Esercizio 1 – Sintesi del codice (10.5 punti)

Si considerino le seguenti strutture dati: la struttura `prodotto` definisce un prodotto con il suo codice, il suo nome e il suo prezzo; `lista_prodotti` definisce una lista concatenata di prodotti; `lista_float` definisce una lista concatenata di numeri in virgola mobile.

```
#define MAX_NOME 15

typedef struct {
     unsigned int codice;
     char nome[MAX_NOME + 1];
     float prezzo;
} prodotto;

typedef struct nodo_prodotto_t {
     prodotto p;
     struct nodo_prodotto_t *next;
} nodo_prodotto;

typedef nodo_prodotto * lista_prodotti;

typedef struct nodo_float_t {
     float v;
     struct nodo_float_t *next;
} nodo_float;

typedef nodo_float * lista_float;
```

(a) Si scriva una funzione

```
lista_prodotti leggi_da_file(char *nome_file)
```

che acquisisce le descrizioni di alcuni prodotti da un *file* testuale il cui nome è passato come parametro e crea una lista dinamica di prodotti.
Ogni rigo del *file* contiene la descrizione di un prodotto come sequenza di:
codice prodotto (sequenza di caratteri indicante un numero intero positivo),
nome prodotto (sequenza di caratteri senza spaziature), e
prezzo in euro (sequenza di caratteri indicante un numero in virgola mobile con due cifre decimali);
un carattere di ***spazio*** è usato per separare ciascun valore. *(2.5 punti)*

Esempio del contenuto del *file* con la descrizione dei prodotti:

```
123456 Prodotto1 20.10
43216 Prodotto2 30.00
3212342 Prodotto3 40.85
32342 Prodotto4 10.00
1245 Prodotto5 100.50
```

Soluzione:

Usando una funzione ausiliaria per l'inserimento "in coda" di un nuovo elemento nella lista di prodotti:

```
lista_prodotti inserisciInCoda(lista_prodotti L, prodotto p) {

    nodo_prodotto *aux;

    if ( L == NULL ) {
        aux = (nodo_prodotto*) malloc(sizeof(nodo_prodotto));
        aux->p = p;
        aux->next = NULL;
        return aux;
    } else {
        L->next = inserisciInCoda(L->next, p);
        return L;
    }
}

lista_prodotti leggi_da_file(char *nome_file) {

    lista_prodotti L = NULL;
    FILE *fp = fopen(nome_file, "r");

    if (fp == NULL)
      fprintf(stderr, "Errore nell'apertura del file\n");
    else {
      prodotto p;
      while ( ! feof(fp) ) {
           fscanf(fp, "%u %s %f", &p.codice, p.nome, &p.prezzo);
           L = inserisciInCoda(L, p);
      }
      fclose(fp);
    }
    return L;
}
```

Soluzione alternativa

Usando un solo sottoprogramma con politica di "Inserimento In Testa" della lista di prodotti

```
lista_prodotti leggi_da_file(char *nome_file) {

    lista_prodotti L = NULL;
    FILE *fp = fopen(nome_file, "r");

    if ( fp == NULL )
      fprintf(stderr, "Errore nell'apertura del file\n");
    else {
      prodotto p;
      nodo_prodotto *aux;
      while ( ! feof(fp) ) {
        fscanf(fp, "%u %s %f", &p.codice, p.nome, &p.prezzo);
        aux = (nodo_prodotto*) malloc(sizeof(nodo_prodotto));
        aux->p = p;
        aux->next = L;
        L = aux;
      }
      fclose(fp);
    }
    return L;
}
```

(b) Si scriva una funzione iterativa

```
lista_prodotti elimina_prodotti(lista_prodotti L, float prezzo_minimo)
```

che modifica la lista passata come parametro rimuovendo tutti i prodotti che hanno un prezzo inferiore a `prezzo_minimo` e ritorna la lista modificata.
Durante la rimozione, prestare attenzione a liberare la memoria non più utilizzata. *(3 punti)*

