Esercizio 1: Banca (1)

Una banca vuole realizzare un programma di ***simulazione di gestione conto-corrente***

A tal scopo il programma simula l'esistenza di un conto corrente, inizialmente vuoto, e offre un menù all'utente in cui si può scegliere di compiere ***4 operazioni diverse***:

- deposito
- prelievo
- visualizzazione del saldo
- calcolo degli interessi composti

L'utente sceglie l'operazione richiesta o l'intenzione di terminare la simulazione specificando un numero o un carattere

Dopo ogni operazione il programma stampa a video l'elenco delle operazioni possibili e attende un nuovo comando dall'utente

Esercizio 1: Banca (2)

1.L'operazione di **`deposito`**, che consiste nel chiedere all'utente una somma, controllare che sia > 0, e depositarla nel conto

2.L'operazione di **`prelievo`**, che consiste nel chiedere la somma all'utente, controllare che sia > 0 e che ci siano fondi a sufficienza nel conto corrente, e prelevarla dal conto

3.L'operazione di **`visualizzazione saldo`**, che consiste nel visualizzare il saldo del conto corrente. Questa operazione deve essere eseguita automaticamente al termine delle operazioni di deposito e prelievo

Esercizio 1: Banca (3)

4. L'operazione di **`calcolo degli interessi`**, che consiste nel calcolare a quanto ammonterebbe il capitale finale qualora si investisse il capitale iniziale (pari ai soldi depositati sul conto) per **`N`** anni con un tasso di interesse pari a **`r`**

Il programma deve quindi chiedere all'utente il tasso di interesse e il numero di anni su cui effettuare la simulazione

È necessario controllare che i dati inseriti corrispondano a valori plausibili (tasso d'interesse >= 0 e <= 100) (anni > 0)

La formula per il calcolo degli interessi composti, dove **`r`** è il tasso di rendimento e **`N`** è il numero di anni a cui gli interessi sono applicati, è la seguente:

$$Cfin = Cin * \left(1 + \frac{r}{100}\right)^{N}$$

Esercizio 1: Soluzione (1)

```
#include <stdio.h>

#define DEPOSITO 1
#define PRELIEVO 2
#define SALDO 3
#define INTERESSI 4
#define FINE 0

void stampa_menu();
void f_deposito(float * soldi);
void f_prelievo(float * soldi);
void f_saldo(float soldi);
void f_interessi(float soldi);
float interessi(float c_in, float tasso, int anni);
```

Esercizio 1: Soluzione (2)

```
int main()
{
   int op;     float soldi = 0;
   stampa_menu();
   printf("Scegliere operazione: ");   scanf("%d", &op);
   while (op != FINE)
   {
      switch (op)
        {
         case DEPOSITO:
            f_deposito(&soldi); break;
         case PRELIEVO:
            f_prelievo(&soldi); break;
         case SALDO:
            f_saldo(soldi); break;
         case INTERESSI:
            f_interessi(soldi); break;
         default:
            printf("Operazione non consentita");
      }
      stampa_menu();
      printf("Scegliere operazione: ");
      scanf("%d", &op);
   }
   return 0;
}
```


Esercizio 1: Soluzione (3)

```
void stampa_menu()
{
      printf("Operazioni disponibili:\n");
      printf("%d.\tDeposito\n", DEPOSITO);
      printf("%d.\tPrelievo\n", PRELIEVO);
      printf("%d.\tSaldo\n", SALDO);
      printf("%d.\tCalcolo interessi\n", INTERESSI);
      printf("%d.\tFine\n", FINE);
}
```


Esercizio 1: Soluzione (4)

```
void f_deposito(float * soldi)
{
      float cash;

      printf("Digitare l'ammontare del deposito: ");
      scanf("%f", &cash);
      while (cash < 0)
      {
            printf("Ammontare negativo.\n");
            printf("Digitare l'ammontare del deposito: ");
            scanf("%f", &cash);
      }
      *soldi = *soldi + cash;
      f_saldo(*soldi);
}
```


Esercizio 1: Soluzione (5)

```
void f_prelievo(float * soldi)
{
      float cash;
      printf("Digitare l'ammontare del prelievo: ");
      scanf("%f", &cash);
      while (cash < 0 || cash > *soldi)
      {
            printf("Ammontare negativo /
                  Non ci sono cosi' tanti soldi. .\n");
            printf("Digitare l'ammontare del prelievo: ");
            scanf("%f", &cash);
      }
      *soldi = *soldi - cash;
      f_saldo(*soldi);
}

void f_saldo(float soldi)
{
   printf("Disponibilita' nel conto corrente: %6.2f\n", soldi);
}
```

Esercizio 1: Soluzione (6)

```
void f_interessi(float soldi)
{
      float tasso, c_fin;     int anni;

      printf("Inserire il tasso d'interesse: ");
      scanf("%f", &tasso);
      while ((tasso<0) || (tasso>100))
      {
            printf("Tasso errato\n ");
            scanf("%f", &tasso);
      }

      printf("Inserire anni: ");
      scanf("%d", &anni);
      while (anni<0)
      {
            printf("Anni negativi.\n");
            scanf("%d", &anni);
      }
```


