Esercitazione 7

Esercizio 1:

Scrivere una funzione ricorsiva che calcoli il MCD di due numeri interi positivi. Ricordare che la funzione MCD(N,M) termina quando N == M e si ritorna N.

Soluzione:

```
int MCD(int N, int M){
   int mcd;
   if(N == M) {
     return (mcd = N);      // terminazione
   } else if (N > M) {
     mcd = MCD(N-M, M); // sottoproblema1
   } else if (M > N) {
     mcd = MCD(N, M-N); // sottoproblema2
   }
   return mcd;
}
```

Esercizio 5:

Scrivere una procedura ricorsiva che riceva un array a lo stampi in senso inverso.

```
void stampaInverso1 (int vet[], int dim);
void stampaInverso2 (int vet[], int i, int dim);
```

Soluzione:

```
void stampaInverso1 (int vet[], int dim) {
  if(dim==0) {
    printf("\n");
    return;
  }
  printf("%d ", vet[dim-1]);
  stampaInverso1(vet, dim-1);
}

void stampaInverso2 (int vet[], int i, int dim) {
  if(i==dim) {
    printf("\n");
    return;
  }
  printf("%d ", vet[dim-(i+1)]);
  stampaInverso2(vet, i+1, dim);
}

/*
Un semplice main che crea un array e lo passa alle funzioni appena definite,
stampando i risultati.
*/
main () {
  int vett[] = {1,2,3,4,5,6,7,8,9,10};
```

```
  stampaInverso1(vett,10);
  stampaInverso2(vett,0,10);
}
```

Esercizio 11:

Scrivere una funzione ricorsiva che azzeri tutti gli elementi pari, raddoppi gli elementi dispari e ritorni il minimo elemento di un array passato.

Soluzione:

```
#include <stdio.h>

int azzeraRaddoppiaTrovaMinimo(int vet[], int dim) {
  int res;

  if(dim==1) {
    res = vet[0];
  }
  else {
    res = azzeraRaddoppiaTrovaMinimo(vet+1, dim-1);
    if(vet[0]<res) res=vet[0];
  }

  if(vet[0]%2==0)
    vet[0] = 0;
  else
    vet[0] *= 2;

  return res;
}


main() {
  int vett[] = {2,12,3,4,5,6,1,7,8,9};
  int res, i;

  printf("L'array e': [ ");
  for(i=0; i<10; i++)
    printf("%d ", vett[i]);
  printf("]\n");


  res = azzeraRaddoppiaTrovaMinimo(vett, 10);
  printf("Il minimo e': %d\n", res);

  printf("L'array e': [ ");
  for(i=0; i<10; i++)
    printf("%d ", vett[i]);
  printf("]\n");
}
```

Esercizio 12:

Scrivere una funzione ricorsiva per la ricerca di un intero all'interno di un vettore di interi passato per parametro. La funzione termina se nel vettore è presente l'elemento ricercato.

Soluzione:

```
int ricerca_ric_dx(int vett[], int len, int idx, int val) {
    if(idx==len) return -1;    // condizione di terminazione
    else {
```

```
        if(vett[idx] == val) return idx;
        else return ricerca_ric_dx(vett,len,idx+1,val);
    }
}
```

Esercizio 15:

Scrivere una procedura ricorsiva che riceva un array di caratteri e stampi
la prima metà su una riga e la seconda metà su una riga diversa.
Se l'array ha un numero dispari di elementi, la lettera centrale dovrà
essere stampata su una riga a parte, tra la prima metà e la seconda metà.

Soluzione:

```
void stampaSuDueRighe(char vet[], int dim) {

  if(dim==0) {
    printf("\n");
    return;
  }

  if(dim==1) {
    printf("\n%c\n", vet[0]);
    return;
  }

  printf("%c ", vet[0]);
  stampaSuDueRighe(vet+1, dim-2);
  printf("%c ", vet[dim-1]);

}

/*
Un semplice main che crea due array e li passa alla funzione appena definita,
stampando i risultati.
Notate le dimensioni diverse dei due array per verificare entrambi i casi,
numero di elementi pari e numero di elementi dispari.
*/
main() {

  char arr1[] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'l', 'm' };
  char arr2[] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'l' };

  stampaSuDueRighe(arr1, 11);
  printf("\n---\n");
  stampaSuDueRighe(arr2, 10);
  printf("\n---\n");

}
```

Esercizio 16:

Scrivere una funzione che chieda un valore N all'utente e quindi
calcoli ricorsivamente l'N-esimo termine della successione di Fibonacci.
F(n) = F(n-1) + F(n-2)

Riuscite a scrivere la funzione fibonacci ottimizzando (riducendo al minimo) il
numero di chiamate ricorsive?

