

## Il linguaggio di programmazione C

Tipi di dati: CHAR – INT – FLOAT – DOUBLE

Operatori: +, -, \*, /, %, ==, !=, >, <, >=, <=, n++, ++n, n--, n--;

Modulo da il resto di ↑ una divisione tra interi

Nel C lo 0 indica FALSO, mentre un qualsiasi altro numero indica VERO

Operatori logici: || (o); && (e); ! (non);

Espressione condizionale: condizione ? espressione1 : espressione2

Se la condizione è vera si avrà come risultato il risultato di espressione1, se invece la condizione risulterà falsa si avrà come risultato il risultato di espressione2.

Assegnamenti compatti:

1.  $a = a + b \longrightarrow a += b$
2.  $a = a - b \longrightarrow a -= b$
3.  $a = a * b \longrightarrow a *= b$
4.  $a = a / b \longrightarrow a /= b$
5.  $a = a \% b \longrightarrow a \% = b$

Le istruzioni semplici sono separate dai ;

I blocchi sono racchiusi dalle {} e possono essere tra loro innestati.

Istruzioni di scelta semplice (IF)

< istruzione di scelta >

```
if (condizione) {istruzione}
else {istruzione}
```

→ se la condizione è vera si esegue il ramo if,  
se la condizione è falsa si esegue il ramo else

Istruzioni di scelta multipla (SWITCH)

< istruzione di selezione >

swith (espressione di selezione)

{case < 1 >: istruzione; break

case < 2 >: istruzione; break

case < 3 >: istruzione; break

default: istruzione; }

scanf ("%c", & scala)

swith (scala)

{case 'c': printf ("gradi celsius \ n"); break

case 'f': printf ("gradi faraday \ n"); break

default: printf ("ERRORE \ n"); }

Istruzioni Iterative:

WHILE:

< istruzione >

while (condizione) → se condizione è vera subito, istruzione 1 non viene mai eseguita  
{istruzione}

DO...WHILE: è uguale al while, solo che l'istruzione 1 viene eseguita almeno una volta

< istruzione >

do < istruzione1 > while (condizione)

FOR: istruzione1 viene eseguita subito dopo la condizione

< istruzione >

for (inizializzazione; condizione; espressione di modifica)

< istruzione1 >

Tipo dato ridefinito: typedef tipo di dato nome (es. typedef int Giordy)  
typedef enum {elenco} nome

Direttive al pre-processor: #include..... #define.....

Vettori: tipo nome [n°] (es. int Giordy [3], i vettori saranno: Giordy [0], Giordy [1], Giordy [2])

Funzioni:  
void char scala (char lettere)  
{if (c = lettera || f = lettara)  
printf("ok \n");  
else printf("ERRORE \n");}

Chiamata di funzioni:

1. Bisogna segnalare la funzione subito dopo le direttive al pre-processor (es. char scala (char););
  2. Quando si chiama la funzione bisogna inserire tra parentesi il valore di ingresso (es. c = scala (z));;
  3. Inserire lo sviluppo della funzione entro la fine del programma;
- NB:** ogni funzione può essere dichiarata più volte, ma definita una sola volta;

Se indico un vettore con v [ ], con la seguente simbologia indico: \*v ≡ v [0];  
\*(v +1) ≡ v [1];...  
\*(v + i) ≡ v [i];

Strutture: collezione finita di variabili non necessariamente dello stesso tipo, ognuna identificata da un nome;

Per definirla subito dopo il pre-processor: struct {char Nome[15], Cognome[15]; int Età;} Persona;

Poi all'interno della funzione bisogna dichiararla: Persona Alunno;

Infine per utilizzare i campi all'interno della struttura si usa il nome dichiarato e il nome definito separati da un punto: scanf ("%d", &Età.Alunno);

Input/Output Standard a caratteri: c = getchar() / putchar(c);

Input/Output Standard a stringhe: gets(s) / puts(s);

Input/Output Formattato: scanf("%...", &...) / printf ("...", ...);

**NB:** L'operatore & non va usato per le stringhe, mentre va sempre usato per i valori scalari;  
Formati più comuni di stringhe di formato:

%c	Carattere Singolo	%d	Interi
%s	Stringa di Caratteri	%f	Float
%lf	Double	%Lf	Long Double

Gestione delle variabili: **static** int...: rende globale e nascosta fuori dal blocco una variabile;

La funzione malloc ( ): chiede al sistema operativo di allocare un'area di memoria grande quanto da noi richiesto, e ci restituisce l'indirizzo dell'area di memoria allocata, se l'allocazione non è stata possibile, ci restituisce un puntatore NULL (0) che non è mai un puntatore valido. In numero racchiuso tra le parentesi tonde che segue malloc indica il numero di byte dell'area di memoria richiesta, per questo motivo è opportuno non inserire mai valori numerici direttamente, ma utilizzare l'operatore **sizeof ( )**: int p; p = (int\*) malloc (5\*sizeof (int)); [allocca spazio per 5 interi].

Ora l'area di memoria è utilizzabile tramite la notazione \*p o p[i]. alla fine l'area allocata dovrà essere esplicitamente liberata tramite la funzione **free (p)**.

