NUMERO: 1112 DATA: 16/09/2014

A P P U N T I

STUDENTE: Rizzi

MATERIA: Informatica

Prof. Servetti

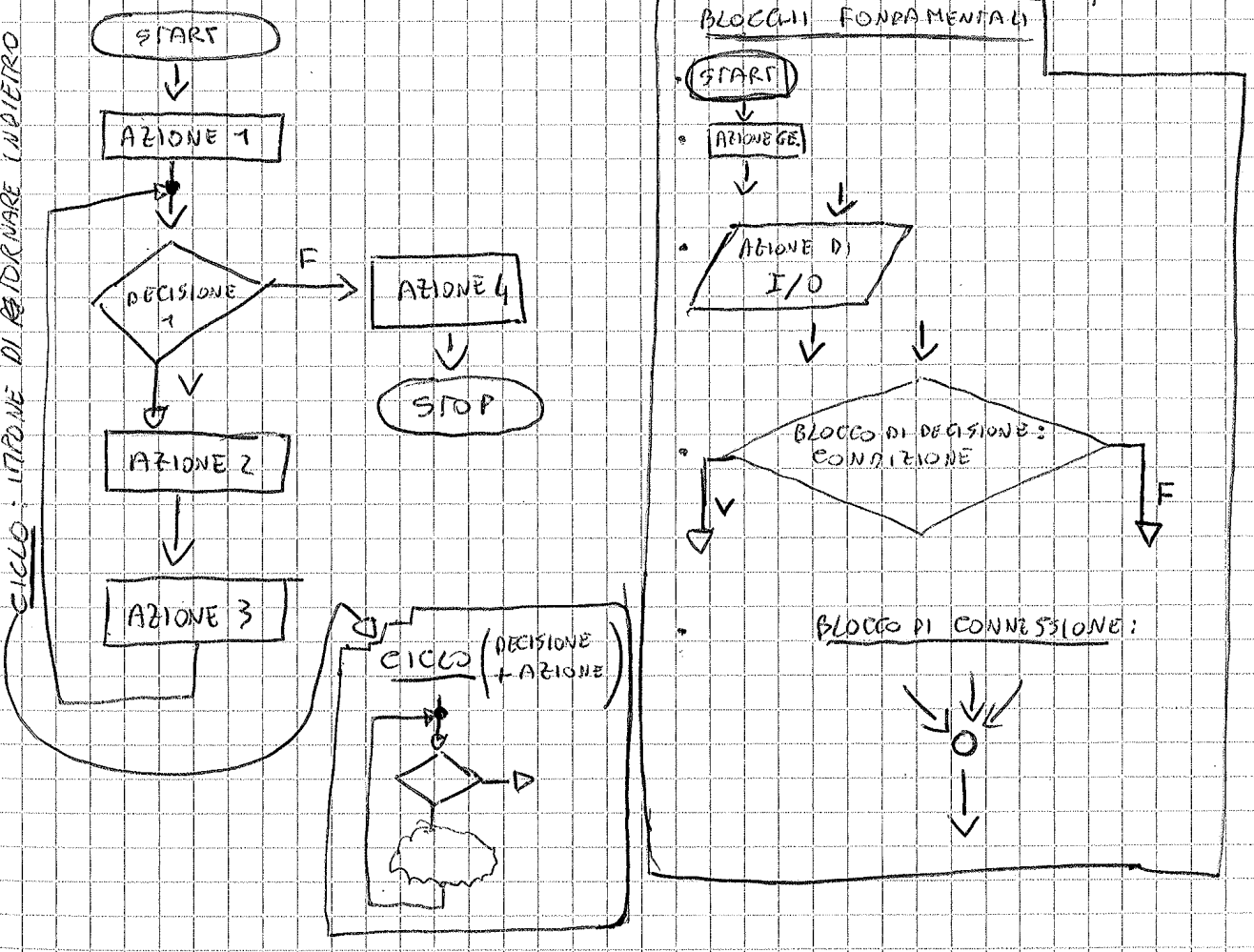
COSTRUZIONE DI UN PROGRAMMA:

ALGORITMO = descrizione precisa di una sequenza (finita) di azioni da eseguire per giungere alla soluzione di un problema. Le azioni devono essere definite e non ambigue, l'algoritmo lavora su DATI. (DATI + ALGORITMO) = PROGRAMMA → USCITA
Esempio = COTTURA DELLA PASTA. INGRESSO (INPUT) (OUTPUT)

Il calcolatore ha bisogno di condizioni e di cui è capace e una serie di azioni.
(differenzia il calcolo del programma)
- Se la CONDIZIONE non è verificata torna indietro, altrimenti continua → si differenzia quello dal caso di INPUT e OUTPUT (a seguito delle quali l'azione è la stessa).

METODI PIÙ FORMALI:
- PSEUDO-CODICE (troppo vicino al linguaggio di programmazione):
- DIAGRAMMA DI FLUSSO (FLOW-CHART): Sono una rappresentazione grafica di un'evoluzione logica di un problema. Sono costituiti da BLOCCHI e ARCHI (collegati da)
  - AZIONI ELEM. costituite da: - azione - decisione - input/output

Esempio:



• TRASCRIZIONE DEL PROBLEMA IN LINGUAGGIO C

```
int main()
{
    /* PARTE DICHIARATIVA */
int A;
int B;
    /* START */
    /* IN A, B */
scanf("%i %i", &A, &B);
    /* CONDIZIONE */
if(A>B)
{ /* RAMO VERO: OUT A */
printf("%i", A);
}
else
{ /* RAMO FALSO: OUT B */
printf("%i", B);
}
return 0;
}
    /* STOP */
```

[NOTE: - le parentesi {} servono per creare dei BLOCCHI.

- Poi l'AZIONE DI INPUT e OUTPUT vi è una azione di ASSEGNAMENTO in senso matematico (espresso da =)]

ESPRESSIONI: combinazioni di VARIABILI, COSTANTI ed OPERATORI.
Il VALORE di una espressione può essere assegnato ad una variabile

<valore> <variabile> = <espressione>.

NOTA: <variabile> e <espressione> devono essere COMPATIBILI ma non necessariamente uguali

Esistono varie categorie di OPERATORI:

- **OPERATORI ARITMETICI**: + - * / %
  - % → forma il RESTO della divisione intera

Esempio:
```
int x=5
int y=2
int q, r
q = x / y ;   // (q=2, TRONCAMENTO)
r = x % y     // (r=1)
```

Quesito: • che operazione svolge il seguente programma?

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

a=10 → b=25 → a=25 → b=25

• come fare a scambiare tra di loro i valori di due variabili? senza una variabile di appoggio.

a [7] b [-12]

```
a = 7
b = -12
c = a   -> c = 7
a = b   -> a = -12
b = c   -> b = 7
```

- **OPERATORI DI CONFRONTO**
  - UGUAGLIANZA: UGUALE `a==b` ; DIVERSO `a!=b`
  - ORDINE

MAGGIORE	a>b
MINORE	a<b
MAGG O UGUALE	a>=b
MIN O UGUALE	a<=b

- PRECEDENZA : NOT > AND > OR

Le espressioni LOGICHE (che derivano dalla combinazione di VARIABILI BOOLEANE con OPERATORI BOOLEANI) sono valutate da SINISTRA a DESTRA. La valutazione viene interrotta non appena il risultato viene univocamente determinato.

esempio: $(X>0)$ && $(X<10)$ [X COMPRESO TRA 0 e 10]

Per effettuare DIMOSTRAZIONI in algebra Booleana si considerano tutti i casi possibili e si confrontano le tabelle di verità.

esempio: DIMOSTRAZIONE $\boxed{A \oplus B = A \times B' + A' \times B}$

A	B	A⊕B	A'	B'	A×B'	A'×B	A×B'+A'×B
0	0	0	1	1	0	0	0
0	1	1	1	0	0	1	1
1	0	1	0	1	1	0	1
1	1	0	0	0	0	0	0

=

- PROPRIETÀ ~~ESPRESSIONI LOGICHE~~ OPERATORI [si dimostrano con tavole di verità]

AND	OR
$A \times 0 = 0$	$A + 0 = A$
$A \times 1 = A$	$A + 1 = 1$
$A \times A = A$	$A + A = A$
$A \times A' = 0$	$A + A' = 1$

- PROPRIETÀ ESPRESSIONI LOGICHE

COMMUTATIVA

$\begin{cases} A \times B = B \times A \\ A + B = B + A \end{cases}$

ASSOCIATIVA

$\begin{cases} A \times (B \times C) = (A \times B) \times C \\ A + (B + C) = (A + B) + C \end{cases}$

DISTRIBUTIVA

$\begin{cases} A \times (B + C) = A \times B + A \times C \\ A + (B \times C) = (A + B) \times (A + C) \end{cases}$

- TEOREMI BASE

1. TEOREMA DELL'INCLUSIONE : $\boxed{A + A \cdot B = A}$

DIMOSTRAZIONE

TABELLA

PROPRIETÀ

$A(1 + B) = A \cdot 1$

b

$A1 = A \cdot 1$

2. TEOREMA DELLA FUSIONE

DIREMA :

$\boxed{A + A' B = A + B}$

DIMOSTRAZIONE

TABELLA

PROPRIETÀ DISTRIBUTIVA

$(A + A') \times (A + B) = (A + B) 1$

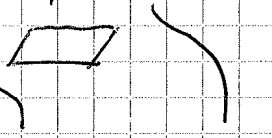
$1 \cdot (A + B) = (A + B) \cdot 1$

ISTRUZIONI ELEMENTARI

ASSEGNAMENTO

[<variabile> = <valore>]
da destra a sinistra

INPUT / OUTPUT



scanf() printf()

L'utilizzo di queste istruzioni richiede l'inserimento di una DIRETTIVA: #include $<stdio.h> all'inizio delle sorgenti.
(che indica di includere il file stdio.h)

- **ISTRUZIONE PRINTF()**:

printf (<formato>, <arg1>, <arg2>, ... <argn>)

es. printf ("%i", val)

direttive che indica il FORMATO e dunque viene sostituito dal valore della variabile che segue.

NOTE: ① Vi possono essere più direttive in una stessa printf: in questo caso occorre riportare il giusto numero tra direttive e variabili e l'ordine. Se si indicano più direttive tra di esse si mette uno spazio altrimenti verrebbero stampati VALORI sbagliati.

② È importante associare correttamente direttive e variabile:
int -> %i, %d
float -> %f
double -> %lf.

CARATTERI SPECIALI:
% -> segue il formato
\n a capo
\t tabulazione

- **ISTRUZIONE SCANF()**:

scanf (<formato>, <arg1>, <arg2>, ... <argn>)

es. scanf ("%i", &x)

NOTE: ①
② con la differenza che double -> %lf

Nella scanf nel 90% delle volte bisogna inserire le DIRETTIVE; se si vogliono far visualizzare dei messaggi all'utente prima della scanf si utilizza una printf.