Esercizio 1: Soluzione (7)

```
   c_fin = interessi(soldi, tasso, anni);
   printf("Interessi simulati, con:\n");
   printf("Capitale iniziale: %6.2f\n", soldi);
   printf("Tasso d'interesse: %6.2f\n", tasso);
   printf("Anni di applicazione degli interessi: %d\n", anni);
   printf("Capitale finale: %6.2f\n", c_fin);
}

float interessi(float c_in, float tasso, int anni)
{
      float tasso_perc, tasso_tot, result=1;    int i;

      tasso_perc = tasso/100;
      tasso_tot = 1 + tasso_perc;

      for (i=0; i<anni; i++)
            result=result*tasso_tot;

      return c_in*result;
}
```


Esercizio 2: Autobus (1)

Una compagnia di autobus che effettua servizio su lunghe distanze vuole realizzare un ***programma di controllo delle prenotazioni dei posti***

A tal scopo rappresenta ogni prenotazione tramite una struttura **`booking`** contenente nome del cliente (al massimo 1023 caratteri, senza spazi) e numero del posto prenotato (un intero)

Le prenotazioni effettuate vengono registrate tramite ***un array (di dimensione prefissata `DIM`)*** di strutture **`booking`**, di dimensione logica iniziale pari a 0

Esercizio 2: Autobus (2)

Si realizzi una funzione:

```
int assegna( booking list[], int * lengthList, char * name, int pref);
```

La funzione riceve in ingresso l'array di prenotazioni, la sua dimensione logica, e il nome del cliente ed il posto da lui indicato
La funzione deve controllare che il posto indicato non sia già stato assegnato e, in caso contrario, deve restituire il valore 1
Qualora invece ***il posto sia ancora libero, la funzione deve assegnare tale posto al cliente*** copiando i dati della prenotazione nell'ultima posizione libera nell'array; deve inoltre aggiornare correttamente la dimensione logica dell'array
In questo secondo caso la funzione deve ***restituire come valore `0`***, per indicare il successo nella prenotazione

Al fine di copiare il nome del cliente, si utilizzi la funzione di libreria

```
char * strcpy(char * s, char * ct);
```

che copia **`ct`** in **`s`** (terminatore compreso)

Esercizio 2: Autobus (3)

Si realizzi un programma **main()** che chieda all'operatore il nome di un utente e il posto prescelto. Il programma deve cercare di registrare la prenotazione tramite la funzione **assegna()**; qualora l'operazione di prenotazione fallisca (perché il posto risulta essere già assegnato), il programma chieda all'operatore un nuovo posto, fino ad effettuare la prenotazione

Qualora l'operatore inserisca il nome "fine", il programma deve terminare; qualora invece venga inserita la stringa "stampa", il programma deve stampare a video le prenotazioni già effettuate
A tal scopo si usi la funzione di libreria:

```
int strcmp(char * ct, char * cs);
```

che restituisce 0 se e solo se le due stringhe sono identiche

Esercizio 2: Soluzione (1)

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

#define MAX 1024
#define DIM 10


typedef struct {
      char name[MAX];
      int seat;
} booking;
```


Esercizio 2: Soluzione (2)

```
int assegna(booking list[], int * lengthList,
                              char * name, int pref)
{
      int i=0, trovato = 0;

      while (( i < *lengthList) && !trovato)
      {
            if (list[i].seat == pref)
                  trovato = 1;
            i++;
      }
      if (!trovato)
      {
            list[*lengthList].seat = pref;
            strcpy(list[*lengthList].name, name);
            (*lengthList)++;
            return 0;
      }
      else return 1;
}
```


Esercizio 2: Soluzione (3)

```
int stampaBooking(booking list[], int lengthList)
{
      int i=0;
      for (i=0; i<lengthList; i++)
            printf("%s: %d\n", list[i].name, list[i].seat);
      return 0;
}

int main()
{
      booking list[DIM];
      int alengthList = 0;
      char nome[MAX];
      int seat = 0;

      printf("Inserire Nome Passeggero: ");
      scanf("%s", nome);
```

Esercizio 2: Soluzione (4)

```
    while (strcmp(nome, "fine"))
    {
          if (strcmp(nome, "stampa"))
          {
             printf("Posto preferito: ");
             scanf("%d%*c", &seat);
             while(assegna(list, &alengthList, nome, seat))
             {
                printf("posto scelto e' gia' occupato.\n");
                printf("specificare un altro posto: ");
                scanf("%d%*c", &seat);
             }
          }
          else stampaBooking(list, alengthList);

          printf("Inserire Nome Passeggero: ");
          scanf("%s", nome);
    }
    return 0;
}
```


Esercizio 3: Matrici A

Una compagnia aerea tiene traccia, ogni giorno, delle prenotazioni dei posti sui suoi voli. In particolare, in una matrice di interi, di dimensione fissata **MAX_FLIGHTS** x **MAX_SEATS**, tiene traccia di quante prenotazioni sono state fatte sull'i-esimo aereo (**0<=i<=MAX_FLIGHTS**), per il posto j-esimo (**0<=j<=MAX_SEATS**)

Capita spesso che uno stesso posto venga richiesto da più clienti: in tal caso la compagnia all'atto della prenotazione concede lo stesso posto a tutti i richiedenti, salvo poi comunicare all'ultimo uno spostamento obbligatorio di posto
Quindi può succedere che:

- se il valore in posizione [i][j] è 0, significa che nessuno ha prenotato il posto j-esimo per il volo i-esimo
- se il valore in posizione [i][j] è 1, significa che una persona ha prenotato il posto j-esimo per il volo i-esimo
- se il valore in posizione [i][j] è 3, significa che tre persone hanno prenotato lo stesso posto j-esimo per il volo i-esimo

Esercizio 3: Matrici A

Si realizzi un programma che:

1.chieda all'utente il codice di un volo (si supponga per semplicità che tale codice coincida con l'indice della matrice), e calcoli se tale volo è effettivamente in overbooking (cioè se la somma di tutte le prenotazioni per quel volo è maggiore della costante **MAX_SEATS**), o se invece vi sono posti disponibili per tutti coloro che hanno prenotato

2.calcoli (e stampi a video) ***qual è il posto più richiesto***, rispetto a tutti i voli della giornata

Al fine di realizzare i punti 1 e 2, il candidato implementi delle opportune funzioni, e poi mostri un programma **main()** di esempio che ne faccia uso

Esercizio 3: Soluzione (1)

