Informatica - CdL in FISICA
II Prova di verifica del 1/6/2012

Scrivere **in stampatello** COGNOME, NOME e MATRICOLA su ogni foglio consegnato

N.B.: Negli esercizi di programmazione, viene valutata anche la leggibilità del codice proposto. Inoltre, non è consentito l'uso di istruzioni che alterino il normale flusso dell'esecuzione (come, ad esempio, `continue`, `break` e istruzioni di `return` all'interno di cicli che ne provochino l'uscita forzata). Infine non è consentito l'uso di variabili statiche.
Laddove è utilizzato, il tipo `boolean` è definito da `typedef enum {false, true} boolean;`

ESERCIZIO 1 (4 punti)

Indicare i valori stampati dal seguente programma `C`.

```
#include <stdio.h>
  main()
  {
    int v = 3;
    int w = 1;
    int s = 0;
     {
       int w;
       w = 5;
       v=w+2;
       s = foo(w,v);
       v = w + s+v;
       printf("Prima stampa: v = %d\n w = %d\n s = %d\n", v,w,s);
        }
      v=  w+s+v;
      printf("Seconda stampa:v = %d\n w = %d\n s = %d\n", v,w,s);
    }

int foo(int v,int w)
   {
     int * s;
     s=&v;
     w = v + 2*w;
     v=2+w+*s;
     return (*s);
 }
```

Soluzione
Nel primo blocco, una nuova variabile w viene dichiarata e questa "fa ombra" alla variabile w originale (che resta intoccata per tutto il blocco). Il valore di v é quindi 7 prima della chiamata a foo.
Notare come le variabili attuali e formali della chiamata a foo sono scambiate, quindi in foo abbiamo $v = 5$ e $w = 7$
In foo, s viene fatto puntare a v, w viene impostato come $5 + 2 * 7$ e quindi assume valore 19 mentre a v viene assegnato il valore $2 + 19 + 5$ cioé 26. Questo é anche il valore di ritorno di foo, visto che s punta proprio a v.
Nel blocco quindi s riceve dalla chiamata a foo il valore 26 e di conseguenza v diventa $5 + 26 + 7 = 38$.
La prima stampa é quindi questa:

```
Prima stampa:   v = 38, w = 5, s = 26
```

Fuori dal blocco, w torna ad assumere valore 1 e quindi v riceve il valore $1 + 26 + 38 = 65$.
Da questo la seconda stampa:

```
Seconda stampa: v = 65, w = 1, s = 26
```

ESERCIZIO 2 (4+2 punti)
Si definisca una funzione che dato un array *vet* di interi ordinati in maniera crescente e tutti diversi tra loro e un intero k, restituisca true se esiste una coppia di interi la cui somma e' esattamente uguale a k.
Ad esempio, se vet e'

2	3	6	8	10

e k e' uguale a 11 la funzione deve restituire *true*.
Se k e' uguale a 6 deve restituire $false$.
N.B. Verranno premiate le soluzioni che scorrono il vettore al piu' una volta.

Soluzione
Soluzione standard con scorrimento multiplo del vettore:

```
boolean somma(int vet[], int dim, int k) {
  boolean trovato = false;
  int i=0, j, s;
  while(i<dim && !trovato) {
    j=i+1;
    while(j<dim && !trovato) {
      if(vet[i] + vet[j] == k)
        trovato = true;
      j++;
    }
    i++;
  }
  return trovato;
}
```

Soluzione efficiente con scorrimento del vettore una sola volta:

```
boolean somma(int vet[], int dim, int k) {
  boolean trovato = false;
  int i=0, j=dim-1;
  while(i<j && !trovato) {
    s = vet[i] + vet[j];
    if (s<k)
      i++;
    else {
      if(s>k)
        j--;
      else
        trovato = true;
    }
  }
  return trovato;
}
```

ESERCIZIO 3 (6 punti)
Definire in modo **ricorsivo** una procedura che, dato un array di caratteri, la sua dimensione, un carattere c e un intero k, verifichi che il carattere c non compaia *piu' di* k volte nell'array.
Ad esempio, dato il vettore

'f'	'a'	'}'	'a'	'c'	'2'	'i'	'a'	'l'	'e'

con il carattere $'a'$ e l'intero 3 la procedura deve restituire *true*, con il carattere $'a'$ e l'intero 4 la procedura deve ancora restituire *true*, mentre con il carattere $'a'$ e l'intero 2 la procedura deve restituire $false$.

