

ALLOCAZIONE DINAMICA, STRUTTURE DATI E FILES

- PUNTATORI

un puntatore e' una entita' software che indica una cella di memoria.

(OPERAZIONI) ESEMPLI:

#Definiamo un puntatore

```
int *ptr;
```

```
ptr = ptr + 1;
```

questo valore sara' incrementato di 4 bytes (DIMENSIONE DI UN INTERO)

#Definiamo un puntatore

```
char *ptr;
```

```
ptr = ptr + 1;
```

questo valore sara' incrementato di 1 byte (DIMENSIONE DI UN char)

- PUNTATORI A GU ARRAY

Quali puntatori in array puntano sempre il suo primo elemento.

ALLOCAZIONE DINAMICA

I dati statici sono allocati nello stack

(si puo' allocare memoria nello stack tramite il comando "alloc()")

I dati dinamici si allocano invece nell'heap.

tramite comandi calloc o malloc

→ DIFFERENZE:

- calloc alloca memoria nell'heap in modo che a 0 le celle di memoria, calloc no.

- In C non esiste il garbage collector quindi

il programmatore deve liberare la memoria dell'heap tramite il comando `free(p)`.
È importante pulire la memoria per avere delle buone prestazioni ma non bisogna pulire due volte la stessa area, questo ha un comportamento non prevedibile. (INDEFINITO)

- Quando allochiamo memoria dinamica il puntatore ritornato è un puntatore NULL, è quindi sempre richiesto il casting.

ESEMPIO:

```
char *strPtr = NULL;  
const int SIZE_OF_ARRAY = 30;  
strPtr=(char *) calloc(SIZE_OF_ARRAY, sizeof(char));
```

REALLOC:

- serve per allargare un'area di memoria dinamica
- IN CASO DI ERRORE VIENE RITORNATO UN PUNTATORE NULLO
- il puntatore in input potrebbe essere diverso dal puntatore in output, perché se la funzione non riesce ad allargare l'area puntata dal puntatore in input allora cambia area restituendo un puntatore diverso.



SURICATI



STRUTTURE DATI

1. VARIABILI STRUTTURA

ESEMPIO:

```
STRUCT { x, y: nomi' membri struttura  
        double x;  
        double y; } point2D // NOME VARIABILE
```

QUESTO TIPO DI STRUTTURA RISULTA POCO PORTATILE IN QUANTO NON PUO' ESSERE RIEMPIUTA PER COSTRUIRE NUOVI DATI MA TUTTI I DATI SONO ESPLICITATI ALLA CREAZIONE DELLA STRUTTURA.

2. TAGGED STRUCTURE

RISOLVE IL PROBLEMA PRECEDENTEMENTE ESPOSTO, PUO' ESSERE RIEMPIUTA LA STRUTTURA CREATA PER CREARE NUOVI DATI

ESEMPIO:

```
STRUCT POINT2D { x, y: nomi' membri struttura  
                double x;  
                double y; }
```

```
STRUCT POINT2D PUNTO;
```

3. TYPEDEF STRUCTURE

simile alle precedenti, ma per essere doti, viene utilizzata la sintassi dei tipi.

ESEMPIO:

```
typedef PUNTO { x, y: nomi membri struttura  
double x;  
double y; }
```

```
// POINT2D PUNTO; PUNTO2;  
(MANCA "STRUCT")
```

CONVENIENZA:

- typedef più comodo in progetti grandi perché meno verbose e più pulite.
- Tagged Structure per avere purezza (non contenere puntatori e re-sterne), chiarezza esplicita.
(DIPENDE DALLA SITUAZIONE.)


FILE

- I FILE possono essere di vario tipo
 - TESTO
 - BINARI
 - BUFFER (File temporanei che contengono informazioni in transito tra memoria PRIMARIA e SECONDARIA)

FILE DI TESTO (COME GESTIRLI)

- tutte le funzioni per la loro gestione sono in stdio.h
- ogni riga termina con '\n'
- il file termina con EOF

La struttura canonica delle operazioni eseguite su un file è:

1. DICHIARAZIONE FILE POINTER (`*file`) [`stdio.h`]
2. APERTURA (COLLEGAMENTO CON FILE POINTER)
3. OPERAZIONI
4. CHIUSURA  IMPORTANTISSIMA.

Ogni volta che vengono completate le operazioni su file vengono automaticamente chiusi, tuttavia è buona norma chiudere i file per evitare malfunzionamenti e/o eventi inaspettati.

- COMANDI

`fscanf`

PROTOTIPO:

```
int fscanf(FILE *stream, const char *format, ...);
```

DESCRIZIONE:

Legge dati formattati da un file.

PARAMETRI:

- `FILE *stream`: PUNTATORE AL FILE DOVE LEGGERE.
- `const char *format`: COME INTERPRETARE I DATI LETTI
- `...`: PUNTATORI AI BUFFER DOVE INSERIRE I DATI LETTI.

RITORNO:

- un intero se non viene raggiunto EOF altrimenti EOF.

'fprintf'

PROTOTIPO:

```
int fprintf(FILE *stream, const char *format, ...)
```

DESCRIZIONE:

Printa sul file *stream i dati passati come arguments

PARAMETRI:

- FILE *stream : FILE DOVE SCRIVERE
- const char *format : FORMATO VALORI DA SCRIVERE
- ... : VALORI DA SCRIVERE

RITORNO:

- Numero caratteri scritti, se errore → negativo.

'fgetc'

PROTOTIPO:

```
char *fgetc(char *s, int size, FILE *stream)
```

DESCRIZIONE:

legge una linea di testo fino a size - 1 caratteri e lo memorizza nella stringa s

RITORNO:

Puntatore a s, Null in caso di ERRORE.

'fputs'

PROTOTIPO:

```
int fputs(const char *s, FILE *stream)
```

DESCRIZIONE:

Scrivere la stringa s sul file

RITORNO:

- numero di caratteri scritti, EOF in caso di ERRORE

`'feof'`

`int feof(FILE *stream)`

DESCRIZIONE:

ritorna valore se è stata raggiunta la fine
altrimenti ritorna 0