Esercizio 2: Intersezione e Differenza fra Liste

- Si leggano da standard input ***due liste di interi positivi*** (l'utente terminerà l'inserimento di ognuna con il valore 0)
- Scrivere le seguenti funzioni:
  - `List intersect(List l1, List l2)` riceve due liste e ***restituisce una terza lista contenente i valori presenti in entrambe***, utilizzando le primitive
  - `List diff(List l1, List l2)` restituisce una ***lista contenente i valori presenti in l1 che NON sono presenti in l2*** (l1- l2), senza usare le primitive
- ***Modificare*** le soluzioni precedenti facendo in modo che la ***lista risultato NON contenga elementi ripetuti***
- Deallocare correttamente le liste utilizzate

Esercizio 2: contains (con primitive)

```
Boolean contains(List l, Element e)
{
   Boolean found = false;
   while(!empty(l) && !found)
   {
       found = (head(l) == e);
       l = tail(l);
   }
   return found;
}
```

Esercizio 2: contains (senza primitive)

```
Boolean contains(List l, Element e)
{
   Boolean found = false;
   while(l != NULL && !found)
   {
       found = (l->value == e);
       l = l->next;
   }
   return found;
}
```


Esercizio 2: Intersezione

```
List intersect(List l1, List l2)
{
   Element cur;
   List intersection = emptyList();
   while(!empty(l1))
   //converrebbe iterare sulla lista più corta...
   {
       cur = head(l1);
       if(contains(l2, cur))
       intersection = cons(cur, intersection);
       l1 = tail(l1);
   }
   return intersection;
}
```

`&& !contains(intersection, cur)`
per evitare elementi ripetuti nel risultato

Esercizio 2: Differenza

```
List diff(List l1, List l2)
{
  Element cur;
  List difference = NULL, temp;
  while(!empty(l1))
      {
      cur = l1->value;
      if(!contains2(l2, cur))
      {
            temp = (List) malloc(sizeof(Item));
            temp->value = cur;
            temp->next = difference;
            difference = temp;
      }
      l1 = l1->next;
  }
  return difference;
}
```

`&& !contains2(difference, cur)`
per evitare elementi ripetuti nel risultato

Esercizio 3: Lista di strutture

Gestione di un negozio di videogame

- Un negozio di videogame vuole automatizzare parte della propria gestione
- Il negozio salva ***mensilmente lo stato dei propri articoli su un file di testo*** e traccia ***su un secondo file tutte le vendite effettuate***
- Lo scopo del programma è
  - ***Acquisire la lista di videogiochi*** aggiornandone il ***numero di copie disponibili*** rispetto alle vendite effettuate
  - Generare una lista di ***videogiochi acquistabili da bambini***
  - Generare un file contenente gli ordini da effettuare per riportare il magazzino ***a un certo numero di copie per ogni gioco***

Esercizio 3 - Element

```
typedef struct
{
    int code;
    char title[31];
    char type;
    int nItems;
    float rate;
} Element;
```


Esercizio 3 - Stampa lista (ricorsiva, con primitive)

```
void printGames(List games)
{
  if(!empty(games))
  {
     Element game = head(games);
     printf("%d) %s \t(%c) %d \t %f\n",
                          game.code,
                               game.title,
                               game.type,
                               game.nItems,
                               game.rate);
     printGames(tail(games));
  }
}
```

Esercizio 3 – Caricamento (iterativo, con primitive)

```
Boolean loadFromFile(char *fileName, List *games)
{
  FILE *fp;
  Element curGame;
  *games = emptyList();
  if((fp = fopen(fileName, "r")) == NULL)
  {
      perror("Error opening file: ");
      return false;
  }
  …
```


Esercizio 3 – Caricamento (iterativo, con primitive)

```
      …
      while(fscanf(fp, "%d%30c%c%d%f",
                       &curGame.code,
                       curGame.title,
                       &curGame.type,
                       &curGame.nItems,
                       &curGame.rate) != EOF)
      {
            curGame.title[30] = '\0';
            *games = cons(curGame, *games);
      }
      fclose(fp);
      return true;
}
```

Esercizio 3 – Aggiornamento (iterativo, senza prim.)

```
Boolean updateAvailability(char *fileName, List games)
{
   FILE *fp;
   int code;
   Element* game;
   if((fp = fopen(fileName, "r")) == NULL)
   { perror("Error opening file: "); return false; }
   while(fscanf(fp, "%d", &code) != 1)
   {
       game = findElementByCode(games, code);
       if(game != NULL)
              game->nItems--;
       else
              printf("CODICE NON RICONOSCIUTO: %d\n", code);
   }
   return true;
}
```


Esercizio 3 - Ricerca per codice (iter., senza prim.)

```
Element* findElementByCode(List games, int code)
{
   List cur;
   Element* found = NULL;
   cur = games;
   while(cur != NULL && found == NULL)
   {
       if(cur->value.code == code)
              found = &cur->value;
       else
              cur = cur->next;
   }
   return found;
}
```

Esercizio 3 - Giochi per bambini (ric., con prim.)

```
List gamesForKids(List games, float threshold)
{
  List kids;
  Element game;
  if(empty(games))
     return emptyList();
  kids = gamesForKids(tail(games), threshold);
  game = head(games);
  if(game.nItems > 0 && game.type != 'P' &&
     (game.type == 'A' || game.rate >= threshold))
  {
     kids = cons(game, kids);
  }
  return kids;
}
```


Esercizio 3 - Salvataggio (iter., con prim.)

```
Boolean saveOrdersToFile
      (char* fileName, List games, int qty)
{
  FILE* fp;
  int orderNumber;
  Element game;
  List gamesToRead = games;
  if ((fp = fopen(fileName,"w")) == NULL)
  {
     perror("Error opening file: ");
     return false;
  }
  …
```

Esercizio 3 - Salvataggio (iter., con prim.)

```
  …
  while(!empty(gamesToRead))
  {
      game = head(gamesToRead);
      orderNumber = qty - game.nItems;
      if(orderNumber > 0)
           fprintf(fp, "%d %s %d\n",
                                  game.code,
                                  game.title,
                                  orderNumber);
      gamesToRead = tail(gamesToRead);
  }
  fclose(fp);
  return true;
}
```