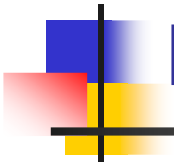


Definizione classi

(Per altri esercizi su questo argomento, si considerino quelli del 2006-2007)

Campionato di calcio

Si vuol realizzare un archivio dati che contiene informazioni su un insieme di campionati di calcio nazionali.

- Per ogni campionato si memorizza la nazione, l'anno, oltre alle squadre partecipanti e il tabellone dei risultati
- Ogni squadra è caratterizzata da un nome e una città
- Delle partite si registrano la data e il risultato
- Per l'insieme delle squadre partecipanti al campionato si usi un vettore
- Per l'insieme dei campionati si usi una lista
- L'applicazione dovrà avere un'interfaccia utente e si vuol seguire il principio di separazione fra interfaccia e dati

Campionato di calcio

Si definiscano i metodi della classe archivio affinché siano possibili le seguenti operazioni

- creare un archivio con un numero massimo di campionati
- inserire un campionato
- inserire una squadra in un campionato
- calcolare i punti di una squadra indicando il nome della squadra, il nome del campionato e l'anno
- cercare una squadra indicando il nome
- cercare tutte le squadra partecipanti ad un campionato un certo anno

- Progettare le classi necessarie indicando le variabili, i metodi e i costruttori senza implementarli
- L'applicazione dovrà avere un'interfaccia utente e si vuol seguire il principio di separazione fra interfaccia e dati

Campionato di calcio: chiarimenti

- Si noti che la descrizione del problema potrebbe essere non esaustiva e che la definizione Java potrebbe richiedere di specificare classi o variabili corrispondenti a concetti non direttamente citati nel testo.
- Ad esempio, si supponga che sia richiesta la realizzazione di un archivio dei corsi e degli studenti, tenendo traccia delle ore seguite da ciascun studente in ciascun corso.
- Esplicitamente sono menzionati i concetti "Studente" e "Corso" a cui corrisponderanno due classi. Mentre, per memorizzare le ore seguite si potrebbe introdurre una classe "Presenze", che abbia come variabili uno studente, un corso e il numero di ore seguite, e inserire un array di Presenze nell'archivio. Un'altra soluzione consiste nell'inserire un array di Presenze nella classe Studente (in tal caso la classe Presenze contiene solo una variabile corso e una ore). Comunque, è essenziale che la soluzione sia tale da poter calcolare quante sono le ore di presenza di un ogni studente a ogni corso, perché ciò è richiesto dal problema.

Campionato di calcio: chiarimenti

- Poiché si richiede di separare l'interfaccia utente e dai dati, occorrerà prevedere una classe separata che implementa l'interfaccia.
- E' ragionevole supporre che l'interfaccia abbia al suo interno un oggetto Archivio che contiene i dati e i metodi descritti dal testo del problema
- E' anche ragionevole pensare che i metodi della classe interfaccia possono essere gli stessi della classe Archivio con la differenza che non hanno parametri in ingresso e non restituiscono e che non restituiscano niente. Di fatti ciascun metodo della classe interfaccia potrebbe essere un metodo che permetta all'utente di chiamare i metodi dell'archivio: per ottenere questo scopo avrà bisogno interagire con l'utente per avere dei dati in ingresso e per stampare i risultati
- E' infine ragionevole pensare che la classe interfaccia contenga un metodo che ha lo scopo di mostrare all'utente un menù di opzioni corrispondenti alle funzionalità disponibili

Schema di un'interfaccia

```
class Interfaccia{
  Archivio A;

void menu(){
   while(true){
      //stampa il menu e attendi
      //che l'utente scelga un metodo da eseguire
      if(metodoScelto==1){metodo1();}
      if(metodoScelto==2){metodo2();}
      }
   }
 void metodo1(){
         // chiedi all'utente i dati necessari a
         //chiamere metodo1 della classe Archivio
        r=A.metodo1(.....)
        //Stampa i risultati di metodo1
}

void metodo2(){
         // simile a metodo1
}
.... //altri metodi
```

Lo schema di una classe generica Archivio con la sua Interfaccia

```
class Archivio{

.... metodo1(....){
     .....
 }

.... metodo1(....){
     .....
 }

.... metodo1(....){
     .....
 }

}
```

