Esercitazione di Reti degli elaboratori

Prof.ssa Chiara Petrioli

SAPIENZA
UNIVERSITÀ DI ROMA

Corso di C

Christian Cardia, Gabriele Saturni

Argomenti

- Funzioni
- Allocazione dinamica della memoria
- File

Le funzioni

- Fino ad ora abbiamo sviluppato molte applicazioni che si potrebbero riutilizzare in altri programmi. Al momento, l'unico modo in cui possiamo farlo è di tagliare ed incollare il codice in un altro programma.
- Altamente sconsigliato e soggetto a molti errori. Il modo più semplice per riutilizzare il codice è di isolare parti di codice in *funzioni* definite dall'utente.

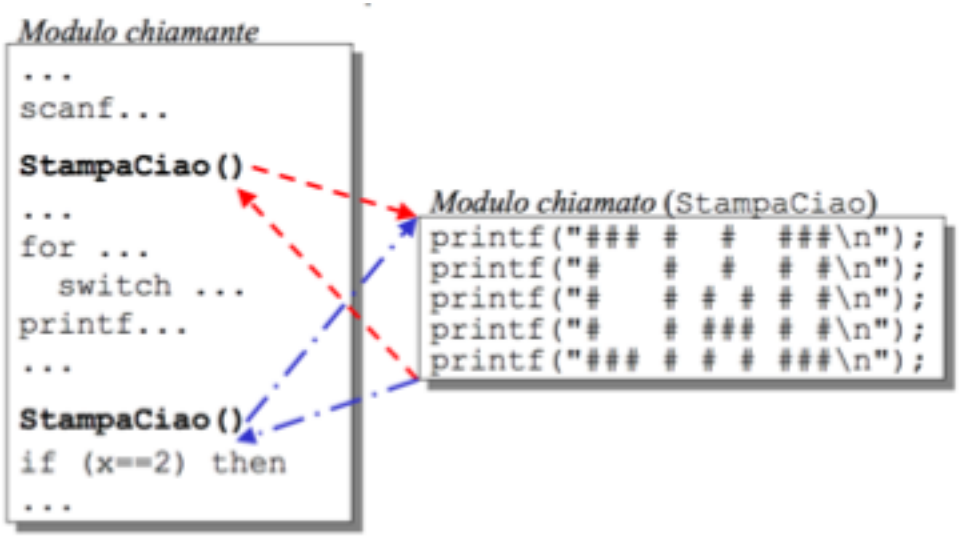
Le Funzioni

Struttura modulare

- Per semplificare la struttura di un programma complesso è possibile suddividerlo in moduli
- Un modulo è un blocco di codice che assolve ad un compito preciso (ad es. calcola la radice quadrata) e a cui è stato dato un nome
- Un programma è composto dal main ed eventuali altri moduli di supporto
- Quando un modulo richiama un altro modulo, il chiamante viene sospeso finché il chiamato non ha terminato la sua esecuzione
- In C i moduli sono chiamati **funzioni**
- Ogni funzione può richiamare (far eseguire) qualsiasi altra funzione (anche se stessa)

Le Funzioni
Struttura modulare

- Le due chiamate del modulo StampaCiao fanno eseguire ogni volta le istruzioni che lo costituiscono (sospendendo il chiamante)



Le Funzioni
I VANTAGGI

- Migliorano la leggibilità del codice organizzando le istruzioni in gruppi logici (funzioni) che risolvono problemi specifici;
- Riducono la duplicazione del codice raggruppando gruppi di istruzioni di uso comune in funzioni che vengono chiamate, invece di copiare ed incollare il codice comune;
- Facilitano l'individuazione degli errori dato che, una volta che si è individuata la funzione problematica, il programmatore si può focalizzare solo su quella parte di codice;
- Facilitano le modifiche e estensioni dato che il codice è organizzato il modo più logico;
- Rendono possibile la riusabilità del codice attraverso librerie di funzioni che si possono usare in programmi distinti.

Le Funzioni

Gli SVANTAGGI

NON

CI

SONO

Le Funzioni
Variabili locali

- Ogni funzione è un piccolo programma indipendente dalle altre funzioni
- All'interno di una funzione possono essere definite delle variabili locali: le altre funzioni non le "vedono"
- Variabili con lo stesso nome in funzioni diverse sono quindi completamente non correlate (possono dunque anche essere di tipo diverso)
- Vengono create ogni volta che si entra nella funzione e distrutte (perdendo il valore) quando si esce
- Le inizializzazioni avvengono ad ogni chiamata, senza inizializzazione il contenuto è indefinito

Le Funzioni
Parametri e valore restituito

- Essendo le variabili interne locali, private, per passare ad una funzione i dati da elaborare è necessario utilizzare variabili speciali dette parametri.
- La funzione comunica al modulo chiamante il risultato della sua elaborazione producendo un unico valore detto valore restituito o valore di ritorno

Le Funzioni

Definizione

```
tipo nomeFunzione (parametri)
{
    definizione_variabili_locali

    istruzioni
    eventuale return
}
```

corpo della funzione

- *tipo* indica il tipo del valore restituito (double, float, int, ecc…)

