NUMERO: 2137A- ANNO: 2017

A P P U N T I

STUDENTE: Barlassina Lorenzo

MATERIA: Informatica - Prof. Acquaviva

IL COMPUTER

L'INFORMATICA: È LA SCIENZA CHE RAPPRESENTA E MANIPOLA LE INFORMAZIONI.

1942 - '57	1 generazione	= TUBI A VUOTO (VALVOLE)
1958 - '63	2 gen.	= TRANSISTORI
1964 - '80	3 gen.	= CIRCUITI INTEGRATI MSI
1980 - OGGI	4 gen.	= CIRCUITI VLSI (VERY LARGE SCALE INTEGRATED)
FUTURO	5 gen.	= ?

ESISTONO DUE GRANDI CLASSI DI ELABORATORI:

- ELABORATORI DI USO GENERALE (GENERAL-PURPOSE COMPUTER)
  ES. PC.
- ELABORATORI DEDICATI (SPECIAL-PURPOSE COMPUTER) EMBEDDED
  ES. COMPUTER DENTRO ALLA LAVATRICE

SERVER: È UN ELABORATORE CHE FORNISCE DEI "SERVIZI" AD ALTRI ELABORATORI (CHIAMATI CLIENTS) ATTRAVERSO UNA RETE (COMPUTER NETWORK).

SERVER FARM: SI FA RIFERIMENTO ALL'INSIEME DI ELABORATORI SERVER COLLOCATI IN UN APPOSITO LOCALE (CENTRO DI CALCOLO) PRESSO UNA MEDIA O GRANDE AZIENDA.

MAINFRAMES (BIG IRON): SONO ELABORATORI DI GRANDI PRESTAZIONI USATI PRINCIPALMENTE DA GRANDI IMPRESE PER RILEVANTI APPLICAZIONI SOFTWARE

SUPER COMPUTER (2012): GRANDISSIMI CALCOLATORI, IBM BLUE GENE/Q, LIVERMORE LABS, POTENZA 16 PFLOPS (PETA FLOPS)

SUPER COMPUTER (2015) JUNE 34 PETAFLOPS: 16 000 000 000 000 000 MOLTIPLICAZIONI AL SECONDO (16 MILA MILIARDI)

FLOATING POINT → CIOÈ OPERAZIONI CON NUMERI CON LA VIRGOLA CIOÈ PIÙ COMPLICATE

FLOPS: OPERAZIONE PER SECONDO

PETA: MILLE MILIARDI DI OPERAZIONI

DOBBIAMO TRADURRE IL PROBLEMA IN UNA SERIE DI PASSI/PASSAGGI COSÌ CHE IL PC COMPRENDA LA SITUAZIONE. → PROGRAMMA

LA SOLUZIONE DI UN PROBLEMA PASSA GENERALMENTE ATTRAVERSO UN **ALGORITMO**. ALGORITH + DATA = PROGRAM

[ALGORITMI CICLI E BIFORCAZIONE]

1) ALGORITMO **SEMPLICE**: GUNNARE LA PASTA
2) " **COMPLESSO**: ORDINARE UN MAZZO DI CARTE

ALGORITMO: DESCRIZIONE PRECISA E FORMALE DI UNA SEQUENZA FINITA DI AZIONI CHE DEVONO ESSERE ESEGUITE PER GIUNGERE ALLA SOLUZIONE DI UN PROBLEMA

ALGORITMI X ORDINAMENTO PIÙ FAMOSI: - BUBBLESORT
- QUICKSORT

DIAGRAMMA DI FLUSSO STRUTTURATI, SENZA SALTI, CHE SEGUANO CIOÉ LE REGOLE DELLA PROGRAMMAZIONE STRUTTURATA (ASSENZA DI SALTI INCONDIZIONATI)

UN DIAGRAMMA DI FLUSSO É DETTO STRUTTURATO SE CONTIENE SOLO UN INSIEME PREDEFINITO DI STRUTTURE ELEMENTARI:

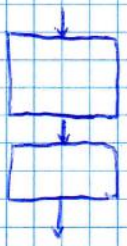
- UNO ED UNO SOLO BLOCCO START
- UNO ED UNO SOLO BLOCCO STOP

SEQUENZA DI BLOCCHI (DI AZIONE E/O DI IMPUT/OUTPUT)

- IF - THEN - ELSE MECCANISMO DI DECISIONE
- WHILE - DO
- REPEAT - UNTIL

MECCANISMO DI ?

