

STATICO → Tempo di compilazione

DINAMICO → dim non note e tempo di comp.

22104122




**UNIVERSITÀ DEGLI STUDI  
DEL SANNIO** Benevento  
**DING**  
DIPARTIMENTO DI INGEGNERIA

**CORSO DI "PROGRAMMAZIONE I"**

Prof. Franco FRATTOLILLO  
Dipartimento di Ingegneria  
Università degli Studi del Sannio


Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio    Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT    1



**Funzioni di allocazione dinamica**

- Una delle maggiori limitazioni delle strutture dati statiche, come gli array, è la necessità di specificarne la dimensione in sede di dichiarazione
  - A tempo di compilazione
- Problemi possono nascere quando non si conosce la quantità precisa dei dati da memorizzare durante l'esecuzione
  - A run-time
- Una possibile soluzione a questo problema è l'uso delle funzioni di allocazione dinamica della memoria

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio    Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT    2




**La funzione malloc**

*memory allocation*

- Sintassi  
#include <malloc.h> ←  
void\* malloc(unsigned size); *puntatore non caratterizzato generico*
- Funzionalità  
• allocazione di un blocco di memoria dell'ampiezza specificata
- Valori di ritorno  
• puntatore al blocco allocato *memorie in byte*  
• se NULL, l'allocazione è fallita
- Esempio:  
char \*stringa;  
stringa = (char \*) malloc(20 \* sizeof(char)); *20 caratteri*

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio    Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT    3



**La funzione calloc**

- Sintassi  
#include <malloc.h>  
void\* calloc(unsigned num, unsigned size);
- Funzionalità  
• allocazione di un blocco di num elementi, ciascuno di ampiezza size, con inizializzazione a 0 della memoria allocata
- Valori di ritorno  
• puntatore al primo elemento  
• se NULL, l'allocazione è fallita
- Esempio:  
char \*stringa;  
stringa = (char \*) calloc(20, sizeof(char));

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio    Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT    4

operazione di  
CAST

&gt; = &lt;

## ridimensionare

### La funzione realloc

- Sintassi
 

```
#include <malloc.h>
void* realloc(void* ptr, unsigned newsize);
```
- Funzionalità
  - ridimensionamento di un blocco precedentemente allocato
- Valori di ritorno
  - puntatore al primo elemento
  - se NULL, l'allocazione è fallita
- Esempio:
 

```
char *ptr;
ptr = (char *) malloc(4*sizeof(char));
ptr = (char *) realloc(ptr, 8*sizeof(char));
```

## deallocazione esplicita

### La funzione free

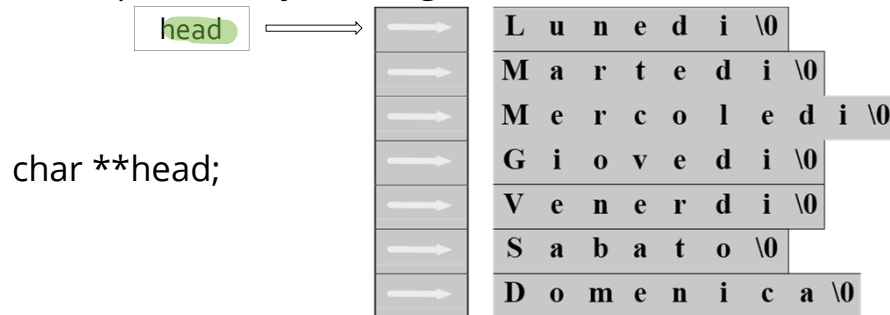
- Sintassi
 

```
#include <malloc.h>
int free(void* ptr);
```
- Funzionalità
  - rilascia la memoria precedentemente allocata con malloc, calloc, realloc
- Valori di ritorno
  - 1 operazione completata
  - 0 errore
- Esempio:
 

```
char *ptr;
ptr = (char *) malloc(4*sizeof(char));
free(ptr);
```

### Esempio di array dinamico

- Esempio di array di stringhe



### Esempio di codice

```

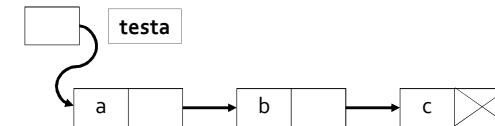
const int N = 300;    /* legge e salva N stringhe inserite da tastiera */
void main() {
    char **head, buffer[N];
    int n, i;
    printf("Quante stringhe vuoi inserire? : "); scanf("%d", &n);
    head = (char**) malloc(sizeof(char*)*n);
    for(i=0; i < n; i++) {
        printf("Inserisci la stringa: "); scanf("%s", buffer);
        *(head+i) = (char*) malloc(sizeof(char)*(strlen(buffer)+1));
        strcpy(*(head+i), buffer);
    }
    printf("\n");
    for(i=0; i < n; i++) printf("%s\n", *(head+i));
}
  