Soluzione:

```
lista_prodotti elimina_prodotti(lista_prodotti L, float prezzo_minimo) {

    lista_prodotti ptr_precedente = NULL,
                   ptr_corrente = L,
                   testa = L,
                   temp;

    while (ptr_corrente != NULL) {

        if (ptr_corrente->p.prezzo < prezzo_minimo) {

          temp = ptr_corrente;
          if (ptr_precedente == NULL)
            testa = ptr_corrente->next;
          else
            ptr_precedente->next = ptr_corrente->next;
          ptr_corrente = ptr_corrente->next;
          free(temp);

        } else {

            ptr_precedente = ptr_corrente;
            ptr_corrente = ptr_corrente->next;

        }
    }

    return testa;
}
```

(c) Si scriva una funzione iterativa

```
lista_float somma(lista_prodotti L)
```

che genera una lista di numeri in virgola mobile a partire dalla lista di prodotti passata come parametro. L'elemento in posizione *i*-esima della lista di numeri in virgola mobile ha come valore la somma dei prezzi di tutti gli elementi della lista di prodotti passata come parametro fino alla posizione *i*. *(3 punti)*

Esempio:
Se la prima lista contenesse 4 prodotti con i seguenti prezzi:
`12.00  3.00  20.00  5.00`
La funzione costruirebbe una seconda lista contenete i seguenti valori:
`12.00  15.00(=12+3)  35.00(=12+3+20)  40.00(=12+3+20+5)`

Soluzione:

```
lista_float somma(lista_prodotti L) {

    lista_float testaListaFloat = NULL;
    nodo_float *ptr = NULL, *aux;
    float totaleCorrente = 0.0;

    while (L != NULL) {
        totaleCorrente += L->p.prezzo;
        aux = (nodo_float*) malloc(sizeof(nodo_float));
        aux->v = totaleCorrente;
        aux->next = NULL;
        if (ptr == NULL) {
          ptr = aux;
          testaListaFloat = aux;
        }
        else {
          ptr->next = aux;
          ptr = ptr->next;
        }
        L = L->next;
    }
    return testaListaFloat;
}
```

Esercizio 2 (8 punti)

Si consideri un file di testo, composto da 11 righe, creato per memorizzare le estrazioni del gioco del lotto in una certa data, come mostrato nell'esempio seguente:

```
BARI 26 55 30 41 15
CAGLIARI 83 44 43 29 37
FIRENZE 35 65 61 2 47
GENOVA 57 28 62 59 37
MILANO 62 42 56 12 21
NAPOLI 55 19 59 72 18
PALERMO 48 34 62 41 55
ROMA 81 53 50 27 51
TORINO 40 84 38 64 35
VENEZIA 5 28 52 3 53
NAZIONALE 39 3 84 75 26
```

Si considerino le seguenti definizioni del tipo di dato strutturato `t_ruota` e del tipo array con `11` celle `t_estrazioneSettimanale`:

```
typedef struct {
                  char nomeRuota[10+1];
                  int numeri[5];
} t_ruota;

typedef t_ruota t_estrazioneSettimanale[11];
```

Si scriva un sottoprogramma C avente come parametri il nome di un *file* di testo (assunto essere stato già creato, nello stesso formato mostrato sopra) e una variabile di tipo `t_estrazioneSettimanale`, con il seguente prototipo:

```
… conteggioAssenti(char *fileName, t_estrazioneSettimanale vett)
```

Il sottoprogramma deve aprire il file `fileName`, copiarne il contenuto nell'array `vett`, e restituire il conteggio dei valori (quanti sono) da 1 a 90 che **NON** compaiono nell'estrazione considerata.