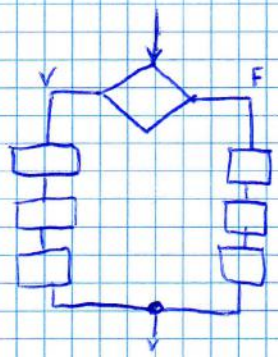
→ STRUTTURE DI CONTROLLO



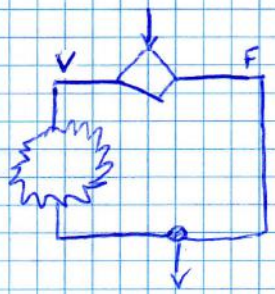
STRUTTURA ELEMENTARE

=

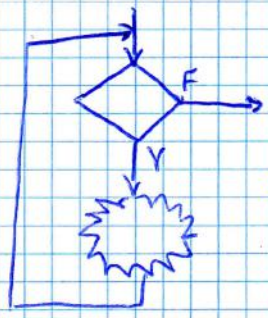
PIÚ STRUTTURE ELEMENTARI MESSE ASSIEME



BLOCCO IF - THEN - ELSE
(SE) (ALLORA) (ALTRIMENTI)



BLOCCO IF - THEN



BLOCCO WHILE - DO
(FINTANTO CHE) (FAI)

LA STRUTTURA VIENE VALUTATA ZERO VOLTE

FINCHÉ LA CONDIZIONE É COSÍ CONTINUA A CICLARE.

LA CONSOLE: È L'INSIEME DI TASTIERA (INPUT) E VIDEO/SCHERMO (OUTPUT)

PER FARE UN PROGRAMMA BISOGNA APRIRE L'APPLICAZIONE "CONSOLE"

LINGUAGGI

- DIVERSI LIVELLI DI ASTRAZIONE
  - LINGUAGGI AD ALTO LIVELLO

    ELEMENTI DEL LINGUAGGIO HANNO COMPLESSITÀ EQUIVALENTE AI BLOCCHI DEI DIAGRAMMI DI FLUSSO STRUTTURATI (CONDIZIONALI, CICLI...)

    ES. C, C++, BASIC, PASCAL, FORTRAN, JAVA

    INDIPENDENTI DALL'HARDWARE

  - LINGUAGGI "ASSEMBLER"

    DIPENDENTI DALL'HARDWARE

    SPECIFICI PER UNA SPECIFICA ARCHITETTURA (MICROPROCESSORE)

ELEMENTI DEL LINGUAGGIO

ESSENDO IL LINGUAGGIO UN'ASTRAZIONE, ESISTONO ALCUNI FONDAMENTALI ELEMENTI SINTATTICI ESSENZIALI PER L'USO DEL LINGUAGGIO STESSO:

- PAROLE CHIAVE (KEYWORD)
- DATI
- IDENTIFICATORI
- ISTRUZIONI

GLI ELEMENTI SINTATTICI

- double → double int RADDOPPIA i byte DELL'int
  DA SOLO RAPPRESENTA UN REALE DA 8 byte (64 bit)
- float 4 byte = 32 bit
- int 4 byte = 32 bit
- char 1 byte = 8 bit

IL DATO IN LINGUAGGIO AD ALTO LIVELLO È INDIVIDUATO DA:

- UN NOME (IDENTIFICATORE)
- UNA INTERPRETAZIONE (TIPO)
- UNA MODALITÀ DI ACCESSO (COSTANTE O VARIABILE)

L'IDENTIFICATORE PERMETTE DI DARE NOMI INTUITIVI AI DATI

IL TIPO INDICA L'INTERPRETAZIONE DEI DATI IN MEMORIA; LEGATO ALLO SPAZIO OCCUPATO DA UN DATO; PERMETTE DI DEFINIRE TIPI "PRIMITIVI" (NUMERI, SIMBOLI) INDIPENDENTEMENTE DAL TIPO DI MEMORIZZAZIONE DEL SISTEMA.

TIPO DI ACCESSO - INDICA LA MODALITÀ DI ACCESSO AI DATI:

- VARIABILI
- COSTANTI

LE ISTRUZIONI INDICANO LE OPERAZIONI CHE IL LINGUAGGIO PERMETTE DI ESEGUIRE (TRADUCENDOLE) A LIVELLO MACCHINA:

- PSEUDO-ISTRUZIONI
- ISTRUZIONI ELEMENTARI
- " DI CONTROLLO DEL FLUSSO

CLOCK

: È IL TEMPORIZZATORE DEL SISTEMA OPERATIVO, (CHE GENERA UN RIFERIMENTO TEMPORALE COMUNE PER TUTTI GLI ELEMENTI COSTITUENTI IL SISTEMA DI ELABORAZIONE)

