

```

23     return 0;
24 }
25
26 int calcola(int op1, char op, int op2)
27 {
28     int res;
29
30     switch (op)
31     {
32         case '+':
33             res = op1 + op2;
34             break;
35         case '-':
36             res = op1 - op2;
37             break;
38         case '*':
39             res = op1 * op2;
40             break;
41         case '/':
42             res = op1 / op2;
43             break;
44         default:
45             break;
46     }
47     return res;
48 }

```

Esercizio 28.2: Valore mediano di un array

Il file di testo `mediana.txt` contiene una sequenza di al più 100 interi, preceduta da un intero n che rappresenta il numero di valori successivamente presenti nel file (non ci saranno inconsistenze, se n vale 12, nel file ci sono poi 12 valori interi).

Scrivere un programma che visualizzi il valore mediano M dei numeri presenti nel file. Il valore mediano cercato M è il valore che occupa la posizione centrale nella distribuzione ordinata dei valori.

Per risolvere il problema, sviluppare due sottoprogrammi: `ordina` e `mediano`, il primo per ordinare un insieme di valori, il secondo per calcolare il valore mediano.

Dopo il valore visualizzato, mettere un 'a-capo'.

Per esempio, se il contenuto del file è il seguente:

```

10
3
5
97
45
68
32
15
20
1000
24

```

il programma visualizza 32.

Ingresso/Uscita:

input: file di testo contenente valori interi

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

```
1 #include <stdio.h>
2 #define MAX 100
3
4 void ordina(int v[], int n);
5 int mediano(int v[], int n);
6
7 int main(int argc, char const *argv[])
8 {
9     int v[MAX];
10    FILE *fp;
11    int i, num_elem;
12
13    fp = fopen("mediano.txt", "r");
14
15    fscanf(fp, "%d", &num_elem);
16    for (i = 0; i < num_elem; ++i)
17        fscanf(fp, "%d", &v[i]);
18
19    printf("%d\n", mediano(v, num_elem));
20
21    return 0;
22 }
23
24
25 void ordina(int v[], int n)
26 {
27     int i, j, x;
28     /*insertion sort*/
29     for (i = 1; i < n; ++i)
30     {
31         x = v[i];
32         for (j = i - 1; j >= 0 && v[j] > x; j--)
33             v[j + 1] = v[j];
34         v[j + 1] = x;
35     }
36 }
37
38
39 int mediano(int v[], int n)
40 {
41     ordina(v, n);
42
43     return v[n / 2];
44 }
45 }
```

Esercizio 28.3: Prodotto tra matrici

Scrivere un programma che acquisisce i dati di due matrici di valori interi da un file binario `mats.bin`, e ne effettua il prodotto, visualizzando il risultato (seguito dal carattere 'a-capo'). Il contenuto del file binario è il seguente:

```
numero di righe matrice 1
numero di colonne matrice 1
numero di colonne matrice 2
valori matrice 1
valori matrice 2
```

(Si noti che il numero di colonne della matrice 1 è pari al numero di righe della matrice 2 affinché il prodotto possa essere fatto.) Le matrici hanno senz'altro dimensione non superiore a 50x50. Per esempio, se il file contiene (qua visualizzato non in binario):

```
3
4
2
12 54 99 5
2 45 68 33
10 22 888 4
197 -85
264 -8
487 1
-4 -8
```

il programma visualizza:

```
64813 -1393
45258 -726
440218 -170
```

Ingresso/Uscita:

input: un file binario

output: una sequenza di interi organizzati su più righe

```
1 #define MAX 50
2 #include <stdio.h>
3
4 void prodottoMatrici(int A[][MAX], int B[][MAX], int n_r_a, int n_rc, int n_c_b,
5     int res[][MAX]);
6
7 int main(int argc, char const *argv[])
8 {
9     int A[MAX][MAX], B[MAX][MAX], risultato[MAX][MAX];
10    FILE * fp;
11    int r, c, n_r_a, n_rc, n_c_b;
12
13    fp = fopen("prodotto-matrici.txt", "r");
14
15    fscanf(fp, "%d %d %d", &n_r_a, &n_rc, &n_c_b);
16
17    /*Leggi matrice 1*/
18    for (r = 0; r < n_r_a; ++r)
19        for (c = 0; c < n_rc; ++c)
20            fscanf(fp, "%d", &A[r][c]);
```



```

21
22  /*Leggi matrice 2*/
23  for (r = 0; r < n_rc; ++r)
24      for (c = 0; c < n_c_b; ++c)
25          fscanf(fp, "%d", &B[r][c]);
26
27
28
29  prodottoMatrici(A, B, n_r_a, n_rc, n_c_b, risultato);
30
31
32  for (r = 0; r < n_r_a; ++r){
33      for (c = 0; c < n_c_b; ++c)
34          printf("%d ", risultato[r][c]);
35      printf("\n");
36  }
37  printf("\n");
38
39
40  return 0;
41 }
42
43 void prodottoMatrici(int A[][MAX], int B[][MAX], int n_r_a, int n_rc, int n_c_b,
44                     int res[][MAX])
45 {
46     int r, c, i;
47
48     for (r = 0; r < n_r_a; ++r)
49         for (c = 0; c < n_c_b; ++c){
50             res[r][c] = 0;
51             for (i = 0; i < n_rc; ++i)
52                 res[r][c] += A[r][i] * B[i][c];
53         }
54     }

```

Esercizio 28.4: Ruota immagine a destra

Un'immagine in bianco e nero è rappresentata in un sistema di calcolo come un array bidimensionale di valori interi tra compresi nell'intervallo $[0, 255]$, ciascuno rappresentante il valore di intensità del pixel in quella posizione. Scrivere un programma che acquisita un'immagine da un file il cui nome viene chiesto all'utente (ed è al più di 30 caratteri), la ruota di 90 gradi in senso orario. L'immagine ha una dimensione massima di 100×100 pixel.