Soluzione:

```
int conteggioAssenti(char nomefile[], t_estrazioneSettimanale vett) {
```

```
  FILE *fp = fopen(nomefile, "r"); // apertura file in sola lettura

  if ( fp == NULL ) {              // controllo d'errore su fopen
    fprintf(stderr, "\n Errore nell' aprire il file!\n ");
    exit(-1);
  }

  int r = 0;                   // assumera' valore 11 al termine del ciclo
  while ( feof(fp) == 0 ) {
    fscanf(fp, "%s", vett[r].nomeRuota);
    for (int j = 0; j < 5; j++)
       fscanf(fp, "%d", &vett[i].numeri[j]);
    r += 1;
  }

  fclose(fp);

  int cont = 0;

  for (int num = 1; num <= 90; num++) {

     flag = 1;

     for (r = 0; (r < 11) && (flag == 1); r++) {
        for (int j = 0; (j < 5) && (flag == 1); j++)
          if (num == vett[r].numeri[j])
            flag = 0;
     } // end for in r

     if (flag == 1)
       cont += 1;
   }
   return cont;
} // end conteggioAssenti
```

Esercizio 3 (5 punti)

Si scriva in C un sottoprogramma

```
...ruotePesanti(...)
```

che riceva come parametri in ingresso un valore intero chiamato `soglia` e una variabile `vett` di tipo `t_estrazioneSettimanale` (vedi def. nella traccia dell'esercizio 2) e restituisca:

- un vettore dinamico di tipo `t_ruota*` contenente la copia delle celle di `vett` in cui la somma dei numeri estratti superi il valore `soglia`.
- la lunghezza del vettore dinamico (numero di elementi presenti nel vettore).

N.B.: è richiesta la **corretta** definizione dei parametri in ingresso ed in uscita dal sottoprogramma.

Soluzione

```
t_ruota* ruotePesanti(int soglia,
                      t_estrazioneSettimanale vett,
                      int* lung) {

     int supera[11] = {0}, cont = 0;
    /*
     l'array supera[…] viene usato di seguito come array di valori 0 o 1:
     se una sua cella conterra' 0 allora si sara' verificato che la cella
     di vett[….] con lo stesso indice ha somma degli estratti che NON
     supera la soglia

     se una sua cella conterra' 1 allora si sara' verificato che la cella
     di vett[….] con lo stesso indice ha somma degli estratti che supera
     la soglia

     La variabile cont, conteggia il numero di celle non nulle dell'array
     supera[…]
    */

    for (int i = 0; i < 11; i++) {
       int sum = 0;
       for (int j = 0; j < 5; j++) {
          sum += vett[i].numeri[j];
       }
       if (sum > soglia) {
         supera[i] = 1;
         cont++;
       }
    } // end for

    *lung = cont; // valore di ritorno passato per indirizzo
    if (cont == 0) return NULL;

    t_ruota* ret = (t_ruota*) malloc(cont * sizeof(t_ruota));
    if (ret == NULL) return NULL;

    int k = 0;
    for (int i = 0; i < 11; i++) {
       if (supera[i] == 1) {
        ret[k] = vett[i];
        k++;
       }
    } // end for

    return ret;

} // end ruotePesanti
```

Esercizio 29 (12 punti)

In enigmistica la cerniera è uno schema per cui, date due parole, si ottiene una terza parola scartando il medesimo gruppo di lettere (e nel medesimo ordine) dalla testa (prefisso) della prima parola e dalla coda (suffisso) della seconda parola, unendo infine i caratteri rimasti ordinatamente nella prima e nella seconda parola. Per es. "**abc**def" /gh**abc**" restituisce "defgh".

1) **[3punti]** Scrivere in C il sottoprogramma

   `...CercaIndice(...)`

   avente come argomenti un carattere e una stringa, che restituisca la posizione della prima occorrenza del carattere nella stringa scorrendola da destra verso sinistra. Se il carattere non è presente nella stringa data, il sottoprogramma restituisce il valore -1.

   Esempi:
   il sottoprogramma invocato con argomenti `'t'`, `"tatto"` restituisce `3`;
   il sottoprogramma invocato con argomenti `'b'`, `"tatto"` restituisce `-1`;

2) **[6 punti]** Scrivere in C il sottoprogramma

   `…cerniera(…)`

   che abbia come parametri due stringhe e che indichi se la cerniera tra le due parole in esse contenute è possibile.
   Per indicare che la cerniera è possibile, il sotto-programma deve restituire la lunghezza della comune sequenza di caratteri che risulta essere a prefisso della prima stringa e a suffisso della seconda; mentre deve restituire -1 nel caso in cui la cerniera non sia possibile.