- Se la funzione non restituisce valori (ad esempio StampaCiao visualizza soltanto) bisogna indicare il tipo *void*: *void StampaCiao(.....)*

Le Funzioni
Chiamata di funzione

- Si chiama una funzione indicandone il nome seguito da una coppia di parentesi contenenti i valori da elaborare (separati da virgole) eleva(y,2);
- Se la funzione non richiede parametri, le parentesi sono vuote, ma devono esserci: StampaCiao();
- Il valore restituito può essere assegnato ad una variabile o utilizzato in un'espressione, altrimenti viene semplicemente scartato.

```
x = eleva(y,2);
y = 3*eleva(2,k)-4*k;
eleva(3,5);
```

Le Funzioni

Ritorno da una funzione

- La funzione termina (cioè l'esecuzione torna al modulo chiamante) quando viene eseguita l'istruzione ***return*** *valore*;
- Una funzione può avere più istruzioni **return**;
- Se il tipo restituito dalla funzione è **void,** la **return** può essere omessa;
- I valori delle variabili locali vengono persi.

Le Funzioni
Un esempio

Funzione che eleva un numero b per un esponente e

```
#include <stdio.h>

int eleva(int b, int e)
{
    int k=1;
    while (e-- > 0)
        k *= b;
    return k;
}
```

Le Funzioni

Un esempio (parte 2)

... Continua da slide precedente

```
int main()
{
    int x=0, y=0, z=0;
    printf("Introduci numero: ");
    scanf("%d", &x);
    printf("Introduci la potenza: ");
    scanf("%d", &y);
    z = eleva(x, y);
    printf("%d^%d = %d\n", x,y,z);
    return 0;
}
```

Le Funzioni
Riepilogo

Definizione della funzione

```
#include <stdio.h>

int eleva(int b, int e)
{
    int k=1;
    while (e-- > 0)
        k *= b;
    return k;
}
```

Le Funzioni
Riepilogo

Tipo di ritorno della funzione

```
#include <stdio.h>

int eleva(int b, int e)
{
    int k=1;
    while (e-- > 0)
        k *= b;
    return k;
}
```

Le Funzioni
Riepilogo

Nome della funzione

```
#include <stdio.h>

int eleva(int b, int e)
{
    int k=1;
    while (e-- > 0)
        k *= b;
    return k;
}
```

Le Funzioni
Riepilogo

Lista parametri input della funzione

```
#include <stdio.h>

int eleva(int b, int e)
{
    int k=1;
    while (e-- > 0)
        k *= b;
    return k;
}
```

Le Funzioni
Riepilogo

Corpo della funzione (variabili definite all'interno della funzione sono visibili sono all'interno della stessa)

```
#include <stdio.h>

int eleva(int b, int e)
{
    int k=1;
    while (e-- > 0)
        k *= b;
    return k;
}
```

Le Funzioni
Riepilogo

Espressione di ritorno

```
#include <stdio.h>

int eleva(int b, int e)
{
    int k=1;
    while (e-- > 0)
        k *= b;
    return k;
}
```

Le Funzioni
prototipo

- Il prototipo di una funzione è una dichiarazione della funzione (nome e tipo). Viene inserito:
  - prima del main
  - In file header separato

```
int eleva(int b, int e);
```

- I prototipi sono simili alle definizioni, salvo che:
  - manca il corpo
  - i nomi dei parametri possono essere omessi (ma i tipi devono essere presenti!)
  - ha un punto e virgola alla fine

Le Funzioni

Esempio con prototipo

```c
#include <stdio.h>

int eleva(int b, int e); //prototipo

int main()
{
    int a=3, b=2;
    int risultato=eleva(a,b);
    printf("%d^%d = %d\n", a,b,risultato);
    return 0;
}

int eleva(int b, int e)
{
    int k=1;
    while (e-- > 0)
        k *= b;
    return k;
}
```

Le Funzioni
Passaggio di parametri

- I parametri di tipo numerico (intero e virgola mobile) vengono ***passati per valore*** alle funzioni. Questo significa che se il parametro viene cambiato durante l'esecuzione della funzione, questo non ha effetto al di fuori della funzione stessa.
- I parametri di tipo array sono invece ***passati per riferimento***. Questo significa che se i valori delle componenti dell'array sono alterati durante l'esecuzione della funzione, gli stessi valori sono alterati nell'array che viene passato come argomento durante la chiamata.

Le Funzioni

Passaggio di parametri – esempi

- Funzione che ritorna la somma dei valori di un array.

```
float sum(float[] v, int n) {
    float s = 0;
    for(int i = 0; i < n; i ++){
        s += v[i];
    }
    return s;
}
```

Le Funzioni

Passaggio di parametri – esempi

- Funzione che imposta gli elementi di un array ad un valore dato

```
void set(float[] v, int n, float value) {
    for(int i = 0; i < n; i ++) {
        v[i] = value;
    }
}
```

Le funzioni
Esempio

Esempio di una funzione che legge dallo standard input una sequenza di caratteri e li scrive nell'array *line* fino a che non legge carattere '\n' o sono stati letti *maxlen* caratteri; la funzione ritorna il numero di caratteri letti.