Il formato del file contenente l'immagine è il seguente:

```

<numero righe immagine> <numero colonne immagine>
<valori dei pixel dell'immagine>

```

Per esempio, se il file contiene:

```

4 6
2 55 79 3 218 24
41 5 46 233 96 9
73 34 11 0 109 68
35 85 17 45 20 219

```

il programma visualizza:

```
35 73 41 2
85 34 5 55
17 11 46 79
45 0 233 3
20 109 96 218
219 68 9 24
```

Ingresso/Uscita:

input: un file di interi

output: sequenze di interi organizzati per righe

```
1 #include <stdio.h>
2 #define MAX 50
3
4 void ruota(int l[][MAX], int n_r, int n_c, int res[][MAX]);
5
6 int main(int argc, char const *argv[])
7 {
8     int l[MAX][MAX], res[MAX][MAX];
9     FILE *fp;
10    int n_r, n_c, r, c;
11
12    fp = fopen("ruota.txt", "r");
13
14    /*Leggi dimensioni immagini*/
15    fscanf(fp, "%d %d", &n_r, &n_c);
16
17    /*Leggi immagine */
18    for (r = 0; r < n_r; ++r)
19        for (c = 0; c < n_c; ++c)
20            fscanf(fp, "%d", &l[r][c]);
21
22    ruota(l, n_r, n_c, res);
23
24    for (r = 0; r < n_c; ++r){
25        for (c = 0; c < n_r; ++c)
26            printf("%d ", res[r][c]);
27        printf("\n");
28    }
29
30    return 0;
31 }
32
33
34
35 void ruota(int l[][MAX], int n_r, int n_c, int res[][MAX])
36 {
37     int r, c;
38
39     for(r = 0; r < n_r; r++)
40         for(c = 0; c < n_c; c++)
41             res[c][n_r-r-1] = l[r][c];
42
43     return ;
44 }
```

Esercizio 28.5: Somma dei quadrati delle differenze tra immagini

Confrontare due immagini (o parti di immagini) e capire quanto sono simili è un problema molto comune in Computer Vision. Un modo per capire quanto due immagini sono simili consiste nel calcolare la somma dei quadrati delle differenze pixel per pixel, metodo noto con il nome di Sum of Squared Differences o SSD.

Scrivere un programma che dati due file contenenti due immagini (con le stesse dimensioni di al più 50x50 pixel) calcoli l'indice SSD, ovvero:

$$SSD = \frac{1}{n} \sum_{i=0}^{n_r} \sum_{j=0}^{n_c} (I_1(i, j) - I_2(i, j))^2 \quad (1)$$

dove n è il numero di pixel di una immagine, n_r è il numero di righe, n_c il numero di colonne e $I_x(i, j)$ è il valore del pixel (i, j) nell'immagine x . Si calcoli il valore di SSD tramite un sottoprogramma opportuno che riceve in ingresso le due immagini e restituisce il valore calcolato. Il programma chiede all'utente il nome dei due file contenenti le immagini (ciascuno al più 30 caratteri).

Il formato del file contenente un immagine è:

```
<numero righe immagine> <numero colonne immagine>
<valori dei pixel dell'immagine>
```

Per esempio, date le immagini:

```
3 4
25 65 48 88
100 251 64 95
65 10 25 45
```

```
3 4
28 65 55 94
100 254 70 95
77 20 30 45
```

il valore visualizzato è 9547.083008

Ingresso/Uscita:

input: due file di testo contenente interi

output: un valore reale (seguito da un carattere 'a-capo')

```
1 #include <stdio.h>
2 #define MAX 50
3
4 float ssd(int I1[][MAX], int I2[][MAX], int n_r, int n_c);
5
6 int main(int argc, char const *argv[])
7 {
8     int I1[MAX][MAX], I2[MAX][MAX];
9     FILE *fp;
10    int n_r, n_c, r, c;
11
```

```

12  fp = fopen("ssd.txt", "r");
13
14  /*Leggi dimensioni immagini*/
15  fscanf(fp, "%d %d", &n_r, &n_c);
16
17  /*Leggi immagine 1*/
18  for (r = 0; r < n_r; ++r)
19      for (c = 0; c < n_c; ++c)
20          fscanf(fp, "%d", &l1[r][c]);
21
22  /*Leggi immagine 2*/
23  for (r = 0; r < n_r; ++r)
24      for (c = 0; c < n_c; ++c)
25          fscanf(fp, "%d", &l2[r][c]);
26
27  printf("%f\n", ssd(l1, l2, n_r, n_c));
28
29  return 0;
30 }
31
32
33
34 float ssd(int l1[][MAX], int l2[][MAX], int n_r, int n_c)
35 {
36     int sum_tot;
37     int r, c;
38
39     sum_tot = 0;
40     for (r = 0; r < n_r; ++r)
41         for (c = 0; c < n_c; ++c)
42             sum_tot += (l1[r][c] - l2[r][c]) * (l1[r][c] - l2[r][c]);
43
44     return (float)sum_tot / (float)(n_r * n_c);
45
46 }

```

Esercizio 29.1: Visualizza in ordine inverso

Scrivere un programma che acquisisce una sequenza di valori interi a priori di lunghezza ignota e che si ritiene terminata quando l'utente inserisce il valore 0, e la visualizza in ordine inverso.