   Esempi:
   `"osti"`, `"mao"` → restituisce `1` (la loro cerniera è : `"stima"`)
   `"mare"`, `"tema"` → restituisce `2` (la loro cerniera è : `"rete"`)
   `"drago"`, `"ladra"` → restituisce `3` (la loro cerniera è ': `"gola"`)
   `"manico"` , `"stamani"` → restituisce `4` (la loro cerniera è : `"costa"`)
   `"papa"`, `"papa"` → restituisce `2` (la loro cerniera è : `"papa"`)
   `"ciao"`, `"ciao"` → restituisce `4` (la loro cerniera è : parola vuota)
   `"ciao"`, `"per"` → restituisce `-1` (la loro cerniera NON è possibile)

   Suggerimento:
   - Se si assume (come negli esempi) di fare la cerniera considerando prefissi e suffissi SENZA ripetizioni di caratteri, l'idea di soluzione potrebbe far uso del sotto-programma richiesto in 1)
   - È possibile far uso delle funzioni incluse nella libreria `string.h` (per es. per ottenere la lunghezza di una stringa).

3) **[3punti]** Scrivere in C un programma principale che faccia uso dei sotto-programmi precedenti e che chieda all'utente di inserire due parole. Il programma deve riportare a video un messaggio che indichi se la cerniera delle due parole sia o non sia possibile, e in caso affermativo visualizzarla.

Soluzione Domanda 1)

```
int cercaIndice(char c, char str[]) {
   int idx = strlen(str)-1;
   while (idx >= 0 && c != str[idx]) {
       idx = idx - 1;
   }
   return idx;
} // end cercaIndice
```

Soluzione Domanda 2)

```
// Assumendo di fare la cerniera considerando
// prefissi e suffissi SENZA ripetizioni di caratteri.
int cerniera(char prima[], char seconda[]) {

   int idxSuffisso = cercaIndice(prima[0], seconda);
   if (idxSuffisso < 0) return -1;

   int lung = strlen(seconda) - idxSuffisso;
   int flag = 1;
   for (int i = 1; i < lung && flag == 1; i++)
      if (prima[i] != seconda[idxSuffisso+i])
        flag = 0;

   if (flag == 0) return -1;
   return lung;
} // end cerniera

// Soluzione generale (NON RICHIESTA)- senza far uso di ...cercaIndice(...)
int cerniera(char prima[], char seconda[]) {

   int idxSeconda = strlen(seconda)-1;
   if (idxSeconda < 0) return -1;

   int riconoscimento_iniziato_su_prima, idxPrima, lung = 0;

   riconoscimento_iniziato_su_prima = idxPrima = strlen(prima)-1;
   while ( idxPrima >= 0 && idxSeconda >= 0) {
     if (prima[idxPrima] == seconda[idxSeconda]) {
       if (lung == 0) // fase di inizio riconoscimento
          riconoscimento_iniziato_su_prima = idxPrima;
       lung++;
       idxPrima--;
       idxSeconda--;
     }
     else {
            idxSeconda = strlen(seconda)-1;
            idxPrima = riconoscimento_iniziato_su_prima - 1;
            riconoscimento_iniziato_su_prima = idxPrima;
            lung = 0;
     } // end 1mo-if-else
    } // end while
    if (lung == 0)
      return -1;
    else
      return lung;
} // end cerniera
```

Soluzione Domanda 3)

```
#include <stdio.h>
#include <string.h>
#define MAX_LUNG_PAROLA 50
```

```c
int cercaIndice(char c, char str[]);
int cerniera(char prima[], char seconda[]);

int main() {
   char primaParola[MAX_LUNG_PAROLA+1] = {'\0'};
   char secondaParola[MAX_LUNG_PAROLA+1] = {'\0'};

   printf("\n Programma di valutazione della cerniera tra due parole \n");
   printf("\n Inserisci la 1ma parola: "); scanf("%s", primaParola);
   printf("\n Inserisci la 2da parola: "); scanf("%s", secondaParola);

   int lung = cerniera(primaParola, secondaParola);

   if (lung < 0) {
     printf("\n La cerniera NON e' possibile! \n");
     return 1;
   }
   if (lung == strlen(primaParola) && lung == strlen(secondaParola)) {
     printf("\n La cerniera e' la stringa vuota! \n");
   }
   else {
         printf("\n La cerniera e': ");
         for (int i = lung; i < strlen(primaParola); i++)
            printf("%c", primaParola[i]);
         for (int i = 0; i < strlen(secondaParola)-lung; i++)
            printf("%c", secondaParola[i]);
         printf("\n");
   }
   return 1;
} // end main
```