```
#define TRUE    1
#define FALSE   0

#define MAX_SEATS 5
#define MAX_FLIGHTS 3

typedef int BOOL;

BOOL canFly(int flightIndex, int bookingMatrix[][MAX_SEATS]) {
      int totalBooking = 0, j;
      for (j=0; j<MAX_SEATS; j++)
            totalBooking = bookingMatrix[flightIndex][j]
                                                + totalBooking;
      if (totalBooking <= MAX_SEATS)
            return TRUE;
      else
            return FALSE;
}
```

Esercizio 3: Soluzione (2)

```
int mostWantedSeat(int bookingMatrix[MAX_FLIGHTS][MAX_SEATS])
{
      int mostWanted = 0; // il posto piu' prenotato

      // il record di prenotazioni per quel posto
      int bookingRecord = 0;

      // variabile accumulatore in cui tengo traccia delle
      // prenotazioni per un certo posto
      int howManyBooking = 0;  int i, j;

      for (j=0; j<MAX_SEATS; j++)
      {
      // per ogni posto sugli aerei, ne calcolo le prenotazioni
            for (i=0; i<MAX_FLIGHTS; i++)
            {
                  // per ogni aereo, conto quante prentoazioni
                  // sono state fatte per quel posto
                  howManyBooking = howManyBooking
                        + bookingMatrix[i][j];
            }
```


Esercizio 3: Soluzione (3)

```
            if (howManyBooking > bookingRecord)
            {
                  // se batto il record precedente
                  // aggiorno il nuovo record
                  mostWanted = j;
                  bookingRecord = howManyBooking;
            }
            howManyBooking = 0;
      }
      return mostWanted;
}
```


Esercizio 3: Soluzione (4)

```
int main(void)
{
      int flight;
      int bookingMatrix[MAX_FLIGHTS][MAX_SEATS] = {
            {2,3,1,0,0},
            {1,0,0,1,1},
            {4,0,0,0,0}
      };
      printf("Insert index of the desired plane: ");
      scanf ("%d", &flight);
      if (canFly(flight, bookingMatrix))
            printf("\nThe plane is overbooked,
                        but enough space is available");
      else
            printf("\nThe plane is too much overbooked!!!\n");

      printf("The most booked place is the number %d\n",
                        mostWantedSeat(bookingMatrix));
      return (0);
}
```


Esercizio 4: Matrici B

Un centro di meteorologia tiene traccia di ***tutte le temperature registrate nell'anno*** in un vettore bidimensionale (matrice) di interi. Tale matrice è costituita di tante righe quanti sono i mesi dell'anno e ha per ogni riga spazio sufficiente a contenere il massimo numero di valori possibile, corrispondente alla durata massima di un mese in giorni

Dopo aver definito ed inizializzato correttamente tale matrice, si realizzi un programma che:

1. effettui il calcolo della **temperatura media di un mese** specificato dall'utente;
2. effettui il calcolo della **temperatura media di un giorno** (specificato dall'utente) **di tutti i mesi** (ad esempio la media delle temperature rilevate il giorno 5 di tutti i mesi)
3. effettui il calcolo della temperatura media dei giorni che risiedono **all'interno di un range** specificato dall'utente (ad es. dal 15 Febbraio al 3 Aprile)

Esercizio 4: Matrici B (Note)

Definizione e inizializzazione della variabile contenente le temperature di tutti i giorni dell'anno:

- matrice bidimensionale di interi
- 12 righe (i mesi) e 31 colonne (i giorni)

```
int temp[12][31]
```

La matrice è composta da 372 (=12*31) interi, in quanto tutti i mesi vengono considerati di 31 giorni. Alcuni giorni però non esistono (ad esempio il 30 Febbraio); per specificare che un certo valore non ha senso (poiché corrisponde ad un giorno che non esiste) viene inserito un valore di controllo (-100) al di fuori del range dei valori di temperatura validi (ad esempio [-30, +50])

Occhio agli indici:
`temp[0][0]` è la temperatura del primo giorno di Gennaio
`temp[11][1]` è la temperatura del secondo giorno di Dicembre

Esercizio 4: Soluzione (1)