```

## Lista

- Una lista è una struttura dati dinamiche che implementa una sequenza di elementi di un determinato tipo
- Una sequenza è una collezione ordinata di elementi, ovvero si può sempre individuare il primo elemento, il secondo, etc...
- Gli elementi sono memorizzati associando a ciascuno di essi l'informazione (riferimento) che permette di individuare la locazione in cui è inserito l'elemento successivo
  - struttura dati linkata

## Notazione

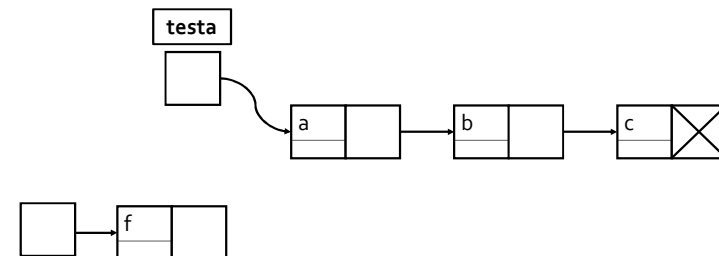
- Elementi della lista sono detti nodi ed i riferimenti agli elementi successivi sono rappresentati come archi

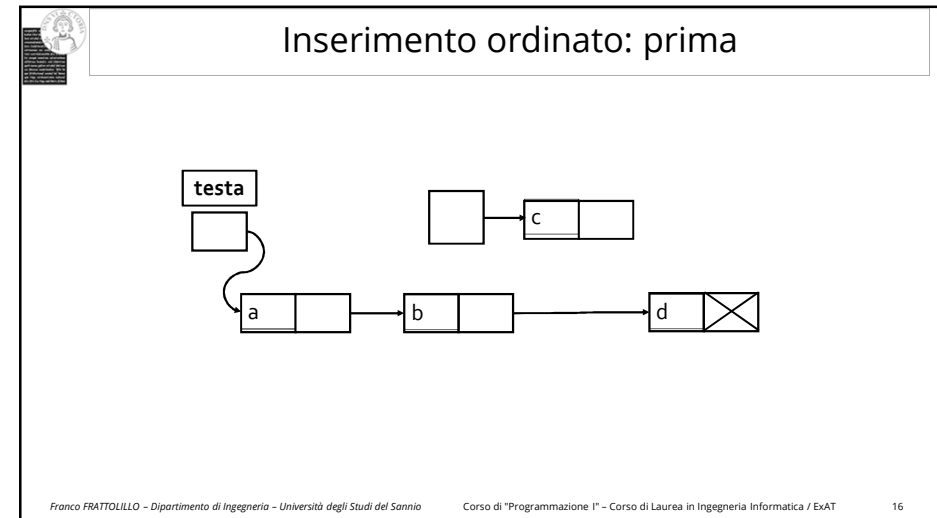
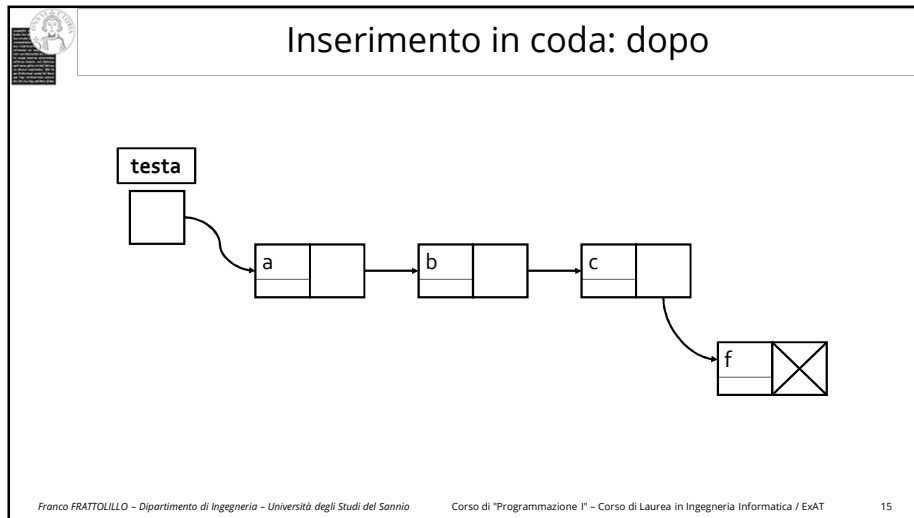
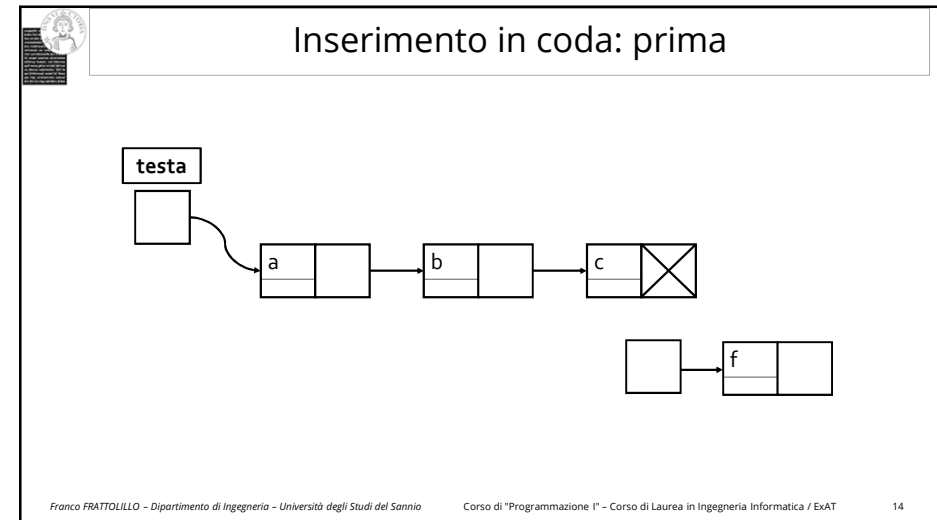
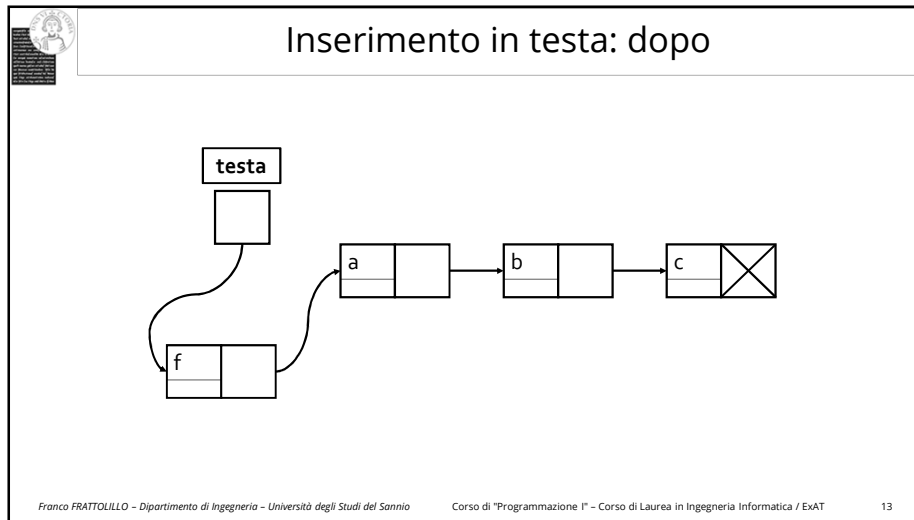