FORMATTAZIONE AVANZATA

esempi: • STAMPARE UN VALORE INTERO SU 3 CIFRE: printf ("%03d", int_val)
indica di riempire le cifre mancanti con 0
indica il numero delle cifre

• STAMPARE UN VALORE ~~DECIMALE~~ REALE CON DUE CIFRE DECIMALI: printf ("%.2f", float_val)
indica il formato decimale
indica il numero delle cifre

CONTROLLI DI FLUSSO

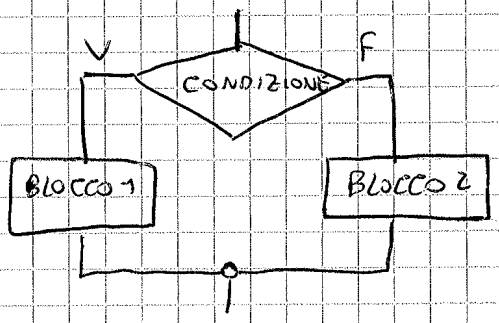
= istruzioni di linguaggio che modificano il PROGRAMMA che strada seguire

- ISTRUZIONE IF

SINTASSI

```
if (<condizione>)
    <blocco 1>
[else
    <blocco 2>]
```

DIAGRAMMA DI FLUSSO

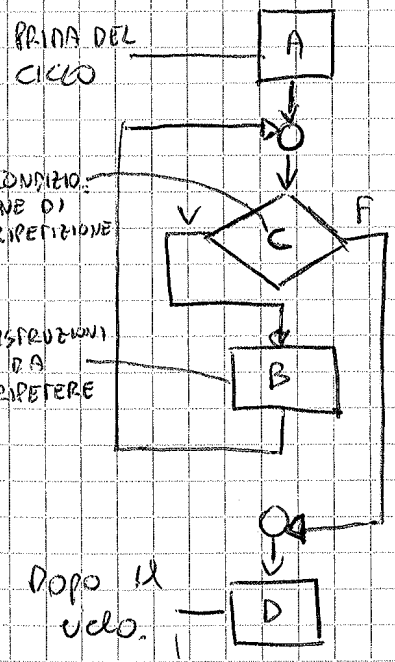


<CONDIZIONE>: è espressa da OPERATORI RELAZIONALI o OPERATORI LOGICI, o eventualmente combinati: tali condizioni sono delle espressioni booleane in quanto restituiscono un valore logico 0 (FALSO) o 1 (VERO).

NOTA: il blocco 1 è racchiuso tra { }. Tale BLOCCO di istruzioni viene eseguito solo se la condizione è vera; se la condizione è FALSA viene eseguito il BLOCCO 2 racchiuso a sua volta tra due { } dopo else.

Se l'intera struttura si ripete, esso rappresenta un CICLO.

FLUSSO DI ESECUZIONE CICLICO



CICLO

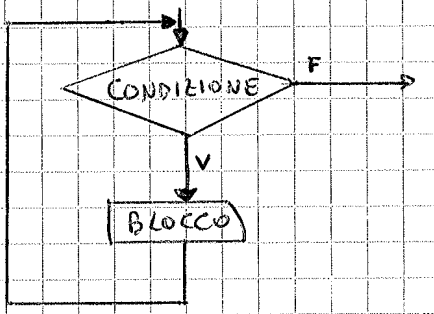
DEFINITO [si ripete n volte]

INDEFINITO [si chiude all'utente quando termina]

(I) ISTRUZIONE WHILE

- Significato: Ripeti <blocco> finché <condizione> è vera.
- Sintassi: while (<condizione espressa su una variabile da inizializzare e (*)modificare prima di ricominciare il ciclo>) {
  <blocco da ripetere>
  }

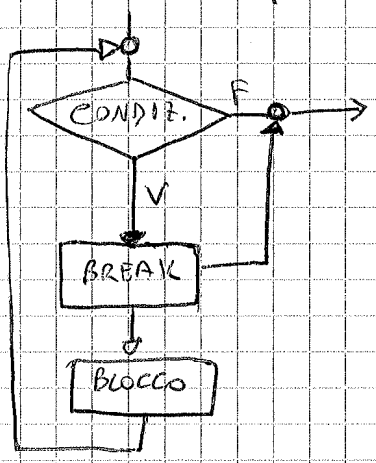
- Si realizza un CICLO con CONDIZIONE in testa.

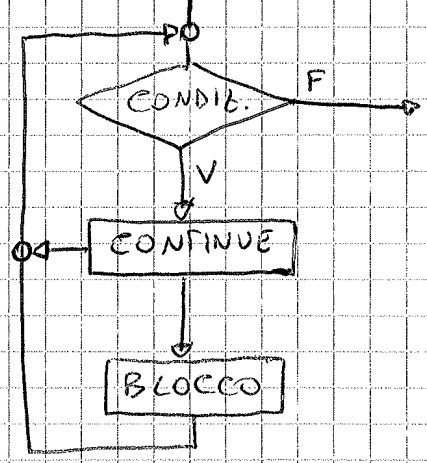


(*)NOTA: Per modificare la variabile si utilizza ad es. un operatore di INCREMENTO alla fine del blocco da

L'ISTRUZIONE BREAK, interrompe il ciclo. L'esecuzione continua dalla prima istruzione dopo la fine del ciclo.

L'istruzione CONTINUE, blocca la ripetizione del ciclo corrente. L'esecuzione continua con la prossima iterazione del ciclo





CICLI ANNIDATI:

Nelle ISTRUZIONI del blocco "VERO" o del blocco "ELSE" è possibile inserire altri blocchi di scelta: in questo caso la seconda scelta risulta annidata all'interno della prima.

ESEMPI:

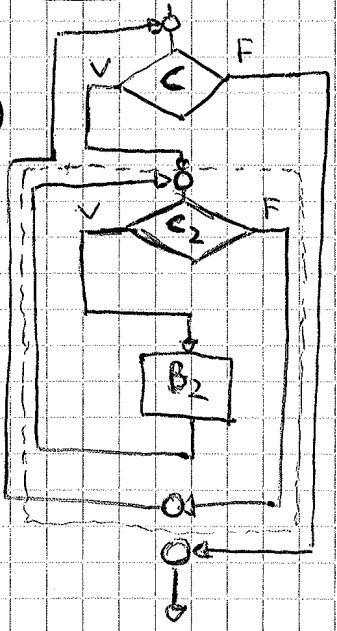
- CICLI FOR ANNIDATI

```
for (...)
{
    for (...)
    {
    ...
    }
}
```

Il CICLO ESTERNO non si ripete finché non è terminato il CICLO INTERNO

- CICLI WHILE ANNIDATI

```
while (C)
{
    while (C2)
    {
        B2;
    }
}
```



(*) ESEMPIO DI UTILIZZO DELLO SWITCH

MENU DEL RISTORANTE

```
int scelta;
int fine = 0;

do{
    printf("1: spaghetti\n");
    printf("2: salmone\n");
    printf("3: carpaccio\n");
    printf("4: arrangiate\n");
    printf("5: caffè\n");
    printf("0: fine\n");

    printf("Cosa desidera?");
    scanf("%d", &scelta);
```

MENU

```
switch (scelta)
{
    case 1:
        printf("Ecco: spaghetti");
        break;
    ...
    case 0:
        fine = 1;
}
} while (fine == 0);
```

• VETTORI E CICLI

I cicli sono particolarmente utili per "scandire" un VETTORE.

STAMPA DI UN VETTORE (attraverso un ciclo for):

```
for (i=0; i<N; i++)
{ printf ("Elemento vettore: %d", vettore[i]);
}
```

LETTURA DI UN VETTORE (attraverso un ciclo for):

```
for (i=0; i<N; i++)
{ scanf ("%d", &vettore[i]);
}
```

[NOTA= Spesso per facilitare la scrittura di programmi in linguaggio C si definisce una costante N attraverso #define N _____, che rappresenta la dimensione del vettore.]

• OPERAZIONI TRA VETTORI

* Per COPIARE un vettore non si può fare ~~[w=u]~~ : è necessario copiare ciascun elemento di un vettore all'altro [utilizzando ad esempio un ciclo for].

```
for (i=0; i<N; i++)
{ w[i] = v[i];
}
```

* Per SOMMARE due vettori non si può fare ~~[w=v1+v2]~~ : è necessario sommare ciascun elemento dei vettori [utilizzando ad esempio un ciclo for].

```
for (i=0; i<N; i++)
{ w[i] = v1[i] + v2[i];
}
```

③ VERIFICARE CHE TUTTI I DATI INSERITI DALL'UTENTE SIANO POSITIVI = basta verificare se c'è almeno un dato non positivo!

```
#define VERO 1;
int main()
{
int tutti_positivi = VERO;
int v[N], i;
--
--

for (i=0; i<N; i++)
   { if ( !(v[i]>0))
        { tutti_positivi = FALSO;
          break;
        }
   }
if ( tutti_positivi == VERO)
  { printf("Sono tutti positivi");
  } else {
     printf("Non sono tutti positivi");
      }
return 0;
}
```

NOTA: al posto del break si potrebbe cambiare la condizione del for: i<N → i<N && tutti_positivi==VERO

④ RICERCA DI DUPLICATI IN UN VETTORE

Si utilizzano due cicli FOR ANNIDATI:

```
for (i=0; i<N; i++)
  { duplicato = 0;
    for (j=0; j<N; j++)
      { if ( (v[i]==v[j]) && (i!=j))
          { duplicato = 1;
          }
      }
    if (duplicato == 1)
      printf("v[%d] è duplicato", i);
  }
```

- FORMALIZZAZIONE RICERCA DI ELEMENTI CHE SODDISFANO UNA DATA PROPRIETÀ:

- $\forall x : P(x)$ → • INIZIALIZZO FLAG = 1;
  • CICLO su tutte le x: se P(x) è FALSA, allora FLAG=0;
  • IF (FLAG=1) → PROPRIETÀ DIMOSTRATA
    ELSE → PROPRIETÀ FALSA

- $\exists x : P(x)$ → • INIZIALIZZO FLAG=0;
  • Ciclo su tutte le x: se P(x) è VERA, allora FLAG=1;
  • IF (FLAG=1) → PROPRIETÀ DIMOSTRATA;
    ELSE → PROPRIETÀ FALSA.