```
int main()
{
      int i, j, giorno, mese, sommatoria, somGiorni, giornoIn,
                                    meseIn, giornoFin, meseFin;
      float media;

int temp[12][31] =
{
// Gennaio (31 giorni)
{5,7,8,6,2,3,4,3,2,2,4,5,3,5,7,1,3,4,7,8,1,2,3,4,2,6,8,3,1,2,3},
// Febbraio (28 giorni)
{5,7,8,6,2,3,4,3,2,2,4,5,3,5,7,1,3,4,7,8,1,2,3,4,2,6,8,3,-100,-100,-100},
{5,7,8,6,2,3,4,13,2,2,14,5,3,15,7,1,3,4,7,8,1,2,3,4,2,6,8,3,1,2,3},
{5,7,8,6,2,3,14,3,2,2,4,15,3,5,17,1,13,4,17,8,1,12,3,4,2,6,8,3,1,2,-100},
{5,7,8,16,21,3,14,3,2,2,4,5,3,15,7,1,3,4,7,8,1,2,13,14,2,6,8,3,1,2,3},
{15,27,8,6,2,3,4,3,2,2,4,5,3,5,7,11,3,4,7,8,1,2,3,4,2,6,8,3,1,2,-100},
{15,27,8,6,2,3,4,3,2,2,4,5,3,5,7,1,3,4,7,8,11,2,3,4,12,6,8,3,1,2,3},
{5,7,8,6,9,8,24,23,22,22,24,25,23,25,72,1,3,4,7,8,1,2,3,4,2,6,8,3,1,2,3},
{25,27,8,6,2,3,4,3,2,2,4,5,3,5,7,11,3,4,7,8,1,2,3,4,2,6,8,3,1,2,-100},
{5,7,8,6,2,3,4,3,2,2,4,5,3,5,7,1,3,14,7,8,1,2,3,4,2,6,8,3,1,2,3},
{5,7,8,6,2,3,4,3,2,2,4,5,3,15,7,1,3,4,7,8,1,2,3,4,2,6,8,3,1,2,-100},
{5,7,8,6,2,3,4,3,12,2,4,5,3,5,7,1,3,4,7,8,1,2,3,4,2,6,8,3,1,2,3}
};
```


Esercizio 4: Soluzione (2)

```
// Calcolo della temperatura media di un particolare mese
      printf("\nInserire un mese \n");
      printf("(1=Gennaio, 2=Febbraio, ... , 12=Dicembre)\n");
      scanf("%d", &mese);

      // Il ciclo for somma il valore delle temp. di tutti i
      // giorni del mese dato; il ciclo for itera finché non si
      // verifica una delle due seguenti condizioni di fine mese:
      // a) la temperatura è -100;
      // b) è stato raggiunto il numero massimo di giorni (31).
      sommatoria=0;
      for(j=0; temp[mese-1][j]!=-100 && j<31; j++)
      {     // e' costante il primo indice (il mese),
            // si itera sul secondo indice (il giorno)
            sommatoria+=temp[mese-1][j];
      }
      media=sommatoria/j;
      printf("numero totale di giorni %d\n",j);
      printf("sommatoria %d\n",sommatoria);
      printf("temperatura media del mese %d: %f\n\n",mese,media);
```


Esercizio 4: Soluzione (3)

```
      // Calcolo della temperatura media di un particolare giorno

      printf("giorno di cui si vuole conoscere la temp media\n");

      // il giorno inserito deve essere valido
      // per tutti i mesi dell'anno (no 29, 30, 31)
      scanf("%d", &giorno);

      sommatoria=0;
      for(i=0;i<12;i++)
      {
            // si itera sul primo indice (il mese),
            // e' costante il secondo indice (il giorno)
            sommatoria+=temp[i][giorno-1];
      }
      media=sommatoria/12; // i mesi dell'anno sono sempre 12
      printf("temperatura media del giorno %d:
                                          %f\n\n",mese,media);
```

Esercizio 4: Soluzione (4)

```
// Calcolo della temperatura media di un particolare range

      printf("range di date \n");
      printf("(giorno e mese iniziale, giorno e mese finale)\n");
      scanf("%d %d,%d %d",
                  &giornoIn, &meseIn, &giornoFin, &meseFin);
      // per ipotesi le date inserite sono valide (no 30 Feb.)
      // la data iniziale non è successiva alla data finale
      sommatoria=0; somGiorni=0; i=meseIn-1; j=giornoIn-1;

      // Il seguente ciclo for somma il valore delle temperatura
      // di tutti i giorni nel range dato, estremi compresi.
      // Il ciclo itera finché la data in esame
      // è precedente o uguale alla data finale, ovvero:
      // a) il mese corrente è precedente al mese finale
      // b) il mese corrente è il mese finale e il giorno
      //          corrente è precedente o uguale al giorno finale
```


Esercizio 4: Soluzione (5)

```
      while(i<meseFin-1 || (i==meseFin-1 && j<=giornoFin-1) )
      {
            printf("%d:%d  ",j,i);
            sommatoria+=temp[i][j];
            somGiorni++;
            j++;
            if(j==31 || temp[i][j]==-100)
            {
                  // Il mese corrente è finito;
                  // riparto dal primo giorno del mese successivo
                  j=0; // (primo giorno)
                  i++; // (mese successivo)
            }
      }
      media=sommatoria/somGiorni;
      printf("temperatura media effettuata %d giorni:
                                    %f\n\n",somGiorni,media);

      return 0;
}
```


Esercizio 5: Allocazione dinamica

Realizzare un programma che, data in input una ***sequenza di N parole*** (di al più 20 caratteri ciascuna), le memorizzi in una ***struttura dati dinamica*** e poi stampi la lunghezza di ogni parola memorizzata

Esercizio 5: Soluzione

