Esercitazione di Reti degli elaboratori

Prof.ssa Chiara Petrioli

SAPIENZA
UNIVERSITÀ DI ROMA

Corso di C

Christian Cardia, Gabriele Saturni

Argomenti

- Allocazione dinamica della memoria

- File

Allocazione dinamica della memoria

Supponiamo di voler scrivere una funzione:

***char** *dupstr(**char** *s)*

che ritorna una copia della stringa di input s.
Potremmo scriverla così?

```
char *dupstr(char *s)
{
        char dup[strlen(s) + 1];
        strcpy(dup, s);
        return dup;
}
```

Allocazione dinamica della memoria

- **Risposta**: **No!**
- **dup è un array locale!**
- La memoria riservata per la variabile locale esiste solo all'interno della funzione che la dichiara.
- Per scrivere una funzione di questo tipo dobbiamo ***allocare memoria il modo dinamico***, cioè riservare locazioni di memoria per la durata dell'esecuzione del programma.
- Inoltre, la dimensione dell'array non è nota a priori (durante la compilazione), quindi non possiamo semplicemente allocare la memoria in modo statico ma dobbiamo allocare una dimensione di memoria determinata durante l'esecuzione del programma.

Allocazione dinamica della memoria

- Il C è un linguaggio molto flessibile e un esempio di questa flessibilità è dato dalla gestione della memoria.
- Rispetto al Java o Python, il C permette di assegnare la giusta quantità di memoria (solo e solamente quella necessaria) alle variabili del programma.
- In particolare l'uso della memoria allocata dinamicamente risulta utile con gli array. Utilizzare queste caratteristiche del C permette di creare programmi altamente portabili, in quanto utilizzano di volta in volta i valori giusti per la piattaforma.

Come è organizzata la memoria

- La memoria è divisa sostanzialmente in due parti:

Tipo Memoria	Descrizione
Stack	È una parte **statica**, che contiene tutto ciò che sappiamo sarà allocato di certo (tutto ciò che è dichiarato nel codice quindi, come una variabile int o un array allocato staticamente come abbiamo visto nelle lezioni precedenti)
Heap	È una parte **dinamica**, in cui la dimensione degli elementi può cambiare a “runtime” ovvero durante l’esecuzione del programma

Allocazione dinamica della memoria

- Il C supporta l'allocazione dinamica si memoria attraverso l'uso delle funzioni definite nella libreria **stdlib.h**
- Le principali funzioni sono:

Funzione	Descrizione
void* malloc(size)	riserva una buffer di size bytes e ne ritorna l'indirizzo.
void* calloc(size)	Come la malloc, solamente che inizializza anche la memoria con tutti 0.
void* realloc(void* ptr, size)	realloca la memoria puntata da ptr alla dimensione size bytes, o realloca la memoria
void free(void* ptr)	libera la memoria riservata da malloc, calloc o realloc

Allocazione dinamica della memoria

- Queste funzioni ritornano o accettano come argomenti puntatori di tipo **void*** → indica un puntatore ad un blocco di memoria generico.

- Se necessario, questo tipo è automaticamente convertito ad altri tipi puntatore.

Allocazione dinamica della memoria

- Per determinare la quantità di bytes necessaria per i vari tipi si usa l'operatore
  - **sizeof**(<tipo>)
  - **sizeof**(<variabile>)

- Ad esempio:
  - **sizeof**(char) ritorna 1;
  - **sizeof**(float) ritorna 4

Allocazione dinamica della memoria

- Invece, per una variabile *v* di un tipo *x*
  → **sizeof**(v) ritorna **sizeof**(*x*). Ad esempio, sia v di tipo **int**:
  - **sizeof**(*v*) = **sizeof**(*int*)

- Nel caso di tipi array,
  l'operatore **sizeof**(<tipo>[<dimensione>]) è uguale a **sizeof**(<tipo>)*<dimensione>.

Allocazione dinamica della memoria

Adesso possiamo scrivere la versione corretta della nostra funzione dupstr():

```
char *dupstr(char *s) {    //ritorna una copia della stringa s
    int n = strlen(s);
    char *dup = malloc((n + 1)*sizeof(char));
    if (dup == NULL) {
        //allocazione fallita
    }
    strcpy(dup, s);
    return dup;
}
```

Allocazione dinamica della memoria

Esercizio

Funzione ***int*** **occ(**int** A[], **int** n, **int** x, **int** *nocc)* che ritorna in un array, allocato dinamicamente, le posizioni dell'array *A* che contengono il valore *x* e in **nocc* restituisce il numero di tali posizioni.