T = PERIODO DI CLOCK

- UNITÀ DI MISURA = S (secondi)

f = FREQUENZA DI CLOCK (= $1/T$)

- UNITÀ DI MISURA = S^{-1} = H_z (CICLI/SECONDO)

IN UN INTEL CORE i7-2700 LA FREQUENZA DI CLOCK È 3,5 GHz
(IN 3,5 MILIARDESIMI DI SECONDO LA LUCE PERCORRE 1 m [1 cm, 93 mm])
GLI INTEL SONO COSTRUITI CON TECNOLOGIE A 20 nm

IL CICLO MACCHINA È L'INTERVALLO DI TEMPO IN CUI VIENE SVOLTA UNA OPERAZIONE ELEMENTARE ED È UN MULTIPLO INTERO DEL PERIODO DEL CLOCK.

L'ESECUZIONE DI UN'ISTRUZIONE RICHIEDE UN NUMERO INTERO DI CICLI MACCHINA.

MEMORIA

- MEMORIZZA I DATI E LE ISTRUZIONI NECESSARIE ALL'ELABORATORE PER OPERARE.
- CARATTERISTICHE:
  1) INDIRIZZAMENTO
  2) FISICO
  3) ACCESSO (SEQUENZIALE O CASUALE)

1) LA MEMORIA È ORGANIZZATA IN CELLE (MINIMA UNITÀ ACCESSIBILE DIRETTAMENTE). AD OGNI CELLA DI MEMORIA È ASSOCIATO UN INDIRIZZO (NUMERICO) PER IDENTIFICARLA UNIVOCAMENTE.

2) OGNI CELLA CONTIENE UNA QUANTITÀ FISSA DI BIT: (IDENTICA PER TUTTE LE CELLE / ACCESSIBILE CON UN'UNICA ISTRUZIONE / È UN MULTIPLO DEL BYTE, MINIMO UN BYTE).

MEMORIA INTERNA: RAM — È ALL'INTERNO DELL'ELABORATORE, È ALLO STATO SOLIDO (CHIP), SOLITAMENTE È VOLATILE, È VELOCE (NANOSECONDI, 10^{-9} S), QUANTITÀ LIMITATA (QUALCHE GB), NON RIMOVIBILE, COSTOSA (0,1 €/MB).

ROM = READ ONLY MEMORY (MEMORIA A SOLA LETTURA) → MEMORIA NON VOLATILE
CONTIENE LE INFORMAZIONI NECESSARIE PER IL FUNZIONAMENTO DEL PC ALL'ACCENSIONE. → PROGRAMMA D'AVVIO (BOOT PROGRAM) DI BASE

BIOS DICE COSA FARE

MEMORIA ESTERNA: HARD DISK O MEMORIA DI MASSA, DVD, CD — È ALL'ESTERNO DELL'ELABORATORE, TALVOLTA RIMOVIBILE, NON ELETTRONICA (es. MAGNETICA O OTTICA) O ELETTRONICA (SSD: SOLID STATE DISK), PERMANENTE, LENTA (MILLISECONDI, 10^{-3} S), GRANDE QUANTITÀ (QUALCHE TB), ECONOMICA (0,1 €/GB).

RAM – RANDOM ACCESS MEMORY

- CIRCUITI INTEGRATI
- IL TEMPO DI ACCESSO È COSTANTE (INDIPENDENTE DALLA CELLA SCELTA)
- T_2 = TEMPO DI ACCESSO È COSTANTE
- MEMORIA INTERNA VOLATILE IN LETTURA E SCRITTURA.
- TUTTO CIÒ CHE È NELL'HARD DISK PER ESSERE ESEGUITI PASSANO E VENGONO SEMPRE COPIATI NELLA RAM.

MEMORIA CACHE (FUNZIONA SIA PER I DATI CHE PER LE ISTRUZIONI)

MEMORIA INTERMEDIA VELOCE, GESTITA IN TOTALE AUTONOMIA

MEMORIZZARE DATI CHE VENGONO PIÙ FREQUENTEMENTE USATI

CALCOLA I DATI CHE IL SISTEMA POTRÀ AVER BISOGNO IN FUTURO (CPU)

LA CACHE USA DUE PRINCIPI: LOCALITÀ SPAZIALE

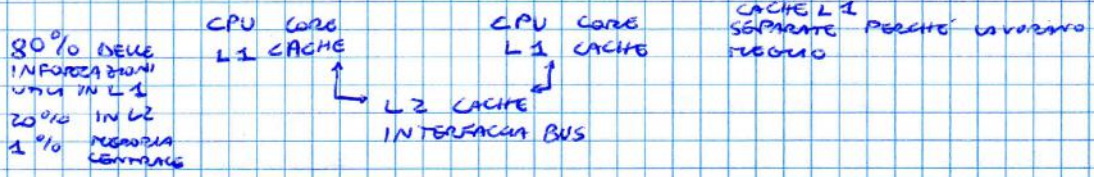
" TEMPORALE

CPU - MULTI - CORE

CON IL CRESCERE DELLA CAPACITÀ DI INTEGRAZIONE È POSSIBILE AUMENTARE IL NUMERO DI UNITÀ OPERATIVE SU UN UNICO CHIP.