Esercizio 13 (5 punti)

Le seguenti dichiarazioni di tipo permettono la rappresentazione dei dati necessari per l'implementazione di una piccola rete sociale che può essere descritta come un insieme di persone e di relazioni di amicizia. Il numero massimo di utenti nell'intera rete sociale è definito con la direttiva al pre-processore C: `MAX_UTENTI`.

Ogni persona è descritta da un `nome`, un `cognome`, un `nickname` e un numero intero positivo che la identifica univocamente (`id`).
Le relazioni di amicizia di ciascuna persona sono descritte tramite un *array* (`amici[...]`) contenente gli identificatori di tutti i suoi amici e un attributo `num_amici` che tiene traccia del numero di amici presenti effettivamente nell'*array.* Ogni utente può avere al più `MAX_AMICI`.

Un utente del sistema è dunque descritto da un tipo di dato (`t_utente`) che aggrega le informazioni di una persona e le sue relazioni di amicizia; mentre, la rete sociale è descritta come un tipo di dato *array* in cui ogni cella contiene le informazioni di un utente.

```
#define MAX_UTENTI 200
#define MAX_AMICI  200
#define LUN_NOME   50

typedef struct {
                char nome[LUN_NOME], cognome[LUN_NOME];
                char nickname[LUN_NOME];
                unsigned int id;                // identificatore univoco
                unsigned int num_amici;         // numero di amici
                unsigned int amici[MAX_AMICI];  // array con gli id degli amici
        } t_utente;

typedef t_utente t_rete_sociale[MAX_UTENTI];    // array di utenti
```

Si codifichi in C la funzione

```
int cerca_utente(unsigned int id, t_rete_sociale rete, unsigned int num_utenti);
```

che, dato l'identificatore univoco di un utente (`id`), una rete sociale (`rete`) e il numero di utenti effettivamente presenti nella rete sociale (`num_utenti`), restituisca il valore intero che rappresenta la posizione dell'utente nell'*array* `rete`, oppure `-1` se l'utente non è presente nell'*array*.

Soluzione

```
int cerca_utente(unsigned int id, t_rete_sociale rete, unsigned int num_utenti){

  if (num_utenti > MAX_UTENTI)  // condizione di errore:
    return -1;                  // il valore del 3zo parametro e' troppo grande!

  unsigned int i = 0;
  while (i < num_utenti && rete[i].id != id)
       i++;

  if (i == num_utenti)          // se l'utente non e' presente nell'array
    i = -1;

  return i;
}
```

Esercizio 14 (8 punti)

Dati gli identificatori di due utenti e la rete sociale a cui appartengono, codificare in C la funzione

```
void amici_comuni(unsigned int id1,
                  unsigned int id2,
                  t_rete_sociale rete , unsigned int num_utenti);
```

che stampa a video l'elenco degli amici **in comune** dei due utenti `id1, id2`.
Stampare un amico per riga e fornire nome, cognome e *nickname* per ciascun amico trovato.
Dove appropriato, è consigliato il ri-uso della funzione `cerca_utente(...)`, come dichiarata nell'esercizio precedente.