Argomenti: lista concatenata semplice. push. emptylist. printlist

```
1 #define STOP 0
2
3 typedef struct listi_s {
4     int dato;
5     struct listi_s * next;
6 } listi_t;
7
8 listi_t * push(listi_t *, int);
9 listi_t * append(listi_t *, int);
10 listi_t * emptylist(listi_t *);
11 void printlist(listi_t *);
12
13 int main(int argc, char * argv[])
14 {
15     listi_t * head = NULL;
16     int n;
17
18     scanf("%d", &n);
19     while(n != STOP){
20         head = push(head, n);
21         scanf("%d", &n);
22     }
23     printlist(head);
24     head = emptylist(head);
25     return 0;
26 }
27
28
29 void printlist(listi_t * h)
30 {
31     listi_t * p;
32     for(p = h; p != NULL; p = p->next)
33         printf("%d -> ", p->dato);
34     printf("|\\n");
35 }
36
37 listi_t * push(listi_t * h, int val)
38 {
39     listi_t * n;
40
41     n = malloc(sizeof(listi_t)); /* A */
42     if(n != NULL){              /* if(n = malloc(sizeof(listi_t))) */
43         n->dato = val;           /* B */
44         n->next = h;             /* C */
45         h = n;                  /* D */
```

```

46 } else
47     printf("push: errore allocazione %d\n", val);
48
49 return h;
50 }
51
52 listi_t * append(listi_t * h, int val){
53     listi_t * n, * p;
54
55     n = malloc(sizeof(listi_t));          /* A */
56     if(n != NULL){
57         n->dato = val;                    /* B */
58         n->next = NULL;                  /* C */
59         if(h != NULL){ /* lista non vuota */
60             for(p = h; p->next != NULL; p = p->next) /* D */
61                 ;
62             p->next = n;                  /* E */
63         } else /* lista vuota */
64             h = n;                      /* X */
65     } else
66         printf("append: errore allocazione %d\n", val);
67     return h;
68 }
69
70
71
72 listi_t * emptylist(listi_t * h)
73 {
74     listi_t * p;
75
76     while(h != NULL){
77         p = h;          /* A */
78         h = h->next;    /* B */
79         free(p);       /* C */
80     }
81     return h;
82 }

```

Esercizio 29.2: push

Scrivere un sottoprogramma `push` che ricevuta in ingresso una lista ed un valore intero, inserisce l'elemento *in testa* alla lista.

```

1 listi_t * push(listi_t * h, int val)
2 {
3     listi_t * n;
4
5     n = malloc(sizeof(listi_t)); /* A */
6     if(n != NULL){              /* if(n = malloc(sizeof(listi_t))) */
7         n->dato = val;          /* B */
8         n->next = h;            /* C */
9         h = n;                  /* D */
10    } else
11        printf("push: errore allocazione %d\n", val);
12
13    return h;
14 }

```

Argomenti: push. malloc. sizeof.

Esercizio 29.3: append

Scrivere un sottoprogramma `append` che ricevuta in ingresso una lista ed un valore intero, inserisce l'elemento *in coda* alla lista.

```
1 listi_t * append(listi_t * h, int val){
2     listi_t * n, * p;
3
4     n = malloc(sizeof(listi_t));           /* A */
5     if(n != NULL){
6         n->dato = val;                     /* B */
7         n->next = NULL;                   /* C */
8         if(h != NULL){ /* lista non vuota */
9             for(p = h; p->next != NULL; p = p->next) /* D */
10                ;
11            p->next = n;                     /* E */
12        } else /* lista vuota */
13            h = n;                          /* X */
14    } else
15        printf("append: errore allocazione %d\n", val);
16    return h;
17 }
```

Argomenti: append. malloc. sizeof.

Esercizio 29.4: emptylist

Scrivere un sottoprogramma `emptylist` che ricevuta in ingresso una lista ne elimina il contenuto, restituendo la lista vuota.

```
1 listi_t * emptylist(listi_t * h)
2 {
3     listi_t * p;
4
5     while(h != NULL){
6         p = h; /* A */
7         h = h->next; /* B */
8         free(p); /* C */
9     }
10    return h;
11 }
```

Argomenti: emptylist. free.

Esercizio 29.5: itos

Scrivere un sottoprogramma che ricevuta in ingresso una lista la restituisce dopo aver invertito l'ordine dei suoi elementi.

```
1
2 listi_t * invertilista(listi_t * h)
3 {
4     listi_t * hinv = NULL;
5     listi_t * p;
```

```
6
7  for(p = h; p != NULL; p=p->next)
8      hinv = push(hinv, p->dato);
9  h = emptylist(h);
10 return hinv;
11
12 }
```

Argomenti: emptylist.

Esercitazione 9 ◇ liste concatenate semplici

19-11-2019

Esercizio 30.1: Quanta memoria!

Si considerino i 4 stralci di codice qua riportati: per ciascuno di essi si vuole calcolare la quantità di memoria allocata per le variabili dichiarate ed utilizzate (quindi la memoria allocata nel momento in cui si arriva all'ultima istruzione del codice riportato). Si utilizzi l'operatore `sizeof`. Per i primi due stralci, il calcolo è già stato fatto, riportare il risultato nell'apposito spazio il calcolo per gli ultimi due stralci.

<pre>1 int a; 2 ... 3 scanf("%d", &a);</pre>	<pre>1 int v[NUM], i; 2 ... 3 for(i = 0; i < NUM; i++) 4 scanf("%d", &v[i]);</pre>	<pre>1 int * p, i, ndati; 2 ... 3 if(p = (int *)malloc(ndati* 4 sizeof(int))) { 5 for (i = 0; i < ndati; i++) 6 scanf("%d", p+i); 7 ...</pre>	<pre>1 typedef struct _s { 2 int val; 3 struct _s * next; 4 } t_listi; 5 ... 6 t_listi * head = 7 NULL; 8 ... 9 ndati = 0; 10 scanf("%d", &n); 11 while(n != STOP){ 12 head = append(head, 13 n); 14 ndati++; 15 scanf("%d", &n); 16 }</pre>
--	---	--	--

quantità di memoria complessiva allocata:

<pre>1 sizeof(int) 2 3 .</pre>	<pre>1 NUM * sizeof(int) 2 + 3 sizeof(int) 4 .</pre>	<pre>1 2 3 .</pre>	<pre>1 2 3 .</pre>
--	--	----------------------------	----------------------------

Esercizio 30.2: Insieme unione

Scrivere un sottoprogramma che ricevette in ingresso due liste che rappresentano insiemi di valori interi, restituisce una nuova lista rappresentante l'unione dei due insiemi.