Campionato di calcio: soluzione

```
class Squadra{
   String nome, citta;
}
```

```
class Partita{
   Squadra squadraDiCasa, squadraOspite;
   Date data;
   int goalSquadraDiCasa, goalSquadraOspite;
}
```

```
class Campionato{
  Squadra  partecipanti[];
  Partita  tabellone[][];
  String nazione, anno;
}
```

```
class ListaCampionati{
 ItemCampionato head;
}
```

```
class ItemCampionato{
  Campionato  value;
  ItemCampionato next;
}
```

La classe Partita contiene le squadre di casa e quella ospite, nonostante questo non fosse indicato espicitamente nel testo.
D'altra parte senza questo inserimento non sarebbe possibile sapere fra quali squadre si tiene la partita

Le partecipanti in Campionato potrebbero essere omesse perchè implicitamente contenute intabellone

Campionato di calcio: una soluzione alternativa

```
class Squadra{
   String nome, citta;
   Partita partite[];
}
```

```
class Partita{
   boolean inCasa;
   Squadra avversaria;
   Date data;
   int goalSquadraDiCasa, goalSquadraOspite;
}
```

```
class Campionato{
  Squadra  partecipanti[];
  String nazione, anno;
}
```

```
class ListaCampionati{
 ItemCampionato head;
}
```

```
class ItemCampionato{
  Campionato  value;
  ItemCampionato next;
}
```

In questa seconda soluzione, le partite sono nella classe squadra

Per questo motivo, in partita, è sufficiente tenere traccia di una delle due squadre (l'avversaria) perchè l'altra si può conoscere guardando in quale squadra si trova la partita.

Inoltre, nel campionato è suficciente indicare la partecipanti, dato che le partite sono dentro le squadre

Campionato di calcio: soluzione

```
class Archivio{
   ListaCampionati lista;
   Archivio(int numeroCampionati)
   void  inserisci(Campionato c)
   void  inserisci(Squadra s, String nazione, String anno)
   Squadra cerca(String nome)
   Squadra[] cerca(String nome, String anno)
}
```

```
class Interfaccia{
   Archivio A;

   void menu();
   void  inserisciCampionato();
   void  inserisciSquadra();
   void cercaSquadraperNome();
   void cercaSquadraInCampionato();
}
```

Olimpiadi

Si vuol realizzare un archivio dati delle olimpiadi

- Per ogni olimpiade si memorizzano il luogo e l'anno
- Si dovranno registrare anche partecipanti e competizioni, tenendo traccia del risultato di ogni partecipante che può essere registrato come una stringa generica
- I partecipanti e le competizioni dovranno essere memorizzati con un vettore. L'insieme di olimpiadi con una lista
- Per i partecipanti si registrano il nome, la data di nascita e la nazionalità
- Per le competizioni si registra il nome e un campo che definisce se la competizione è dimostrativa o meno

Olimpiadi

Si definiscano i metodi della classe archivio affinché siano possibili le seguenti operazioni

- creare un archivio
- inserire un'olimpiade
- inserire una competizione indicandone il nome, l'olimpiade e se è dimostrativa o no
- calcolare il numero delle presenze di un certo atleta
- cercare tutte le olimpiadi a cui ha partecipato un atleta
- Progettare le classi necessarie indicando le variabili, i metodi e i costruttori senza implementarli
- L'applicazione dovrà avere un'interfaccia utente e si vuol seguire il principio di separazione fra interfaccia e dati
- Si vedano anche i suggerimenti dati per l'esercizio precedente

Olimpiadi: soluzione

```
class Olimpiade{
   String luogo;
   int anno;
   Atleta atleti[];
   Partecipazione partecipazioni[];
   Competizione competizioni[];
}
```

```
class List{
 Item head;
}
```

```
class Item {
 Olimpiade  value;
 Iitem next;
}
```

```
class Atleta{
   String nome, mazionalita;
   Date dataNascita;
}
```

```
class Partecipazione{
  Altleta  partecipante;
  String risultato;
  Competizione gara;
}
```

```
class Competizione{
  String nome;
  boolean dimostrativa;
}
```