Soluzione

```
boolean ktimes(char vet[], int dim, char c, int k) {
  if(k<0)
    return false;
  else
      if(dim==0)
```

```
        return true;
    else
        if(vet[0]==c)
            return ktimes(vet+1, dim-1, c, k-1);
        else
            return ktimes(vet+1, dim-1, c, k);
}
```

ESERCIZIO 4 (16 punti)
Si vuole modellare un gioco con le *carte* mediante una lista concatenata. Una carta e' rappresentata dal suo valore (un intero da 1 a 10) e dal suo seme (Quadri,Cuori, Picche e Fiori). Ogni nodo della lista deve rappresentare una carta con in aggiunta l'informazione di quante carte dello stesso seme seguono nella lista.

(i) (2 punti) Definire i tipi opportuni per rappresentare il gioco.

```
typedef enum { Quadri, Cuori, Picche, Fiori } Seme;

typedef struct carta {
  int val;
  Seme seme;
} Carta;

typedef struct nodo {
  Carta carta;
  int carteSuccessive;
  struct nodo* next;
} NodoMazzo;
typedef NodoMazzo* Mazzo;
```

(ii) (3 punti) Scrivere una funzione *ricorsiva* che data una lista di carte, un valore e il seme di una carta, controlli che tale carta appartenga alla lista *sfruttando al meglio tutte le informazioni contenute nella lista.*

```
boolean appartiene (Mazzo mazzo, int val, Seme seme) {
  if(mazzo==NULL)
    return false;
  else
        {if (mazzo->carta.seme==seme)
           { if  (mazzo->carta.val==val)
               return true;
           else
              {if( mazzo->carteSuccessive==0)
               return false;
              else   return appartiene(mazzo->next, val, seme);}
         }
         else   return appartiene(mazzo->next, val, seme);
          }}
```

(iii) (5 punti) Scrivere una procedura che data una lista di carte, un valore e il seme di una nuova carta, la inserisca prima della prima carta con lo stesso seme, se esiste, altrimenti la inserisca in coda.

```
void inserisci(Mazzo* p_mazzo, int val, Seme seme) {
  Mazzo  aux, nuova, prec;
  boolean done;
  if(p_mazzo!=NULL)  /* sanity check */
{
    nuova = malloc(sizeof(NodoMazzo));
    nuova->carta.val = val;
    nuova->carta.seme = seme;
    nuova->next = NULL;
```

```
if(*p_mazzo==NULL) {
  nuova->carteSuccessive = 0;
  *p_mazzo = nuova;
}
```

```
    else if( *p_mazzo ->carta.seme==seme) {
      nuova->carteSuccessive = *p_mazzo->carteSuccessive + 1;
      nuova->next = *p_mazzo;
      *p_mazzo = nuova;
    }
    else {
      prec=*p_mazzo;
      aux=*p_mazzo->next;
      done = false;
      while(aux!=NULL && !done) {
        if(aux->carta.seme==seme) {
          prec->next = nuova;
          nuova->next = aux;
          nuova->carteSuccessive = aux->carteSuccessive + 1;
          done = true;
        }
        prec = aux;
        aux = aux->next;
      }
      if(!done) {
        prec->next = nuova;
        nuova->carteSuccessive = 0;
      }
    }
  }
}
```

(iv) (6 punti) Scrivere una funzione *ricorsiva* che data una lista di carte, un valore e il seme di una carta, cancelli la prima occorrenza di tale carta, se esiste, nella lista. La funzione deve restituire *true* se la cancellazione e' avvenuta.

```
boolean cancellaPrima(Mazzo* p_mazzo, int val, Seme seme) {
  Mazzo mazzo;
  if(p_mazzo==NULL) return false; /* sanity check */

  if(*p_mazzo\not=NULL)

  { if(mazzo->carta.seme==seme && mazzo->carta.val==val) {
     mazzo=*p_mazzo;
    *p_mazzo = *p_mazzo->next;
    free(mazzo);
    return true;
  }

  else
     {if(cancellaPrima(&(*p_mazzo->next), val, seme)) {
           if(*p_mazzo->carta.seme==seme) /* aggiorniamo il campo carteSuccessive */
               *p_mazzo->carteSuccessive--;
           return true;}
      else return false;
  } }
  else return false;
}
```