Argomenti: Unione di insiemi. find.

```
1 listi_t * find(listi_t *, int);  
2  
3 listi_t * setunion(listi_t * set1, listi_t * set2)  
4 {  
5   listi_t * u = NULL;  
6   listi_t * p;  
7  
8   /* tutto il primo insieme */  
9   for(p = set1; p; p = p->next)
```

```

10     u = append(u, p->val);
11
12     /* aggiungo quelli del secondo insieme che non appartengono al primo */
13     for(p = set2; p; p = p->next)
14         if(!find(set1, p->val))
15             u = append(u, p->val);
16
17     return u;
18 }
19
20 listi_t * find(listi_t * h, int val)
21 {
22     listi_t * p;
23
24     for(p = h; p; p = p->next)
25         if(p->dato == val)
26             return p;
27
28     return NULL;
29 }

```

Esercizio 30.3: Elimina doppioni adiacenti

Scrivere un sottoprogramma che ricevuta in ingresso una lista elimina dalla lista le ripetizioni di elementi *adiacenti*.

Argomenti: lista concatenata semplice. deleteptr.

Per esempio, se la lista in ingresso è quella di seguito riportata e `start` è 1:

$3 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow 2 \rightarrow 1 \rightarrow |$

il sottoprogramma restituisce la lista seguente

$3 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow 1 \rightarrow |$

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct listi_s {
5     int dato;
6     struct listi_s * next;
7 } listi_t;
8
9 listi_t * fill(int);
10 listi_t * emptylist(listi_t *);
11 listi_t * removedup(listi_t *);
12 listi_t * append(listi_t *, int);
13 listi_t * deleteptr(listi_t *, listi_t *);
14 void printlist(listi_t *);
15
16 int main(int argc, char * argv[])
17 {
18     listi_t * h = NULL;
19     int num;
20

```

```

21  scanf("%d", &num);
22  h = fill(num);
23  h = removedup(h);
24  printlist(h);
25  h = emptylist(h);
26
27  return 0;
28 }
29
30 listi_t * removedup(listi_t * h)
31 {
32     listi_t * p;
33
34
35     for(p = h; p; )
36         if(p->next && p->dato == p->next->dato) /* se c'e' un elemento dopo ed e'
           uguale */
37             h = deleteptr(h, p->next);
38         else
39             p = p->next;
40
41     return h;
42 }
43
44
45 listi_t * deleteptr(listi_t * h, listi_t * d)
46 {
47     listi_t * p;
48
49     if(h == d){ /* se devo eliminare il primo */
50         h = h->next;
51         free(d);
52     } else {
53         p = h;
54         while(p->next != d)
55             p = p->next;
56         /* p punta all'elemento che precede quello da eliminare */
57         p->next = d->next;
58         free(d);
59     }
60     return h;
61 }
62
63 listi_t * emptylist(listi_t * h)
64 {
65     listi_t * p;
66
67     while(h != NULL){
68         p = h;      /* A */
69         h = h->next; /* B */
70         free(p);    /* C */
71     }
72     return h;
73 }
74
75 void printlist(listi_t * h)
76 {

```

```

77  listi_t * p;
78  for(p = h; p != NULL; p = p->next)
79      printf("%d -> ", p->dato);
80  printf("\n");
81  }
82
83  listi_t * fill(int num){
84      listi_t * h = NULL;
85      int i, val;
86
87      for(i = 0; i < num; i++){
88          scanf("%d", &val);
89          h = append(h, val);
90      }
91      return h;
92  }
93
94  listi_t * append(listi_t * h, int val){
95      listi_t * n, * p;
96
97      n = malloc(sizeof(listi_t));          /* A */
98      if(n != NULL){
99          n->dato = val;                    /* B */
100         n->next = NULL;                   /* C */
101         if(h != NULL){ /* lista non vuota */
102             for(p = h; p->next != NULL; p = p->next) /* D */
103                 ;
104             p->next = n;                   /* E */
105         } else /* lista vuota */
106             h = n;                        /* X */
107     } else
108         printf("append: errore allocazione %d\n", val);
109     return h;
110 }

```

Esercizio 30.4: Da lista a insieme

Scrivere un sottoprogramma che ricevuta in ingresso una lista la manipola facendola diventare un insieme (elimina gli elementi ripetuti).

Argomenti: lista concatenata semplice.

Per esempio, se la lista in ingresso è quella di seguito riportata e `start` è 1:

$$3 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow 2 \rightarrow 1 \rightarrow |$$

il sottoprogramma restituisce la lista seguente

$$3 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow |$$

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct listi_s {
5      int dato;
6      struct listi_s * next;

```

```

7 } listi_t;
8
9 listi_t * fill(int);
10 listi_t * emptylist(listi_t *);
11 listi_t * removedup(listi_t *);
12 listi_t * append(listi_t *, int);
13 listi_t * deleteptr(listi_t *, listi_t *);
14 void printlist(listi_t *);
15
16 int main(int argc, char * argv[])
17 {
18     listi_t * h = NULL;
19     int num;
20
21     scanf("%d", &num);
22     h = fill(num);
23     h = removedup(h);
24     printlist(h);
25     h = emptylist(h);
26
27     return 0;
28 }
29
30 listi_t * removedup(listi_t * h)
31 {
32     listi_t * p;
33
34
35     for(p = h; p; )
36         if(p->next && p->dato == p->next->dato) /* se c'e' un elemento dopo ed e'
           uguale */
37             h = deleteptr(h, p->next);
38         else
39             p = p->next;
40
41     return h;
42 }
43
44
45 listi_t * deleteptr(listi_t * h, listi_t * d)
46 {
47     listi_t * p;
48
49     if(h == d){ /* se devo eliminare il primo */
50         h = h->next;
51         free(d);
52     } else {
53         p = h;
54         while(p->next != d)
55             p = p->next;
56         /* p punta all'elemento che precede quello da eliminare */
57         p->next = d->next;
58         free(d);
59     }
60     return h;
61 }
62