- $\forall x : \neg P(x)$ → • INIZIALIZZO FLAG=1
  • CICLO su tutte le x: se P(x) è VERA, allora FLAG=0;
  • if (flag=1) → proprietà dimostrata
    else → proprietà falsa

- $\exists x : \neg P(x)$ → • INIZIALIZZO FLAG=0; ciclo su x: se P(x) è FALSA → FLAG=1 → proprietà dimostrata;

FUNZIONI

Un PROGRAMMA reale può consistere di migliaia di ISTRUZIONI.
Per evitare di ripetere del CODICE e per rendere più comprensibile il codice stesso è consigliato procedere secondo un APPROCCIO TOP-DOWN ovvero, decomporre il problema in SOTTOPROBLEMI più semplici in modo che si parli del generale al particolare.
Si fa uso pertanto di SOTTOPROGRAMMI che nel linguaggio C sono detti FUNZIONI (se ritornano un risultato) o PROCEDURE (se non ritornano un risultato).

Una FUNZIONE è un insieme di operazioni e il suo "BIGLIETTO da visita" [il nome] deve essere invocato per poter svolgere le istruzioni contenute in esso. [L'INVOCAZIONE definisce le informazioni su come gli input devono essere passati.]

La SINTASSI del PROTOTIPO di una funzione è la seguente:

<Tipo risultato> <nome funzione> (<parametri formali>) → [Il PROTOTIPO definisce come le info devono essere lette.]
(parametri formali = inizializzati)

Al PROTOTIPO segue fra { } il BLOCCO DI ISTRUZIONI da eseguire. [Al termine delle ISTRUZIONI vi deve essere return <valore>;]

NOTA: ① Se la funzione non ha risultato, <Tipo risultato> deve essere void.

② Tutte le FUNZIONI devono essere definite allo stesso livello del main() e non si possono definire l'una dentro l'altra [devono essere INDIPENDENTI!]

③ Il main() è una funzione che restituisce come tipo del valore di ritorno int.

Ogni FUNZIONE gestisce le proprie variabili [la VISIBILITÀ delle VARIABILI è limitata].
(È consigliabile evitare di utilizzare delle VARIABILI GLOBALI (visibili da tutti) in quanto ciò produce POLLUTION.)

PARAMETRI FORMALI / PARAMETRI ATTUALI

PARAMETRI FORMALI: Sono le VARIABILI specificate nella definizione di una funzione

PARAMETRI ATTUALI: Sono le variabili specificate durante l'UTILIZZO della FUNZIONE
↓
Sono i parametri che vengono "passati" alla FUNZIONE per far sì che essa lavori su di essi.

ESEMPIO:
- funzione FUNC
  - def: float FUNC (int x, double y)
  - utilizzo: float z = FUNC (a*2, 1.34)

PARAMETRI FORMALI: x, y

PARAMETRI ATTUALI: a*2, 1.34

Come "passare" un VETTORE ad una funzione?

Per "passare" un VETTORE ad una funzione si utilizza il PASSAGGIO BY REFERENCE perché sarebbe impossibile copiare un numero elevatissimo di elementi utilizzando il PASSAGGIO BY VALUE.
In questo modo si lavora direttamente sul vettore originale.

NOTA: Poiché il VETTORE è "passato" per INDIRIZZO, la funzione che riceve il VETTORE come argomento non ne conosce la lunghezza! → occorre "passare" alla funzione anche la dimensione del vettore

PARAMETRI FORMALI → (nome del vettore con [ ] (senza dimensione), dimensione)

PARAMETRI ATTUALI → (nome del vettore senza [ ] (indica l'INDIRIZZO), lunghezza vettore)

ESEMPIO

```
int somma(int v[], int len){
    int i;
    int s;
    for(i=0; i<N; i++)
    {   s+= v[i];
    }
}
int main()
{
    int vett[N]={1,2,3};
    int s;
    int l=N;
    s= somma(vett, l);
    return 0;
}
```

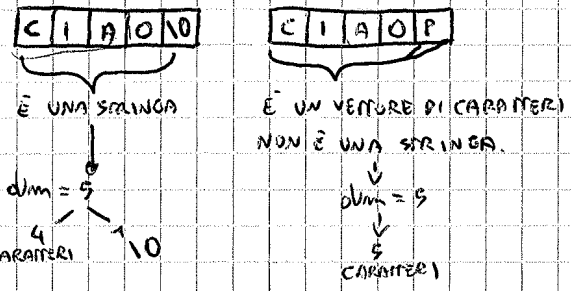
(all'interno della funzione, uso il vettore normalmente)

l → LUNGHEZZA VETTORE.

STRINGHE:

Le STRINGHE sono dei VETTORI DI CARATTERI terminato da \0.

Dato ad es. char saluto[5];



NOTA: ① La STRINGA può ~~essere~~ occupare tutta la dimensione del VETTORE o NO; essa termina con \0.

es 

② La STRINGA VUOTA non è un VETTORE VUOTO! Contiene \0.

③ Gli APICI '' indicano il CARATTERE.
I doppi APICI "" indicano la STRINGA (con \0 finale)

- SSCANF: (char* <stringa>, char <formato> <espressioni>)
  Legge da una STRINGA
- SPRINTF: (char* <stringa>, char <formato>, <espressioni>)
  Scrive su una STRINGA

FORMATTAZIONE DI STRINGHE

La DIRETTIVA che indica il FORMATO della STRINGA è %s.

scanf ("%s", &stringa); → Legge i caratteri fino al primo carattere di SPAZIATURA (senza &)

printf ("%s", stringa); → Scrive tutto ciò che c'è prima del \0.

(*) Le OPERAZIONI DI I/O possono anche essere effettuate da/su STRINGHE.

Dato la loro natura di tipo "aggregato", le STRINGHE non possono essere usate come variabili qualunque. Per eseguire determinate operazioni si utilizzano delle ~~f~~ apposite

FUNZIONI PER LA MANIPOLAZIONE DELLE STRINGHE:

Sono utilizzabili includendo al programma #include <string.h>.

es.
- strcpy (char* S_1, char* S_2): Il CONTENUTO di S_2 è copiato in S_1.
  NOTA: In S_1 ci deve essere abbastanza spazio per contenere S_2.
- int strcmp (char* S_1, char* S_2): Confronta i CARATTERI fino al \0.
  Restituisce un valore: >0 [contenuto S_2 >] / =0 [uguale contenuto] / <0 [contenuto S_2 <]
- int strlen (char* S_1): Indica la lunghezza della stringa contenuta nel vettore prima del \0.

VETTORI MULTIDIMENSIONALI - MATRICI

- **SINTASSI:**

  <tipo> <nome vettore> [<dim 1>][<dim 2>] ... [<dim n>];

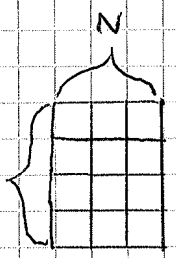
- **ACCESSO AD UN ELEMENTO:**

  <nome vettore> [<pos 1>][<pos 2>] ... [<pos n>]

  ↳ occorre specificare le posizioni di tutte le dimensioni.

- I VETTORI BIDIMENSIONALI si chiamano **MATRICI**:
  - LA 1ª DIMENSIONE specifica le RIGHE;
  - LA 2ª DIMENSIONE specifica le COLONNE;

  → `int m [n][N]` (4 × 3)

  4 × 3 = n° tot di CELLE (ogni cella contiene un elemento)

- **INIZIALIZZAZIONE** (come per vettori MATRICI):

  L'INIZIALIZZAZIONE viene fatta per RIGHE quindi la MATRICE viene riempita da sinistra e dunque a dall'alto verso il basso →↓

  Esempio: `int v[3][2] = {{8,1},{1,9},{0,3}};`

8	1
1	9
0	3

- **ACCESSO AGLI ELEMENTI DI UNA MATRICE**

  Per scorrere tutti gli elementi di una MATRICE servono **2 CICLI FOR** e quindi 2 CONTATORI, 1 per le righe e 1 per le colonne

  Es: Data `int m[3][5];` (MATRICE 3×5)

	0	1	2	3	4
0	[ ]	[ ]	[ ]	[ ]	[ ]
1	[ ]	[ ]	[ ]	[ ]	[ ]
2	[ ]	[ ]	[ ]	[ ]	[ ]

  - SCORRERE LA RIGA 1: (RIGA 1)

```
for (i=0; i<5; i++){
    printf("%d", m[1][i]);
}
```

  - SCORRERE LA COLONNA 2: (COLONNA 2)

```
for (j=0; j<3; j++){
    printf("%d", m[j][2]);
}
```

  - SCORRERE LA MATRICE per RIGHE o PER COLONNE:

PER RIGHE

```
for (i=0; i<3; i++){
    for (j=0; j<5; j++){
        printf("%d", m[i][j]);
    }
    printf("\n");
}
```

PER COLONNE

```
for (j=0; j<5; j++){
    for (i=0; i<3; i++){
        printf("%d", m[i][j]);
    }
    printf("\n");
}
```

```
        /* inizio riga */   somma = 0;

        /* per ogni colonna j */
        for(j=0;j<n;j++) {
            printf("%4d", (mat[i][j]) );
            somma += mat[i][j];
        }


        /* fine riga */ printf("| %d", somma);
        printf("\n");
    }

    /* per ogni colonna j */
    for(j=0;j<n;j++) {

        /* inizio colonna */   somma = 0;

        /* per ogni riga i */
        for(i=0;i<n;i++) {
            /* aspetto int * , ma ho trovato int */
            /* printf("%4d", (mat[i][j]) ); */
            somma += mat[i][j];
        }


        /* fine colonna */ printf("%4d", somma);
    }
    printf("\n");

    return 0;
}
```

```
Inserisci dimensione matrice: 3
Inserisci 3x3 valori interi: 1 2 3 4 5 6 7 8 9
La matrice da te inserita e`:
1   2   3
4   5   6
7   8   9
La matrice completata con le relative somme di righe e colonne e`:
1   2   3| 6
4   5   6| 15
7   8   9| 24
12  15  18
Program ended with exit code: 0
```