La classe partecipazione serve a stabilire una connessione fra chi ha partecipato, a cosa ha partecipato e il risultato

Gli atleti e le competizioni potrebbero anche essere omessi, perchè già presenti in partecipazioni

Olimpiadi: altra soluzione

```
class Olimpiade{
   String luogo;
   int anno;
   Atleta atleti[];
}
```

```
class List{
 Item head;
}
```

```
class Item {
 Olimpiade  value;
 Iitem next;
}
```

```
class Atleta{
   String nome, mazionalita;
   Date dataNascita;
   Particazione partecipazioni[];
}
```

```
class Partecipazione{
  String risultato;
  Competizione gara;
}
```

```
class Competizione{
  String nome;
  boolean dimostrativa;
}
```

In questa seconda soluzione, le partecipazioni sono negli atleti

Per questo motivo, in partecipazione, non è necessario tenere traccia dell'atleta perchè questo può essere dedotto dall'oggett in cui si trova.

Inoltre, nell'olimpiade è sufficiente tenere trraccia degli atleti, perchè le altre informazioni sono presenti al loro interno

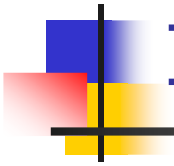
Olimpiadi: soluzione

```
class Archivio{
   List olimpiadi;
   Atleta atleti[];

   Archivio();
   void  inserisci(Olimpiade o);
   void  inserisciCompetizione(String nome, boolean demo. String luogoOlimpiade, int annoOlimpiade)
   int contaPresenze(String nome, Date dataNascita )
   Olimpiade[] cercaPresenze(String nome, Date dataNascita )
}
```

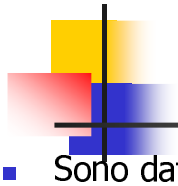
```
class Interfaccia{
   Archivio A;

   void menu();
   void  inserisci();
   void  inserisciCompetizione();
   void contaPresenze();
   void cercaPresenze();
}
```



Implementazione metodi

(Per altri esercizi su questo argomento, si considerino quelli del 2006-2007)



Esami degli studenti

- Sono date le classi sottostanti che definiscono uno studente e un esame
- Si osservi che ogni studente contiene un insieme di esami organizzati in una lista
- Si scriva un metodo che preso in ingresso un array di studenti e cerchi tutti gli esami superati con la lode, stampando il nome e il cognome dello studente e il nome dell'esame.

```
class Studente{
   String nome, cognome;
   ListaEsami esami;
}

class Esame{
   int corso;
   int voto;
   boolean lode;
}
```

```
class ListaEsami{
   ItemEsami head;
}

class ItemEsami{
   ItemEsami next;
   Esame value;
}
```

metodo da implementare

```
void stampaLodi(Studente s[])
```

Esami degli studenti: soluzione

```
void stampaLodi(Studente s[]){

  for(int i=0;i<s.length;i++){
      for(ItemEsami it=s[i].esami.head;it!=null;it=it.next){
            if(it.value.lode){
                System.out.println(s[i].nome+" "+s[i].cognome+" "+it.value.corso)
            }
      }
  }
}
```

Numero elementi di un albero ternario

- Sono date le classi che implementano un albero ternario (ogni nodo a tre figli) di interi
- Si implementi un metodo ricorsivo che calcola il numero dei nodi dell'albero
- Il metodo da realizzare è ricorsioneNumeroElmenti (vedi sotto)

```
class Node{
   int value;
   nodo child1, child2, child3;
}
```

```
class Tree{
  root;
}
```

```
int numeroElementi(Tree T){
            return ricorsioneNumeroElmenti(T.root);
}

int ricorsioneNumeroElmenti(nodo n);
```

Numero elementi di un albero ternario: soluzione

```
int ricorsioneNumeroElmenti(nodo n){

  if(n==null){return 0;}

  else {return 1 + ricorsioneNumeroElmenti(n.child1)
                 + ricorsioneNumeroElmenti(n.child2)
                 + ricorsioneNumeroElmenti(n.child3);}


}
```