## Operazioni

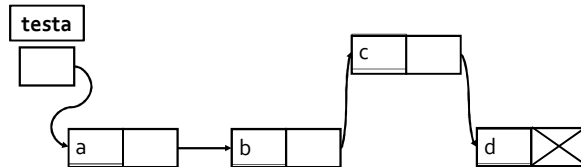
- Inserimento di un elemento in testa, in coda o ordinato
- Eliminazione di un elemento
- Ricerca di un elemento

## Inserimento in testa: prima

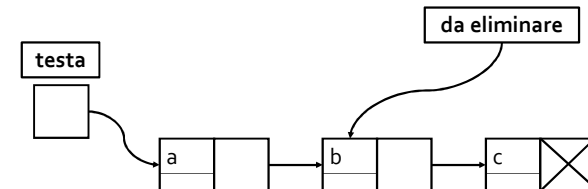




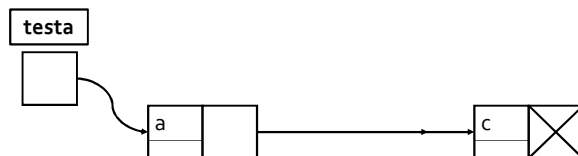
### Inserimento ordinato: dopo



### Eliminazione: prima



### Eliminazione: dopo




### Esempio di lista di stringhe

- Elemento:
 

```
struct stringa {
    char* str;
    struct stringa* next;
};
```
- La testa:
 

```
struct stringa* head;
```
- Allocazione di un nodo:
 

```
struct stringa* p;
p = (struct stringa *) malloc(sizeof(struct stringa));
```