```
typedef char parola[20];

int main()
{
      parola  *p;
      int i,  N;

      printf("Quante parole? ");
      scanf("%d", &N);

      // allocazione del vettore
      p=(parola *)malloc(N*sizeof(parola));

      // lettura della sequenza
      for(i=0; i<N; i++) scanf("%s", p[i]);
      for(i=0; i<N; i++) printf("\n%d", strlen(p[i]));

      free(p); // deallocazione
      return 0;
}
```

Esercizio 6: Liste di interi (1)

Si scriva una funzione ricorsiva **`crossSelection()`** che, ricevute in ingresso due liste di interi positivi **`l1`** e **`l2`**, restituisca una terza lista (eventualmente non ordinata) contenente ***gli interi di `l2` che sono nelle posizioni indicate dai valori di `l1`*** (si assuma per convenzione che il primo elemento di una lista sia in posizione 1)

Ad esempio, date due liste: **`l1=[1,3,4]`** e **`l2=[2,4,6,8,10,12]`**, la lista risultante deve contenere gli elementi di **`l2`** che sono in prima, terza e quarta posizione, cioè**`: [2,6,8]`**

Esercizio 6: Liste di interi (2)

A tal scopo si realizzi una funzione ricorsiva di supporto **`select()`** che, ricevuti in ingresso una lista e un intero positivo rappresentante una posizione, ***restituisca l'intero della lista posto alla posizione specificata***. La funzione deve restituire -1 qualora l'intero passato non corrisponda a nessuna posizione valida (si assuma comunque positivo l'intero passato)

Le funzioni **`crossSelection()`** e **`select()`** devono essere realizzate in modo ricorsivo, utilizzando il tipo di dato astratto **`list`**. Si possono utilizzare le sole operazioni primitive definite durante il corso (che quindi possono NON essere riportate nella soluzione). Non si possono usare altre funzioni di alto livello

Esercizio 6: Soluzione (1)

```
// Versione con primitive
element select(list l, int pos)
{
      if (empty(l)) return -1;
      else if (pos == 1) return head(l);
      else return select(tail(l), pos-1);
}

list crossSelection(list l1, list l2)
{
      if (empty(l1))
            return emptylist();
      else
            return cons(select(l2, head(l1)),
                  crossSelection(tail(l1), l2));
}
```


Esercizio 6: Soluzione (2)

```
// Versione con puntatori
element select(list l, int pos)
{
      if (l==NULL) return -1;
      else if (pos == 1) return l->value;
      else return select(l->next, pos-1);
}
list crossSelection(list l1, list l2)
{
      list l;
      if (l1==NULL) return NULL;
      else
      {
            l=(list) malloc(sizeof(item));
            l->value=select(l2, l1->value);
            l->next= crossSelection(l1->next, l2));
            return l;
      }
}
```

Esercizio 7: File di testo (1)

Un sito Internet salva in un file di testo lo ***stato delle piste di località sciistiche***
La **prima riga** di tale file contiene un intero rappresentante il numero di località registrate nel file
Ciascuna riga successiva contiene il nome di una località sciistica (al più 30 caratteri senza spazi), la quantità minima e massima in centimetri della neve sulle piste di tale località (due float separati dal carattere '/') e un intero che indica il numero di giorni trascorsi dall'ultimo aggiornamento dello stato della neve
(se non specificato diversamente i vari elementi sono separati da spazi)

Definire correttamente una struttura dati chiamata **localita** in grado di contenere il ***nome di una località sciistica e la quota minima e massima della neve***

Esercizio 7: File di testo (2)

Realizzare:

```
void onlyMax(localita* locs, int* res, float max)
```

che, ricevuti come parametri di ingresso **locs**, un vettore di strutture dati **localita** già correttamente popolato con **res** elementi, e **max**, un float, modifichi il vettore **locs** al fine di ***eliminare quelle località la cui quantità massima di neve sia inferiore a*** ***max***. A tale scopo la procedura **onlyMax** deve spostare gli elementi all'interno del vettore **locs** in modo tale che tutti i primi elementi abbiano una quota di neve massima uguale o superiore a **max** (non è necessario tenere traccia degli elementi che non soddisfano il requisito)

Al termine dell'esecuzione della procedura il valore puntato da **res** deve rappresentare il ***numero dei soli elementi che soddisfano il requisito sulla quota massima***

Esercizio 7: File di testo (3)

```
localita* meteo(char* filename, int maxGiorni, int* res)
```

che, ricevuti come parametri di ingresso il nome del file di testo sopra descritto e un intero **maxGiorni**, restituisca un vettore di strutture dati **localita**, allocato dinamicamente, composto dalle località sciistiche presenti nel file di testo il cui ***stato della neve sia stato aggiornato al più*** ***maxGiorni*** ***prima*** e la cui ***quota di neve non sia inferiore di più di 30 centimetri rispetto alla località con più neve*** (considerando la quota massima e non quella minima delle sole località aggiornate recentemente); restituisca inoltre **res**, il numero di elementi presenti nel vettore restituito

A tal fine la funzione **meteo** allochi dinamicamente un vettore che sia in grado di contenere tutte le località contenute nel file di testo e lo popoli con le località aggiornate da al più **maxGiorni**; in seguito invochi opportunamente la procedura al punto **onlyMax** per eliminare le località che non soddisfano il requisito sulla quantità di neve

Esercizio 7: Soluzione (1)