SI REALIZZANO PERTANTO DISPOSITIVI CHE CONTENGONO PIÙ CPU (DETTE CORE), IN GRADO CIOÈ DI ESEGUIRE PIÙ PROGRAMMI O PIÙ PEZZI DI PROGRAMMA IN PARALLELO.

MULTI-CORE: 2 O PIÙ PROCESSORI INCLUSI NELLO STESSO CHIP



GPU: PROCESSORE GRAFICO (QUAD-CORE) / HA MOLTI CORE (CENTINAIA) MA PIÙ PICCOLI E SEMPLICI (ELEMENTARI)

(GRAPHICS PROCESSOR UNIT)

DATI

I DATI NUMERICI SONO I PIÙ USATI E TUTTI GLI ALTRI TIPI DI DATI VENGONO TRASFORMATI IN NUMERICI.

IN C TUTTI I DATI DEVONO ESSERE DICHIARATI PRIMA DI ESSERE USATI.

LA DICHIARAZIONE DI UN DATO MI INDICA:

- L'ALLOCAZIONE DI UNO SPAZIO IN MEMORIA ATTO A CONTENERE IL DATO
- L'ASSEGNAZIONE DI UN NOME A TALE SPAZIO IN MEMORIA

LA DICHIARAZIONE VA NELLA PARTE DICHIARATIVA (GLOBALE O LOCALE). OCCORRE SPECIFICARE: NOME, TIPO E MODALITÀ DI ACCESSO.

SE LA MODALITÀ DI ACCESSO NON È SPECIFICATA È SOTTOINTESA: VARIABILE.

IDENTIFICATORI DA PAROLE CHIAVE

TIPI BASE O PRIMITIVI

- **char** INTERI A 8 bit USATI PER I CARATTERI (CODIFICA ASCII) ← ASSOCIA IL NUMERO ALLA LETTERA
- **int** INTERO
- **float** REALE (FLOATING POINT SINGOLA PRECISIONE, EX. 32 bit)
- **double** REALE (" " DOPPIA " , EX. 64 bit)

LA DIMENSIONE DI QUESTI TIPI DIPENDE DALL'ARCHITETTURA (NON DEFINITA DAL LINGUAGGIO), AD ECCEZIONE DEL CHAR: |char| = 8 bit = 1 byte SEMPRE

SONO PREVISTI DEI MODIFICATORI AI TIPI BASE:

signed / unsigned (+ o −) APPLICABILI A char e int

- SEGNO → VALORE NUMERICO CON SEGNO
- → VALORE NUMERICO SENZA SEGNO

signed 2^{16} → −, 2^{16} → +

unsigned 2^{32} → +

short / long APPLICABILI A int UTILIZZABILI ANCHE SENZA SPECIFICARE int.

- DIMENSIONE

ES. long int NUMERO INTERO CON IL DOPPIO DI BIT 64 BIT
short int " " " LA METÀ DI BIT 16 BIT
long long int " " " IL QUADRUPLO DI BIT 128 BIT

ES. [signed/unsigned] short [int] con [...] OPZIONALE
" int SI PUÒ ANCHE OMETTERE
" long [int]
" long long [int]

ES. int x;
short int stipendio;

$6{,}02 \cdot 10^{23}$ = 6.02 e 23
"REALTÀ" IN C

1 0 1 1 0 1 1 0
↑ ↑
MSB LSB
MOST (PIÙ) SIGNIFICANT BIT — LEAST (MENO) SIGNIFICANT BIT

0 + 0 = 0
1 + 0 = 1
0 + 1 = 1
1 + 1 = 0 CARRY = 1 (RIPORTO)

1 + 1 = 2 NO 1 + 1 = 0 RIPORTO CARRY = 1

0 − 0 = 0
0 − 1 = 1 BORROW = 1
1 − 0 = 1
1 − 1 = 0

OVERFLOW

IL TERMINE VIENE USATO PER INDICARE L'ERRORE CHE SI VERIFICA IN UN SISTEMA DI CALCOLO AUTOMATICO QUANDO IL RISULTATO DI UN'OPERAZIONE NON È RAPPRESENTABILE CON LA MEDESIMA CODIFICA E STESSO NUMERO DI BIT DEGLI OPERANDI.