## Inserzione in testa

```


void insertT(char* buf) {
    struct stringa* p;

    p = (struct stringa*) malloc (sizeof(struct stringa));
    p->str = (char*) malloc((strlen(buf)+1)*sizeof(char));
    strcpy(p->str, buf);

    p->next = head;
    head = p;
}

```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio    Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT    21



## Ricerca

```


struct stringa* search(char * buf) {
    struct stringa* curr = head;

    while(curr && strcmp(curr->str, buf))
        curr = curr->next;

    return curr;
}

```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio    Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT    22



## Visita con stampa


```

void stampa( ) {
    struct stringa* curr = head;

    while(curr) {
        printf("stringa: %s\n", curr->str);
        curr = curr->next;
    }
    printf("\n");
}

```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio    Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT    23



## Inserzione in coda

```

void insertC(char* buf) {
    struct stringa* p;    struct stringa* curr = head;
    p = (struct stringa*) malloc(sizeof(struct stringa));
    p->str = (char*) malloc((strlen(buf)+1)*sizeof(char));
    strcpy(p->str, buf);
    p->next=NULL;
    if (!head)
        head = p;
    else {
        while ((curr) && (curr->next)) curr = curr->next;
        curr->next = p;
    }
}

```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio    Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT    24

## Inserzione ordinata

```

void insertO(char* buf) {
    struct stringa* p;    struct stringa* curr;    struct stringa* prev;
    p = (struct stringa*) malloc(sizeof(struct stringa));
    p->str = (char*) malloc((strlen(buf)+1)*sizeof(char));
    strcpy(p->str, buf);
    p->next=NULL;
    curr=head;    prev=NULL;
    if (!head || (strcmp(buf, curr->str) < 0) ) {
        p->next=head;    head = p;
    }
    else {
        while (curr && (strcmp(curr->str, buf) < 0) ) {
            prev=curr;    curr = curr->next;
        }
        p->next=curr;    prev->next = p;
    }
}

```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio      Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT      25

## Eliminazione

```

int elimina(char *buf) {
    struct stringa* prev=NULL;    struct stringa* curr = head;
    if (!head) return 0;
    if (!strcmp(curr->str, buf)) {
        head = curr->next;    free(curr->str);    free(curr);    return 1;
    }
    else {
        while (curr && strcmp(curr->str, buf)) {
            prev = curr;    curr = curr->next;
        }
        if (!curr)
            return 0;
        else {
            prev->next = curr->next;    free(curr->str);    free(curr);    return 1;
        }
    }
}

```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio      Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT      26

## File ...

- I file sono sequenze di byte su cui è possibile effettuare operazioni di lettura, scrittura e aggiornamento
- Sono strutture non volatili, in quanto allocati in memoria secondaria
- L'uso dei file si realizza mediante un insieme di funzioni di libreria
- L'accesso ai file è per default sequenziale, anche se è possibile effettuare un accesso di tipo casuale attraverso opportune funzioni

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio      Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT      27

## File

- Un file è riferito nei programmi tramite un "file pointer"  
FILE \*fp;
  - FILE è un tipo predefinito contenuto nella libreria stdio.h
- All'atto dell'apertura del file, il file pointer acquisisce un valore e si stabilisce il collegamento con il file presente in memoria secondaria
  - da questo momento, il puntatore al file punta ad una struttura che contiene informazioni sul file da gestire, quali l'indirizzo del buffer associato, la posizione corrente nel buffer, la modalità di apertura, ecc.

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio      Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT      28

## Funzioni I/O standard per i file

- Le funzioni basilari per controllare l'I/O da file sono:
  - fopen()      fclose()
  - getc()      putc()
  - fgets()      fputs()
  - fprintf()      fscanf()
  - fseek()      ferror()
  - fwrite()      fread()

## fopen

- La prima operazione da compiere quando si usa un file consiste nella sua apertura:  
 FILE \*fopen(char \*nomefile, char \*modo);
  - la funzione restituisce un puntatore ad un oggetto di tipo FILE, la cui struttura è definita in stdio.h; NULL in caso di problemi
  - il primo parametro rappresenta il nome del file, il secondo la modalità di apertura del file
- Le modalità di apertura previste sono:
  - "r" per file aperto in sola lettura
  - "w" per file aperto in scrittura; se il file esiste se ne perde il contenuto
  - "a" per file aperto in scrittura, per l'aggiunta di nuovi dati
- Alcuni sistemi distinguono tra file binari e testuali
  - nel caso di uso di file binari occorre aggiungere alle modalità precedenti il suffisso "b"

## Esempio

```
#include <stdio.h>
FILE *ingresso, *uscita;
ingresso = fopen("alfa", "r"); /* lettura */
uscita = fopen("beta", "w"); /* scrittura */
```