```
#define DIM 31
typedef struct
{
   char nome[DIM];
   float min;
   float max;
}localita;

void onlyMax(localita* locs, int*res, float max)
{
   int dim=0, i;
   for(i=0;i<*res;i++)
      if(locs[i].max>=max)
      {
            locs[dim]=locs[i];
            dim++;
      }
   *res=dim;
}
```

Esercizio 7: Soluzione (2)

```
localita* meteo(char* filename, int maxGiorni, int* res){
   int righe,update; float max;
   localita *resLoc; FILE* file;
   if( (file=fopen(filename,"r"))==NULL ) exit(-1);

   fscanf(file,"%d",&righe);
   resLoc=(localita*)malloc(sizeof(localita)*righe);
   max=0; *res=0;
   while( (fscanf(file,"%s %f/%f %d",
                  resLoc[*res].nome, &(resLoc[*res].min),
                  &(resLoc[*res].max), &update)) >=0){
      if(update<maxGiorni){
         if(resLoc[*res].max>max)
             max=resLoc[*res].max;
         (*res)++;
      }
   }
   fclose(file);
   onlyMax(resLoc,res,max-30);
   return resLoc;
}
```


Esercizio 8: Liste di interi e File (1)

Un file di testo **ARCHIVIO.TXT** contiene i dati (primo autore, titolo, numero di copie possedute, numero di copie in prestito) relativi ai ***differenti volumi conservati presso una biblioteca***

Più precisamente, ogni riga del file contiene nell'ordine, separati da uno spazio bianco:

- **autore** (al più di 20 caratteri senza spazi)
- **titolo** (al più di 50 caratteri senza spazi)
- **numero_possedute** (intero)
- **numero_prestito** (intero)

Esercizio 8: Liste di interi e File (2)

Si realizzi un programma C che:

1.Legga il contenuto di **ARCHIVIO.TXT** e costruisca in memoria centrale un vettore **V** di strutture corrispondenti (si supponga che il file **ARCHIVIO.TXT** non possa contenere più di 30 righe). Si stampi a video il contenuto del vettore

2.A partire da **V**, costruisca una lista **L** di interi contenente per ciascun volume il numero di copie disponibili nella biblioteca, ovvero la differenza fra il numero di copie possedute e il numero di copie in prestito. Si stampi a video il contenuto della lista **L**

Esercizio 8: Liste di interi e File (3)

3. Utilizzando **L** per ottenere la somma delle copie disponibili e **V** per la somma delle copie possedute, calcoli il ***rapporto fra volumi disponibili e volumi posseduti***

4. Utilizzando la lista di interi **L**, stampi il numero di riga di **ARCHIVIO.TXT** relativo al ***volume con più copie disponibili***. In caso di più volumi con pari numero di copie disponibili, qualunque riga relativa a questi ultimi è considerata una risposta corretta

Esercizio 8: Liste di interi e File (4)

Ad esempio: contenuto di ARCHIVIO.TXT

Salinger IlGiovaneHolden 10 8
Wallace InfiniteJest 12 3
Carver Cattedrale 12 12
Baricco Seta 6 0
Hornby ComeDiventare 9 9
Sartre LaNausea 3 1
Robbins NaturaMorta 7 7

Stampa di L:
[2, 9, 0, 6, 0, 2, 0]

Esercizio 8: Soluzione (1)

Suddivido il programma nei seguenti file:

- **list.c**: funzioni di libreria per la gestione di liste
- **list.h**: header file associato a list.c
- **element.h**: dichiarazione di element
- **mainLibri.c**: programma principale

Esercizio 8: Soluzione (2)

```
/* PROGRAMMA PRINCIPALE - file mainLibri.c */
#include <stdio.h>
#include <stdlib.h>
#include "list.h"
#define MAX 20

typedef struct{
      char autore[20];
      char titolo[50];
      int possedute;
      int prestito;
} volume;
```


Esercizio 8: Soluzione (3)

```
int main()
{
     volume e; list L,L1; FILE *f;
     volume V[MAX]; int elementi=0,i,pos,max;
     int somma_possedute, somma_disponibili;
     L=emptylist();


     // DOMANDA 1
     f = fopen("ARCHIVIO.TXT", "r");
     if (f==NULL)
     {
          printf("Impossibile aprire file di ingresso");
          exit(1);
     }
```

Esercizio 8: Soluzione (4)

```
while (fscanf(f,"%s%s%d%d\n",
      e.autore, e.titolo, &e.possedute, &e.prestito)>0)
{
      V[elementi++] = e;
}
fclose(f);

for (i=0; i<elementi; i++)
      printf("Volume %d: %s\t%s\t%d\t%d\n",
                              i,
                              V[i].autore,
            V[i].titolo,
            V[i].possedute,
            V[i].prestito);
```


Esercizio 8: Soluzione (5)

```
// DOMANDA 2
for (i=0; i<elementi; i++)
      L = cons(V[i].possedute-V[i].prestito,L);
showlist(L);
// in che ordine viene stampata la lista?