**NB** per utilizzare l'allocazione di memoria statica è necessario inserire nelle direttive al pre-processor l'inserimento della libreria **stdlib.h**.

Lista: sequenza, multi-insieme finito e ordinato di elementi di un certo tipo, dove un certo elemento può anche comparire più volte. Un vettore di valori, è una lista sequenziale.

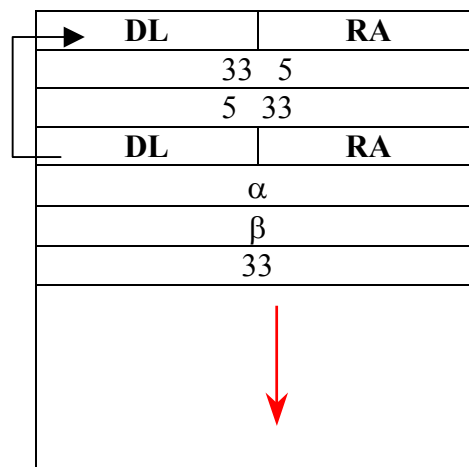
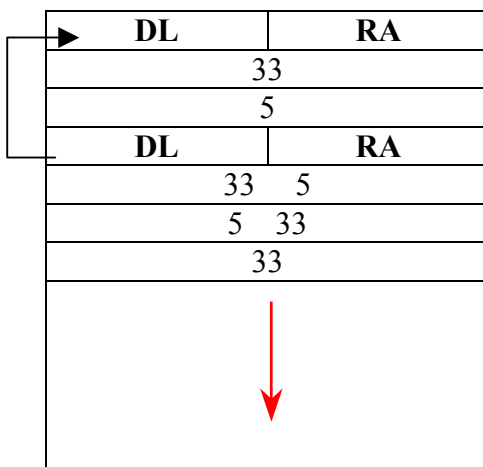
Una struttura si dice **Iterativa** se viene sviluppata con cicli while, for, e do ... while, mentre si dice **Ricorsiva** se una funzione richiama se stessa più volte per essere risolta, cioè un servitore è cliente di se stesso.

Record di Attivazione: rappresenta il "Mondo" della funzione e al suo interno vale la regola LIFO, ossia, last in first out. La sua dimensione varia a seconda della funzione e ad ogni attivazione di funzione viene creato un nuovo record di attivazione. Funzioni che chiamano altre funzioni danno luogo ad una sequenza di record di attivazione allocati secondo l'ordine di chiamata e deallocati in ordine contrario (LIFO).

Esempi

```
main () {
    int X = 33, Y = 5;
    scambia (X, Y) ; }
void scambia (int A, int B) {
    int T;
    if (A > B) { T=A; A=B; B=T} }
```

```
main () {
    int X = 33, Y = 5;
    scambia (X, Y) ; }
void scambia (int &A, int &B) {
    int T;
    if (A > B) { T=A; A=B; B=T} }
```



### Gestione di File in C:

Per aprire un file in C, stdio.h dichiara la funzione fopen():

FILE\* fopen (char fname [], char modo []) questa funzione apre il file fname nel modo specificato e restituisce un puntatore a FILE. La stringa modo specifica come aprire il file:

r (lettura), w (scrittura), a (aggiunta) eventualmente seguiti da: t (testo), b (binario).

Per operare su file binari vi sono 2 apposite funzioni, fread () ed fwrite ().

Apertura di un file di testo in lettura: FILE\* f; f = fopen ("nomefile.txt", "r");

Apertura di un file di testo in scrittura: FILE\* f; f = fopen ("nomefile.txt", "w");

Apertura di un file di testo in aggiunta: FILE\* f; f = fopen ("nomefile.txt", "a");

Apertura di un file di testo in lettura con aggiunta: FILE\* f; f = fopen ("nomefile.txt", "r+");

Chiusura di un file: fclose (f);

**Esempio:** archiviare ciò che viene inserito da console su un file di testo.

```
#include <stdio.h>
main ()
{
FILE*fp;
fp = fopen ("C:/Giordy.txt","w");
if (fp==NULL)
printf ("Errore di apertura file\n");
else {
int c;
while ((c=getchar()) != EOF) fputc (c, fp);
fclose (fp);
}
}
```

Per operare su file binari vi sono 2 apposite funzioni, fread () ed fwrite ().

int fwrite (addr, int dim, int n, FILE\*f); la funzione scrive sul file n elementi ognuno grande dim byte, gli elementi da scrivere vengono prelevati in memoria a partire dall'indirizzo addr.

int fread (addr, int dim, int n, FILE\*f); la funzione legge sul file n elementi ognuno grande dim byte, gli elementi da leggere vengono posti in memoria a partire dall'indirizzo addr.

**Esempio:** archiviare su un file binario il contenuto di un vettore di interi.

```
#include <stdio.h>
main ()
{
FILE*fp;
int vet[10] = {1,2,3,4,5,6,7,8,9,10};
if ((fp = fopen ("C:/Giordy.dat","wb"))==NULL){
fprintf (stderr, "Errore di apertura file\n");
}
fwrite (vet, sizeof(int), 10, fp);
fclose (fp);
}
```