Allocazione dinamica della memoria

Soluzione

```
int *occ(int A[], int n, int x, int *nocc) {
    int *pos = NULL;
    int no = 0;
    for (int i = 0 ; i < n ; i++) {
        if (A[i] == x) {
            pos = realloc(pos, (no + 1)*sizeof(int));
            pos[no++] = i;
        }
    }
    *nocc = no;
    return pos;
}
```

Allocazione dinamica della memoria
Esempio

Funzione ***char*** **inputstr()* che ritorna in una stringa, allocata dinamicamente, la sequenza di caratteri letti dall'input (fino o '\n').

Allocazione dinamica della memoria

Esempio

```
char *inputstr() {
    char *str = NULL;
    int c, n = 0;
    do {
        c = getchar();
        if (c != '\n') {
            str = realloc(str, (n + 1)*sizeof(char));
            str[n++] = c;
        }
    } while (c != '\n');
    str[n] = '\0';
    return str;
}
```

Allocazione dinamica della memoria
Esercizi

- **Esercizio 1** Scrivere una funzione ***int*** **mulArrayD(**int** A[], **int** n)* che ritorna un array, allocato dinamicamente, in cui ogni elemento dell'array A viene moltiplicato per 2.

Allocazione dinamica della memoria

Soluzione esercizio 1

```c
/* Ritorna un array, allocato dinamicamente.
 Gli elementi dell'array A, di dimensione n, vengono moltiplicati per 2
*/
int *mulArrayD(int A[], int n)
{
    int *doppio = malloc(n*sizeof(int));
    for (int i = 0 ; i < n ; i++)
        doppio[i] = 2*A[i];
    return doppio;
}
```

Allocazione dinamica della memoria
Esercizi

- **Esercizio 2** Scrivere una funzione ***char*** **concat(**char** *s1, **char** *s2)* che ritorna una nuova stringa che contiene la concatenazione delle stringhe s1 e s2. Ad esempio, se *s1* è "Prima parte" e *s2* è "Seconda parte" allora la funzione ritorna la stringa "Prima parteSeconda parte".

Allocazione dinamica della memoria

Soluzione esercizio 2

```c
#include <string.h>

/* Ritorna, in una stringa allocata dinamicamente, la concatenazione
 * delle stringhe s1 e s2. */
char *concat(char *s1, char *s2) {
    int n1 = strlen(s1), n2 = strlen(s2);
    char *str = malloc((n1 + n2 + 1)*sizeof(char));
    for (int i = 0 ; i < n1 ; i++)
        str[i] = s1[i];
    for (int i = 0 ; i < n2 ; i++)
        str[n1 + i] = s2[i];
    str[n1 + n2] = '\0';
    return str;
}
```

File I/O

- I file mantengono nella memoria di un computer qualunque tipo di informazioni: testi, dati, immagini, video, musica, ecc.
- Il C mette a disposizione alcune funzioni per creare, leggere e scrivere sui File.

File I/O

- Vediamo come gestire i dati e le informazioni presenti in un file in C.
- Tipi di file:
  - accesso sequenziale (file di testo).
  - accesso casuale (file binari).

File I/O
Funzioni principali in stdio.h

- FILE *fopen(**const char** *fname, **const char** *mode)*: apre un file di nome *fname* in modalità mode e ritorna il riferimento ad esso. Modalità di apertura per file di testo:"r": lettura file esistente (se non esiste ritorna NULL)
  - "w": scrittura (se il file esiste lo tronca, altrimenti lo crea)
  - "a": scrittura in append: scrive a partire dalla fine del file (se non esiste lo crea)
  - "r+": lettura e scrittura file esistente (se non esiste ritorna NULL)
  - "w+": lettura e scrittura (se il file esiste lo tronca, altrimenti lo crea)
  - "a+": lettura e scrittura in append: scrive (e legge) a partire dalla fine del file (se non esiste lo crea)
- Per i file binari bisogna aggiungere il carattere 'b', ad es. "r+b".