NELLA SOMMA DI BINARIO PURO SI HA OVERFLOW QUANDO: - SI LAVORA CON NUMERO FISSO DI BIT
- SI HA CARRY SUL MSB

ES. OPERAZIONI SU NUMERI DA 4 BIT CODIFICATI IN BINARIO PURO

RIPORTO (CARRY)
0 1 0 1 +
1 1 1 0 =
OVERFLOW → 1 0 0 1 1

OPERANDI DA 4 BIT E RISULTATO DA 5 BIT
4 BIT + 1 OVERFLOW

IL SISTEMA OTTALE

- BASE = 8 (TALVOLTA INDICATA CON Q PER OCTAL)
  - CIFRE = { 0, 1, 2, 3, 4, 5, 6, 7 }
  - UTILE PER SCRIVERE IN MODO COMPATTO I NUMERI BINARI (3 : 1)

10 111 001 = 2
2 7 1 8

IL SISTEMA ESADECIMALE

- BASE = 16 (TALVOLTA INDICATA CON H PER HEXADECIMAL)
  - CIFRE = { 0, 1, ..., 9, A, B, C, D, E, F } (10 11 12 13 14 15)
  - UTILE PER SCRIVERE IN MODO COMPATTO I NUMERI BINARI (4 : 1)

1011 1001 = 2
B 9 16

I NUMERI CON SEGNO (+ o −) — CODIFICA "MODULO E SEGNO" MS

- UN BIT PER IL SEGNO (TIPICAMENTE IL MSB):
  - 0 = SEGNO POSITIVO (+)
  - 1 = " NEGATIVO (−)

SEGNO	MODULO
1 BIT	N − 1 BIT

- N − 1 bit PER IL VALORE ASSOLUTO O MODULO

NUMERO DI BIT CHE OCCUPANO:

- BINARIO = $0 \dots [(2^N) - 1]$
- CA2 = $-(2^{(N-1)}) \dots + (2^{(N-1)} - 1)$
- MS = $-(2^{(N-1)} - 1) \dots + (2^{(N-1)} - 1)$

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 15 di 40

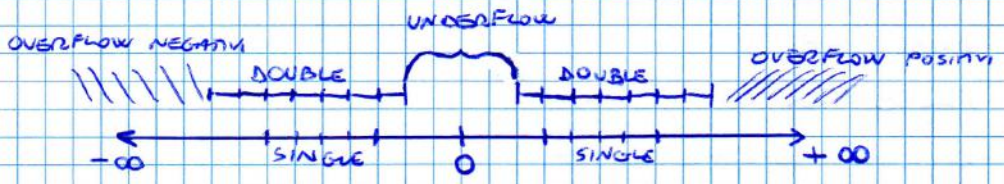
NOTAZIONE IN VIRGOLA MOBILE (FLOATING POINT)

$N = \pm M \cdot 2^{E}$ M: MANTISSA

E: ESPONENTE

- UN NUMERO IN TALE NOTAZIONE UTILIZZA DI NORMA 32 BIT PARI AD UN INTERVALLO $-10^{38}, +10^{38}$

$\pm \; 1.0110 \cdot 2^{7}$



CARATTERI

OCCORRE UNA CODIFICA STANDARD PERCHÉ È IL GENERE DI INFORMAZIONE PIÙ SCAMBIATA:

1) CODICE ASCII (AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE)
↓ /HUSKY/ O /ASCI/

2) CODICE EBCDIC (EXTENDED BCD INTERCHANGE CODE)

1) ASCII USATO NELLE TELECOMUNICAZIONI, USA 8 BIT (ORIGINARIAMENTE 7 BIT PER US-ASCII) PER RAPPRESENTARE:

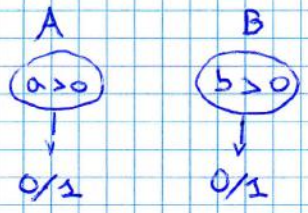
- 52 CARATTERI ALFABETICI (a... z, A... Z)
- 10 CIFRE (0... 9)
- SEGNI DI INTERPUNZIONE (, ; ! ? ecc.)
- CARATTERI DI CONTROLLO

CR	(13)	CARRIAGE RETURN
LF, NL	(10)	NEW LINE, LINE FEED
FF, NP	(12)	NEW PAGE, FORM FEED
HT	(9)	HORIZONTAL TAB
VT	(11)	VERTICAL TAB
NUL	(0)	NULL
BEL	(7)	BELL
EOT	(4)	END-OF-TRANSMISSION
...	...	...