Suggerimento: Per verificare quante chiamate vengono eseguite, usate una
variabile globale che viene incrementata ad ogni chiamata.

Suggerimento 2: potete sfruttare il fatto che per calcolate F(n-1) dovete sapere

F(n-2)?

Suggerimento 3: una funzione in C non puo' ritornare piu' di un valore ma se riceve dei puntatori puo' modificare i valori delle variabili originali.

Soluzione:

```
int classic_fibonacci(int n)
{
   if (n==1) return 0;
   else if (n==2) {
      return 1;
   }
   else {
      return fibonacci(n-1) + fibonacci(n-2);
   }
}


int optimized_fibonacci_forward_worker(int n, int a, int b) {
  int t;
  counter++;

  if(n==0)
    return b;

  if(n==1)
    return a;

  t = a+b; /* a e b sono rispettivamente l'ultimo e il penultimo numero di
fibonacci quindi t diventa il successivo   */

  /* e quindi la chiamata avviene "dimenticandosi" di b, scalando a come
penultimo e impostando t come ultimo numero di fibonacci calcolato */
  return optimized_fiboacci_forward_worker(n-1, t, a);

}

/*
Interfaccia per rendere il prototipo della funzione ottimizzata identica
a quella della funzione classica
 */
int optimized_fibonacci_forward(int n) {
  return optimized_fibonacci_forward_worker(n, 1, 0);
}


/*
Un semplice main che crea un array e lo passa alla funzione appena definita,
stampando i risultati.
*/
int main(void) {
  int n;

  printf("Inserisci un numero: ");
  scanf("%d", &n);

  counter = 0;
```

```
  printf("Fibonacci classico.\n");
  printf("Il numero di Fibonacci corrispondente è %d\n", classic_fibonacci(n));
  printf("Numero di chiamate ricorsive = %d\n\n", counter);

  counter = 0;
  printf("Fibonacci ottimizzato (all'indietro).\n");
  printf("Il numero di Fibonacci corrispondente è %d\n",
optimized_fibonacci_backward(n));
  printf("Numero di chiamate ricorsive = %d\n\n", counter);

  counter = 0;
  printf("Fibonacci ottimizzato (in avanti).\n");
  printf("Il numero di Fibonacci corrispondente è %d\n",
optimized_fibonacci_forward(n));
  printf("Numero di chiamate ricorsive = %d\n\n", counter);

  return 0;
}
```

Esercizio 17:

Scrivere una funzione che ricevuto un array di interi, calcoli e stampi ricorsivamente tutte le permutazioni dell'array.

Soluzione:

```
int counter=0;


void permuta (int arr[], int first, int dim) {
  int temp, k, i;

  counter++;

  if (first == dim-1) {
    /* il sotto-array ha un elemento solo, non ci sono altre permutazioni
possibili,
    stampiamo tutto l'array. */
    printf("[ ");
    for (i=0; i < dim; i++)
      printf ("%d ", arr[i]);
    printf ("]\n");
  }
  else {

    /* ricorro sul sotto-array immutato a partire dall'elemento successivo a
first */
    permuta (arr, first + 1, dim);

    /* poi scambio l'elemento first con ciascuno dei successivi
    e ogni volta ricorro sul sotto-array a partire da quello dopo */
    for (k = first+1; k < dim; k++) {
      temp = arr[k];
      arr[k] = arr[first];
      arr[first] = temp;

      permuta (arr, first + 1, dim);

      temp = arr[k];
      arr[k] = arr[first];
      arr[first] = temp;
```

```
    }
  }

}

/*
Un semplice main che crea un array e lo passa alla funzione appena definita,
stampando i risultati.
*/
int main (void) {
  int arr[4] = {1,2,3,4};

  permuta (arr, 0, 4);
  printf("Numero di chiamate ricorsive effettuate: %d\n", counter);

  return 0;
}
```