- fopen() restituisce NULL, se il file non può essere aperto, o comunque in qualsiasi situazione d'errore
  - se il file "alfa" non esiste, si verifica una situazione d'errore
- Se il file "beta" non esiste, la funzione ne crea uno nuovo, altrimenti cancella il suo contenuto ed il puntatore al buffer si riposiziona all'inizio

## fclose

- La funzione duale di fopen() è  
 int fclose(FILE \*file);
- che chiude un file precedentemente aperto
  - in caso di successo fclose() restituisce il valore 0, altrimenti restituisce il valore EOF

```
#include <stdio.h>
FILE *input;
input = fopen("maggio.doc", "wb");
...
fclose(input);
```






## getc e putc

- Sono definite in `stdio.h` e servono per trasferire caratteri da o verso un file aperto con `fopen()`

```
#include <stdio.h>
int getc(FILE *filepunt);
int putc(char c, FILE *filepunt);
```

- La funzione `getc()` legge un carattere dal file puntato da `filepunt`
- La funzione `putc()` trasferisce un carattere `c` nel file puntato da `filepunt`


Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio    Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT    33



## Esempio

```
#include <stdio.h>
main() {
    FILE *file;
    char c;
    file = fopen("temp", "w");
    while ((c = getchar()) != EOF) putc(c, file);
    fclose(file);
    file = fopen("temp", "r");
    while ((c = getc(file)) != EOF) putchar(c);
    fclose(file);
}
```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio    Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT    34




## fgets e fputs

- Sono analoghe alla `getc` e `putc`, ma operano sulle stringhe:  
`char* fgets(char* str, int len, FILE* fp);`  
`int fputs(char *str, FILE* fp);`

```
#include <stdio.h>
main() {
    FILE *file; char stringa[80];
    file = fopen("temp", "w");
    while (gets(stringa) != NULL) fputs(stringa, file);
    fclose(file);
    file = fopen("temp", "r");
    while(fgets(stringa, 20, file) != NULL) puts(stringa);
}
```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio    Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT    35




## fprintf

- `fprintf()` si comporta esattamente come la `printf()`, utilizzando il file invece dello standard output
- `fprintf()` restituisce il numero di caratteri memorizzati sul file

```
FILE *file;
char *nome = "Marco";
file = fopen("miofile.txt", "w");
fprintf(file, "Il mio nome è %s", nome);
```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio    Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT    36




## fscanf

- `fscanf()` si comporta come la `scanf()` relativamente al file
 

```
int fscanf(FILE *fp, char *form, arg0,...);
```
- Il valore restituito da `fscanf()` è il numero di elementi, letti da file, che hanno soddisfatto le specifiche imposte dalla stringa formato
- In caso di fine file, `fscanf()` restituisce EOF


Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio    Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT    37



## Esempio

```
#include <stdio.h>
main() {
    FILE *file;
    int i;
    file = fopen("temp", "w");
    for(i=0; i < 985; i+=123) fprintf(file, "%d ", i);
    fclose(file);
    file = fopen("temp", "r");
    while(fscanf(file, "%d ", &i) != EOF) printf("%d\n", i);
}
```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio    Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT    38




## ferror

- `ferror()` consente di controllare lo stato delle operazioni di accesso ai file
 

```
int ferror(FILE *filepunt);
```
- In situazioni normali `ferror()` restituisce il valore 0
- Quando un'operazione di lettura ha raggiunto la fine del file, `ferror()` restituisce un valore diverso da 0 se una qualche operazione di I/O ha incontrato una condizione di errore

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio    Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT    39




## fseek

- `fseek()` implementa l'accesso casuale ai file, cioè la possibilità di spostarsi in una qualsiasi posizione di un file
  - restituisce un valore diverso da 0 se lo spiazamento comporta l'uscita dal file

```
int fseek(FILE *fp, int spz, int modo);
```
- Il parametro `spz` consente di posizionare la successiva operazione di lettura o scrittura in base al valore di *modo*
  - se *modo* vale 0, `spz` indica di quanti byte ci si deve spostare dall'inizio del file
  - se *modo* vale 2, `spz` indica a quanti byte dalla fine del file ci si deve posizionare
  - se *modo* vale 1, `spz` indica di quanti byte ci si deve spostare dalla posizione attuale nel file


Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio    Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT    40