File I/O

Funzioni principali in stdio.h

- ***int*** *fclose(**FILE** *f)*: chiude il file f e ritorna 0, se si verifica un errore ritorna EOF.
- ***void*** *rewind(**FILE** *f)*: riporta il cursore di f all'inizio del file.
- ***int*** *fgetc(**FILE** *f)*: ritorna il prossimo carattere (byte) del file f e avanza il cursore. Se il cursore è alla fine del file o si verifica un errore ritorna EOF.
- ***int*** *fputc(**int** c, **FILE** *f)*: scrive il carattere c nella posizione del cursore del file f, avanza il cursore e ritorna il carattere scritto. Se si verifica un errore, ritorna EOF.

File I/O
Funzioni principali in stdio.h

- ***int*** *fscanf(**FILE** *f, **const char** *format, ...)*: del tutto simile alla scanf, solo che legge dal file f invece che dallo standard input. Ritorna il numero di assegnamenti effettuati.
- ***int*** *fprintf(**FILE** *f, **const char** *format, ...)*: del tutto simile alla printf, solo che scrive nel file f anziché nello standard output. Ritorna (in quasi tutte le implementazioni) il numero di caratteri scritti, in caso di errore ritorna un valore negativo.

File I/O
Funzioni principali in stdio.h

- ***int*** *fscanf(**FILE** *f, **const char** *format, ...)*: del tutto simile alla scanf, solo che legge dal file f invece che dallo standard input. Ritorna il numero di assegnamenti effettuati.
- ***int*** *fprintf(**FILE** *f, **const char** *format, ...)*: del tutto simile alla printf, solo che scrive nel file f anziché nello standard output. Ritorna (in quasi tutte le implementazioni) il numero di caratteri scritti, in caso di errore ritorna un valore negativo.

File I/O

Funzioni principali in stdio.h

- ***char** *fgets(**char** *s, **int** n, **FILE** *f)*: legge una linea dal file f e la copia in s.
- Legge e copia i caratteri nel blocco puntato da s finché incontra:
  - un ‘\n’
  - la fine del file
  - ha letto n - 1 caratteri.
- La stringa s è sempre terminata da '\0’.
- Se incontra la fine del file senza aver letto alcun carattere ritorna NULL(e s rimane invariata), altrimenti ritorna s.

File I/O
Esempio

Programma che prende come argomento del main il nome di un file (di testo) e permette all'utente di stampare a video le linee del file pagina per pagina (uso di fopen(), fgets()e fclose()).

File I/O
Esempio

Programma che legge un file e ne stampa il contenuto.

```c
#include <stdio.h>

#define MAX_LINE_SIZE 80

int main()
{
    char buf[MAX_LINE_SIZE];
    FILE *fp;
    fp=fopen("testo.txt", "r");
    if(fp!=NULL) {
        while (fscanf(fp,"%s",buf)>0)
            printf("%s\n", buf);
        fclose(fp);
    } else {
        printf("Error opening the file\n");
    }
    return 0;
}
```

File I/O
Esercizio

Programma che scrive in un file la lista della spesa.

File I/O

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

#define MAX_LINE_SIZE 80

int main()
{
    //scrittura in un file della lista della spesa
    FILE *fp;
    char item[MAX_LINE_SIZE];
    int fine=0;
    fp=fopen("testo.txt", "w");
    if( fp != NULL) {
        printf(" Lista della spesa: ");
        do
        {
            printf("\nprossimo articolo ? ");
            scanf("%s", item);
            fprintf(fp, "%s\n", item);
            printf("\nFinito (si=1, no=0)? ");
            scanf("%d", &fine);
        } while (!fine);
        fclose(fp);
        printf("Fine scrittura\n");
    } else {
        printf("Errore nell'apertura del file\n");
    }
    return 0;
}
```

File I/O
Esercizio

Esercizio. Scrivere un programma che prende come input i nomi di due file di testo e appende il contenuto del secondo file al primo file. Ad esempio, se il primo file contiene *ciao* primo file e il secondo file contiene *bella* allora il programma modifica il primo file così con il testo *ciaobella*.

File I/O
Soluzione

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

#define MAX_STR_SIZE 80

//Concatena due files, appendendo il contenuto del secondo file al primo.
int main() {
    char filename1[MAX_STR_SIZE];
    char filename2[MAX_STR_SIZE];
    printf("Inserisci il primo filename\n");
    scanf("%s", filename1);
    printf("\nInserisci il secondo filename\n");
    scanf("%s", filename2);
    FILE *f1 = fopen(filename1, "a");    //Apre il primo file in append
    FILE *f2 = fopen(filename2, "r");    //Apre il secondo file in lettura
    if (f1 == NULL || f2 == NULL) {
        printf("Impossibile aprire uno dei files!\n");
        return 0;
    }

    int c;
    while ((c = fgetc(f2)) != EOF)    //Legge carattere per carattere il secondo file
        fputc(c, f1);                 //e appende i caratteri al primo file
    fclose(f1);
    fclose(f2);
    return 0;
}
```