// DOMANDA 3
for (i=0; i<elementi; i++)
      somma_possedute += V[i].possedute;
L1=L;
while (!empty(L1))
{
      somma_disponibili += head(L1);
      L1=tail(L1);
}
printf("Rapporto disponibili/possedute = %f\n",
(float)somma_disponibili/somma_possedute);
```


Esercizio 8: Soluzione (6)

```
      // DOMANDA 4
      i=0;
      max=-1;

      while (!empty(L))
      {
            if (head(L)>max)
            {
                  max = head(L);
                  pos=i;
            }
            L=tail(L);
            i++;
      }
      printf("Volume con più copie disponibili: %d",
                                          elementi-pos-1);
      return 0;
}
```


Esercizio 9: File di testo e binari (1)

Una ditta di impianti elettrici tiene un ***registro dei lavori effettuati*** per i diversi clienti su ***diversi file binari*** e in un file di testo registra ***per ogni cliente il nome del file in cui sono registrati i lavori che lo riguardano***

In particolare, per ogni lavoro effettuato la ditta salva su un **file binario** strutture dati di tipo **`Work`**: ogni struttura contiene il nome del cliente (stringa di al più 64 caratteri utili), il giorno in cui il lavoro è stato fatto (int che rappresenta il giorno nell'anno corrente) e l'importo ai fini della fatturazione (float)
I lavori relativi ad uno stesso cliente risiedono tutti nello stesso file, ma uno stesso file contiene i dati dei lavori relativi a più clienti

La ditta salva poi, su ogni riga di un **file di testo** che funge da "chiave", il nome del cliente (sempre una stringa di al più 64 caratteri utili) e, separato da uno spazio, il nome del file in cui i lavori del cliente sono salvati (stringa di al più 64 caratteri utili). Si deve realizzare un programma che calcoli ***l'ammontare delle fatture per tutti i clienti della ditta***

Esercizio 9: File di testo e binari (2)

Esempio di contenuto del file di testo:

```
PaoloBellavista marzo.dat
PaolaMello aprile.dat
FedericoChesani marzo.dat
CarloGiannelli marzo.dat
Marco Montali giugno.dat
```

Realizzare:

`list findBills(char * fileName, char * clientName)`

che, ricevuti in ingresso il nome del file binario e il nome del cliente, restituisca una lista contenente ***i soli importi relativi ai lavori eseguiti per il cliente specificato*** (si presti attenzione al fatto che in uno stesso file ci possono essere i dati relativi a più clienti)

Si supponga a tal fine di poter disporre del tipo di dato astratto **`list`** visto a lezione, definito per il tipo primitivo **`float`**, e si supponga di disporre anche di tutte le funzioni primitive presentate a lezione

Esercizio 9: File di testo e binari (3)

Un programma **`main()`**, che chieda all'utente il nome del file di testo in cui sono registrate le coppie nome cliente-nome file; per ogni cliente registrato in tale file, il programma deve stampare a video il nome cliente, la lista importi relativi, e la loro somma

Al fine di stampare le liste, il candidato ipotizzi di avere a disposizione la funzione **`showlist(…)`** opportunamente modificata, e che quindi non deve essere riportata nella soluzione

```
#define DIM 65
typedef struct work
{
      char name[DIM];
      int day;
      float bill;
} Work;
```


Esercizio 9: Soluzione (1)

```
list findBills(char * fileName, char * clientName)
{
   FILE * fWorks;
   list result;
   Work temp;
   if ((fWorks = fopen(fileName, "rb")) == NULL)
   {
      printf("Error opening the file %s\n", fileName);
      exit(-1);
   }
   result = emptylist();
   while (fread(&temp, sizeof(Work), 1, fWorks) > 0 )
      if (strcmp(temp.name, clientName) == 0)
            result = cons(temp.bill, result);
   fclose(fWorks);
   return result;
}
```


Esercizio 9: Soluzione (2)

```
int main() {
      FILE * fClients; list tempBill; float totalBill;
      char fileClients[DIM], clientName[DIM], fileName[DIM];

      printf("Inserire il nome del file dei clienti: ");
      scanf("%s", fileClients);

      if ((fClients = fopen(fileClients, "r")) == NULL)
      {
          printf("Error opening the file %s\n", fileClients);
          exit(-1);
      }
```

Esercizio 9: Soluzione (3)

```
        while (fscanf(fClients, "%s %s",
                                clientName, fileName) != EOF )
        {
                totalBill = 0;
                tempBill = findBills(fileName, clientName);
                printf("%s: ", clientName);
                showlist(tempBill);
                while(! empty(tempBill))
                {
                        totalBill = totalBill + head(tempBill);
                        tempBill = tail(tempBill);
                }
                printf(" Total bill: %6.2f\n", totalBill);
        }
        fclose(fClients);
        return 0;
}
```


Esercizio 10: Coda di float

Realizzare una funzione

```
float piuVicino(startQueue start, endQueue end);
```

che, data una coda FIFO definita per i **`float`**, individui l'elemento che ***più si avvicina alla media di tutti gli elementi***

Realizzare anche un **`main()`** di prova che inserisca **`float`** all'interno della coda FIFO, invochi la funzione **`piuVicino()`** e stampi sullo standard output il valore restituito

Esercizio 10: Soluzione (1)