Esempio

```
#include <stdio.h>
main( ) {
    FILE *file;
    file = fopen("esempio.dat", "r");
    fseek(file, 0L, 2);    /* fine file */
    do {
        putchar(getc(file));
    } while( !fseek(file, -2L, 1) );    /* arretra di due caratteri */
}
```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio
Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT
41




fread e fwrite

- `fread()` e `fwrite()` sono pensate per gestire informazioni costituite sia da singoli byte che da gruppi di byte (carattere, intero, struttura)
 

```
int fread(void *buf, int len, int cont, FILE *fp);
int fwrite(void *buf, int len, int cont, FILE *fp);
```
- Entrambe le funzioni utilizzano gli stessi parametri
  - *buf* è la zona di memoria dove sono contenuti i dati
  - *len* è la dimensione, in numero di byte, degli oggetti da leggere o da scrivere
  - *cont* è il numero degli oggetti, tutti di lunghezza len, da leggere o da scrivere
  - *fp* è il valore restituito da `fopen()`


Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio
Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT
42



Esempio

```
#include <stdio.h>
main( ) {
    int numero;
    FILE *file;
    file = fopen("esempio", "rb");
    fread(&numero, sizeof(int), 1, file);
    printf("Il numero letto è %d", numero);
}
```

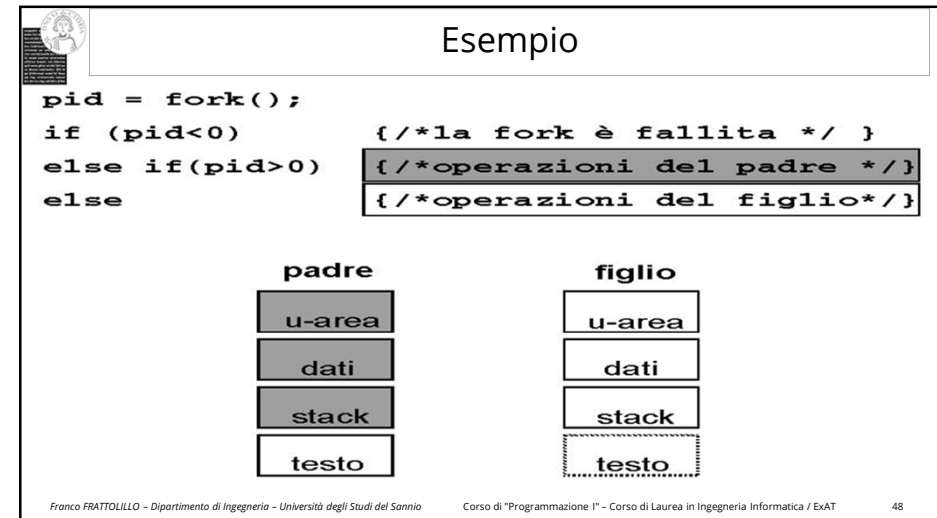
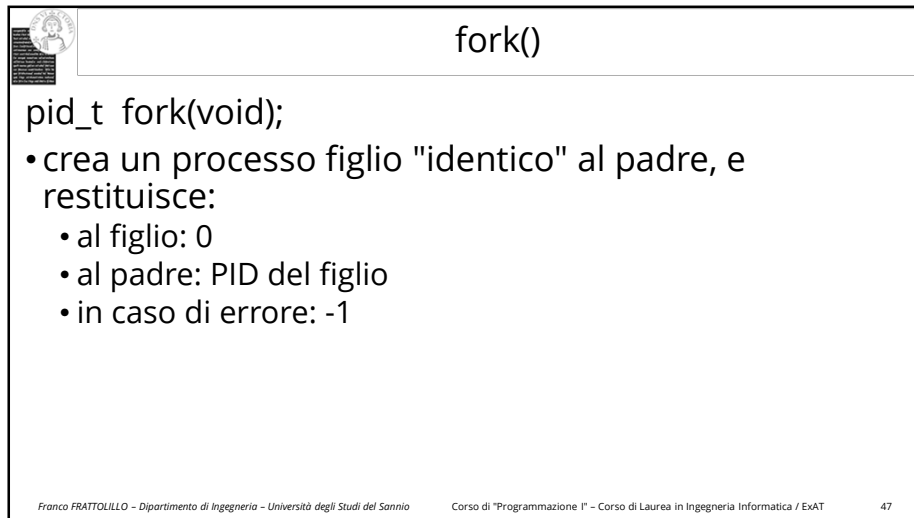
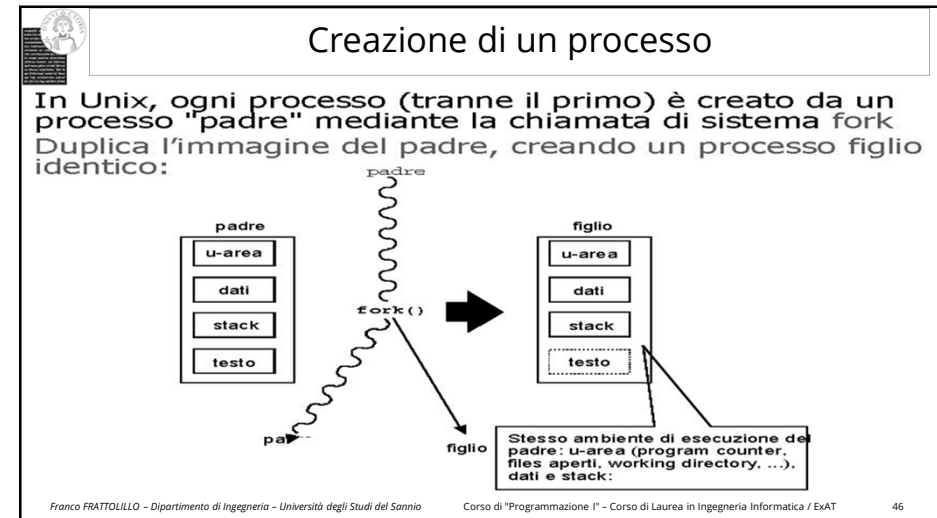
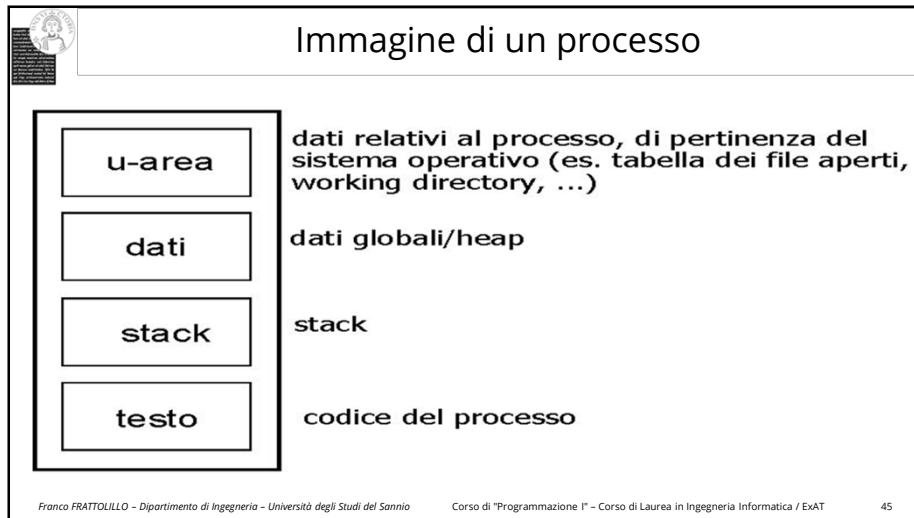
Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio
Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT
43