Soluzione

```
void amici_comuni(unsigned int id1,
                  unsigned int id2,                        // identificatori utenti
                  t_rete_sociale rete, unsigned int num_utenti)  // rete sociale
{
  int i = 0;                        // indice nell'array dell arete sociale
  int pos1 = pos2 = -1;             // inizializzazione indici di posizione degli
                                    // utenti con gli identificatori dati

  // ricerca posizione utenti conoscendo gli id
  while ( i < num_utenti && (pos1 < 0 || pos2 < 0)) {
    if (rete[i].id == id1) pos1 = i;
    if (rete[i].id == id2) pos2 = i;
    i++;
  }
  if (pos1 == -1) {
    printf("L'utente con id: %d, non esiste nella rete sociale!\n\n", id1);
    return;
  }
  if (pos2 == -1) {
    printf("L'utente con id: %d, non esiste nella rete sociale!\n\n", id2);
    return;
  }

  printf("Gli amici in comune dei due utenti sono:\n\n");
  int amico, flag_almeno_uno = 0;
            // ricerca degli id degli amici di utente1 fra gli amici di utente2
  for (i = 0; i < rete[pos1].num_amici; i++) {
    for (int j = 0; j < rete[pos2].num_amici; j++) {

      if (rete[pos1].amici[i] == rete[pos2].amici[j]) {
                                    // ottieni indice amico e stampa info amico
        amico = cerca_utente(rete[pos1].amici[i], rete, num_utenti);
        if (amico != -1) {
          printf("%s %s, %d anni\n", rete[amico].nome,
                                     rete[amico].cognome,
                                     rete[amico].nickname);
          flag_almeno_uno = 1;
        } // end if
      } // end if

    } // end for
  } // end for
                    // se non é stato trovato nessun amico in comune, stampa info
  if (flag_almeno_uno == 0)
    printf("\n I due utenti non hanno amici in comune.\n");
}
```

Esercizio 15 (6 punti)

Assumiamo ora che la rete sociale non sia più modellizzata con un tipo di dato "*array* di utenti", ma con un tipo di dato "lista dinamica di utenti", come segue.

```
#define MAX_UTENTI 200
#define MAX_AMICI  200
#define LUN_NOME   50

typedef struct {
                char nome[LUN_NOME], cognome[LUN_NOME];
                char nickname[LUN_NOME];
                unsigned int id;                  // identificatore univoco
                unsigned int num_amici;           // numero di amici
                unsigned int amici[MAX_AMICI];    // array con gli id degli amici
        } t_utente;

typedef struct t_struct_nodo {
                 t_utente info;
                 struct t_struct_nodo* next;
        } t_nodo;

typedef t_nodo* lista;
```

Definire un sottoprogramma che abbia un parametro di tipo `lista` e che restituisca il numero di amici dell'utente più popolare della rete sociale.

Soluzione

```
unsigned int num_amici_utente_piu_popolare(lista rete) {
     unsigned int massimo = 0;

     while (rete != NULL) {
       if ( rete->info.num_amici > massimo )
          massimo =  rete->info.num_amici;

       rete = rete->next;
     } // end while
     return massimo;
}
```

Esercizio 4 (5 punti)

Si assuma che la rete sociale sia rappresentanta con il tipo di dato "lista dinamica di utenti" specificato nell'esercizio 3.
Si scriva un sottoprogramma `... stampa_popolari(...)` che abbia un parametro di tipo `lista` e che stampi a schermo il *nickname* dell'utente più popolare (o i *nickname* degli utenti più popolari, se ve né più di uno).
Il sottoprogramma deve far uso di un ulteriore sottoprogramma **ricorsivo** `...stampa_con_max_amici(...)` che abbia come parametri la lista che rappresenta la rete sociale e il numero massimo di amici che un utente ha nella rete stessa.

Soluzione

```
void stampa_con_max_amici(lista rete, unsigned int massimo) {
    if (rete == NULL) return;
    if (rete->info.num_amici == max) {
      printf("%s\n", rete->info.nickname);
    }
    stampa_con_max_amici(rete->next, massimo);
}

void stampa_popolari(lista rete) {
    unsigned int max = num_amici_utente_piu_popolare(rete);
    stampa_con_max_amici(rete, massimo);
}
```

Contesto

Il G.I.S. (*Geographical Information System*), o sistema informativo geografico, è uno strumento informatico che permette di rappresentare, analizzare, e interrogare entità o eventi che si verificano sul territorio.
In questo contesto, un modello digitale di elevazione rappresenta la mappa di un territorio attraverso una matrice rettangolare con `NUMR` × `NUMC` celle, in cui ogni cella memorizza un valore in virgola mobile corrispondente alla misura, in centimetri, della quota sul livello del mare rilevata nell'omologo punto sul territorio.