PROPRIETÀ COMMUTATIVA E ASSOCIATIVA

- $A \cdot B = B \cdot A$
  $A + B = B + A$

- $A \cdot B \cdot C = (A \cdot B) \cdot C = A \cdot (B \cdot C) = (A \cdot C) \cdot B$
  $A + B + C = (A + B) + C = A + (B + C) = (A + C) + B$



DISTRIBUTIVA

- $A \cdot (B + C) = A \cdot B + A \cdot C$
  $A + (B \cdot C) = (A + B) \cdot (A + C)$

TEOREMI BASE

$A \cdot A' \overset{\text{NOT}}{=} \text{FALSO} = 0 = \varnothing$

$A + A \cdot B = A$ $\quad B = 0 \Rightarrow A, \quad B = 1 \Rightarrow A$

$A + A' \cdot B = A + B$

$A + A' = \text{VERO} = 1$

$A(A + B) = A$

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

$A + 1 = 1$

$A \cdot 1 = A$

$A \cdot 0 = 0$

$A + 0 = A$

TEOREMA DI DE MORGAN

$f(a, b, \dots, z; \overset{\text{OR}}{+}, \overset{\text{AND}}{\cdot}) = f'(a', b', \dots, z'; \cdot, +)$ $\quad f' \overset{\text{NOT}}{=} f!$

OVVERO (NEGANDO TUTTI I MEMBRI)

$f'(a, b, \dots, z; +, \cdot) = f(a', b', \dots, z'; \cdot, +)$

ES. $A + B = (A' \times B')'$

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

OPERATORI DI CAST

IN ALCUNI CASI, PUÓ ESSERE NECESSARIO CONVERTIRE ESPLICITAMENTE UN'ESPRESSIONE IN UNO SPECIFICO TIPO.

SINTASSI : (<tipo>) <espressione>

ES.
```
int a, b;
printf("A/B = %f \n", ((float) a)/b);
```

SIZEOF (SIZE = TAGLIA)

É POSSIBILE CALCOLARE IL NUMERO DI BYTE UTILIZZATO DAI TIPI DI DATO DI BASE UTILIZZANDO L'OPERATORE: sizeof.

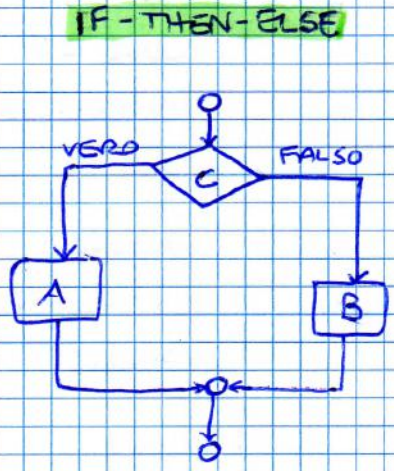
SINTASSI : sizeof (<tipo>)

RITORNA IL NUMERO DI BYTE OCCUPATO DA <tipo>

ES.
```
printf (" float %d byte \n", sizeof (float));
```
%d = 4 byte

IF - THEN - ELSE

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



ES.
```
int A, B, D;
if (A > B)
    D = A - B;
else
    D = B - A
printf ("%d \n", D);
```

WHILE

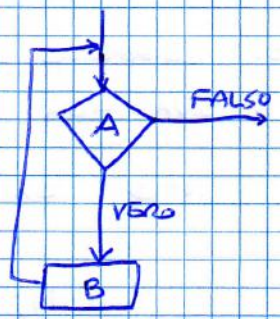
SINTASSI :

```
<inizializzazioni>
while (<condizione>)
    <blocco>
    <incremento>
```

ES. DATO N STAMPA LA SOMMA S DEI PRIMI N NUMERI INTERI.

```
int N, S, i;

i = 1
S = 0
scanf ("%d \n", &N);
   while (i <= N)
     { S = S + i;
       i ++;
     }
     printf ("Somma = %d \n", S);
```



WHILE

DO

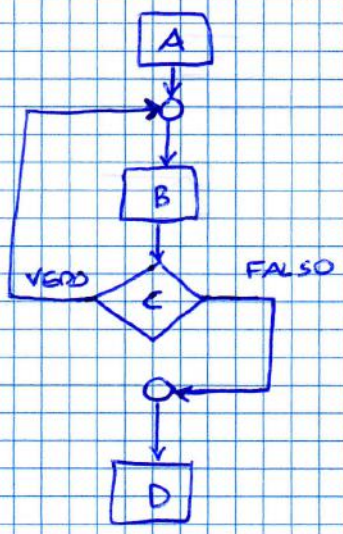
SINTASSI :

```
do
    <blocco>
while (<condizione>)
```

ES. LETTO n CONTROLLA CHE SIA POSITIVO, SE NO RIPETI LA LETTURA

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

DO-WHILE



VETTORI

SINGOLE VARIABILI (ELEMENTI) INDIVIDUATE DA UN INDICE, CORRISPONDENTE ALLA LORO POSIZIONE RISPETTO AL PRIMO ELEMENTO.

