ESERCIZIO 1

La seguente dichiarazione definisce il catalogo prezzi nel sistema informativo di un piccolo supermercato, su una piattaforma in cui

```
sizeof(long)==8,  sizeof(float)==sizeof(int)==sizeof(void *)==4

typedef char Descrizione[40];
typedef struct { unsigned long barcode;
                 Descrizione descr;
                 float prezzo;
                 int noPunti;
                 char * descrEstesa;
               } Articolo;
typedef struct {
                 Articolo list[1000];
                 int numArticoli;
               } CatalogoGenerale;

CatalogoGenerale cat;
```

`noPunti` è un booleano che identifica gli articoli su cui per legge non si possono effettuare promozioni o fidelizzazioni; numArticoli indica quanti dei 1000 elementi di list sono effettivamente utilizzati.

(a) Si calcoli quanto vale `sizeof(cat)`

```
sizeof(cat) == sizeof(Articolo)*1000+sizeof(int)
sizeof(Articolo) == 8+40+4+4+4 == 60

sizeof(cat) == 60004
```

(b) Sapendo che cat è allocato a partire dall' indirizzo 2348, si calcoli l' indirizzo della cella contenente la variabile cat.list[20].noPunti

```
2348 + 20 * sizeof(Articolo)
     + sizeof(barcode)
     + sizeof(descr)
     + sizeof(prezzo)

    == 2348 + 1252 == 3600
```

(c) Si implementi una funzione C++ ...stampa(...) che riceve come parametro un Articolo (passato per indirizzo) e stampa su *stdout* una riga così come deve apparire sullo scontrino.

```
void stampa(Articolo* a) {

   cout << (*a).descr << "\t \t";
   cout << setw(5) << setprecision(2) << (*a).prezzo;
}
```

(d) Si consideri che:

(1.) i cataloghi contengono gli articoli in ordine di codice a barre crescente, e

(2.) non possono esservi due articoli con lo stesso codice a barre.

Si implementi una funzione C ...trova(...)che riceve come parametri un catalogo di articoli (passato per indirizzo) e un codice a barre, e restituisce
al chiamante l' indirizzo dell' articolo identificato dal codice, o NULL se il codice non corrisponde a nessun articolo in catalogo.

```
Articolo* trova(CatalogoGenerale* c, unsigned long bar) {

   for (int i = 0; i < c->numArticoli; ++i)
       if ( bar == c->list[i].barcode)
         return &c->list[i];

   return NULL;
}
```

(e) Si implementi una funzione ...scan(...) che legge da *stdin* una sequenza di interi positivi (di lunghezza ignota e a priori illimitata), terminata dal valore 0, che rappresentano i codici a barre nell' ordine in cui sono letti dallo scanner laser di una delle casse del supermercato. La funzione riceve come parametro il catalogo del supermercato (passato per indirizzo), e:

- Restituisce al chiamante l' importo totale da pagare.
- Stampa su *stdout* lo scontrino (attenzione: non si perda tempo a organizzare intestazioni elaborate!).
- Stampa su *stdout* i punti fedeltà conseguiti dal cliente (si ha diritto a un punto fedeltà per ogni euro di spesa oltre i primi 5€, senza considerare i prodotti "non promozionabili", come ad esempio giornali e medicinali, per i quali l' attributo noPunti ha valore 1).

```
float scan(CatalogoGenerale* c) {
    unsigned long bar;
    Articolo* addr;
    float tot = 0.0, perPunti = 0.0;
    int puntiAcquisiti = 0;

    do {
      cin >> bar;

      addr = trova(c, bar);
      if (addr != NULL) {
         tot += addr->prezzo;

         if (addr->noPunti != 1)
           perPunti += addr->prezzo;

         stampa(addr);
      }
    } while (bar != 0L);

    if (perPunti > 5.0)
      cout << "\n Pti: " << (unsigned int)(perPunti-5.0);
    return tot;
}
```

f) Si spieghi brevemente perché è particolarmente opportuno che trova() e scan() ricevano il catalogo per indirizzo.

...

ESERCIZIO 2

Dichiarare un tipo per rappresentare i punti in un Piano Cartesiano.

(a) Realizzare un sottoprogramma per l'acquisizione da *stdin* di un punto del Piano Cartesiano, esibendo due versioni del sottoprogramma: procedurale e funzionale:

```
void procedureGetPoint(Point *p)
  Point functionGetPoint(void)
```

(b) Realizzare un sotto-programma

```
Point meanPoint(Point p1, Point p2)
```

che dati due punti restituisce il punto medio del segmento che li congiunge.

