Esercizi di programmazione in linguaggio C – Ricerca

Nota: memorizzare i programmi in cartelle separate. Ogni cartella deve contenere, oltre al codice sorgente e al file eseguibile, gli eventuali file di input indicati nel testo e file di output prodotti dal programma. I file di input devono essere generati con un apposito editor prima della scrittura del codice.

Ricerca sequenziale

1. **seq1** ★ Dopo aver definito una funzione che trovi un numero all'interno di un vettore di interi usando un algoritmo di ricerca sequenziale, scrivere un programma che inizializzi staticamente un vettore di N elementi, chieda di inserire un numero da tastiera e dica se tale numero sia presente nel vettore. Eseguire il programma con N = 10.

2. **seq2** – Ripetere l'esercizio **seq1** con dati di tipo *double* e modificando la funzione affinché la ricerca inizi dall'ultimo elemento del vettore.

3. **seq3** – Ripetere l'esercizio **seq1** inizializzando il vettore con valori casuali. Eseguire il programma con N = 500.

4. **seq-file1** – Il file di testo *numeri.txt* contiene un certo numero di valori interi (non più di 10000). Scrivere un programma che, utilizzando apposite funzioni, esegua le seguenti operazioni:

   a) caricamento del contenuto del file in un vettore;
   b) ricerca del numero 0 all'interno del vettore;
   c) scrittura dell'esito della ricerca nel file *esito.txt*: se il numero è presente si riporta nel file la sua posizione (one-based); in caso contrario, si scrive la stringa `NUMERO NON TROVATO`.

5. **seq-file2** ★ Il file di testo *invitati.txt* contiene i nomi degli invitati a una festa (si suppone che i nomi siano privi di spazi, scritti in maiuscolo e che non siano più di 100). Scrivere un programma che, utilizzando una funzione di ricerca sequenziale, dica se ANNA, BRUNO e CARLO sono invitati alla festa.

6. **seq-punti** – Dopo aver definito una struttura per memorizzare le coordinate reali di un punto del piano cartesiano, scrivere un programma che:

   a) acquisisca i punti contenuti nel file di testo *punti.txt* (in cui ogni riga contiene l'ascissa e l'ordinata di un punto) e li memorizzi in un'apposita tabella;
   b) indichi se l'origine è presente nella tabella (si utilizzi una funzione di ricerca lineare);
   c) chieda all'utente le coordinate di un punto P e indichi se P è presente nella tabella.

Ricerca binaria

7. **bin1it** – Definire una funzione che trovi un numero all'interno di un vettore di *float* usando un algoritmo iterativo di ricerca binaria. Scrivere un programma che inizializzi un vettore di N elementi ordinati, chieda di inserire un numero *float* da tastiera e dica se tale numero sia presente nel vettore. Eseguire il programma con N = 8.

8. **bin1ric** – Ripetere l'esercizio **bin1it** ma con un algoritmo ricorsivo.

9. **bin2** ★ Dato un vettore V di elementi *unsigned int* non ordinato, scrivere un programma che: ordini V e ne stampi il contenuto finale; determini e visualizzi la media aritmetica M (troncata all'intero) tra i valori minimo e massimo presenti in V; indichi se nel vettore esiste un elemento di valore pari a M utilizzando un algoritmo di ricerca dicotomica.

Nelle pagine successive sono indicate le soluzioni degli esercizi che riportano il simbolo ★.

10. **incasso** – Nel file di testo *incassi.txt* sono registrati gli incassi giornalieri di un certo negozio. Il tracciato record del file è il seguente:

```
gg  mm  aaaa   totale_giorno
```

I primi tre campi rappresentano la data di registrazione e sono di tipo intero; il quarto campo (totale in Euro incassato in quel giorno) è invece di tipo *float*. I dati sono memorizzati nel file in ordine cronologico.