Esempio

```
#include <stdio.h>
main( ) {
    int numero;
    FILE *file;
    numero = 100;
    file = fopen("esempio", "wb");
    fwrite(&numero, sizeof(int), 1, file);
    printf("Il numero scritto è %d", numero);
}
```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio
Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT
44



## Esempio

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
main()
{
    int pid=fork();
    if(pid<0)
    {
        perror("main");
        exit(2);
    }
    else if(pid>0)
        while(1)
            printf("Sono il padre\n");
    else
        while(1)
            printf("Sono il figlio\n");
}
```

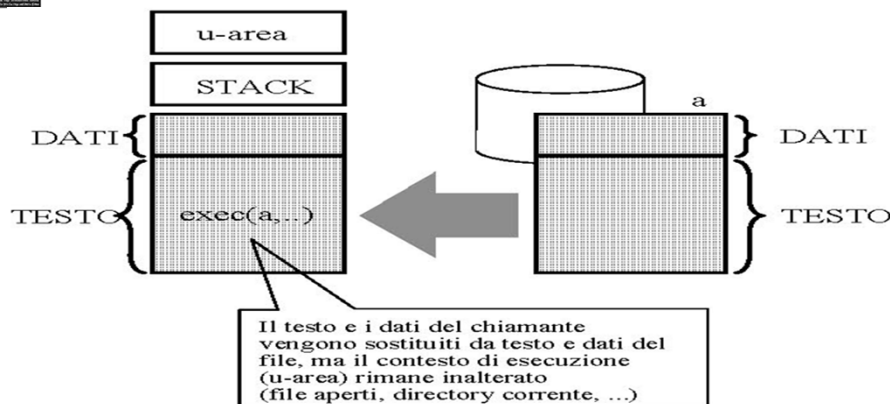
Sono il padre  
 Sono il padre  
 Sono il figlio  
 Sono il figlio  
 Sono il padre  
 Sono il padre  
 Sono il padre  
 Sono il figlio  
 Sono il figlio  
 Sono il padre  
 Sono il padre  
 Sono il padre  
 Sono il figlio  
 Sono il figlio  
 Sono il padre  
 Sono il padre  
 Sono il padre  
 Sono il figlio

## exec()

int exec(pathname, argomenti)

- Rappresenta una "famiglia" di funzioni
  - exec più una combinazione delle lettere "l", "e", "v", "p"
  - invoca un programma dall'interno di un programma
  - sostituisce l'immagine del chiamante con il file eseguibile indicato con pathname e lo manda in esecuzione passandogli gli argomenti

## Effetto della exec



## La famiglia exec ...