```

```

63 listi_t * emptylist(listi_t * h)
64 {
65     listi_t * p;
66
67     while(h != NULL){
68         p = h;          /* A */
69         h = h->next;     /* B */
70         free(p);        /* C */
71     }
72     return h;
73 }
74
75 void printlist(listi_t * h)
76 {
77     listi_t * p;
78     for(p = h; p != NULL; p = p->next)
79         printf("%d -> ", p->dato);
80     printf("\n");
81 }
82
83 listi_t * fill(int num){
84     listi_t * h = NULL;
85     int i, val;
86
87     for(i = 0; i < num; i++){
88         scanf("%d", &val);
89         h = append(h, val);
90     }
91     return h;
92 }
93
94 listi_t * append(listi_t * h, int val){
95     listi_t * n, * p;
96
97     n = malloc(sizeof(listi_t));          /* A */
98     if(n != NULL){
99         n->dato = val;                     /* B */
100        n->next = NULL;                    /* C */
101        if(h != NULL){ /* lista non vuota */
102            for(p = h; p->next != NULL; p = p->next) /* D */
103                ;
104            p->next = n;                     /* E */
105        } else /* lista vuota */
106            h = n;                          /* X */
107        } else
108            printf("append: errore allocazione %d\n", val);
109        return h;
110 }

```

Esercizio 30.5: Riempi lista

Scrivere un sottoprogramma che ricevuto in ingresso un valore intero `num` crea una lista chiedendo all'utente `num` valori interi e la restituisce all'utente. Anche se sappiamo quanti dati vogliamo memorizzare (e quindi una unica istruzione `malloc` sarebbe appropriata) lo scopo è creare una lista da usare per collaudare gli altri sottoprogrammi.

Argomenti: lista concatenata semplice. `fill`.

```
1 /* riempie una lista con num valori chiesti all'utente */
2 listi_t * fill(int num){
3     listi_t * h = NULL;
4     int i, val;
5
6     for(i = 0; i < num; i++){
7         scanf("%d", &val);
8         h = append(h, val);
9     }
10    return h;
11 }
```

- ◇ passo base
- ◇ passo induttivo
- ◇ condizione di termine della ricorsione
- ◇ stack delle chiamate
- ◇ visibilità variabili locali
- ◇ variabili `static`

Esercizio 31.1: Fattoriale – versione ricorsiva

Scrivere un sottoprogramma ricorsivo che calcola e restituisce il fattoriale di un intero.

Argomenti: sottoprogramma. ricorsione.

```
1
2 int fattoriale_r(int n)
3 {
4     if(n == 0 || n == 1)
5         return 1;
6
7     return n * fattoriale_r(n-1);
8 }
```

Esercizio 31.2: Lunghezza di una lista – versione ricorsiva

Scrivere un sottoprogramma ricorsivo che ricevuta in ingresso una lista, calcola e restituisce la sua lunghezza.

Argomenti: sottoprogramma. ricorsione. length

```
1 int strlen_r(char s[])
2 {
3     if(s[0] == '\0')
4         return 0;
5     return 1 + strlen_r(&s[1]);
6 }
7
8 int strlen_r(char s[])
9 {
10     int i;
11     i = 0;
12     if(s[i] == '\0')
13         return 0;
14     return 1 + strlen_r(&s[i+1]);
15 }
16
17 int strlen_r(char s[])
```



```

18 {
19     return strlen2_r(s, 0);
20 }
21
22 int strlen2_r(char s[], int pos)
23 {
24     if(s[pos] == '\0')
25         return 0;
26
27     return 1 + strlen2_r(s, pos+1);
28 }
29
30 int strlen_rs(char s[]) {
31     static int pos = 0;
32
33     if(s[pos] == '\0')
34         return 0;
35     pos++;
36     return 1 + strlen_rs(s);
37 }
38 }

```

Esercizio 31.3: Lunghezza di una stringa – versione ricorsiva

Scrivere un sottoprogramma ricorsivo che ricevuta in ingresso una stringa, calcola e restituisce la sua lunghezza.

Argomenti: sottoprogramma. ricorsione. stringhe.

```

1 int strlen_r(char s[])
2 {
3     if(s[0] == '\0')
4         return 0;
5     return 1 + strlen_r(&s[1]);
6 }
7
8 int strlen_r(char s[])
9 {
10     int i;
11     i = 0;
12     if(s[i] == '\0')
13         return 0;
14     return 1 + strlen_r(&s[i+1]);
15 }
16
17 int strlen_r(char s[])
18 {
19     return strlen2_r(s, 0);
20 }
21
22 int strlen2_r(char s[], int pos)
23 {
24     if(s[pos] == '\0')
25         return 0;
26
27     return 1 + strlen2_r(s, pos+1);

```

```

28 }
29
30 int strlen_rs(char s[]) {
31     static int pos = 0;
32
33     if(s[pos] == '\0')
34         return 0;
35     pos++;
36     return 1 + strlen_rs(s);
37
38 }

```

Esercizio 31.4: Carattere nella stringa – versione ricorsiva

Scrivere un sottoprogramma ricorsivo che ricevuta in ingresso una stringa ed un carattere, calcola e restituisce il numero di volte che il carattere compare nella stringa.