L'INDICE DEGLI ELEMENTI PARTE DA 0.

SINTASSI: <tipo> <nome vettore> [<dimensione>];

ES. int dato [10]

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

ESEMPI DI UTILIZZO:

```
TOT = TOT + dato[i];
dato[0] = 0;
printf("%d\n", dato[k]);
scanf("%d\n", &dato[k]);
```

I CICLI SONO PARTICOLARMENTE UTILI PER "SCANDIRE" UN VETTORE.

ES.

```
int data [10];
for (i=0; i<10; i++)
{ ...
}
```

LETTURA DEL VETTORE

```
for (i=0; i<N; i++)
{ printf("Elemento %d: ", i+1);
  scanf("%d\n", &v[i]);
}
```

STAMPA DEL VETTORE

```
for (i=0; i<N; i++)
{ printf("Elemento %d: ", i+1);
  printf("%d\n", v[i]);
}
```

STAMPA DEL VETTORE AL CONTRARIO

```
for (i=0; i<10; i++)
  scanf("%d", &dato[i]);

for (i=9; i>=0; i--)
  printf("%d\n", dato[i]);
```

FUNZIONI - PROTOTIPI

SINTASSI: <tipo risultato> <nome funzione> (<parametri formali>)

{

<istruzioni>

}

<TIPO RISULTATO> : void (SE LA FUNZIONE NON HA RISULTATO)
int
float
double

IN <istruzioni> DEVE COMPARIRE: ① return <valore>; SE NON void
② return; SE void

ES.

```
void swap (int a, int b);

main ()
{   int x, y;
    scanf(" %d %d", &x, &y);
    printf(" %d %d", x, y);   /* x e y non vengono modificati */

    swap (x, y);
    printf("%d %d", x, y);   /* vengono invertiti */
}

void swap (int a, int b)
{   int tmp;
    tmp = a;
    a = b;
    b = tmp;
    return;
}
```

MATRICI - VETTORI MULTIDIMENSIONALI

SINTASSI: `<nome vettore> [<dim1>] [<dim2>];`
(dim1 = RIGA, dim2 = COLONNA)

ES. `int v[3][2];`
- v[3] → RIGA
- [2] → COLONNA

ARGOMENTI SULLA LINEA DI COMANDO

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

- **argc**: NUMERO DI ARGOMENTI SPECIFICATI
  - ESISTE SEMPRE ALMENO UN ARGOMENTO (IL NOME DEL PROGRAMMA)

- **argv**: VETTORE DI STRINGHE
  - argv[0]: PRIMO ARGOMENTO
  - argv[i]: GENERICO ARGOMENTO
  - argv[argc-1]: ULTIMO ARGOMENTO

ES.

```
c:\> prog.exe 3 file.dat 3.2
```

argv[0] → "prog.exe \0"
argv[1] → "3 \0"
argv[2] → "file.dat \0"
argv[3] → "3.2 \0"

argc = 4

CICLO DI ELABORAZIONE

```
for (i=0; i<argc; i++)
{ELABORA argv[i] COME STRINGA
}
```

TIPI AGGREGATI - STRUTTURE

IN C È POSSIBILE DEFINIRE DATI COMPOSTI DA ELEMENTI ETEROGENEI DETTI **RECORD**, AGGREGANDOLI IN UNA SINGOLA VARIABILE, INDIVIDUATA DALLA KEYWORD **STRUCT**.