Si considerino dunque i seguenti tipi di dato, utili per rappresentare in `C` la mappa di un territorio.

```
#define NUMR  50
#define NUMC  80

typedef float mappa[NUMR][NUMC];
typedef struct { int r, c; } posizione;
```

La mappa, `m`, di un qualunque territorio è assunto che abbia un numero di righe minore o uguale a `NUMR` e un numero di colonne minore o uguale a `NUMC`.
Un **percorso carrabile valido**, `pcv`, è definito come un vettore di tipo `posizione`, in cui celle consecutive memorizzano posizioni a cui corrispondono celle contigue di `m` (in orizzontale o in diagonale) e aventi differenze di quota minore o uguale a 15 cm.

Nota:
Per semplicità, su una mappa si considerano solo percorsi che vanno da sinistra a destra, cioé dalla colonna `0` alla colonna (`NUMC-1`), senza cicli e senza tratti verticali.
Dunque, nella creazione di un percorso, essendo possibili solo movimenti in avanti, se si parte dalla posizione `[riga][colonna]` ci si può spostare solo in una cella di posizione

- `[riga-1][colonna+1]` (solo se la riga non è la prima)
- `[riga][colonna+1]`
- `[riga+1][colonna+1]` (solo se la riga non è l'ultima)

Esercizio 27 (8 punti)

Si scrivano in C i due sottoprogrammi seguenti.

- Considerando come parametri in ingresso una mappa `m` e un vettore di posizioni, il sottoprogramma

  `...valido(...)`

  deve ritornare al chiamante il valore intero `1`, se il vettore di posizioni rappresenta un percorso carrabile valido sulla mappa; in caso contrario deve ritornare `0`.

- Considerando come parametri in ingresso una mappa `m` e un vettore di posizioni rappresentante un percorso valido, il sottoprogramma

  `...stampaPercorso(...)`

  deve visualizzare a video la mappa del territorio, riportando ogni cella lungo il percorso con un carattere `'*'` e ogni altra cella con un carattere `'.'`.

Soluzione:

```
float absDiff(float v1, float v2) {
    if (v1 < v2) return v2-v1;
    else return v1-v2;
}

int valido(mappa m, posizione p[NUMC]) {
   for (int j = 1; j < NUMC; j++) {
      if ( p[j].c != 1 + p[j-1].c          ||
           absDiff(p[j].r, p[j-1].r) > 1   ||
           absDiff(m[p[j].r][p[j].c],
                   m[p[j-1].r][p[j-1].c]) > 15
         )
        return 0;
   }
   return 1;
}

void stampaPercorso(mappa m, posizione p[NUMC]) {
    if ( valido(mappa, p) == 0 ) {
      printf("\n Il percorso indicato non e' valido ! \n");
      return;
    }
    for (int i = 0; i < NUMR; i++) {
      for (int j = 0; j < NUMC; j++) {
         if (i == p[j].r && j == p[j].c)
           printf("*");
         else
           printf(".");
      }
      printf("\n");
    }
}
```

Esercizio 2 (10 punti)

Si assuma di avere a disposizione un *file* di caratteri che contiene la mappa di un territorio.
Il *file* contiene valori di quota organizzati in esattamente `NUMR` righe e `NUMC` colonne, più precisamente:

- la prima riga contiene `NUMC` valori di quota da memorizzare nella prima riga della matrice; ciascun valore è separato dal successivo da uno spazio (`' '`), tranne l'ultimo valore che è invece seguito da un fine-riga (`'\n'`);
- la seconda riga contiene i valori di quota da memorizzare nella seconda riga della matrice ; ciascun valore è separato dal successivo da uno spazio (`' '`), tranne l'ultimo valore che è invece seguito da un fine-riga (`'\n'`); ecc.

Si assuma inoltre di avere a disposizione il sottoprogramma

```
int cercaPercorso(mappa m, int riga, int col, posizione cammino[NUMC]);
```

la cui invocazione consente di cercare un percorso carrabile valido a partire dalla posizione `[riga][col]` sulla mappa `m`, e memorizzare nell'*array* `cammino` le posizioni che lo compongono. Il sottoprogramma ritorna `1` se un attraversamento valido viene trovato, `0` altrimenti.