Le funzioni
Esempio

```c
/* Legge dallo standard input nell'array line una sequenza di caratteri
  fino a quando incontra il carattere di fine linea '\n' (escluso)
 o ha letto maxlen caratteri. Ritorna il numero di caratteri letti.
 L'array deve avere dimensione almeno pari a maxlen.
*/
int inputline(char line[], int maxlen) {

    int n = 0;
    char c;
    do {
        c = getchar();
        if (c != '\n')
            line[n++] = c;
    } while (c != '\n' && n < maxlen);
    return n;
}
```

Le funzioni
Esempio

```
int main()
{

    printf("Inserire una linea di testo: ");
    char line[MAXLEN];
    int n = inputline(line, MAXLEN);
    printf("Il numero di caratteri da cui la stringa e' composta: %d\n",n);
    return 0;
}
```

Le funzioni
Esempio

MAXLEN è una costante definita a priori tramite il costrutto #define

```
int main()
{

    printf("Inserire una linea di testo: ");
    char line[MAXLEN];
    int n = inputline(line, MAXLEN);
    printf("Il numero di caratteri da cui la stringa e' composta: %d\n",n);
    return 0;
}
```

Le funzioni
Esempio

#define definisce una costante visibile in tutto il programma

```
#define MAXLEN    1000


int main()
{

    printf("Inserire una linea di testo: ");
    char line[MAXLEN];
    int n = inputline(line, MAXLEN);
    printf("Il numero di caratteri da cui la stringa e' composta: %d\n",n);
    return 0;
}
```

Le funzioni
Esempio

```c
#define MAXLEN    1000

/* Legge dallo standard input nell'array line una sequenza di caratteri
  fino a quando incontra il carattere di fine linea '\n' (escluso)
 o ha letto maxlen caratteri. Ritorna il numero di caratteri letti.
 L'array deve avere dimensione almeno pari a maxlen.
*/
int inputline(char line[], int maxlen) {

    int n = 0;
    char c;
    do {
        c = getchar();
        if (c != '\n')
            line[n++] = c;
    } while (c != '\n' && n < maxlen);
    return n;
}

int main()
{

    printf("Inserire una linea di testo: ");
    char line[MAXLEN];
    int n = inputline(line, MAXLEN);
    printf("Il numero di caratteri da cui la stringa e' composta: %d\n",n);
    return 0;
}
```

Le funzioni

Esercizio. Scrivere una funzione *void rot(char C[], int n, int k)* che ruota i caratteri del vettore C di k posizioni a destra. Ad esempio, se i caratteri nel vettore C sono "programma" (n = 9) e k = 3, allora la funzione modifica il vettore in modo che contenga "mmaprogra".

Le funzioni

```c
#include <stdio.h>

int inputline(char line[], int maxlen) {
    . . .
}

void rot(char C[], int n, int k) {
    char copy[n];
    for (int i = 0 ; i < n ; i++) copy[i] = C[i];
    for (int i = 0 ; i < n ; i++)
        C[(i + k) % n] = copy[i];
}

#define MAX_CHARS     100

// stampa una linea di testo ruotata
int main() {
    char line[MAX_CHARS];
    printf("Inserire una linea di testo: ");
    int r, n = inputline(line, MAX_CHARS);
    if (n == 0) return 0;
    printf("Inserire la rotazione: ");
    scanf("%d", &r);
    rot(line, n, r);
    for (int i = 0 ; i < n ; i++)
        printf("%c", line[i]);
    printf("\n");
}
```

Le funzioni

Esercizio. Scrivere una funzione *int included(int A[], int n, int B[], int m)* che ritorna 1 se tutti i valori nel vettore A (di dimensione n) sono anche contenuti nel vettore B (di dimensione m), altrimenti ritorna 0. Ad esempio, se A = {2, 1, 3, 2, 1} e B = {5, 1, 3, 2} allora la funzione ritorna 1, se invece B = {5, 3, 2, 0} ritorna 0.

Le funzioni

```
// Ritorna 1 se tutti i valori di A sono inclusi in B, 0 altrimenti
int included(int A[], int n, int B[], int m) {
    for (int i = 0 ; i < n ; i++) {
        int j = 0;
        while (j < m && B[j] != A[i]) j++;
        if (j == m) return 0;
    }
    return 1;
}
```

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

void* malloc**(int** size**)**	riserva una buffer di size bytes e ne ritorna l'indirizzo
void free(**void*** ptr)	libera la memoria riservata da malloc o realloc per altro uso
void* realloc(**void*** ptr, **int** size)	realloca la memoria puntata da ptr alla dimensione size bytes, o realloca la memoria

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
Esempio

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

Esempio

```
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
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 1

```
/* 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

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

- 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.

```
#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
Esempio

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;
}
```