File I/O
FSEEK

Quando si lavora con i file spesso può risultare necessario accedere ad una posizione specifica di un determinato file aperto e creato precedentemente. Di seguito mostreremo le funzioni messe a dispozione dal linguaggio C per implementare queste funzionalità.

In particolare, andiamo a vedere la funzione fseek().

File I/O
FSEEK

Funzioni per il posizionamento del cursore dei files aperti.

*int fseek(FILE *f, long offset, int wherefrom)*: posiziona il cursore del file f secondo wherefrom e offset. Il parametro wherefrom specifica da dove deve essere calcolato l'offset e può assumere solamente tre valori costanti con i seguenti significati:

- SEEK_SET: inizio del file
- SEEK_CUR: posizione corrente
- SEEK_END: posizione finale del file

File I/O
FSEEK

Ad esempio, se f è un file binario:

- fseek(f, 100, SEEK_SET): posiziona il cursore 100 bytes dall'inizio del file
- fseek(f, 100, SEEK_CUR): posiziona il cursore 100 bytes dopo la posizione corrente
- fseek(f, 100, SEEK_END): posiziona il cursore 100 bytes dopo la fine del file in tutti i casi se la posizione è oltre la fine del file, il file è esteso con contenuto non specificato.
- fseek(f, -100, SEEK_CUR): posiziona il cursore 100 bytes prima delle posizione corrente

File I/O
FSEEK

Per i file di testo sono permesse solamente le seguenti forme (eccetto per quelle piattaforme, come Unix/Linux, in cui non c'è differenza tra file di testo e file binari):

- fseek(f, 0, SEEK_SET): equivale a rewind(f)
- fseek(f, 0, SEEK_END): posiziona il cursore alla fine del file
- fseek(f, ftell_pos, SEEK_SET): dove ftell_pos è una posizione ritornata da ftell(f), e quindi riporta il cursore in quella posizione

La funzione fseek ritorna 0, ma se si verifica un errore ritorna un valore diverso da 0.

File I/O

FTELL

long ftell(FILE *f):

ritorna la posizione del cursore del file f, se c'è un errore ritorna -1. Per i file binari il valore ritornato è il numero di bytes dall'inizio del file, per i file di testo questo non è garantito. Comunque il valore ritornato può essere usato come parametro offset della funzione fseek.

File I/O
Esercizio

- Usare le funzioni fseek e ftell per scrivere una funzione che cerca la prima occorrenza di una stringa in un file di testo. Se la trova ritorna la posizione (data da ftell()), altrimenti ritorna -1

File I/O
Soluzione

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

/* Cerca la prima occorrenza della stringa str nel file di testo f a partire
 * dalla posizione corrente. Se la trova ritorna la posizione (data da ftell()),
 * altrimenti ritorna -1.
 * Usa fgets() per leggere un blocco di caratteri di lunghezza uguale alla
 * lunghezza della stringa e poi usa fseek() e fgetc() per riposizionarsi per la
 * prossima chiamata di fgets(). */
long filefind(FILE *f, char *str) {
    long n = strlen(str) + 1, pos = 0;
    char buffer[n];
    fseek(f, pos, SEEK_SET);
    while (fgets(buffer, n, f) != NULL) {
        if (strcmp(buffer, str) == 0)
            return pos;
        else {
            fseek(f, pos, SEEK_SET);
            fgetc(f);
            pos = ftell(f);
        }
    }
    return -1;
}
```

File I/O
Soluzione

```
//Legge dalla linea di comando il nome di un file e una stringa da cercare
int main(int argc, char *argv[]) {
    if (argc != 3) return 0;
    FILE *f = fopen(argv[1], "r");
    if (f == NULL) {
        printf("Impossibile aprire file \"%s\"\n", argv[1]);
        return 0;
    }
    long pos = filefind(f, argv[2]);
    if (pos >= 0)
        printf("Trovata in posizione %ld\n", pos);
    else
        printf("Non trovata\n");
    fclose(f);
    return 0;
}
```