```
int execl(const char *path, const char *arg0, ...);
int execlp(const char *file, const char *arg0, ...);
int execlen(const char *path, const char *arg0, ..., char * const
    envp[]);
int execlv(const char *path, char *const argv[]);
int execlvp(const char *file, char *const argv[]);
int execlve(const char *filename, char *const argv[], char
    *const envp[]);
```

## Gli argomenti della exec

- path specifica il pathname del file da eseguire come processo figlio
- Gli argomenti `arg0, ..., argN` sono una lista di puntatori agli argomenti da passare al processo figlio
- `argv` è un array di puntatori agli argomenti
- `envp` è un array di puntatori alla configurazione dell'ambiente attuale

## Esempio execl

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

main()
{
    int pid=fork();
    if(pid<0)
    {
        perror("main");
        exit(2);
    }
    else if(pid>0)
        /*padre*/
        while(1)
            printf("Sono il padre\n");
    else
        /*figlio*/
        execl("/usr/bin/ls", "-l", 0);
}
```

## Considerazioni

- Il padre stampa ripetutamente un messaggio (all'infinito)
- Il figlio stampa il contenuto della directory corrente e poi muore
- Padre e figlio sono eseguiti concorrentemente
  - l'output sarà "interleaved"...
- Per eseguire `ls` occorre specificarne il percorso `"/usr/bin/ls"`, altrimenti il binario non viene trovato
  - il primo parametro deve essere il nome del programma
  - la lista di parametri deve terminare con uno 0

## L'output

```
Sono il padre
Sono il padre
Sono il padre
Sono il padre
Sono i#cp.c#
a
arc
archive.c
archive.c~
...
asasu
at
ausu
ciao
copia.c
-
Sono il padre
Sono il padre
Sono il padre
Sono il padre
```

## execvp()

```
int execvp(const char *path, char *const argv[]);
```

- Gli argomenti del comando sono passati in un vettore
- Il vettore deve contenere:
  - in prima posizione, il comando da eseguire
  - in ultima posizione, un NULL
  - ... esattamente il formato di argv, poiché il vettore passato diventa l'argv del programma eseguito

## Esempio execvp

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
main(int argc, char **argv)
{
    char *vec[]={ "ls", "-l", NULL };
    int pid=fork();
    if(pid<0)
    {
        perror("main");
        exit(2);
    }
    else if(pid>0)
        /*padre*/
        printf("Sono il padre\n");
    else
        /*figlio*/
        if(execvp("/usr/bin/ls",vec)==-1)
            perror("main");
}
```

## execve()

```
int execve(const char *filename, char *const argv[], char *const envp[]);
```

- Passa variabili d'ambiente della shell al processo chiamato

```
#include <unistd.h>

main(){
    char *cmd[] = { "ls", "-l", (char *)0 };
    char *env[] = { "HOME=/home/max", (char *)0 };
    execve ("/usr/bin/ls", cmd, env);
}
```

## Sospensioni

```
pid_t wait(int *statloc);
```

- Sospende il processo chiamante, fino a che uno dei suoi figli non termina
  - restituisce il PID del figlio terminato, o -1 se non ci sono figli
  - assegna a *statloc* l'exit status del figlio

```
pid_t waitpid(pid_t pid, int *statloc, int options);
```

- Permette di specificare di quale figlio si attende la terminazione

## Esempio wait

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
main()
{
    int stat;
    int pid=fork();
    if(pid<0)
    {
        perror("main");
        exit(2);
    }
    else if(pid>0)
    { /*padre*/
        wait(&stat);
        printf("ls concluso con esito %u\n",stat);
    }
    else /*figlio*/
    { if(execvp("ls", "-l", 0)==-1)
        perror("main");
    }
}
```

## exit()

void exit(int status);

- Termina il processo chiamante e rende disponibile il valore di status al processo padre, che lo otterrà tramite wait

## Processi zombie e orfani

- Un processo terminato passa nello stato di "zombie", e viene rimosso, dopo che il padre ha ricevuto il suo stato di terminazione con una wait
- Un processo "orfano", cioè il cui padre è terminato, viene "adottato" dal processo init
  - un processo ha sempre un padre
  - init ha PID 1