Scrivere un programma che legga il contenuto del file e lo memorizzi in un'apposita tabella. Successivamente, il programma, dopo aver chiesto all'utente l'inserimento da tastiera di una data, deve ricercare la data e stamparne l'incasso (oppure un messaggio d'errore nel caso in cui la data non sia presente). La ricerca deve essere eseguita con un algoritmo efficiente in tempo mediante una funzione avente il seguente prototipo:

```
int ricerca_incasso(struct incasso vett[], int dim, int giorno, int mese, int anno)
```

11. **incasso-mese** – Ripetere l'esercizio precedente, nell'ipotesi che l'utente inserisca solo un mese (e il relativo anno, per esempio 10 2016) e che voglia visualizzare tutti gli incassi registrati nei giorni di quel mese.

12. **elezioni** ★ In vista delle prossime elezioni degli organi collegiali, una scuola ha deciso di automatizzare le procedure di conteggio dei voti e vi ha chiesto di sviluppare un opportuno software. Le preferenze indicate nelle schede dagli elettori sono memorizzate in un file di testo nel seguente modo:
    - se la preferenza è valida, si riporta su una linea del file il solo cognome del candidato (il cognome non può essere più lungo di 30 caratteri e può contenere spazi);
    - se la preferenza non è valida (scheda nulla) si riporta la sequenza speciale `*N*` ;
    - se l'elettore non ha indicato alcuna preferenza (scheda bianca) si riporta la sequenza speciale `*B*` .

    Scrivere un programma che, dopo aver predisposto una tabella per il conteggio dei voti assegnati ai candidati, esegua le seguenti operazioni:

    a) acquisisca il contenuto del file di input, distinguendo per ogni linea letta i seguenti casi:
        - se il candidato indicato è già presente in tabella si incrementa il suo contatore di voti;
        - se il candidato non è presente nella tabella, lo si aggiunge con un algoritmo di *insertion-sort* che preservi l'ordine alfabetico dei candidati e gli si assegna un voto (la tabella non contiene più di 100 candidati);
        - se la linea contiene una sequenza speciale, si aggiornano i corrispondenti contatori.

    b) terminata la fase di lettura, ordini la tabella in base al numero di voti ottenuti e in modo decrescente;

    c) stampi a video i risultati delle elezioni, riportando i voti ottenuti di ciascun candidato il numero di schede bianche e nulle, secondo il seguente formato:

```
============================
  Risultati delle elezioni
============================

Pos.  Candidato        Voti
----------------------------
  1.  Lazzaro           47
  2.  Pellitteri        10
  3.  Molino             6
  ...
  ...
----------------------------

Schede bianche:    2
  Schede nulle:    9
```

Esercizio n. 1 (seq1)

```
/* seq1.c
 *
 * Ricerca di un numero con algoritmo sequenziale.
 */

#include <stdio.h>

#define N 10

int ricerca_numero(int vett[], int dim, int numero) {

    for (int i = 0; i < dim; i++)
        if (vett[i] == numero)
            return i;

    return -1;
}

int main( void ) {
    int v[N] = {13, 2, -19, 0, 8, 5, -7, 76, -12, 3};
    int pos, num;

    printf("Inserire il numero da ricercare: ");
    scanf("%d", &num);

    pos = ricerca_numero(v, N, num);

    if (pos < 0)
        printf("Numero non trovato.\n");
    else
        printf("Numero trovato in posizione %d.\n", pos);

    return 0;
}
```