Polinomio calcolato ricorsivamente: soluzione

- Si implementi un metodo ricorsivo che data una lista di interi che contiene i coefficienti di un polinomio e numero reale x, calcoli $\sum_{i=0}^{n} a_i x^i$ dove a_i è l'intero contenuto nella posizione i della lista
- Suggerimento: si implementi un metodo ricorsivo (vedi polinomioRicorsivo sotto) che prende in ingresso il numero reale x, un nodo e un indice i che indica la posizione del nodo della lista
- Si usi Math.pow(x,i) per calcolare x^i

```
int polinomio(float x, List L){
            return polinomioRicorsivo(x,L.head,0);
}

int polinomioRicorsivo (float x, Item it, int i);
```

```
class List{
 Item head;
}
```

```
class Item {
 int  value;
 Iitem next;
}
```

Polinomio calcolato ricorsivamente

```
int polinomioRicorsivo (float x, Item it, int i){

if(it==null) {return 0;}
   else {return it.value*Math.pow(x,i)+ polinomioRicorsivo (x,it.next,i+1);

}
```

Labirinto

- Un labirinto è definito da un array di array di interi A. Di fatti A è lungo come il numero dei bivi che ci sono nel labirinto. Il vettore A[i] è lungo quanto il numero delle strade possibili che partono dal bivio i. In A[i][j] c'è un numero che indica quale bivio si raggiunge scegliendo la j-esima strada quando siamo nel bivio i
- Si implementi un metodo iterativo che a partire da un bivio dato "Start" simuli un cammino a caso nel labirinto fino a quando non arriva alla posizione 0. Si conti il numero di bivi incontrati nel percorso
- Si utilizi Random.nextInt(b) per generare un valore intero a caso compreso fra 0 (incluso) e b (escluso)

```
int cammino(int A[][], int start)
```

Labirinto: esempio di cammino

- (Si supponga di partire dalla posizione 4)
- Si sceglie una strada a caso fra quelle disponibili in 4, che sono definite dall'array A[4]. (Si supponga di scegliere 3.)
- Allora si va al bivio A[4][3]. (Si supponga che A[4][3] sia 7.)
- Si sceglie una strada a caso fra quelle disponibili in 7, che sono definite dall'array A[7]. (Si supponga di scegliere 2.)
- Allora si va al bivio A[7][2].
- …. si continua così, contando i bivi incontrati

Labirinto: soluzione

```
int cammino(int A[][], int start){
int numBivi=0;
int bivioCorrente=start;

while(bivioCorrente!=0){
   int stradaScelta=Random.nextInt(A[bivioCorrente]);
   bivioCorrente=A[bivioCorrente][stradaScelta];
   numBivi++;
  }

return numBivi;

}
```

Labirinto ricorsivo

- Si risolva l'esercizio precedente usando una funzione ricorsiva
- Suggerimento: è possibile scrivere un metodo ricorsivo che prenda in ingresso l'array che definisce il labirinto e la posizione attuale

```
int cammino(int A[][], int bivioCorrente)
```

Labirinto ricorsivo: soluzione

```
int cammino(int A[][], int bivioCorrente){

 if (bivioCorrente==0){ return 0;}
  else{stradaScelta=Random.nextInt(A[bivioCorrente]);
        return 1+cammino(A, A[bivioCorrente][stradaScelta]);
       }

}
```

Complessità

(Per altri esercizi su questo argomento, si considerino quelli dello 2006-2007)

Serie utili

1) $$\sum_{i=1}^{a} i = \frac{a(a+1)}{2}$$

2) $$\sum_{i=1}^{a} i^2 = \frac{2a^3+3a^2+a}{6}$$

3) $$\sum_{i=1}^{a} b^i = \frac{b^{a+1}-1}{b-1}$$

Complessita' di un programma: schema generale per for annidati

- Calcolare l'ordine asintotico (notazione O(f(n))) delle operazioni svolte dal seguente metodo rispetto al valore del parametro n
- Siano date due funzioni h e g, vediamo cosa succede al variare di h e g
- Si osservi che le operazioni più numerose sono quelle del ciclo più interno. Per cui ci si limiterà a calcolare il numero di volte che viene eseguita la moltiplicazione indicata dalla freccia

```
void metodo1(int n){

  int res=1;
  for(int i=1;i<=h(n)=++){
       for(j=1;j<g(i,n);j++){
          res=res*2;
       }
   }
}
```