SINTASSI:

```
struct <identificatore>
{ campi [ <tipo> <nome campo>;
};
```

- UNA DEFINIZIONE DI struct EQUIVALE AD UNA DEFINIZIONE DI TIPO.
- SUCCESSIVAMENTE, UNA STRUTTURA PUÒ ESSERE USATA COME UN TIPO PER DICHIARARE VARIABILI.

```
ES.   struct complex
      { double re;
        double im;
      }
      struct complex num1, num2;
```

- UNA STRUTTURA PERMETTE DI ACCEDERE AI SINGOLI CAMPI TRAMITE L'OPERATORE '.', APPLICATO A VARIABILI DEL CORRISPONDENTE TIPO struct

```
<variabile>.<campo>
```

```
struct complex num1, num2;
num1.re = 0.33;
num1.im = -0,43943;
num2.re = -0,133;
num2.im = -0,49;
```

PUNTATORI

- VARIABILI CHE CONTENGONO INDIRIZZI DI MEMORIA.
- DEFINITI TRAMITE L'OPERATORE UNARIO '*' POSTO ACCANTO AL NOME DELLA VARIABILE

SINTASSI: <tipo> * <identificatore>

ES. int x; → int *px;
double y; → double *py;

- L'INDIRIZZO DI UNA VARIABILE (DA ASSEGNARE, PER ESEMPIO, AD UN PUNTATORE) È DETERMINABILE TRAMITE L'OPERATORE UNARIO '&' POSTO A FIANCO DELL'IDENTIFICATORE DELLA VARIABILE.

SINTASSI: & <identificatore>

ES.
```
int a, *ptr;
ptr = &a;
```

```
int x=5, *px;
px = &x;    /*px È L'INDIRIZZO DI x*/

printf("%d\n", x); → 5
printf("%d", px); → 10016
printf("%d", &x); → 10016
printf("%d", *px); → 5
```

- GLI OPERATORI * e & SONO UNO L'INVERSO DELL'ALTRO

* OPERATORE DI INDIREZIONE
OPERA SU UN INDIRIZZO
RITORNA IL VALORE CONTENUTO IN QUELL'INDIRIZZO

& OPERATORE DI INDIRIZZO
OPERA SU UNA VARIABILE
RITORNA L'INDIRIZZO DI UNA VARIABILE

SELECTION SORT

```
for ( i=0; i<n-1 ; i++)
{
    for ( J = i+1; J<n ; J++)
    {
        if (V[J] < V[i])
        {
            tmp = V[i];
            V[J] = V[i];
            V[i] = tmp;
        }
```

INSERTION SORT

```
for ( J = 1 ; J<n ; J++)
{
    key = V[J];
    for( i = J-1 ; i>=0 && V[i] >key ; i--)
    {
        V[i+1] = V[i]
    }
    V[i+1] = key;
}
```

SCRITTO IN DUE MODI DIVERSI MA E' LA STESSA COSA.

```
char str1 [9] = 'OLIMPICO'
char str1 [ ] = "OLIMPICO"
```

ESERCIZIO 2 — str1 OLIMPICO, str2 OIO, str3 IMPC

```
for ( i=0; str1[i] != '\0'; i++)
{
    for ( J = 0; (str2[J] != '\0') && (str1[i] != str2[J]); J++)
if (str2[J] == '\0')
{
str1 [K] = str1 [i];
  K = K+1;
}
str1 [J] = '\0';
return str1; }
```

TIPI AGGREGATI

SONO VETTORI

IN C È POSSIBILE DEFINIRE DATI COMPOSTI DA ELEMENTI ETEROGENEI (DETTI RECORD), AGGREGANDOLI IN UNA SINGOLA VARIABILE.

- INDIVIDUATA DALLA KEYWORD struct

- SINTASSI (DEFINIZIONE DI TIPO)

```
                                  ES. STUDENTE
struct <IDENTIFICATORE>
{   campi  →  <tipo> <nome campo>;
};
```

ES. ①

```
struct studente
{   char cognome [21];
    char nome [21];
    int matricola;
    float media;
}
```

CON struct studente HO CREATO UN NUOVO TIPO

ES.

```
struct complex
{   double re;
    double im;
}
```

② struct complex = NOME DEL NUOVO TIPO CREATO

" " num1, num2 SONO I NUMERI

③

```
num1.re = 0,30;      num1.im = 10;
num2.re = 0,5;
```

UNA STRUTTURA PERMETTE DI ACCEDERE AI SINGOLI CAMPI TRAMITE L'OPERATORE '.', APPLICATO A VARIABILI DEL CORRISPONDENTE TIPO struct

<variabile> . <campo>

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 37 di 40

typedef

① • typedef struct complex {
double re;
double im;
} compl;

↓ TUTTO CIÒ SI CHIAMA compl

||
V

• TUTTA LA STRUTTURA LA POSSO FARE AL POSTO DI QUELLA VISTA PRIMA struct complex { ... }

• QUINDI INVECE DI ② struct complex num1;
SCRIVO
compl num1;

QUINDI INVECE DI ③ int func (struct studente num1);
SCRIVO
int func (compl num1);

ES.
main () {
compl z_1, z_2, z_3;
$z_3.re = z_2.re + z_1.re$;
$z_3.im = z_2.im + z_1.im$;