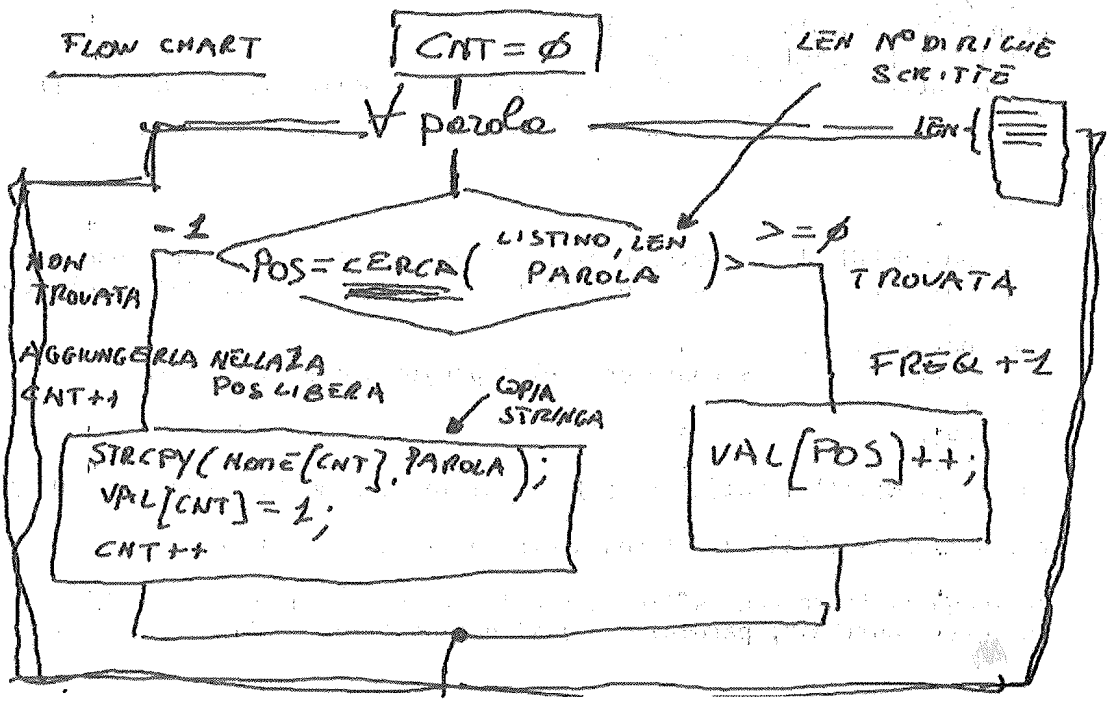
ESERCIZIO RICERCA DI UN CARATTERE IN UNA STRINGA

- Posso risolvere l'esercizio normalmente seguendo un FLOW-CHART simile a quello visto in «RICERCA DI UN ELEMENTO ALL'INTERNO DI UN VETTORE», scrivendo un CODICE come quello nell'ESEMPIO.

- Posso risolvere l'esercizio invocando una FUNZIONE RICERCA, opportunamente definita che lavora:
  - in INPUT su un VETTORE (i metodi penso l'estensione e la lunghezza) e su uno SCALARE da cercare;
  - in OUTPUT su un FLAG (TROVATO o non trovato → restituisce 0 o 1)

VEDERE GLI ESEMPI SU STAMPA.

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

/* AS 21.05.2014 */

/* PROGRAMMA: il programma deve leggere il testo "parola per parola" e
deve contare quante volte (frequenza) si ripete la stessa parola
fornendo una statistica finale.

ad esempio: ciao 123 ciao mondo

 SVOLGIMENTO PROGRAMMA: a) utilizzo un LISTINO (vettore di stringhe)
                           inizialmente vuoto, in cui inserisco le parole nuove;
                        b) utilizzo un VETTORE (val[] per le FREQUENZE
                           allineato al LISTINO;
                        c) utilizzo una VARIABILE CNT(inizializzato a 0) per
                           contare il numero di parole trovate;

             1) ciao --> a) la inserisco nel listino in posizione [0];
                         b) val[0]=1;
                         c) cnt++; (cnt==1)

             2) 123  --> a) la inserisco nel listino in posizione [1];
                         b) val[1]=1;
                         c) cnt++; (cnt==2)

             3) ciao --> a) 'ciao' gia' presente in posizione [0], non la
                            inserisco nel listino;
                         b) val[0]=2;
                         c) NON aggiorno cnt;

             4) mondo --> a) la inserisco nel listino in posizione [2];
                          b) val[2]=1;
                          c) cnt++; (cnt==3);

 FLOW CHART:
```



```
{
        printf("\n[%d] parola: %s\n", cnt, parola);

        printf("Cerco '%s' nel listino:\n", parola);
        stampa(nome,val,cnt);

        /* ricerca ritorna un intero che viene assegnato a pos, poi pos viene
           valutato >= 0 */
        if( (pos = ricerca(nome, cnt, parola)) >= 0)
            {
            printf("  Parola doppione in posizione %d.\n\n", pos);
            val[pos] ++; /* aggiorno la frequenza della parola doppione */
            } else {
                  printf("  Parola nuova: la memorizzo.\n\n");
                  strcpy(nome[cnt], parola);
                  val[cnt] = 1;
                  cnt++;
                   }
}

    /* se arrivo al momento in cui cnt(che conta le parole nuove e quindi le
      righe del listino) e`uguale ad N (dimensione massima righe) devo
      interrompere il programma */

    if(cnt == N) {
        printf("Vettore di stringhe pieno.\n");
    }

    printf("La statistica e`:\n\n");
    stampa(nome,val,cnt);

    return 0;
}
```

ESECUZIONE:

```
ciao 123 ciao mondo

[0] parola: ciao
Cerco 'ciao' nel listino:
Pos Parola    Freq
  Parola nuova: la memorizzo.


[1] parola: 123
Cerco '123' nel listino:
Pos Parola    Freq
[0]  ciao     [1]
  Parola nuova: la memorizzo.


[2] parola: ciao
Cerco 'ciao' nel listino:
Pos Parola    Freq
[0]  ciao     [1]
[1]  123      [1]
  Parola doppione in posizione 0.


[2] parola: mondo
Cerco 'mondo' nel listino:
Pos Parola    Freq
[0]  ciao     [2]
[1]  123      [1]
  Parola nuova: la memorizzo.

La statistica e`:

Pos Parola    Freq
[0]  ciao     [2]
[1]  123      [1]
[2]  mondo    [1]
```


© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 35 di 126

```
/* AS 8.5.2014
 - Ricerca all'interno di una stringa attraverso una funzione
 */

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

#define MAXLINE 100
#define VERO    1
#define FALSO   0

int ricerca( char fz_line[], int fz_len, char fz_c)
{
    int fz_i;
    int fz_pos, fz_trovato;

    /* ricerca */
    fz_pos = -1;   /* inizializzo la posizione ad un valore impossibile
*/
    fz_trovato = FALSO;

    for(fz_i=0;fz_i<fz_len;fz_i++)
        {
        if(fz_line[fz_i] == fz_c)
            {
             fz_trovato = VERO;
             fz_pos = fz_i;    /* mi segno anche dove l'ho trovato */

             break; /* interrompiamo il ciclo */
            }
         }

   return fz_pos;
}

int main()
{
    char line[MAXLINE+1];   /* ricordiamo il /0 */
    char c;
    int len;
    int pos;


    printf("Immetti linea: ");
    gets(line);         /* leggiamo la linea intera */

    printf("Immetti char da cercare: ");
    c = getchar();
    getchar();  /* butta via l'invio */

    printf("Cerco %c in '%s'.\n", c, line);
    len = strlen(line); /* calcola la lunghezza della str */
```


© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 37 di 126

```
for(fz_i=0;fz_i<fz_len;fz_i++) {
        if(fz_line[fz_i] == fz_c) {
            if(fz_trovato == FALSO)
                {fz_pos = fz_i; /* mi segno anche dove l'ho trovato*/
                }
            fz_trovato = VERO;
            (*fz_occorrenze)++;
        }
    }

    /* return fz_trovato; */
    return fz_pos;

}

int main()
{
    char line[MAXLINE+1];   /* ricordiamo il /0 */
    char c;
    int len; int pos;
    int occorrenze;

    printf("Immetti linea: ");
    gets(line);         /* leggiamo la linea intera */

    printf("Immetti char da cercare: ");
    c = getchar();
    getchar();  /* butta via l'invio */

    printf("Cerco %c in '%s'.\n", c, line);
    len = strlen(line); /* calcola la lunghezza della str */

    /* line -> fz_line , len -> fz_len, c -> fz_c */
    pos = ricerca(line, len, c, &occorrenze);

    if(pos != -1) {   /* if(trovato==VERO) */
        printf("Carattere %c trovato, "
               "pos vale %d, "
               "occorrenze vale %d.\n" ,c ,pos, occorrenze);
    } else {
        printf("Carattere %c non trovato, pos vale %d.\n",c, pos);
    }

    return 0;
}
```

Immetti linea: ciao mondo
Immetti char da cercare: o
Cerco o in 'ciao mondo'.
Carattere o trovato, pos vale 3, occorrenze vale 3.
Program ended with exit code: 0

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 39 di 126

CONVERSIONE DEGLI ARGOMENTI (da stringa a numeri)

Se inserisco in una stringa 1.0 esso viene interpretato non come il numero 1.0 ma come la SERIE di CARATTERI terminata col \0: ["1" "." "0" \0].

PER INTERPRETARE I NUMERI:

① Utilizzo la funzione SSCANF:

```
gets(line);
sscanf(line; "%f"; &v);
```

② Utilizzo le FUNZIONI presenti in <stdlib.h>

int atoi	→	alpha to int
long atol	→	alpha to long
double atof	→	alpha to float

ARGOMENTI SULLA LINEA DI COMANDO

- FUNZIONE MAIN

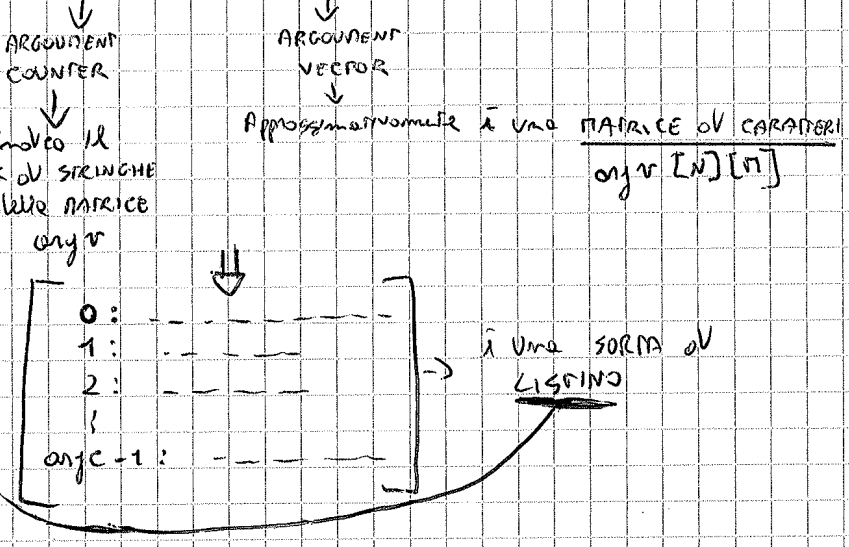
Il MAIN è una FUNZIONE, in particolare la 1ª FUNZIONE ESEGUITA che viene invocata dal SISTEMA OPERATIVO.

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

- argc: ARGOMENTI COUNTER → Indica il n° di STRINGHE della MATRICE argv
- argv: ARGOMENTI VECTOR → Approssimativamente è una MATRICE di CARATTERI `argv[N][n]`

Può essere STAMPATO nel seguente modo (creando un VETTORE di STRINGHE)

```
int n;
for(n=0; n<argc; n++)
{
  printf("%d: %s", n, argv[n]);
}
```



NOTA = Nel CONTRATTO del MAIN è specificato che il PROGRAMMA riceve argc ARGOMENTI dall'ESTERNO (chiamati PARAMETRI) che rappresentano delle ISTRUZIONI fornite dal SISTEMA OPERATIVO per il corretto FUNZIONAMENTO del programma. Questi argc argomenti vengono sistemati nel VETTORE di STRINGHE argv[].

- DEVO DIVIDERE IL PROGRAMMA IN DUE, ① per -l, ② per -m
Occorre quindi distinguere se il PARAMETRO passato in argv[argc-1] è -l o -m.