Argomenti: sottoprogramma. ricorsione. stringhe. indirizzo di un elemento dell'array.

```

1
2
3
4
5 int contaoccorrenze_r(char s[], char c)
6 {
7     int count;
8
9     if(s[0] == '\0')
10         return 0;
11
12     if(s[0] == c)
13         count = 1;
14     else
15         count = 0;
16
17     return count + contaoccorrenze_r(&s[1], c);
18 }

```

Esercizio 31.5: Variabili static

Programma che chiama un sottoprogramma con una variabile dichiarata `static`, che viene allocata non sullo stack e il cui valore è persistente rispetto a chiamate successive.

```

1 #include <stdio.h>
2 #define N 10
3
4 void teststatic();
5
6 int main(int argc, char * argv[])
7 {
8     int val;
9     val = N;
10    printf("[%p] %d\n", &val, val);
11    teststatic(val);
12
13    return 0;
14 }

```

```

15
16 void teststatic(int num)
17 {
18     static int count = 0; /* inizializzata in fase di dichiarazione */
19     /* puo' essere inizializzata una volta sola ... */
20     int change = 0;
21
22     if (num == 0)
23         return ;
24
25     printf("[%p] %d\t[%p]%d\n", &change, change, &count, count);
26     change++;
27     count++;
28     teststatic(num - 1);
29     return;
30 }

```

Esercizio 31.6: Scompatta lista

Scrivere un sottoprogramma `extract` che riceve in ingresso una lista per la gestione dei numeri interi, e un intero `start` che vale senz'altro 0 o 1 (non è necessario gestire il caso in cui non sia così). La lista *codifica* un'informazione binaria: il valore del primo elemento indica quante volte consecutive compare il bit `start`, il secondo elemento indica quante volte compare il complemento di `start`, il terzo elemento quante volte compare il bit `start` e così fino alla fine. Il sottoprogramma scompatta tale informazione e restituisce **una nuova lista** i cui elementi contengono ciascuno 0 o 1.

Non è necessario definire due tipi diversi di dato, poichè il contenuto dell'elemento della lista è comunque sempre un intero.

Per esempio, se la lista in ingresso è quella di seguito riportata e `start` è 1:

$$3 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow 2 \rightarrow 1 \rightarrow |$$

il sottoprogramma restituisce la lista seguente

$$1 \rightarrow 1 \rightarrow 1 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 1 \rightarrow 0 \rightarrow 0 \rightarrow 1 \rightarrow 1 \rightarrow 0 \rightarrow |$$

Argomenti: lista concatenata semplice.

Tratto dal tema d'esame del 09/09/2019

```

1 typedef struct listi_s {
2     int dato;
3     struct listi_s * next;
4 } listi_t;
5
6 listi_t * extract(listi_t * h, int start)
7 {
8     listi_t * bin = NULL, *p;
9     int i;
10
11     for(p = h; p ; p = p->next){
12         for(i = 0; i < p->dato; i++){
13             bin = append(bin, start);
14             if(start == 0) /* start = !start; */
15                 start = 1;

```

```

16     else
17         start = 0;
18     }
19     return bin;
20 }
21
22 listi_t * extract(listi_t * h, int start)
23 {
24     listi_t * bin = NULL;
25     int i;
26
27     for(; h ; h = h->next){
28         for(i = 0; i < h->dato; i++)
29             bin = append(bin, start);
30         if(start == 0)    /* start = !start; */
31             start = 1;
32         else
33             start = 0;
34     }
35     return bin;
36 }
37
38 listi_t * extract(listi_t * h, int start)
39 {
40     listi_t * bin = NULL, * p;
41     int i;
42     int dis, opp;
43
44     if(start == 1)
45         opp = 0;
46     else
47         opp = 1;
48     for(p = h, dis = 1; p; p = p->next, dis++)
49         if(dis % 2)
50             for(i = 0; i < p->dato; i++)
51                 bin = append(bin, start);
52         else
53             for(i = 0; i < p->dato; i++)
54                 bin = append(bin, opp);
55
56     return bin;
57 }
58
59 listi_t * extract(listi_t * h, int start)
60 {
61     listi_t * bin = NULL, * p;
62     int i;
63     int dis, opp;
64
65     if(start == 1)
66         opp = 0;
67     else
68         opp = 1;
69     for(p = h; p; ){
70         for(i = 0; i < p->dato; i++)
71             bin = append(bin, start);
72         p = p->next;

```

```
73     if(p){
74         for(i = 0; i < p->dato; i++)
75             bin = append(bin, opp);
76         p = p->next;
77     }
78 }
79 return bin;
80 }
```

Esercizio 32.1: Campi e alberi

Alberi

Si consideri il gioco “Alberi”, costituito da una griglia quadrata, in cui in ogni elemento può esserci un albero o meno. Le dimensioni della griglia variano da gioco a gioco, ma sono di al più 12 elementi per lato. La griglia è occupata da aree di colore diverso, cui appartengono uno o più elementi, e le regole per la corretta presenza degli alberi sono le seguenti: i) ci devono essere 2 alberi per riga, ii) ci devono essere 2 alberi per colonna, iii) ci devono essere 2 alberi per colore, iv) negli elementi adiacenti ad un albero non ci devono essere alberi. Gli schemi più piccoli (con dimensione minore o uguale a 6) contengono 1 solo albero per riga, colonna e colore.

Si realizzi un programma che acquisisce in ingresso la specifica della disposizione delle aree e dei colori (nel formato indicato di seguito) e visualizza 1 se lo schema ricevuto in ingresso è corretto, 0 altrimenti, seguito dal carattere 'a capo'. Si organizzi il codice in sottoprogrammi.

Si suggerisce di sviluppare, tra gli altri, i seguenti sottoprogrammi:

`checkGriglia`

restituisce 1 se il numero di alberi e la loro disposizione è corretta

`checkRiga`

restituisce 1 se il numero di alberi nella riga della griglia è quello richiesto

`checkColonna`

restituisce 1 se il numero di alberi nella colonna della griglia è quello richiesto

`checkColore`

restituisce 1 se il numero di alberi in un'area di colore della griglia è quello richiesto

`checkDistanza`

restituisce 1 se quando si incontra un albero nella griglia esso non ha alberi negli elementi adiacenti.