```
float piuVicino(startQueue start, endQueue end){
    startQueue startTemp; endQueue endTemp;
    element el;   float count=0, media, res;    int n=0;
    createEmptyQueue(&startTemp, &endTemp);
    while(!isEmptyQueue(start))
    {
        el=deQueue(&start,&end);
        count+=el;
        n++;
        enQueue(el,&startTemp,&endTemp);
    }
    media=count/n;
    res=deQueue(&startTemp,&endTemp);
    while(!isEmptyQueue(startTemp))
    {
        el=deQueue(&startTemp,&endTemp);
        if(fabs(media-el)<fabs(media-res))
            res=el;
    }
    return res;
}
```


Esercizio 10: Soluzione (2)

```
int main()
{
    startQueue start;
    endQueue end;
    float res;

    createEmptyQueue(&start, &end);
    enQueue(3.2,&start,&end);
    enQueue(3.5,&start,&end);
    enQueue(3.8,&start,&end);

    res=piuVicino(start, end);
    printf("piu' vicino %f\n",res);

    return 0;
}
```

Esercizio 11: Code di stringhe

Realizzare una funzione

```
void filtra(startQueue start1, endQueue end1,
            startQueue start2, endQueue end2,
            startQueue* startRes, endQueue* endRes);
```

che, date due code FIFO della stessa lunghezza e contenenti stringhe ben formate, ***crei una nuova coda FIFO che abbia gli elementi della prima coda che precedono lessicografica-mente i corrispondenti elementi della seconda coda*** (non è necessario mantenere invariato il contenuto delle due liste iniziali)

Realizzare anche un **`main()`** di prova che inserisca stringhe ben formate all'interno di due code FIFO, invochi la funzione **`filtra()`** e stampi sullo standard output la coda restituita

Esercizio 11: Soluzione (1)

```
void filtra(startQueue start1, endQueue end1,
            startQueue start2, endQueue end2,
            startQueue* startRes, endQueue* endRes)
{
    element s1,s2;
    createEmptyQueue(startRes, endRes);
    while(!isEmptyQueue(start1))
    {
        s1=deQueue(&start1,&end1);
        s2=deQueue(&start2,&end2);
        if(strcmp(s1,s2)<0)
        {
            enQueue(s1,startRes,endRes);
        }
    }
}
```


Esercizio 11: Soluzione (2)

```
int main()
{
    startQueue start1, start2, startRes;
    endQueue end1, end2, endRes; element el;
    createEmptyQueue(&start1, &end1);
    enQueue("ciao",&start1,&end1);
    enQueue("a",&start1,&end1);
    enQueue("tutti",&start1,&end1);
    createEmptyQueue(&start2, &end2);
    enQueue("aciao",&start2,&end2);
    enQueue("a",&start2,&end2);
    enQueue("ztutti",&start2,&end2);

    filtra(start1, end1, start2, end2, &startRes, &endRes);
    while(!isEmptyQueue(startRes))
    {
        el=deQueue(&startRes,&endRes);
        printf("%s\n",el);
    }
    return 0;
}
```


Esercizio 12: Stack

Realizzare una funzione

```
Stack filterStack(Stack* s);
```

che, dato un generico **`Stack`**, ne restituisca ***una copia filtrata (mantenendo l'ordine degli elementi, e senza perdere i dati nello stack originario).*** Il filtro deve essere dichiarato in un'apposita funzione

```
Boolean isMaintained(StackElement el);
```

e richiamato opportunamente dalla funzione di filtraggio

Testare il funzionamento del filtro implementando un filtro per valori di tipo **`double`** che mantenga ***i valori positivi*** e un filtro per valori di tipo stringa che mantenga le ***stringhe che cominciano con una lettera maiuscola***

Come si può restituire una copia intera dello stack?

Esercizio 12: Soluzione (1)

```
void filterStackRecursive(Stack* src, Stack* dest)
{
      if( !isEmptyStack(*src) )
      {
            StackElement top = pop(src);
            filterStackRecursive(src, dest);
            if( isMaintained(top) )
                  push(top, dest);
            push(top, src);
      }
}

Stack filterStack(Stack* s)
{
      Stack s2 = newStack();
      filterStackRecursive(s, &s2);
      return s2;
}
```


Esercizio 12: Soluzione (2)

```
//filtro float
Boolean isMaintained(StackElement el)
{
      return el > 0;
}

//filtro stringhe
Boolean isMaintained(StackElement el)
{
      return      strlen(el) > 0 &&
                  el[0] >= 'A' && el[0] <= 'Z';
}

//filtro indipendente dal tipo per copiare lo stack
Boolean isMaintained(StackElement el)
{
      return true;
}
```


Esercizio 13: Stack

Realizzare una funzione

`double stackAVG(*Stack s);`

che, dato uno stack di **`float`** contentente almeno un elemento, ne calcoli la media utilizzando lo stack stesso come contenitore dei calcoli parziali (alla fine della computazione, lo stack sarà vuoto). In particolare, si utilizzi per il calcolo una funzione di appoggio

`int sumStack(*Stack s);`

che restituisca il numero di elementi presenti nello stack originario e agisca sullo stack in modo che, alla fine della computazione, questo contenga come ***unico elemento la somma di tutti gli elementi presenti originariamente***

Esercizio 13: Soluzione

```
int sumStack(Stack* s)
{
       StackElement el = pop(s);
       if(isEmptyStack(*s))
       {
              push(el, s);
              return 1;
       }
       else
       {
              push(pop(s) + el, s);
              return sumStack(s) + 1;
       }
}

StackElement stackAVG(Stack* s)
{
       int dim = sumStack(s);
       return pop(s)/dim;
}
```