```
if ((strcmp(argv[1], "-l") == 0) ①
  { to_minuscolo = 1; ————→ RICORDA: inizializzare to_minuscolo = 0;
  } else if ((strcmp(argv[1], "-m") == 0) ②
        {
          to_minuscolo = 0;
        } else {
                 printf("UTILIZZO: -l | -m errato\n");
                 exit(EXIT_FAILURE);
               }
```

- ESEGUIRE LA CONVERSIONE (ISTRUZIONE DEL PROGRAMMA)

```
while (c != EOF) ————→ c inizializzato a un valore != EOF.
  {
    c = getchar();
    if (to_minuscolo == 1)
      {
        c = tolower(c);
      } else {
               c = toupper(c);
             }
    putchar(c);
  }
return 0;
}
```

NOTA = ① L'esecuzione del PROGRAMMA può essere lanciata dal TERMINALE come segue:

C:\ posizione del file ↓ (-l) o (-m)
SPAZIO

VANTAGGIO

① Non è necessario ricompilare il CODICE come si dovrebbe fare su CODEBLOCK ogni volta che si vogliono gli ARGOMENTI inseriti da LINEA DI COMANDO.

② Si possono creare delle SCORCIATOIE org-l, org-m cambiando il Target nelle PROPRIETÀ, aggiungendo il 2° argomento.

NOTE:

(1) Se non si vuole utilizzare la STRUTTURA invocabile con il TIPO creato struct _____ si può CREARE un "SOPRANNOME" nel seguente modo:

typedef struct ___ {

} <nome nuovo>;

In questo modo invocando <nome nuovo> si fa accenno alla STRUTTURA struct _____.

(2) Le STRUCT vengono passate alle funzioni "PER COPIA"

(3) • NON È POSSIBILE CONFRONTARE due variabili dello stesso tipo di struct con $S_1 == S_2$ o $S_1 != S_2$ ⇒ Il CONFRONTO deve avvenire CONFRONTANDO i CAMPI uno ad uno.

Es. $[(S_1.re == S_2.re) \;\&\&\; (S_2.nome == S_2.nome)]$

• È POSSIBILE invece COPIARE due variabili dello stesso ~~tipo~~ tipo di ~~struct~~ struct.

$[S_1 = S_2]$ (ASSEGNAZIONE OK).

• L'INIZIALIZZAZIONE come per i vettori, avviene inserendo una lista di valori tra { }

Es compl $S_1 = \{0.12\text{-}13, 2.655\}$;
↓ re ↓ Im

(a) DICHIARAZIONE DEL FILE

L'utilizzo di un FILE avviene tramite una variabile di tipo FILE * in cui viene copiato il RIFERIMENTO (e non il FILE stesso) del FILE.

In genere si utilizza come nome della VARIABILE *fp:

```
FILE *fp;
```

(b) APERTURA DEL FILE

Come richiedere il RIFERIMENTO del FILE al SISTEMA OPERATIVO?

Si utilizza la funzione *fopen così definita:

```
FILE *fopen(char *<nomefile>, char *<modo>);
```

SPECIFICHE

(1) La FUNZIONE *fopen restituisce il RIFERIMENTO del FILE che deve essere copiato nella variabile di tipo FILE*. Quindi `*fp = *fopen("<nome file>", "<modo>");`

(2) In "<nome file>" posso inserire:

- NOME COMPLETO DEL FILE ricordandomi che il FILE si trova:
  - nella CARTELLA dove si trova il main.c (se lancio il PROGRAMMA da CODEBLOCKS)
  - nella CARTELLA dove c'è l'.exe ESEGUIBILE (se lancio il PROGRAMMA da COMMAND LINE).
- RIFERIMENTO ASSOLUTO del FILE (nome disco + percorso file).

(3) In "<modo>" va specificato il MODO con cui si vuol trattare il file.

- "r": SOLA LETTURA;
- "w": SOLA SCRITTURA (-> IN TORO -> cancella ciò che eventualmente c'è nel file);
- "a": MODIFICA il FILE aggiungendo IN CODA.

(1*) La FUNZIONE *fopen in caso di ERRORE restituisce il VALORE NULL

Perciò prima di utilizzare il FILE si fa:

```
if (*fp == NULL)
  { printf("Errore nell'apertura.\n");
    exit(EXIT_FAILURE);
  }
```

• fgetc : LETTURA A CARATTERE

UTILIZZO PER LEGGERE TUTTI I CARATTERI DI UN FILE:

```
while ( (c = fgetc(fp)) != EOF )
  {
    printf("Ho letto %c\n", c);
  }
```

fgetc restituisce un VALORE < EOF (fine file o ERRORE) che deve essere memorizzato in una char c. (CARATTERE LETTO)

NOTA: getchar() ⇔ fgetc(stdin)

• fputc : SCRITTURA A CARATTERI

```
int fputc(int c, FILE *fp);
```

NOTA: putchar(c) ⇔ fputc(c, stdout)

• fscanf : LETTURA FORMATTATA

• È come la scanf() con un parametro aggiuntivo che rappresenta un FILE.

• Restituisce il numero di CAMPI convertiti, oppure EOF in caso di fine FILE.

```
int fscanf(FILE *fp, "<FORMATO>", &<campo_da_convertire>)
```

PER I FORMATI: %c, %d

UTILIZZO PER LEGGERE TUTTI I CARATTERI DI UN FILE:

```
while ( fscanf(fp, "%s", parola) != EOF )
  {
    printf("Ho letto: %s", parola);
  }
```

NOTA: scanf("%s", parola); ⇔ fscanf(stdin, "%s", parola);

• fprintf : SCRITTURA FORMATTATA

• È come la printf() con un parametro aggiuntivo che rappresenta un FILE.

• Restituisce il numero di BYTE SCRITTI, oppure EOF in caso di ERRORE.

```
fprintf(FILE *fp, "<FORMATO>", <parola>);
```

NOTA: printf("%s", parola) ⇔ fprintf(stdout, "%s", parola);

```
system("chdir");
}

int main(int argc, char *argv[])
{
    /* matrice di caratteri -> vettore di stringhe */
    char nome[N][M+1];
    int  val[N];        /* contatore delle frequenze
                        per ciascuna parola trovata */
    char parola[M+1];
    int  cnt;
    int  i;
    int  pos;

    FILE *fp;

    /* Leggo il nome del file da argv (dove?) */
    /* argv[0]: sempre il nome del programma
       argv[1]: e' il primo argomento, la stringa con il nome del file */

    if(argc < 1) {
        printf("Utilizzo errato.\n");
        system("PAUSE");
        exit(EXIT_FAILURE);
    }

    /* for(i=1;i<argc;i++) {
       apro, leggo e calcolo argv[i] */

    printf("argv[1]: %s", argv[1]);

    /* Apertura del file */
    if( (fp=fopen(argv[1],"r")) == NULL ) {
        printf("File non trovato.\n");
        system("PAUSE");
        exit(EXIT_FAILURE);
    }

    /* inizializziamo le frequenze a 0 */
    for(i=0;i<N;i++) { val[i] = 0; }

    cnt = 0;
    /* leggo il testo con %s */
    while( fscanf(fp,"%s", parola) == 1 && cnt<N ) {
        fprintf(stdout, "[%d] parola: %s\n", cnt, parola);

        /* stampa(nome,val,cnt); */

        /* ricerca ritorna un intero che viene assegnato a pos, ed e' poi
           pos che viene valutato >= 0 */
        if( (pos = ricerca(nome, cnt, parola)) >= 0) {
```

ALTRO MODO PER LEGGERE UN FILE: fgets + sscanf

- LETTURA DI UN FILE FORMATTATO in cui ogni riga contiene un DATO NUMERO DI CAMPI DI TIPO NOTO (es. un intero, ed una stringa)

Ad es.
1 ANTONIO
2 STEFANO
3 GIOVANNI

→
```
while ((fgets(s, 80, fp)) != NULL)
{
    sscanf(s, "%d %s", &intero, &stringa);
}
```

Se invece avessi es. 4. JUAN CARLOS → NON POSSO LEGGERLO perché il formato [%d %s %s]

[OSSERVAZIONE FINALE: Questi vari metodi di lettura/scrittura non sono sovrapponibili perché l'ACCESSO AL FILE È SEQUENZIALE.]

SALVARE UNA STRUTTURA SU UN FILE (⇔ SCRIVERE SU UN FILE)

(1) APERTURA DEL FILE IN SCRITTURA

```
fp = fopen("<file su cui scrivere>", "w");
```

(2) SCRITTURA FINCHÉ CI SONO DATI DA SCRIVERE

```
for (i=0; i<cnt; i++){        cnt memorizza il n° di RIGHE del FILE.
    fprintf(fp, "[%d] %s\n", vet[i], nome[i]);
```

(3) CHIUSURA

