Esercizio 1

Gestione degli esami di uno studente

- Realizzare un programma che permetta di gestire gli esami di uno studente
- Funzionalità richieste
  - Caricamento degli esami sia da file di testo che da file binario
    - Si assuma che la prima riga del file da cui leggere gli esami contenga **IL NUMERO DI ESAMI** presenti nel file
  - Stampa degli esami
  - Calcolo della ***media pesata sul numero di crediti***
  - Salvataggio su ***file di testo*** dell'elenco degli ***esami la cui dicitura contiene una stringa data***, unitamente alla ***media calcolata solo su questi esami***

Esercizio 1 - Strutture dati

Tipi di dato utilizzati

```
typedef struct
{
   char dicitura[36];
   int crediti;
   int voto;
} Esame;

typedef struct
{
   int dim;
   Esame* esami;
} VettoreEsami;
```

35 caratteri fissi + terminatore

Esercizio 1 - Lettura (1)

```
boolean leggiEsamiTxt(char *nomeFile, VettoreEsami* vett)
{
  FILE *fp;
  int i;
  if((fp = fopen(nomeFile, "r")) == NULL)
  {
      perror("Errore di accesso al file: ");
      return false;
  }
  fscanf(fp,"%d",&vett->dim);
  vett->esami =
      (Esame*) malloc(vett->dim * sizeof(Esame));
```

Lettura del numero di esami (che corrisponde alla dimensione del vettore)

Allocazione dello spazio necessario per contenere *dim* esami

Esercizio 1 - Lettura (2)

```
  ...

  for(i = 0; i < vett->dim; i++)
  {
      fgetc(fp);
      fscanf(fp, "%35c%d%d",
                  vett->esami[i].dicitura,
                  &vett->esami[i].crediti,
                  &vett->esami[i].voto);
       vett->esami[i].dicitura[35]= '\0';
  }
  fclose(fp);
  return true;
}
```

Lettura del newline (necessaria perché la stringa viene letta a caratteri)

Poiché la stringa viene letta a caratteri, il terminatore va aggiunto esplicitamente!

Esercizio 1 - Lettura binaria

```
boolean leggiEsamiBin(char *nomeFile, VettoreEsami *vett)
{
   FILE *fp; int i;
   if((fp = fopen(nomeFile, "rb")) == NULL)
   {
       perror("Errore di accesso al file: ");  return false;
   }
   fread(&vett->dim, sizeof(int), 1, fp);
   vett->esami = (Esame*) malloc(vett->dim * sizeof(Esame));
   for(i = 0; i < vett->dim; i++)
   {
       fread(&vett->esami[i], sizeof(Esame), 1, fp);
       vett->esami[i].dicitura[35]= '\0';
       //per sicurezza! Quando sarebbe invece necessario?
   }
   fclose(fp);
   return true;
}
```


Esercizio 1 - Stampa strutture dati

```
void stampaEsame(Esame esame)
{
   printf("%s (%d): %d\n",
              esame.dicitura, esame.crediti, esame.voto);
}

void stampaEsami(VettoreEsami vett)
{
   int i;
   for(i = 0; i < vett.dim; i++)
       stampaEsame(vett.esami[i]);
}
```

Esercizio 1 – calcolo media

```
float media(VettoreEsami vett)
{
   int num = 0, den = 0, i;
   for(i = 0; i < vett.dim; i++)
   {
       num = num + vett.esami[i].crediti * vett.esami[i].voto;
       den = den + vett.esami[i].crediti;
   }
   return ((float) num) / den;
}

boolean matches(char* str, char* pattern)
{
   return (strstr(str, pattern) != NULL);
}
```


Esercizio 1 - Filtro (1)

```
VettoreEsami filtra(VettoreEsami vett, char* pattern)
{
   int i, j = 0, dimFiltro = 0;
   VettoreEsami filtro;
   for(i = 0; i < vett.dim; i++)
   {
       if( matches(vett.esami[i].dicitura, pattern) )
             dimFiltro++;
   }
   filtro.dim = dimFiltro;
   …
```

Esercizio 1 - Filtro (2)

```
   …
   filtro.esami =
       (Esame*) malloc(dimFiltro * sizeof(Esame));
   for(i = 0; i < vett.dim; i++)
   {
       if(matches(vett.esami[i].dicitura, pattern))
       {
             filtro.esami[j] = vett.esami[i];
             j++;
       }
   }
   return filtro;
}
```


Esercizio 1 - Salvataggio (1)

```
void salvaEsame(FILE *fp, Esame *esame)
{
  fprintf(fp, "%s (%d): %d\n",
            esame->dicitura, esame->crediti, esame->voto);
}

boolean salvaReport(VettoreEsami vett, char* nomeFile)
{
  int i;
  FILE *fp = fopen(nomeFile, "w");
  if(fp == NULL)
  {
      perror("errore durante il salvataggio: ");
      return false;
  }
…
```

Esercizio 1 - Salvataggio (2)

```
   …
   for(i = 0; i < vett.dim; i++)
       salvaEsame(fp, &vett.esami[i]);
   fprintf(fp, "MEDIA: %f", media(vett) );
   fclose(fp);
   return true;
}
```

Filtraggio & Salvataggio (esempio):

```
VettoreEsami filtro = filtra(vett, “L-A”);
salvaReport(filtro, “report.txt”);
free(filtro); //DEALLOCAZIONE!
```