Complessita' di un programma: schema generale per for annidati

- Per ogni iterazione del ciclo esterno, il ciclo interno viene eseguito g(i,n) volte. Poiché quello esterno è eseguito h(n) volte, avremo

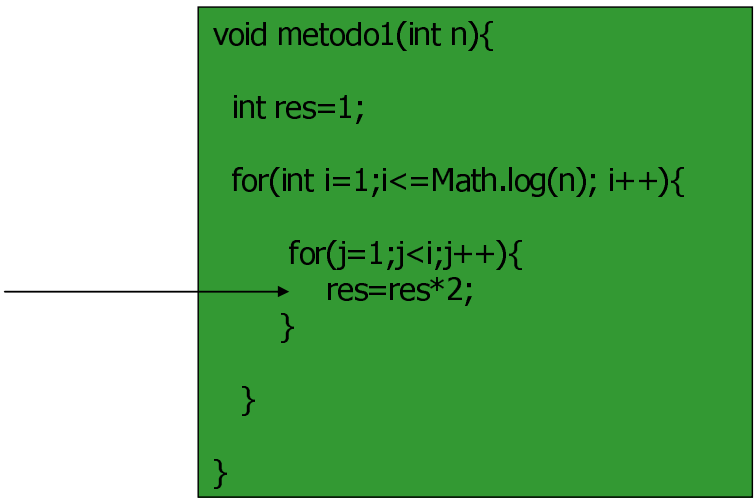
$$T(n) = \sum_{i=1}^{h(n)} g(i,n)$$

- Per sviluppare la sommatoria dobbiamo controllare se è una serie nota oppure se ha delle caratteristiche particolari che ci permette di calcolarla
- Ad esempio, se g è indipendente da i, cioè g(i,n)= r(n) per una qualche funzione r, avremo

$$T(n) = \sum_{i=1}^{h(n)} r(n) = r(n) \cdot h(n)$$

Complessita' di un programma: un esempio

- Calcolare l'ordine asintotico (notazione O(f(n))) del numero di volte che viene eseguita la moltiplicazione indicata dalla freccia rispetto al valore del parametro n



```
void metodo1(int n){

  int res=1;

  for(int i=1;i<=Math.log(n); i++){

        for(j=1;j<i;j++){
            res=res*2;
        }

   }

}
```

Complessita' di un programma: soluzione

- Il numero delle iterazioni del for interno dipende dal valore i definito dal for esterno. Applicando la formula generale e la serie (1) abbiamo

$$T(n) = \sum_{i=1}^{\log n} i = (\log n)(1+\log n)$$

Quindi, la risposta è $O\left((\log n)^2\right)$

Complessita' di un programma: un'altro esempio

- Calcolare l'ordine asintotico (notazione O(f(n))) delle operazioni svolte dal seguente metodo rispetto al valore del parametro n
- Si osservi che le operazioni più numerose sono quelle del ciclo più interno. Per cui ci si limiterà a calcolare il numero di volte che viene eseguito il ciclo interno

```
void metodo1(int n){

  int res=1;

  for(int i=1;i<=Math.pow(n,3); i++){
        j=1
        while(j<Math.pow(n,2)){
            res=res*2;
            j=j*2;
        }
   }
}
```

Complessita' di un programma: soluzione

- Il numero delle iterazione del ciclo esterno è n^3.
- Il numero delle iterazioni del for interno è indipendente da quello esterno.
- Nel ciclo interno la variabile j viene tutte le volte moltiplicata per 2 e quindi segue una serie geometrica, cioè
  1, 2, 2^2, 2^3, 2^4,... Quindi, il ciclo interno verrà eseguito $\log_2 n^2$ volte.
- Poiché i due cicli sono indipendenti, il numero delle esecuzioni è data dal prodotto delle iterazioni dei due cicli.

$$T(n) = n^3 \log n^2 = 2n^3 \log n$$

Quindi, la risposta è $O\left(n^3 \log n\right)$

Complessita': esempio di un programma ricorsivo

- Calcolare l'ordine asintotico (notazione O(f(n))) delle operazioni svolte dal seguente metodo2 rispetto al valore del parametro n
- Si osservi che le operazioni più numerose sono quelle del ciclo for. Per cui ci si limiterà a contare il numero di iterazioni del for

```
int metodo2(int n){
        return metodo1(n);
}

int metodo1(int m){
  if(m==1) {return 0}
  else{int res=0;
        for(int i=1;i<m;i++){
                res=res+i;
        }
        return res+metodo1(m/2);
    }
}
```