```
fclose(fp);
```

ALTRE FUNZIONI UTILI PER I FILE

- rewind(FILE* <file>) → RIPOSIZIONA IL PUNTATORE ALL'INIZIO DEL FILE.
  equivale alla sequenza delle istruzioni → ① fclose(fp); → ② fopen(fp);

- int feof(FILE* <file>) → SERVE A CAPIRE SE SI È ARRIVATI ALLA FINE DEL FILE.
  quindi restituisce → 0 (se sei a fine FILE).
  → 1 (se non sei a fine FILE).

NOTA: Questa FUNZIONE è utile per i FILE di binari.

PUNTATORI E VETTORI

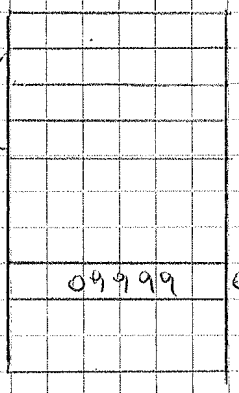
- in un MAIN

int vett[3];

(le [] sostituiscono *)

(in quanto int vett[3] == int* vett)

incrementa (vett)



Il nome di un VETTORE rappresenta l'INDIRIZZO del 1° elemento

*(vett) = vett[0]
*(vett+1) = vett[1]
*(vett+2) = vett[2]

↓ INDIRIZZO

- in una FUNZIONE (in incrementa)

void incrementa (int *z - vett[]);

(p *z - vett[] == **z - vett)

OSS: vett e *z - vett[] sono dello stesso TIPO (PUNTATORI): pertanto non scriviamo * e & (ma il PASSAGGIO AVVIENE PER RIFERIMENTO)

↓ int* ↓ int*

ANALOGIE PUNTATORI - VETTORI

Il nome del vettore rappresenta il PUNTATORE al 1° elemento del vettore; i suoi incrementi rappresentano PUNTATORI agli elementi successivi.

ES a[i];

* a(ptr) = a (ptr = PUNTATORE)

* (a ptr + 1) = a + 1

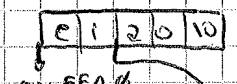
DIFFERENZE PUNTATORI - VETTORI

Un PUNTATORE è una variabile (che contiene INDIRIZZI) che può assumere valori diversi, il nome del vettore è fisso e non può mai essere modificato.

PUNTATORI E STRINGHE

Avendo dichiarato i PUNTATORI (INDIRIZZI) si possono utilizzare a pieno le FUNZIONI della LIBRERIA delle STRINGHE <string.h>

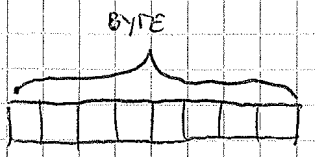
ES: char* strchr (char* s, char c);



strchr (s, 'a') ritorna 0XFFA2 (il PUNTATORE al 3° elemento del vettore)

la FUNZIONE STRCHR ritorna il PUNTATORE all'elemento cercato

SISTEMA BINARIO



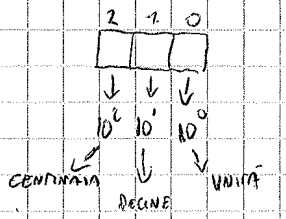
Le variabili sono CELLE DI MEMORIA di dimensione **BYTE** = 8 BIT (0 1)

- Una WORD ha dimensione 2 BYTE = 16 BIT; una DOUBLE WORD ha dimensione 1 WORD = 32 BIT.

I BIT vanno interpretati diversamente a seconda del tipo di DATO memorizzato:
- NUMERI NATURALI
- NUMERI REALI
- CHAR

Come rappresentiamo noi i numeri?

- Il SISTEMA DI NUMERAZIONE del MONDO OCCIDENTALE è il SISTEMA DECIMALE, POSIZIONALE.
  - da 0 a 9 CIFRE [10 CIFRE] → deduciamo la BASE del SISTEMA
  - da destra a sinistra → ad ogni posizione è associato un PESO



Il VALORE è rappresentato dal prodotto cifra * peso.

[ad es: $5*100 + 4*10 + 3*1 = (54)_{10}$]

Come si rappresentano i numeri in INFORMATICA.

- Il SISTEMA DI NUMERAZIONE UTILIZZATO in INFORMATICA è il SISTEMA BINARIO, POSIZIONALE

CARATTERISTICHE

- 2 CIFRE: 0 1 ⇒ BASE = (2) ⇒ i PESI sono potenze di 2.

Il VALORE è dato dalla FORMULA $A = \sum_{i=0}^{n-1} a_i B^i$ (a_i: CIFRA nel posto i-esimo; B^i: PESO del posto i-esimo)

Usando la FORMULA GENERALE per il calcolo del valore

$$\left[A = \sum_{i=0}^{n-1} a_i B^i\right]$$

ESEMPI (PASSAGGIO DA BINARIO A DECIMALE)

- $(101)_2 = 1*2^0 + 0*2^1 + 1*2^2 = (5)_{10}$ (3 BIT)
- $0000 = (0)_{10}$ (4 BIT)
- $0001 = (1)_{10}$ (4 BIT)
- $1011 = 1*2^0 + 1*2^1 + 0*2^2 + 1*2^3 = (11)_{10}$ (4 BIT)

- Dati N BIT quanti numeri NATURALI posso rappresentare? (2^N)
- Data una rappresentazione di INTERI POSITIVI su N BIT:
  - il MINIMO numero che posso rappresentare è 0
  - il MASSIMO NUMERO che posso rappr. è $(2^N - 1)$

ad es su 8 BIT → MAX NUM = (255)

SOTTRAZIONE

REGOLE

$0 - 0 = 0$

$0 - 1 = 1$ (borrow = 1)

$1 - 0 = 1$

$1 - 1 = 0$

es: ① $1010 - 0001 = 1001 = (9)$

② $11000 - 00111 = 10001 = (17)$

UNDERFLOW: si verifica quando si cerca di rappresentare i NUMERI NEGATIVI.

Spesso per comodità i numeri BINARI, si utilizzano dei SISTEMI DI NUMERAZIONE potenze del SISTEMA BINARIO:

- **SISTEMA OTTALE**:

  CIFRE da 0 a 7 ⇒ BASE 8

  Raggruppo i NUMERI BINARI a 3 alla volta: ogni cifra del SISTEMA OTTALE si ottiene a 3 CIFRE del sistema BINARIO

  ES: $\underbrace{10}_{2}\underbrace{111}_{7}\underbrace{001}_{1} \rightarrow (271)_8$

- **SISTEMA ESADECIMALE**:

  CIFRE da 0 a 9 + da A a F → BASE 16

  Raggruppo i numeri BINARI a 4 alla volta: ogni cifra del SISTEMA ESADECIMALE si ottiene a 4 CIFRE del SISTEMA BINARIO.

  ES. $\underbrace{1011}_{11=B}\underbrace{1001}_{9}{}_2 \rightarrow (B9)_{16}$

OPERAZIONI IN M&S

SVANTAGGI

- 0 DOPPIO (+0, -0)
- OPERAZIONI COMPLESSE → A+B

	A ≥ 0	A < 0
B ≥ 0	A+B	-|A|+B
B < 0	A ⊖ |B|	-(|A|+|B|)

↓ Si complementa il bit del segno di B

A causa di questi svantaggi si preferisce usare la RAPPRESENTAZIONE IN COMPLEMENTO A 2

Nella RAPPRESENTAZIONE in C_{A2} il MSB oltre ad avere un SEGNO, ha anche un PESO!

Se SEGNO = 0 (+) → CONSIDERO IL NUMERO COME M&S

1 (-) → PESO NEGATIVO

NUMERI RAPPRESENTABILI SU N BIT

- NUMERO + GRANDE = $2^{N-1}-1$ (Bit per il segno +)
- NUMERO + PICCOLO: $(-)2^{N-1}$ (l'ultimo bit ha segno 1 e ha anche un peso.)

[NOTA = lo -0 manca → è un numero negativo in più.]

N. su 8 BIT (1 BYTE)

NUMERO + GRANDE = 2^7-1 → 0 1 1 1 1 1 1 1

NUMERO + PICCOLO = -2^7 → 1 0 0 0 0 0 0 0

CONVERSIONE DA DECIMALE A CA2:

- Se il n. è positivo → lo faccio come un BINARIO PURO.
- Se il n. è negativo → mi FACCIO il COMPLEMENTO A 2

LO 1. RAPPRESENTO LO STESSO VALORE POSITIVO
2. INVERTO TUTTI I BIT
3. SOMMO 1 AL LSB.

ES:

(-5) su 4 BIT → +5: 0 1 0 1 → INVERTO I BIT: 1 0 1 0 → +1: [1 0 1 1]$_{CA2}$

$-2^3+2^1+2^0 = -5$

RAPPRESENTAZIONE NUMERI REALI (RAPPRESENTAZIONE FLOATING POINT)

Il FLOATING POINT è il modo di rappresentare dei numeri reali che può anche considerarsi l'analogo binario della NOTAZIONE SCIENTIFICA in base 10. Esso si contrappone alla RAPPRESENTAZIONE in "virgola fissa": Entrambe le rappresentazioni ~~sono~~ prevedono alla sinistra della virgola un PESO con potenze negative: $2^{-1}, 2^{-2}, 2^{-3}$, ecc.. [ad es. la RAPPRESENTAZIONE in VIRGOLA FISSA di 3.5 è 11.1]

- 2 BIT PARTE INTERA (PESO CON POTENZE POSITIVE)
- 1 BIT PER LA PARTE DECIMALE (PESO CON POTENZE NEGATIVE)

- **FIXED-POINT**

VANTAGGI: le OPERAZIONI fra due numeri allineati sono + veloci.

SVANTAGGI: il numero di valori rappresentabili è minore, in quanto la PRECISIONE ASSOLUTA è FISSA (posso rappresentare con precisione solo i numeri che hanno come parte DECIMALE potenze negative di 2).

- **FLOATING POINT**

VANTAGGI: nella rappresentazione FLOATING POINT essendo la PRECISIONE ASSOLUTA VARIABILE posso rappresentare una maggiore quantità di NUMERI.

I FLOAT sono rappresentati su 4 BYTE = 32 BIT

- 1 BIT per il SEGNO
- 23 BIT per la MANTISSA (precisione)
- 8 BIT ESPONENTE

Essi sono rappresentati nella FORMA: $N = \pm \cdot \text{MANTISSA} \cdot 2^{\text{ESPONENTE}}$

I DOUBLE sono rappresentati su 8 BYTE = 64 BIT

- 1 BIT per il SEGNO
- 52 BIT per la MANTISSA (quasi il doppio dei FLOAT)
- 11 BIT esponente

ES: RAPPRESENTARE 13.25_{10} in BASE (2)

13	6	3	1	0
	1	0	1	1

1101

$.25 = 01$

$13.25_{10} = 1101.01 = 1.10101 \cdot 2^3$

Teoria – Rappresentazione Binaria

Domanda 1 – Esame del 4.2.2013 – Turno A

Dati i due numeri in complemento a 2 scritti in formato Esadecimale AF04, 8711: - dire qual è il valore maggiore dei due - calcolare la somma dei due numeri e rappresentarla in formato esadecimale	*Risultato:* *maggiore:* AF04 *somma:* 0011 0110 0001 0101

Passaggi più significativi per arrivare al risultato