Esercizio 1 - Salvataggio binario

```
boolean serialize(VettoreEsami vett, char* nomeFile)
{
   FILE *fp = fopen(nomeFile, "wb");

   if(fp == NULL)
   {
       perror("errore durante il salvataggio: ");
       return false;
   }
   fwrite(&vett.dim, sizeof(int), 1, fp);
   fwrite(vett.esami, sizeof(Esame), vett.dim, fp);
   fclose(fp);
   return true;
}
```

Esercizio 2

Gestione articoli in vendita

Realizzare un programma che permetta di gestire gli articoli in vendita con le seguenti funzionalità:

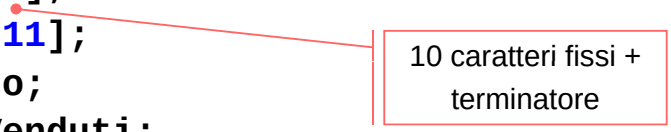
- Caricamento del prezzo e della quantità degli articoli già venduti da due file di testo **`listino.txt`** e **`venduti.txt`**
  - Ciascuna riga di **`listino.txt`** specifica, separati tra loro da uno spazio, la tipologia di articolo in vendita (al più dieci caratteri senza spazi), la sua marca (al più 10 caratteri senza spazi) e il suo prezzo in euro (**`float`**)
  - Ciascuna riga di **`venduti.txt`** specifica, separati tra loro da uno spazio, la tipologia e la marca di ciascun articolo venduto
- Stampa dell'elenco degli articoli già venduti suddivisi per marca e tipo con prezzo unitario e quantità totale
- Salvataggio su file binario dell'elenco precedente
- Calcolo dell'incasso ottenuto suddiviso per **marca**

Esercizio 2 - Strutture dati

- Tipi di dato utilizzati

```
typedef struct
{
  char tipo[11];
  char marca[11];
  float prezzo;
  int totaleVenduti;
} item;

typedef struct
{
  char marca[11];
  float euro;
} income;
```



Esercizio 2 - Lettura (1)

```
item* articoli(FILE* listino, FILE* venduti, int* len)
{
    char tipo_temp[11], marca_temp[11];
    item *res, *resTemp, temp;

    *len = 0;

    while(fscanf(listino,"%s %s %f\n", temp.tipo,
                         temp.marca, &(temp.prezzo))==3)
        (*len)++;
    res = (item*) malloc( sizeof(item)*(*len) );
    resTemp = res;
    rewind(listino);
```

Lettura del numero di oggetti nel listino prezzi (che corrisponde alla dimensione del vettore)

Allocazione dello spazio necessario per contenere **len* item

Esercizio 2 - Lettura (2)

```
while( fscanf( listino,"%s %s %f\n",
            temp.tipo, temp.marca, &(temp.prezzo))==3)
    {
      temp.totaleVenduti=0;
      while( fscanf(venduti, "%s %s\n",
                         tipo_temp, marca_temp) == 2 )
        if( strcmp(temp.tipo, tipo_temp) ==0 )
            if( strcmp(temp.marca, marca_temp) ==0 )
                (temp.totaleVenduti)++;
        rewind(venduti);
        (*resTemp) = temp;
        resTemp++;
    }
    return res;
}
```

Iterazione su tutti gli oggetti presenti nel file *listino*

Iterazione su tutti gli oggetti presenti nel file *venduti*

Stessa marca e stessa tipologia, quindi aumento di uno il contatore degli oggetti venduti

Esercizio 2 - Scrittura

```
Boolean scriviArticoli( char* nomeFileBinario,
                                    item* vett, int len)
{
    int i;
    FILE*bin;

    bin = fopen(nomeFileBinario,"wb");

    if(bin == NULL)
        return false;
    for(i=0; i<len; i++)
        fwrite(vett+i, sizeof(item), 1, bin);
    close(bin);
    return true;
}
```

ricordarsi di chiudere il file

Esercizio 2 - Calcolo incasso (1)

```
income* calcolaIncasso(item vett[], int lenVett,
                                    int* lenIncasso)
{
    income *res;  int i, j, trovato;
    *lenIncasso = 0;
    res = (income*)malloc(sizeof(income)*lenVett);

    for(i=0; i<lenVett; i++)
    {
        trovato=0;
        for(j=0; j<*lenIncasso && trovato==0 ;j++)
           if( strcmp((res+j)->marca, vett[i].marca)==0)
               trovato=1;
```

Allocazione dello spazio necessario per contenere **lenVett* income

Iterazione sul vettore res per cercare un elemento di marca *vett[i].marca*

Il vettore *res* contiene già un elemento con marca uguale a quella in esame

Esercizio 2 - Calcolo incasso (2)

```
  if(trovato == 0)
  {
    strcpy( (res+(*lenIncasso))->marca, vett[i].marca );
    (res+(*lenIncasso))->euro =
                  (vett[i].prezzo*vett[i].totaleVenduti);
    (*lenIncasso)++;
  }
  else
    (res+j-1)->euro +=
                  (vett[i].prezzo*vett[i].totaleVenduti);
  }
  return res;
}
```

Prima volta in cui viene esaminato un oggetto di marca *vett[i].marca*

Un oggetto di marca *vett[i].marca* è già stato esaminato