Complessita' di un programma ricorsivo: soluzione

- Il ciclo for viene eseguito m volte, per ogni chiamata di metodo1
- metodo1 si richiama ricorsivamente dividendo per 2 il parametro m. Le chiamate ricorsive si fermano per m=1.
- Quindi partendo da n ci saranno delle chiamate ricorsive per i seguenti valori n, n/2 $n/2^2$, $n/2^3$, $n/2^4$… Alla k-esima chiamata, m sarà uguale a $n/2^k$ e ci si fermerà per $n/2^k = 1$, cioè alla chiamata $k=\log_2 n$
- Da questo si ricava la seguente formula che può essere calcolato usando la serie (3) data in una slide precedente

$$T(n) = \sum_{k=1}^{\log_2 n} \frac{n}{2^k} = n \sum_{k=1}^{\log_2 n} \left(\frac{1}{2}\right)^k = n \left(\frac{\left(\frac{1}{2}\right)^{1+\log_2 n} - 1}{\frac{1}{2} - 1} \right) = n \left(\frac{\left(\frac{1}{2}\right) \cdot \left(\frac{1}{2^{\log_2 n}}\right) - 1}{-\frac{1}{2}} \right) = n\left(2 - \frac{1}{n}\right)$$

Quindi, la risposta è $O(n)$

Complessita': esempio di un programma ricorsivo

- Calcolare l'ordine asintotico (notazione O(f(n))) delle operazioni svolte dal seguente metodo2 rispetto al valore del parametro n
- Si osservi che le operazioni più numerose sono quelle del ciclo for. Per cui ci si limiterà a contare il numero di iterazioni del for

```
int metodo2(int n){
            return metodo1(Math.pow(n,4));
}

int metodo1(int m){
  if(m==1) {return 0}
  else{int res=0;
            for(int i=1;i<Math.pow(m,2);i++){
                        res=res+i;
            }
            return res+metodo1(m-1);
      }
}
```

Complessita' di un programma ricorsivo: soluzione

- Il ciclo for viene eseguito m^2 volte, per ogni chiamata di metodo1
- metodo1 si richiama ricorsivamente decrementando di il parametro m. Le chiamate ricorsive si fermano per m=1. Quindi ci saranno chiamate ricorsive per m uguale a n^4, n^4-1, n^4-2, ….
- Da questo si ricava la seguente formula che può essere calcolato usando la serie (2) data in una slide precedente

$$T(n) = \sum_{k=n^4}^{1} k^2 = \frac{2\left(n^4\right)^3 + 3\left(n^4\right)^2 + n^4}{6}$$

Quindi, la risposta è $O\left(n^{12}\right)$

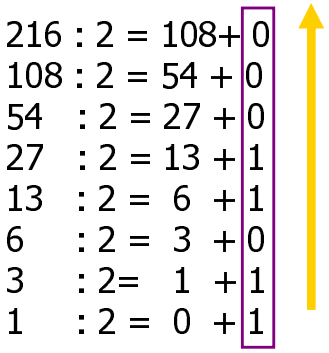
Rappresentazioni e operazioni su numeri binari

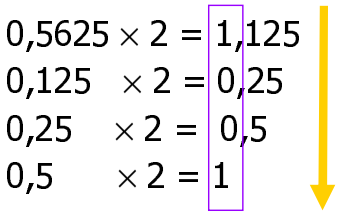
Conversioni

- Dati li seguente numero in base 10 convertirli in binario e in esadecimale. Si usi una rappresentazione in virgola fissa dedicando 8 bit alla parte intera e 4 alla parte frazionaria.
- 216,5625

Soluzioni ...