```
      1   1 111
AF04 =  1010 1111 0000 0100 = -2^16 + 0010 1111 0000 0100 => maggiore
8711 =  1000 0111 0001 0001 = -2^16 + 0000 0111 0001 0001
        -------------------------
        0011 0110 0001 0101 (overflow, segno discorde)
```

Domanda 1 – Esame del 4.2.2013 – Turno B

Date le seguenti rappresentazioni: 10011, 11001 e 00001 dire qual è il valore rappresentato maggiore in modulo, considerando le tre rappresentazioni in: a. modulo e segno b. complemento a 2	*Risultato:* *a.* -11001 *b.* -10011

Passaggi più significativi per arrivare al risultato

a. (1)0011 = -3, (1)1001 = -9, (0)0001 = 1

b. 10011 = -16+3 = -13, 11001 = -16+8+1 = -7, 00001 = 1

Domanda 1 – Esame del 4.2.2013 – Turno C

Rappresentare in CA2 (Complemento a due) e M&S (Modulo e segno) su 6 bit il numero $(-15)_{10}$	*Risposta* *CA2* 110001 *M&S* 101111

Passaggi più significativi per arrivare al risultato

CA2 XXXXXX -15 = -32 + 17 = -32 + 16 + 1 = -2^5 + 2^4 + 2^0 = 110001

M&S XXXXXX -15 = (-) 15 = (-) 8 + 4 + 2 + 1 = (-) 2^3 + 2^2 + 2^1 + 2^0 = 101111

(4) **Domanda 1 – Esame del 3.9.2012 – Turno B/B**

Rappresentare in CA2 (Complemento a due) e M&S (Modulo e segno) su 5 bit il numero $(-13)_{10}$	*Risultato* *CA2* *M&S*

Domanda 1 – Esame del 3.9.2012 – Turno C/A

Effettuare le seguenti operazioni in Complemento a 2. Sapendo che il risultato deve essere rappresentato su 5 bit, indicare se l'operazione genera overflow e motivare la risposta. a. 11011+01111 b. 10111-10011	*Risultato* a. 01010 b. 00100

Passaggi più significativi per arrivare al risultato	
11111 11011 + 01111 ----- 01010 Segno operandi discorde Overflow non possibile Verifica: -5+15=+10	Calcolo -(10011) = 01100+1 = 01101 11111 10111 + 01101 ----- 00100 Segno operandi discorde Overflow non possibile Verifica: -9-(-13)=+4

(6) **Domanda 1 – Esame del 3.9.2012 – Turno C/B**

Effettuare le seguenti operazioni in Complemento a 2. Sapendo che il risultato deve essere rappresentato su 5 bit, indicare se l'operazione genera overflow e motivare la risposta. a. 00011+01110 b. 10111-00011	*Risultato*

(7) **Domanda 1 – Esame del 13.7.2012 – Turno A/A**

(10)

Dati i seguenti numeri binari se ne faccia la somma ipotizzando siano scritti in Modulo e Segno e CA2: a. 01001 b. 10110	*Risultato* 01001_{MS} + 10110_{MS} 01001_{CA2}+ 10110_{CA2}:
Passaggi più significativi per arrivare al risultato	

Domanda 1 – Esame del 13.7.2012 – Turno C/A

(11)

Effettuare le seguenti operazioni in CA2. Indicare se l'operazione genera overflow e motivare la risposta. a. 11011+01111 b. 10111-10011	*Risultato*
Passaggi più significativi per arrivare al risultato	

Domanda 1 – Esame del 13.7.2012 – Turno C/B

(12)

Effettuare le seguenti operazioni in CA2. Indicare se l'operazione genera overflow e motivare la risposta. a. 00011+01111 b. 00111-10011	*Risultato*
Passaggi più significativi per arrivare al risultato	

Domanda 1 – Esame del 27.6.2012 – Turno A/A

	Risultato
Date le seguenti coppie di rappresentazioni in modulo e segno; per ciascuna coppia si determini la relazione maggiore, minore o uguale, tra i valori rappresentati: a. 10110 <?> 11010 b. 101 <?> 11101	a.: b.:
Passaggi più significativi per arrivare al risultato	

Domanda 2 – Esame del 27.6.2012 – Turno C/A

Indicare l'intervallo di rappresentazione di un numero, N, rappresentato su 8 bit nel caso si utilizzi la rappresentazione in binario puro oppure complemento a due (CA2).	Binario puro: $da\ 0\ a\ 2^8-1$ CA2: $da\ -2^{8-1}\ a\ 2^8-1$

Passaggi più significativi per arrivare al risultato
 	
`BIN: 0..(2^N)-1 = 0 .. 255`
 	
`CA2: -(2^(N-1)) .. +(2^(N-1)-1) = -128 .. +127`	

Domanda 2 – Esame del 27.6.2012 – Turno C/B

Indicare l'intervallo di rappresentazione di un numero, N, rappresentato su 9 bit nel caso si utilizzi la rappresentazione in binario puro oppure complemento a due (CA2).	*Risultato* Binario puro: 0 ... 511 CA2: -256 ... 255
Passaggi più significativi per arrivare al risultato	

(2) CA2

10011

00111

$-2^4 + 3$

$-32 + 3 = -29$

29	16	8	4	2	1
	1	0	1	1	1

11101

NOT

11101

00111

(3) $(-12)_{10}$

SVSBIS

NOT

11100

12	8	4	2	1
	0	0	1	1

1100

CA2

01100

10011

10100

-16+4

(5) CA2

SBIS

a. 11011 +

00111

01010

Segno OVSCODE degli operandi

OVERFLOW

Il risultato ha segno OVSCODE

overflow non possibile

b.

-10011

01100 → 01101

10 111 +

01101

00100

Segno OVSCODE operandi

OVERFLOW non possibile

(17)

8 BIT

BINARIO PURO

da 0 a $2^N - 1 = 255$

CA2

da -2^7 a $2^7 - 1$

Come accede il MICROPROCESSORE alla MEMORIA RAM?

Attraverso i BUS: (SISTEMA CIRCUITALE PC) sono dei fili che collegano il MICROPROCESSORE alla RAM e su ogni filo è presente un determinato valore di tensione corrispondente in BIT.

- La FORMA di un BUS (che può rappresentare un solo dato per volta), i dati del modello che ne è un BIT di cui è costituito un singolo DATO (AMPIEZZA) e il n° di dati trasmessi al secondo (FREQUENZA): P.T. N · freq = ~~bit~~ (bit/s)

TIPI DI BUS: BUS → ABUS, DBUS, CBUS

Un singolo BUS è suddiviso in 3 "sotto BUS":

- A-BUS (ADDRESS BUS): per gli INDIRIZZI (INDIRIZZI) — IL PARALLELISMO è il n° di BIT presenti per gli indirizzi: in genere sono 32 = 4 BYTE che consentono di accedere a 4 GB di RAM o qualcosa in più.
- D-BUS (DATA BUS): per i DATI — IL PARALLELISMO del D-BUS è 64 BIT = 8 BYTE
- C-BUS (CONTROL BUS): Read/Write? TEMPISTICHE

DIMENSIONE MASSIMA MEMORIA (INTERNA) $2^{ABUS} \cdot D\text{-BUS}$ BIT

N BIT (PARALLELISMO D-BUS)

$2^{(A)}$ [ ] PARALLELISMO A-BUS

Es: ABUS = 20 BIT
DBUS = 16 BIT
↓
Max Memory = $2^{20} \cdot 2$ BYTE = 2 MB

Ricorda: 1K = 2^{10} 1M = 2^{20} 1G = 2^{30}

MASSIMA MEMORIA ESTERNA: Non dipende dall'ABUS perché viene vista come una periferica. Essa dipende dal BUS esterno (quelli di I/O).

CENNO SUI DISPOSITIVI PERIFERICI

CPU ← INTERFACCIA (CONTROLLER) → PER

CONVERTE TUTTI I DATI in DIGITALE: ad es. CAVO VGA → converte da digitale in analogico

si differenzia dal CAVO HDMI (è digitale) → il MONITOR ad effettuare la conversione

Es 3 LAB 4

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

int main()
{
    int a;
    int somma = 0;
    while (scanf("%d", &a) != '0') {
        somma = somma + a;
    }
    printf("%d", somma);
return 0;
}
```

Es 1 LAB 6

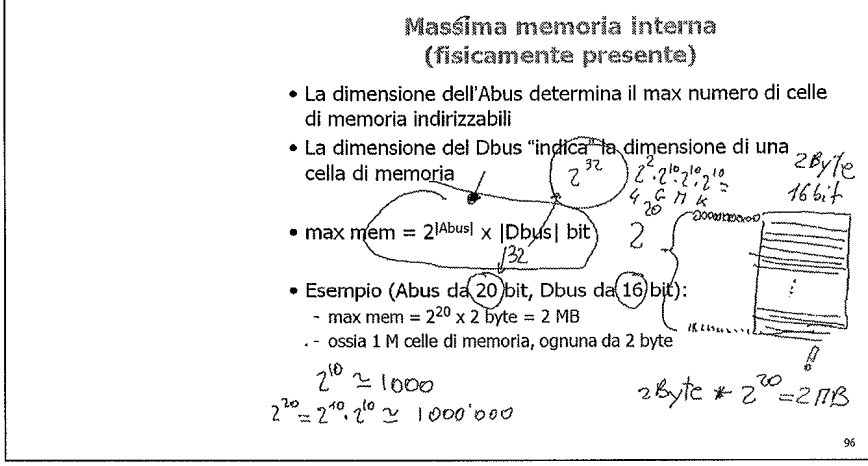
```
include <stdlib.h>

define N 5
int main()
{ int vett[N];
  int i;
  int palindromo = 1;

printf("Inserisci %d valori del vettore", N);
for (i = 0; i < N; i++) {
        scanf("%d", &vett[i]);
    }

for (i = 0; i < N/2; i++) {
        if (vett[i] != vett[N - 1 - i])
            palindromo = 0;
            break;
    }
}
```

```
if (palindromo = 0) {
printf("sveq. non palindroma");
} else { printf("sveq. palindroma"); }
return 0;
}
```

Teoria – Architettura del calcolatore

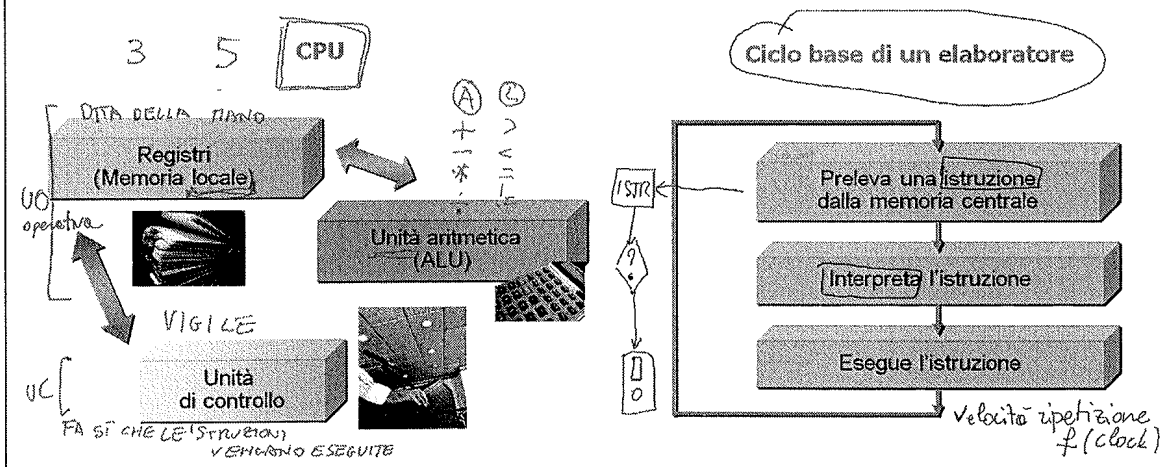
Domanda 3 – Esame del 4.2.2013 – Turno B

Come si può determinare la quantità massima di memoria di sistema (memoria centrale) che può essere installata su di un elaboratore?



