1 Esercizio 1

1. Immaginate di avere una calcolatore che invece che poter rappresentare i classici due valori per ogni bit (0/1) possa rappresentare 5 valori per ogni bit.

   (a) Quanti bit gli occorreranno per rappresentare i numeri naturali fino a 600 ?

   Per rappresentare 600 in base 5 sono necessari 4 bit infatti con 3 bit posso rappresentare numeri fino a 124 ($5^3 = 125$) mentre con 4 bit posso rappresentare numeri fino a 624 ($5^4 = 625$).

   (b) Quanti bit risparmia rispetto ad una macchina che invece usa la classica rappresentazione binaria?

   Per rappresentare 600 in base 2 ho bisogno almeno di 10 bit ($2^9 = 512$ e $2^{10} = 1024$). Quindi risparmio 6 bit.

2. Spiegare brevemente qual è la funzione del compilatore e la sua importanza per il programmatore.

   Vedere libro di testo.

3. Spiegare, magari con un esempio, il significato di ognuna delle fasi del ciclo fetch-decode-execute nella macchina di Von Neumann.

   Vedere libro di testo.

2 Esercizio 2

Si scrivano, nel linguaggio **C**, le dichiarazioni di tipo necessarie per rappresentare un punto in uno spazio tridimensionale usando la rappresentazione in coordinate cartesiane (si chiami **Punto** il tipo corrispondente) ed un insieme di, al massimo, 100 punti (si chiami **InsiemePunti** il tipo corrispondente).

Si scriva un programma in linguaggio C che, data una variabile di tipo **InsiemePunti**, permetta all'utente di immettere un numero grande a piacere di elementi di tipo **Punto** (ovviamente tale numero non può essere superiore a 100), e che, alla fine dell'immissione, stampi quanti, tra gli elementi di tipo **Punto** immessi, hanno la caratteristica di avere lo stesso valore delle tre coordinate cartesiane (si pensi ad esempio ad un punto con coordinate x=3 y=3 z=3) e quale di questi è il più vicino dall'origine degli assi.

```
#include <stdio.h>

#define MAX_PUNTI 100
#define MAX_DIST 10000

typedef struct {
  int x;
  int y;
  int z;
} Punto;

typedef struct {
  Punto lista[MAX_PUNTI];
  int dim;
} InsiemePunti;

void main() {

  int i, contaUguali, distanza;
  int min, minIndice;;
  char risposta;
  InsiemePunti punti;
  /* L'array è vuoto, quindi inizializzo a zero la sua dimensione
     effettiva */
  punti.dim = 0;

  contaUguali = 0;
  i = 0;

  /* Suppongo che nessun numero abbia distanza dall'origine superiore
     a MAX_DIST */
  min = MAX_DIST;
```

```
printf("Vuoi inserire un punto (s per continuare) ? ");
scanf("%c%*c", &risposta);

/* Leggo le coordinate dei punti */
while((i < MAX_PUNTI) && (risposta == 's')) {
  printf("Inserisci la coordinata x: ");
  scanf("%d%*c", &punti.lista[i].x);
  printf("Inserisci la coordinata y: ");
  scanf("%d%*c", &punti.lista[i].y);
  printf("Inserisci la coordinata z: ");
  scanf("%d%*c", &punti.lista[i].z);
  /* Ogni volta che inserisco un elemento devo incrementare la
     dimensione effettiva */
  punti.dim = punti.dim + 1;

  printf("Vuoi inserire un punto (s per continuare) ? ");
  scanf("%c%*c", &risposta);
  i = i + 1;
}

/* Cerco gli elementi con coordinate uguali e tra questi trovo
   quello con distanza minima dall'origine (NB: il for procede fino
   a punti.dim, cioè fino alla dimensione effettiva) */
for(i = 0; i < punti.dim; i = i + 1) {
  if((punti.lista[i].x == punti.lista[i].y) &&
     (punti.lista[i].y == punti.lista[i].z)) {

    contaUguali = contaUguali + 1;

    /* Questo test mi serve per gestire correttamente i punti con
       coordinate negative */
    if(punti.lista[i].x < 0) {
      distanza = punti.lista[i].x * (-1);
    } else {
      distanza = punti.lista[i].x;
    }

    if(distanza < min) {
      min = distanza;
      minIndice = i;
    }
  }
}
printf("Hai inserito %d punti uguali\n", contaUguali);
```

```
  if(contaUguali > 0) {
    printf("Tra questi il punto più vicino è il punto %d (%d, %d, %d)\n",
           minIndice, punti.lista[minIndice].x,punti.lista[minIndice].y,
           punti.lista[minIndice].z);
  }
}
```

3 Esercizio 3

Sia dato il seguente programma in C:

```
void main(){

  int x, y, z;
  int *p1, *p2, *p3;

  x=10;
  y=9;
  z=8;

  /* Queste sono le istruzioni da inserire */
  p1 = &z;
  p3 = &x;
  p2 = &y;
  /****************************************/


  printf("%d  %d  %d %d %d %d \n",x,y,z,*p1,*p2,*p3);

  if (*p2 == 9) {

    p1 = p3;

  } else {

    *p1=7;

  }

  printf("%d  %d  %d %d %d %d \n",x,y,z,*p1,*p2,*p3);

}
```

1. Lo si completi scrivendo le istruzioni che fanno puntare `p1` a `z`, `p3` a `x` e `p2` a `y` e, dopo averlo completato, si dica qual è l'output del programma.

   La prima `printf` stampa `10, 9, 8, 8, 9, 10`
   La seconda `printf` stampa `10, 9, 8, 10, 9, 10`

2. Se avessi un puntatore dichiarato come

   ```
   int ** qq;
   ```

   potrei assegnare a tale puntatore l'indirizzo di p2? e potrei assegnare, sempre a tale puntatore, il valore contenuto in p1? Giustificare brevemente le risposte (non saranno accettate risposte del tipo si/no)

   La prima risposta è si: `qq` è un puntatore ad un puntatore ovvero può contenere l'indirizzo di un puntatore. Quindi l'istruzione `qq = &p2` è corretta.
   Viceversa la seconda risposta è no: infatti non si può assegnare il valore contenuto in `p1`, ovvero l'indirizzo di una variabile intera, ad un puntatore a puntatore, che come detto può contenere solo indirizzi di altri puntatori.