Esercizio n. 5 (seq-file2)

```
/* seq-file2.c
 *
 * Ricerca gli amici invitati alla festa.
 */

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

#define MAX_INVITATI 100
#define MAX_NOME 20
#define NUM_RICERCHE 3

#define NOME_FILE_INVITATI "invitati.txt"

// Prototipi delle funzioni
int carica_invitati(char [], char [][MAX_NOME+1]);
int ricerca_nome(char [][MAX_NOME+1], int, char []);

// Funzione principale

int main( void ) {
    char nomi_da_ricercare[NUM_RICERCHE][MAX_NOME+1] = { "ANNA", "BRUNO", "CARLO" };

    char invitati[MAX_INVITATI][MAX_NOME+1];
    int dim_inv;

    dim_inv = carica_invitati(NOME_FILE_INVITATI, invitati);

    if (dim_inv < 0) {
        printf("Errore nell'elaborazione del file di input.\n");
        return 1;
    }

    for (int i = 0; i < NUM_RICERCHE; i++) {
        int pos;
        pos = ricerca_nome(invitati, dim_inv, nomi_da_ricercare[i]);

        printf("%s %s invitato/a alla festa\n",
            nomi_da_ricercare[i], (pos >= 0 ? "e'" : "non e'"));
    }

    return 0;
}

int carica_invitati(char nome_file[], char vett[][MAX_NOME+1]) {

    FILE *fp;
    int i = 0;

    if ((fp = fopen(nome_file, "r")) == NULL)
        return -1;
```

```
    while(fscanf(fp, "%s", vett[i]) > 0)
        i++;

    fclose(fp);

    return i;
}

int ricerca_nome(char vett[][MAX_NOME+1], int dim, char nome[]) {
    for (int i = 0; i < dim; i++)
        if (strcmp(vett[i], nome) == 0)
            return i;

    return -1;
}
```

Esercizio n. 9 (bin2)

```
/* bin2.c
 *
 * Ricerca la media artimetica dei valori min e max di un vettore.
 */

#include <stdio.h>
#include <stdbool.h>

#define MAX_VETTORE 16

void ordina_vettore(unsigned int vett[], int dim) {
    bool scambi;
    do {
        scambi = false;
        for(int i = 0; i < dim-1; i++)
            if (vett[i+1] < vett[i]) {
                 unsigned int temp = vett[i];
                 vett[i] = vett[i+1];
                vett[i+1] = temp;
                scambi = true;
            }
     }
     while (scambi == true);
}

void stampa_vettore(unsigned int vett[], int dim) {
    for (int i = 0; i < dim; i++)
        printf("%4u", vett[i]);

    printf("\n");
}

int ricerca_bin(unsigned int vett[], int dim, int numero) {
    int i_inf = 0;
    int i_sup = dim - 1;
    int i_medio;

    while (i_inf <= i_sup) {

        i_medio = (i_inf + i_sup) / 2;

        if (numero > vett[i_medio])
            i_inf = i_medio + 1;
        else if (numero < vett[i_medio])
            i_sup = i_medio - 1;
        else
            return i_medio;
    }

    return -1;
}
```

```
int main( void ) {
    unsigned int v[MAX_VETTORE] = {
        45u,  16u, 10u,  34u, 190u, 12u, 285u, 101u,
        56u, 149u,  1u, 209u, 143u, 98u, 155u,  77u
    };

    unsigned int media;
    int pos;

    ordina_vettore(v, MAX_VETTORE);
    stampa_vettore(v, MAX_VETTORE);

    media = (v[0] + v[MAX_VETTORE-1]) / 2;
    printf("\nMedia dei valori estremi: %u\n\n", media);

    pos = ricerca_bin(v, MAX_VETTORE, media);

    if (pos < 0)
        printf("Nessun elemento e' pari alla media calcolata.\n");
    else
        printf("La media e' stata trovata in posizione %d (zero-based).\n", pos);

    return 0;
}
```