Domanda 3 – Esame del 4.2.2013 – Turno C

Descriverne brevemente le funzionalità dell'unità di controllo.



Descrivere brevemente le caratteristiche di un bus.
3 tipi

Domanda 3 – Esame del 13.7.2012 – Turno A/B

Descrivere brevemente le caratteristiche principali della memoria RAM.

Domanda 3 – Esame del 13.7.2012 – Turno B/A

Spiegare brevemente che cos'è un clock di sistema e qual'è la sua unità di misura.

Domanda 3 – Esame del 13.7.2012 – Turno C/A

Descrivere brevemente l'architettura di un elaboratore.

Domanda 3 – Esame del 13.7.2012 – Turno C/B

Descrivere le caratteristiche principali di una CPU.

Passaggi più significativi per arrivare al risultato

Domanda 2 – Esame del 26.6.2012 – Turno B/A

Si supponga di utilizzare un microprocessore in cui i float sono codificati su 32 bit. Qual è il numero minimo di byte occupati dalla struttura in memoria?
struct dipendente {
char Nome [16]; 16
char Cognome [16] 16
char Cod_INPS [8]; 8
char DataAssunzione [8]; 8
float Stipendio; 4 BYTE
}
	Risultato
	52 BYTE

Passaggi più significativi per arrivare al risultato	

Domanda 3 – Esame del 26.6.2012 – Turno B/A

Descrivere il ciclo macchina base di un elaboratore. E' consigliato corredare la spiegazione con un'illustrazione grafica del ciclo macchina.

Domanda 2 – Esame del 26.6.2012 – Turno B/B

Si supponga di utilizzare un microprocessore in cui i double sono codificati su 64 bit. Qual è il numero minimo di byte occupati dalla struttura in memoria? struct artista { char Nome [16]; 16 char Cognome [16] 16	*Risultato* 592 BYTE

Domanda 1 – Esame del 26.6.2012 – Turno C/B

Si supponga di utilizzare un microprocessore in cui i float sono codificati su 32 bit. #define M 24 typedef struct F{ char Piva [10]; char data [8]; float ImportoNetto; float IVA; } Fattura; Fattura ff_clienti[M]; Qual è il numero minimo di byte occupati dalla struttura dati ff_clienti[M] in memoria?	*Risultato* 624 BYTE
Passaggi più significativi per arrivare al risultato	

Domanda 3 – Esame del 26.6.2012 – Turno C/B

Disegnare l'architettura di una "Central Processing Unit (CPU)" in un elaboratore mediante i blocchi fondamentali. Descrivere brevemente ogni singolo blocco.

Nome e Cognome	
Matricola	
Corso Poli@Home☐1(AAAA-BARB)☐2(BARC-BOT)☐3(BOU-CASA)☐4(CASB-CHZ)☐5(CIA-COND)☐6(CONE-DELR)☐7(DELS-FEQ)☐8(FER-GEQ)☐9(GER-JOZ)☐10(JPA-MALI)☐11(MALJ-MOD)☐12(MOE-PAK)☐13(PAL-PORS)☐14(PORT-ROQ)☐15(ROR-SIGN)☐16(SIGO-TRIO)☐17(TRIP-ZZZ)☐18(Automotive)☐Solo Prog☐	

Teoria

Domanda 1

	Risultato
• Eseguire la seguente operazione tra numeri binari interpretandoli prima in binario puro (come numeri interi senza segno) e poi in complemento a 2 (come numeri interi con segno) indicando se si verifica overflow: 11110101 + 01101101	Binario: overflow (si/no): CA2: overflow (si/no):
Passaggi più significativi per arrivare al risultato	

Domanda 2

Cosa contengono il Program Counter (PC) e l'InstructionRegister (IR)?	*PC:* *IR:*

Domanda 3

Stabilire se nell'algebra booleana vale l'uguaglianza xy + xyz=xyz	*Risposta (si/no)*
Passaggi più significativi per arrivare al risultato	

PROVA D'ESAME 22-07-2011

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

int main (int argc, char * argv[]){
  FILE * fp;
  if (argc != 2){
     exit (EXIT_FAILURE);
  }
  if ((fp = fopen (argv[1], "r")) == NULL){
     exit (EXIT_FAILURE)
  }
printf("File aperto con successo\n");
for (i=0; i<N; i++){
   for(j=0; j<N; j++){
      tabella_noleggio [i][j] = 0;

while ((fgets (stringa_tmp, N, fp)) != NULL){
  if ((sscanf (stringa_tmp, "%d %d %d %d", &x1, &y1, &x2, &y2)) != 4){
     exit(EXIT_FAILURE); }
    flag = 0;
    for ( i = x1; i < x2+1 ; i++){
        for ( j = y1 ; j < y2+1 ; j++){
            if (m[i][j] == 0){
                m[i][j] = 1;
            } else { flag = 1;
                break;
            }

if (flag == 1){
   printf("I noleggi hanno almeno una sovrapposizione");
   } else { printf("I noleggi non hanno sovrapposizione"); }
fclose (fp);
return 0;
}
```

ALTRE DICHIARAZIONI

```
char stringa_tmp [N];
int tabella_noleggio [N][N];
int i, j;
int x1, y1, x2, y2;
int flag;
```

* aggiungere anche

```
if ( (x1<0)||(x1>N)||(x2<0)||(x2>N)||
     (y1<0)||(y1>N)||(x2<0)||(y2>N) )
   printf ("Errore");
   exit(1);
```

Nome e Cognome	
Matricola	

Corso

1(AAAA - BARA) ☐ 2 (BARB – BOTS) ☐ 3 (BOTT – CAR) ☐ 4 (CAS – CORD) ☐ 5 (CORE – DIF) ☐

6 (DIG – FIOR) ☐ 7 (FIOS - GIORD) ☐ 8 (GIORE – LANE) ☐ 9 (LANF – MARA) ☐ 10 (MARB – MOH) ☐

11 (MOI – PAK) ☐ 12 (PAL – POLH) ☐ 13 (POLI – ROSA) ☐ 14 (ROSB – SIL) ☐ 15 (SIM – TR) ☐

16 (TS – ZZ) ☐ E1 (AA – LZ) ☐ E2 (MA – ZZ) ☐ Poli@Home ☐ Es. (5 crediti) ☐

Teoria

Domanda 1

	Risultato
Dati i seguenti numeri in complemento a 2, determinare la loro rappresentazione in decimale n1: 10111001 n2: 010101 n3: 1111	n1: n2: n3:
Passaggi	

Domanda 2

Dati i seguenti numeri in complemento a 2 su 8 bit X = 10010111 Y = 01011101 calcolare Z = X+Y verificando la presenza di overflow.	Z: overflow (si/no):
Passaggi	

Domanda 3

Descrivere vantaggi e svantaggi delle possibili rappresentazioni dei numeri interi con segno.

Nome e Cognome	
Matricola	

Corso

1(AAAA - BARA) ☐ 2 (BARB – BOTS) ☐ 3 (BOTT – CAR) ☐ 4 (CAS – CORD) ☐ 5 (CORE – DIF) ☐

6 (DIG – FIOR) ☐ 7 (FIOS - GIORD) ☐ 8 (GIORE – LANE) ☐ 9 (LANF – MARA) ☐ 10 (MARB – MOH) ☐

11 (MOI – PAK) ☐ 12 (PAL – POLH) ☐ 13 (POLI – ROSA) ☐ 14 (ROSB – SIL) ☐ 15 (SIM – TR) ☐

16 (TS – ZZ) ☐ E1 (AA – LZ) ☐ E2 (MA – ZZ) ☐ Poli@Home ☐ Es. (5 crediti) ☐

Teoria

Domanda 1

	Risultato
Dato il seguente numero su 6bit: 110101 Determinare il valore decimale interpretandolo come - Binario puro (BIN) - Modulo e segno (MS) - Complemento a 2 (CA2)	BIN: MS: CA2:
Passaggi	

Domanda 2

Siano dati i numeri in base 10 X1 = +253 X2 = -310 Dopo averli rappresentati nel formato in complemento a due su nove bit, si cacloli la seguente operazione verificando la presenza di overflow: X3 = X1 + X2	X3: overflow (si/no):
Passaggi	

Domanda 3

Si supponga di utilizzare un calcolatore in cui gli interi sono raprresentati su 32bit. Qual è il numero minimo di byte occupato dalla seguente struttura dati? typdef struct { char nome[20]; char cognome[20]; char matricolo[8]; int eta; } studente; studente registro[100];	#Byte:
Passaggi	

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 95 di 126

PROGRAMMAZIONE 20-1-2014 TURNO A

```
Include <stdio.h>
Include <stdlib.h>

# define R 17
# define C 10

# define MARE 'O'
# define MARE_COLPITO 0
# define NAVE 'N'
# define NAVE_COLPITA 1

int main (int argc, char * argv[]){
    FILE * fp;
    char mappa[R][C+1];
    int programma_terminato;
    int riga_colpito, colonna_colpito;
    int affondato;
    int riga, colonna;

if (argc != 2){
    printf("Errore nell'apertura del programma\n");
    exit (EXIT_FAILURE);
    }

if ((fp = fopen(argv[1], "r")) == NULL){
    printf("Errore nell'apertura del file\n");
    exit(EXIT_FAILURE);
    }

printf("File aperto con successo\n");
leggi_mappa( fp, mappa, R);
fclose(fp);
programma_terminato = 1;
for (riga = 0; riga < R; riga++){
    for (colonna = 0; colonna < C; colonna++){
        if (mappa[riga][colonna] == NAVE){
            programma_terminato = 0;
```

FUNZIONI

```
void leggi_mappa (FILE * fp, char mappa[][C+1], int r){
    int i;
    for (i=0; i < r; i++){
        fscanf (fp, "%s", mappa[i]);
        }
    }
```