Scrivere un programma principale, `main(){ ... }`, che chieda all'utente il nome del *file* contenete la mappa di un territorio, lo apra e carichi in una variabile `m`, di tipo mappa, la matrice in esso riportata. Il programma deve inoltre riportare a video i percorsi validi che vanno dalla colonna `0` alla colonna `NUMC-1` sulla mappa.

Nota: fare uso del sottoprogramma `...cercaPercorso(...)` assunto a disposizione per quest'esercizio e del sottoprogramma `...stampaPercorso(...)`, la cui funzionalità è stata descritta nell'esercizio precedente.

Soluzione:

```
#include <stdio.h>   // per utilizzare: scanf, printf, fscanf,
                     // fprintf, fopen, ferror, fclose
#include <stdlib.h>  // per utilizzare: exit

#define NUMR  50
#define NUMC  80
typedef float mappa[NUMR][NUMC];
typedef struct { int r, c; } posizione;

int cercaPercorso(mappa, int, int, posizione cammino[NUMC]);
int valido(mappa, posizione p[NUMC]);
void stampaPercorso(mappa, posizione p[NUMC]);
```

```
int main() {

   char nomeFile[30+1] = { '\0' };
```

```
    printf("\n Nome del file con la mappa(max 30car): ", nomeFile);
    scanf("%31s", nomeFile);

    FILE* fp = fopen(nomeFile, "r");
    if ( fp == NULL ) {
      fprintf(stderr, "\n Il file %s non esite o e' corrotto! ");
      exit(-1);
    }

    mappa m;

    for (int i = 0; i < NUMR; ++i) {
       for (int j = 0; j < NUMC; ++j) {
          fscanf(fp, "%f", &m[i][j]);
          if ( ferror(fp) != 0 ) {
            fprintf(stderr, "\n Errore nella lettura da file! ");
            exit(-1);
          } // end if
       } // end for j
    } // end for i

    fclose(fp);

    int trovatoValido = 0, almenoUno = 0;
    posizione cammino[NUMC] = { 0 };

    for (int i = 0; i < NUMR; i++) {
       trovatoValido = cercaPercorso(m, i, 0, cammino);
       if ( trovatoValido == 1 ) {
         almenoUno = 1;
         printf("\n Percorso con inizio da cella [%d][%d]\n", i, 0);
         stampaPercorso(m, cammino);
       } // end if
    } // end for i

    if ( almenoUno == 0 )
      printf("\n Non ci sono percorsi carrabili validi !\n");

    return 0;

} // end main


// ... qui la definizione dei sottoprogrammi utilizzati ...
```

Esercizio 3 (6 punti)

Scrivere un sottoprogramma `C` che prendendo come parametri una mappa `m` e un vettore di posizioni, restituisca una lista dinamica semplicemente concatenata non vuota, se il vettore contiene un percorso valido. I nodi della lista dinamica devono memorizzare le posizioni presenti nel percorso nello stesso ordine in cui compaiono nel vettore di posizioni.

Definire anche le strutture dati opportune, tenendo conto che ogni nodo della lista deve memorizzare una posizione lungo il percorso e il valore di quota a esso corrispondente sulla mappa `m`.

Soluzione:

```
typedef struct nodo_t { posizione p;
                        float quota;
                        struct nodo_t* next;
        } nodo;

nodo* creaLista(mappa m, posizione cammino[NUMC]) {

     nodo *testa = NULL, *nuovo, *ultimo;

     if (valido(m, cammino) == 0) return testa;

     for (int y = 0; y < NUMC; y++) {
        nuovo = (nodo*) malloc(sizeof(nodo));
        nuovo->p = cammino[y];
        nuovo->quota = m[(nuovo->p).r][(nuovo->p).c];
        nuovo->next = NULL;

        if (testa == NULL) {
          testa = ultimo = nuovo;
        }
        else {
          ultimo->next = nuovo;
        }

     return testa;
}
```