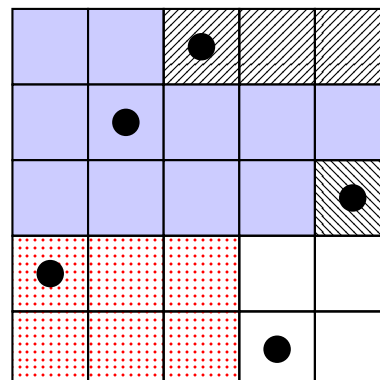
Ingresso/Uscita:

input: un insieme di numeri interi e caratteri

output: un intero

Alcuni casi di test per il collaudo:

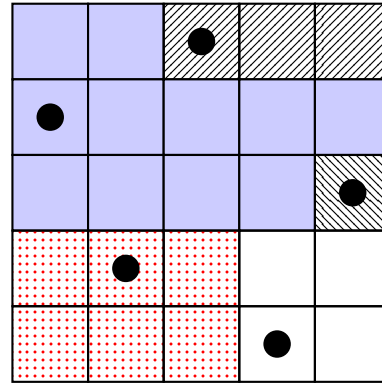
```
5 % dimensione del campo di gioco
AABBB % un carattere per ogni colore
AAAAA
AAAAC
DDDEE
DDDEE
0 2 % coordinate degli alberi
1 1
2 4
3 0
4 3
0
```



```

5 % dimensione del campo di gioco
AABBB % un carattere per ogni colore
AAAAA
AAAAC
DDDEE
DDDEE
0 2 % coordinate degli alberi
1 0
2 4
3 1
4 3
1

```



```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX_GAME 10
5 #define MAX_TREES 2 * MAX_GAME
6 #define TREES_ELEM_LARGE 2
7 #define TREES_ELEM_SMALL 1
8 #define THRDIM 5
9
10 /**
11  * Struttura per tener traccia delle posizioni degli alberi nella griglia
12  */
13 typedef struct point_s{
14     int x;
15     int y;
16 } point_t;
17
18 int loadGrid(int[][MAX_GAME], int[], int);
19
20 void loadTrees(point_t[], int);
21
22 int contains(int[], int, int);
23
24 int checkRow(point_t[], int, int);
25 int checkColumn(point_t[], int, int);
26 int checkColor(int[][MAX_GAME], int[], int, point_t[], int, int);
27 int checkDistance(point_t[], int);
28 int checkGrid(int[][MAX_GAME], int[], int, point_t[], int, int);
29
30 int main(int argc, char * argv[]) {
31     int mat[MAX_GAME][MAX_GAME];
32     int colors[MAX_GAME];
33     point_t trees[MAX_TREES];
34     int gameDimension, nColors, nTrees, result;
35     int treesElem; /* il numero di alberi per elemento in base alla dimensione
36                    del gioco, 1 o 2*/
37
38     scanf("%d", &gameDimension);
39     nColors = loadGrid(mat, colors, gameDimension);
40     scanf("%d", &nTrees);
41     loadTrees(trees, nTrees);

```

```

42     if(gameDimension > THRDIM)
43         treesElem = TREES_ELEM_LARGE;
44     else
45         treesElem = TREES_ELEM_SMALL;
46
47     if(treesElem * gameDimension == nTrees && nColors == gameDimension)
48         result = checkGrid(mat, colors, nColors, trees, nTrees, treesElem);
49     else
50         result = 0;
51     printf("%d\n", result);
52
53     return 0;
54 }
55
56 int loadGrid(int solMat[][MAX_GAME], int colors[], int gameDim)
57 {
58     int i, j, ncol;
59     int colcode;
60     char gridRow[MAX_GAME+1];
61
62     ncol = 0;
63     for(i = 0; i < gameDim; i++){
64         /* se i numeri sono scritti uno di seguito all'altro li leggo come
65         stringhe e recupero gli interi */
66         scanf("%s", gridRow); /* sequenze di numeri per i colori */
67         for(j = 0; j < gameDim; j++){
68             solMat[i][j] = gridRow[j] - '0'; /* sequenza di numeri per i colori
69             */
70             /* scanf("%d", &solMat[i][j]); se numeri interi separati da spazi */
71             if(!contains(colors, ncol, solMat[i][j])){
72                 colors[ncol] = solMat[i][j];
73                 ncol++;
74             }
75         }
76     }
77     printf("numero colori: %d\n", ncol);
78     return ncol;
79 }
80
81 void loadTrees(point_t trees[], int nTrees)
82 {
83     int i;
84
85     for(i = 0; i < nTrees; i++){
86         scanf("%d", &trees[i].x);
87         scanf("%d", &trees[i].y);
88     }
89     return;
90 }
91
92 int contains(int vet[], int dim, int val)
93 {
94     int i;
95
96     for(i = 0; i < dim; i++)
97         if(vet[i] == val)
98             return 1;

```



```

97     return 0;
98 }
99
100 int checkRow(point_t trees[], int nTrees, int treesElem)
101 {
102     int i, j, n, currRow;
103
104     for (i = 0; i < nTrees; i++) {
105         currRow = trees[i].x;
106         for (j = i, n = 0; j < nTrees; j++)
107             if (trees[j].x == currRow)
108                 n++;
109
110         if (n != treesElem)
111             return 0;
112     }
113     return 1;
114 }
115
116 int checkColumn(point_t trees[], int nTrees, int treesElem){
117     int i, j, n, currColumn;
118
119     for (i = 0; i < nTrees; i++) {
120         currColumn = trees[i].y;
121         for (j = i, n = 0; j < nTrees; j++)
122             if (trees[j].y == currColumn)
123                 n++;
124
125         if (n != treesElem)
126             return 0;
127     }
128     return 1;
129 }
130
131 int checkColor(int solMat[][MAX_GAME], int colors[], int nColors, point_t trees
132 [], int nTrees, int treesElem)
133 {
134     int icolor, itree, n;
135
136     for(icolor = 0; icolor < nColors; icolor++) {
137         for (itree = 0, n = 0; itree < nTrees; itree++)
138             if (solMat[trees[itree].x][trees[itree].y] == colors[icolor])
139                 n++;
140         if (n != treesElem)
141             return 0;
142     }
143     return 1;
144 }
145
146 int checkDistance(point_t trees[], int nTrees)
147 {
148     int i, j;
149
150     for(i = 0; i < nTrees - 1; i++)
151         for(j = i + 1; j < nTrees; j++) {
152             if(abs(trees[i].x - trees[j].x) <= 1 && abs(trees[i].y - trees[j].y)
153 <= 1)