$(216{,}5625)_{10} = (1101\ 1000\ ,\ 1001)_2 = (D\ 8\ ,\ 9)_{16}$



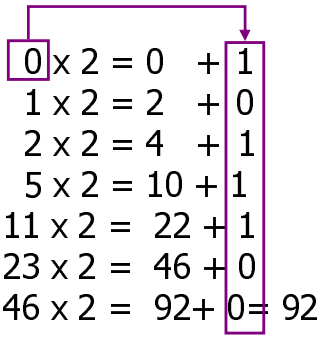


Conversioni

- Dati il seguente numeri in binario convertirlo in base 10
- 0101 1100

Soluzioni ...

$(0101\ 1100)_2 = (92\ )_{10}$



Rappresentazione in complemento a 2

- Dati i seguenti numeri in base 10 rappresentarli in complemento a 2 e in modulo e segno
- i numeri 92, -92 usando 8 bit

Soluzione ...

- $(92)_{10} = (0101\ 1100)_2$

In modulo e segno (8 bit) 92 è 0101 1100, -92 è 1101 1100
In complemento a 2, 92 è 0101 1100, -92 è 1010 0100

Per calcolare il complemento a 2 di un numero

- si scorre il numero da destra fino a quando non si trova un uno: i numeri successivi si complementano
- oppure si complementa tutto il numero e si somma 1

0101 1100 ← complemento le cifre dopo il primo uno

1010 0100

0101 1100 ← complemento tutte le cifre

1010 0011

\+ 1

1010 0100

Somma in complemento a 2

- Calcolare la sottrazione 45 - 92, usando 8 bit e rappresentazioni in complemento a 2

Soluzione …

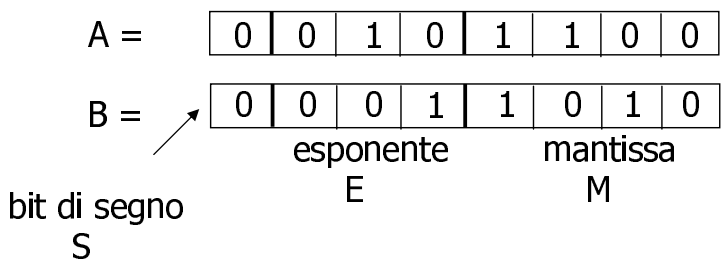
- 45 in complemento a 2 è 0010 1101
- -92 in complemento a è 1010 0100 (vedi l'esercizio precedente)
- 45-92 si ottiene sommando le rappresentazioni di 45 e -92

```
  0010 1101
  1010 0100
  ---------
  1101 0001
```

- La somma non ha generato overflow (ovvio visto che si sottrae due numeri positivi) perché i due bit di riporto relativi al nono e all'ottavo bit della rappresentazione sono entrambi 0

Somma in virgola mobile

- Dati i due seguenti numeri rappresentati in virgola mobile in forma standard, con 1 bit per il segno, 3 bit per l'esponente (rappresentazione in eccesso a 7), 4 bit per la mantissa dire quali numeri rappresentano e sommarli



Soluzione ...

- Si ricorda che la formula che descrive il numero rappresentato dalla forma standard è $(-1)^s$ 1.M 2^{E-7}
- Per

0	0	1	0	1	1	0	0

- il numero è positivo visto che il bit di segno è 0
- l'esponente è il valore di $(10)_2$ meno l'eccesso 7, cioè 2-7= -5
- dato che la mantissa è 11, il numero senza la parte dell'esponente è $(1.11)_2$, cioè 1,75
- quindi A rappresenta il numero 1,75 x 2^{-5}
- Similmente, B rappresenta il numero 1,625 x 2^{-6}

Soluzione ...

Per sommare i numeri A e B

- l'esponente di B è più piccolo di quello di A di un'unita, per cui si sposta la virgola della mantissa di B (cioè 1.1010) a sinistra di una posizione. Si ottiene M_B=0.1101
- Quindi si sommano le mantisse

$$\begin{array}{ll} M_A & 1.1100 \\ M_B & 0.1101 \\ \hline M_R & 10.1001 \end{array}$$

- L'esponente del risultato è quello di A, E_R=010
- A questo punto occorre riportare il numero in forma normale: si sposta a sinistra la virgola della mantissa e si incrementa di uno l'esponente
- Alla fine abbiamo E_R=011 e M_R=1.01001
- Quindi, il risultato è

0	0	1	1	0	1	0	0