(c) Realizzare un sotto-programma

```
double distance(Point p1, Point p2)
```

che dati due punti restituisce la distanza tra i due. Un triangolo può essere rappresentato con un vettore di tre punti:

```
typedef Point Triangle[3]
```

(d) Realizzare un sotto-programma per l'acquisizione delle coordinate dei vertici di un triangolo nelle due versioni:

```
void GetTriangleWithFunction(Triangle t)
void GetTriangleWithProcedure(Point *t)
```

(e) Scrivere un sotto-programma che verifichi se due triangoli sono uguali.

```
int testTriangles(Triangle t1, Triangle t2)
```

```
#include <iostream>
#include <cstdlib>
#include <cmath>

typedef struct punto {
   double x;
   double y;
} Point;

typedef Point Triangle[3];

Point meanPoint(Point p1, Point p2);
double distance(Point p1, Point p2);
void procedureGetPoint(Point *p);
Point functionGetPoint(void);
void GetTriangleWithFunction(Triangle t);
void GetTriangleWithProcedure(Point *t);
int testTriangles(Triangle t1, Triangle t2);

int main() {

    Point a,b,c;
    a = functionGetPoint();
    b = functionGetPoint();
    cout << "\n dist = " << distance(a, b);
    c = meanPoint(a,b);
    cout << "\n mean = (" << c.x << ", " << c.y << ")";

    Triangle Fig1, Fig2;
    cout << "\n-- Triangle 1 -- \n";
    GetTriangleWithFunction(Fig1);
    cout << "\n-- Triangle 2 -- \n";
    GetTriangleWithProcedure(&Fig2);
    cout << "\n Test Result = " << testTriangles(Fig1, Fig2);

    system("PAUSE");
    return 0;
} // end main

Point meanPoint(Point p1, Point p2) {
    Point mean;
    mean.x = (p1.x + p2.x)/2;
    mean.y = (p1.y + p2.y)/2;
    return mean;
} // end meanPoint
```

```
double distance(Point p1, Point p2) {
     double d, xdiff, ydiff;
     xdiff = p1.x - p2.x;
     ydiff = p1.y - p2.y;
     d = sqrt(xdiff*xdiff + ydiff*ydiff);
     return d;
} // end distance

void procedureGetPoint(Point *p) {
    cout << "\n Point coordinates: ");
    cout << "\n x = "; cin >> (p->x); // cin >> ((*p).x))
    cout << "\n y = "; cin >> (p->y);
} // end procedureGetPoint


Point functionGetPoint(void) {
    Point p;
    cout << "\n Point coordinates: ";
    cout << "\n x = "); cin >> p.x;
    cout << "\n y = "); cin >> p.y;
    return p;
} // end functionGetPoint

void GetTriangleWithFunction(Triangle t) {
     int i;
     for(i = 1; i <= 3; i++) {
        cout << "\n Vertex : " << i;
        t[i] = functionGetPoint();
        }
     return ;
} // end GetTriangleWithFunction

void GetTriangleWithProcedure(Point *t){
     int i;
     for(i = 1; i <= 3; i++) {
        cout << "\n Vertex : " <<  i;
        procedureGetPoint(&(*(t+i)));
     }
     return ;
} // end GetTriangleWithProcedure


void swapDoubles(double *one, double *two) {
     double t = *one;
     *one = *two;
     *two=t;
} // end swapPoints
```

```
int testTriangles(Triangle t1, Triangle t2) {

    double edgesT1[3], edgesT2[3];
    int i, count;

    for (i=0; i < 3; i++) {
       edgesT1[i] = distance(t1[i], t1[(i+1) % 3]);
       edgesT2[i] = distance(t2[i], t2[(i+1) % 3]);
    }

    if (edgesT1[0] < edgesT1[1])
      swapDoubles(&edgesT1[0], &edgesT1[1]);
    if (edgesT1[0] < edgesT1[2])
      swapDoubles(&edgesT1[0], &edgesT1[2]);
    if (edgesT1[1] < edgesT1[2])
      swapDoubles(&edgesT1[1], &edgesT1[2]);

    if (edgesT2[0] < edgesT2[1])
      swapDoubles(&edgesT2[0], &edgesT2[1]);
    if (edgesT2[0] < edgesT2[2])
      swapDoubles(&edgesT2[0], &edgesT2[2]);
    if (edgesT2[1] < edgesT2[2])
      swapDoubles(&edgesT2[1], &edgesT2[2]);

    for( i=0; (i<3) && (edgesT1[i] == edgesT2[i]); i++);

    return (!(i == 3));
} // end testTriangles
```