```

```

152         return 0;
153     }
154     return 1;
155 }
156
157 int checkGrid(int solMat[][MAX_GAME], int colors[], int nColors, point_t trees
158 [], int nTrees, int treesElem)
159 {
160     if(checkRow(trees, nTrees, treesElem) &&
161        checkColumn(trees, nTrees, treesElem) &&
162        checkColor(solMat, colors, nColors, trees, nTrees, treesElem) &&
163        checkDistance(trees, nTrees))
164         return 1;
165     return 0;
166 }

```

Esercitazione 10 ◇ variabili globali

25-11-2019

◇ variabili globali

Esercizio 33.1: Da array a lista istogramma

Scrivere un sottoprogramma `array2istogramma` che riceve in ingresso un array di valori interi e qualsiasi altro parametro ritenuto strettamente necessario e restituisce una lista *istogramma* che rappresenta un'unica volta tutti i valori presenti nell'array con il numero di occorrenze. La lista è ordinata in senso crescente rispetto ai valori (non rispetto al numero di occorrenze).

Scrivere poi un programma che acquisisce da file binario `dati.bin` 100 valori interi, chiama il sottoprogramma `array2istogramma` e visualizza un istogramma orizzontale utilizzando il carattere `*`, ordinato prima rispetto al valore, poi ordinato in senso crescente in base al numero di occorrenze.

Per esempio, se il file contiene i seguenti valori (20 per questioni di spazio):

4 5 1 2 6 8 9 1 7 2 4 5 6 1 4 8 2 0 3 11

il programma visualizza:

```

0 : *
1 : ***
2 : ***
3 : *
4 : ***
5 : **
6 : **
7 : *
8 : **
9 : *
11 : *

```

```

0 : *
3 : *
7 : *

```

```
9 : *
11 : *
5 : **
6 : **
8 : **
1 : ***
2 : ***
4 : ***
```

Argomenti: Lista concatenata semplice. Conta occorrenze. increasing.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define N 20
5 #define SYM '*'
6
7 typedef struct _s {
8     int valore;
9     int volte;
10    struct _s * next;
11 } stat_t;
12
13 stat_t * increasing(stat_t *, int);
14 stat_t * find(stat_t *, int);
15 stat_t * empty(stat_t *);
16 stat_t * gethistogram(int [], int);
17 void printhistogram(stat_t *);
18
19 int main(int argc, char * argv[])
20 {
21     int v[N], i;
22     stat_t * h;
23
24     for(i = 0; i < N; i++)
25         scanf("%d", &v[i]);
26
27     h = gethistogram(v, N);
28     printhistogram(h);
29     h = empty(h);
30
31     return 0;
32 }
33
34 stat_t * gethistogram(int v[], int dim)
35 {
36     int i;
37     stat_t * hlist = NULL;
38     stat_t * p;
39
40     for(i = 0; i < dim; i++)
41         if(p = find(hlist, v[i]))
42             p->volte++;
43         else
44             hlist = increasing(hlist, v[i]);
```

```

45     return hlist;
46 }
47
48 stat_t * increasing(stat_t * h, int val){
49     stat_t * pre, * elem;
50
51     if(elem = malloc(sizeof(stat_t))){
52         elem->valore = val;
53         elem->volte = 1;
54
55         if(!h || val < h->valore){ /*diventa il primo elemento della lista */
56             elem->next = h;
57             h = elem;
58         } else { /* e' uno qualsiasi */
59             pre = h;
60             while(pre->next && pre->next->valore < val)
61                 pre = pre->next;
62             /* ci si ferma su quello prima di dove inserire , eventualmente l'ultimo */
63             elem->next = pre->next;
64             pre->next = elem;
65         }
66     } else
67         printf("increasing: errore di memoria %d\n", val);
68
69     return h;
70 }
71
72 stat_t * find(stat_t * h, int val)
73 {
74     for(; h; h = h->next)
75         if(h->valore == val)
76             return h;
77     return NULL;
78 }
79
80 stat_t * empty(stat_t * h)
81 {
82     stat_t * p;
83     while(h){
84         p = h->next;
85         free(h);
86         h = p;
87     }
88     return h;
89 }
90
91
92 void printhistogram(stat_t * h)
93 {
94     int i;
95     for(; h; h = h->next){
96         printf("%d\t: ", h->valore);
97         for(i = 0; i < h->volte; i++)
98             printf("%c", SYM);
99         printf("\n");
100     }
101 }

```

Esercizio 33.2: Valori sotto la media

Scrivere un sottoprogramma che riceve in ingresso un array di valori interi e qualsiasi altro parametro ritenuto strettamente necessario e restituisce una lista contenente tutti e soli i valori dell'array che sono minori o uguali alla media dei valori contenuti nell'array stesso.

Argomenti: ista concatenata semplice. algoritmo. calcolo della media dei valori.

Tratto dal tema d'esame del 04/09/2015

```
1 typedef struct listi_s {
2     int dato;
3     struct listi_s *next;
4 } listi_t;
5
6 listi_t * append(listi_t *, int);
7 listi_t * below_eq_average(int [], int);
8
9 listi_t * below_eq_average(int a[], int dim){
10     int tot, i;
11     float media;
12     listi_t * head = NULL;
13
14     for(i = 0, tot = 0; i < dim; i++)
15         tot += a[i];
16     if(dim > 0)
17         media = (float) tot / dim;
18
19     for(i = 0; i < dim; i++)
20         if(a[i] <= media)
21             head = append(head, a[i]);
22
23     return head;
24 }
```

Esercitazione 11 ♦ riepilogo

26-11-2019

Esercitazione 12 ♦ riepilogo

28-11-2019

Esercitazione 14 ◇ liste ripasso

09-12-2019 (CB)

Analisi esercizi: partecipazione

Numero esercizi svolti: **084**
Numero esercizi proposti: **008**