Esercizio n. 12 (elezioni)

```
/* elezioni.c
 *
 * Visualizza i risultati delle elezioni degli organi collegiali.
 */

#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX_COGNOME 15

#define MAX_CANDIDATI 100
#define NOME_FILE "preferenze-d.txt"
#define SIGLA_SCHEDA_BIANCA "*B*"
#define SIGLA_SCHEDA_NULLA "*N*"

struct candidato {
    char cognome[MAX_COGNOME + 1];
    int voti;
};

// Funzione ausiliaria di insertion-sort.
// Riceve in input un vettore di candidati ordinato per cognome
// e vi inserisce un nuovo candidato mantenendo l'ordine degli elementi.

int inserisci_candidato(struct candidato vett[], int dim, struct candidato nuovo_c) {
    int i;

    if (dim == MAX_CANDIDATI) {
        printf("Tabella piena, inserimento annullato.\n");
        return dim;
    }

    i = dim;

    while (i > 0 && strcmp(vett[i-1].cognome, nuovo_c.cognome) > 0) {
        vett[i] = vett[i-1];
        i--;
    }

    vett[i] = nuovo_c;

    return dim+1;
}

int ricerca_candidato(struct candidato vett[], int dim, char cognome[]) {
    // Ricerca un candidato per cognome usando l'algoritmo di ricerca binario.
    // Restituisce la posizione del candidato nel vettore oppure -1 se
    // il cognome non e' presente.

    int i_inf, i_sup, i_medio;
```

```
    i_inf = 0;
    i_sup = dim-1;

    while (i_inf <= i_sup) {
        int cmp;
        i_medio=(i_inf + i_sup) / 2;
        cmp = strcmp(cognome, vett[i_medio].cognome);

        if (cmp == 0)
            return i_medio;     // Candidato trovato, si restituisce la sua posizione
        else if (cmp < 0)
            i_sup = i_medio - 1;
        else
            i_inf = i_medio + 1;
    }

    // Se il ciclo termina, il candidato non e' presente nel vettore
    return -1;
}

void ordina_per_voti(struct candidato vett[], int dim) {
    // Ordina nuovamente la tabella, questa volta in base al
    // numero di voti e in modo decrescente.
    // Algoritmo di Bubble Sort.

    bool scambi;
    do {
        scambi = false;
        for(int i = 0; i < dim-1; i++)
            if (vett[i+1].voti > vett[i].voti) {    // 'maggiore di' -> ord. decrescente
                struct candidato temp = vett[i];
                vett[i] = vett[i+1];
                vett[i+1] = temp;
                scambi = true;
            }
    }
    while (scambi == true);
}

void stampa_risultati(struct candidato vett[], int dim, int bianche, int nulle) {

    printf("============================\n");
    printf("  Risultati delle elezioni\n");
    printf("============================\n\n");

    printf("Pos.  %-*s  Voti\n", MAX_COGNOME, "Candidato");

    // Stampa una linea adattata alla dimesione max. del cognome
    for (int i = 0; i < 13 + MAX_COGNOME; i++) printf("-");
    printf("\n");

    for(int i = 0; i < dim; i++)
```

```
        printf("%3d.  %-*s %4d\n",
            i+1,
            MAX_COGNOME, vett[i].cognome,
            vett[i].voti
        );

    for (int i = 0; i < 13 + MAX_COGNOME; i++) printf("-");
    printf("\n");

    printf("\n");
    printf("Schede bianche: %4d\n", bianche);
    printf("  Schede nulle: %4d\n", nulle);
}


int main( void ) {
    struct candidato tab[MAX_CANDIDATI];
    int dim_tab = 0;
    int num_bianche = 0;
    int num_nulle = 0;
    char linea[MAX_COGNOME+2];  // Un carattere in piu' perche' contiene \n

    FILE *fp;

    if ((fp = fopen(NOME_FILE, "r")) == NULL) {
        printf("Errore nell'apertura del file.\n");
        return 1;
    }

    while (fgets(linea, MAX_COGNOME+2, fp) != NULL) {
        int pos;
        int lun = strlen(linea);

        if (linea[lun-1] == '\n')
            linea[lun-1] = '\0';    // Rimuove il carattere '\n'

        if (strcmp(linea, SIGLA_SCHEDA_BIANCA) == 0)
            num_bianche++;

        else if (strcmp(linea, SIGLA_SCHEDA_NULLA) == 0)
            num_nulle++;

        else {
            pos = ricerca_candidato(tab, dim_tab, linea);

            if (pos >= 0)
                tab[pos].voti++;    // Incrementa il numero di voti se il candidato
                                    // e' gia' nella tabella
            else {
                // Inserisce un nuovo candidato con una sola preferenza
                struct candidato c;
                strcpy(c.cognome, linea);
                c.voti = 1;
```

```
                dim_tab = inserisci_candidato(tab, dim_tab, c);
            }
        }
    }
    fclose(fp);

    ordina_per_voti(tab, dim_tab);
    stampa_risultati(tab, dim_tab, num_bianche, num_nulle);

    return 0;
}
```