Tutorato Lezione 7

Miscellanea

Docente di tutorato: Alessia Antelmi
aantelmi@unisa.it

Parte I - Progettazione

ADT Casa Editrice

Esercizio 1

Implementare l'ADT Casa Editrice e definire le tabelle di specifica Sintattica e Semantica.

Esercizio 1

CasaEditrice
- Nome → stringa
- Libri → lista di *Libro*

Libro
- Titolo → stringa
- Anno → intero
- Prezzo → double
- Autore → *Autore*

Autore
- Nome → stringa
- Cognome → stringa

ADT CasaEditrice - Possibile specifica

L'ADT «CasaEditrice» fornirà operatori per:

1. Creare una casa editrice (indicata da un nome e una lista di libri);
   - Un libro è caratterizzato dal titolo, dall'anno di pubblicazione, dal prezzo e da un autore
2. Aggiungere un libro alla casa editrice;
3. Rimuovere un libro dalla casa editrice dato il titolo;
4. Ordinare i libri della casa editrice in base al titolo.

Per sperimentare le funzionalità dell'ADT è necessario sviluppare un programma che istanzia e popola una nuova casa editrice:

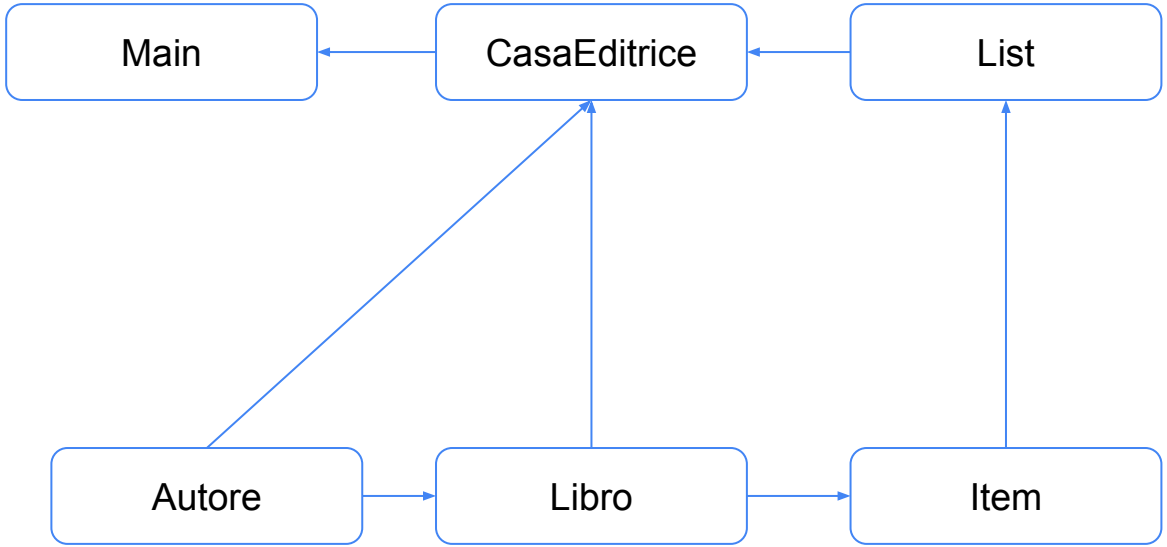
1. Ad esempio facendo inserire all'utente il nome della casa editrice e 4 libri.
2. Successivamente facendo rimuovere un libro e stampando i libri ordinati.

ADT CasaEditrice - Progettazione

1. Tipi Preesistenti:
   a. Item (realizzare item-libro)
   b. List

2. Nuovi tipi di dati:
   a. Autore
   b. Libro
   c. CasaEditrice

ADT CasaEditrice - Progettazione dei Moduli



ADT CasaEditrice - Interfaccia

```c
#include "libro.h"

typedef struct casaEditrice *CasaEditrice;

CasaEditrice createCasaEditrice(char*);
void addLibro(CasaEditrice, Libro);
void removeLibro(CasaEditrice, char*);
void sortLibri(CasaEditrice);
void printCasaEditrice(CasaEditrice);
```

CasaEditrice

```c
#include "autore.h"

typedef struct libro *Libro;

Libro createLibro(char*, int, double, Autore);
char* titolo(Libro);
int anno(Libro);
double prezzo(Libro);
Autore autore(Libro);
```

Libro

```c
typedef struct autore *Autore;

Autore createAutore(char*, char*);
char* nome(Autore);
char* cognome(Autore);
```

Autore

ADT CasaEditrice - Specifica Sintattica e Semantica

Sintattica	Semantica
Nome del tipo: CasaEditrice Tipi usati: Autore, Libro, String	Dominio: insieme di coppie <nome, libri> *nome* è una stringa, *libri* è una lista di Libro
createCasaEditrice(String) → CasaEditrice	createCasaEditrice(nome) → ce • Post: ce = <nome, nil>
addLibro(CasaEditrice, Libro) → CasaEditrice	addLibro(ce, l) → ce' • Post: ce.libri = <a1, a2, … an> AND ce'.libri = <l, a1, …, an>
removeLibro(CasaEditrice, String) → CasaEditrice	removeLibro(ce, l.titolo) → ce' • Pre: ce.libri = <a1, a2, …, l, …, an> n>0 • Post: ce'.libri = ce.libri - <l>
sortLibri(CasaEditrice) → CasaEditrice	sortLibri(ce) → ce' • Pre: n>0 • Post: ce'.libri = <a1, a2, …, an> titolo(ai) < titolo(ai+1) per ogni 1<= i <=n

Parte II - Alberi

Binary Tree e Binary Search Tree

Esercizi su alberi

1. Si implementi un algoritmo che, dato un albero binario, restituisca, per tutti i nodi con chiave **minore** ad una data chiave **k**, una stringa data dalla concatenazione di tali chiavi (ed un **separatore** specificato dall'utente).
    a. Realizzare prima una versione **ricorsiva** e poi una **iterativa**.
    b. **Nota**: aggiungere una funzione toString(Item) ad Item.h (l'algoritmo deve funzionare per qualsiasi tipo di Item, non solo gli Item stringa).

2. Si implementi un algoritmo ricorsivo simile al punto 1, ma per l'ADT BST, in modo che produca un output ordinato e che funzioni in maniera efficiente.
    a. Approfondimento: realizzare una versione **iterativa**.

3. Si implementi una procedura ricorsiva che inserisca in una lista solo il contenuto dei nodi foglia dell'albero.
    a. Approfondimento: realizzare una versione **iterativa**.

4. Realizzare l'ordinamento di una lista impiegando un albero binario di ricerca.
    a. Nota: utilizzare l'ADT **BST**.

Esercizi su alberi

1. Si implementi un algoritmo che, dato un albero binario, restituisca, per tutti i nodi con chiave **minore** ad una data chiave **k**, una stringa data dalla concatenazione di tali chiavi (ed un **separatore** specificato dall'utente).
   a. Realizzare prima una versione **ricorsiva** e poi una **iterativa**.
   b. **Nota**: aggiungere una funzione toString(Item) ad Item.h (l'algoritmo deve funzionare per qualsiasi tipo di Item, non solo gli Item stringa).

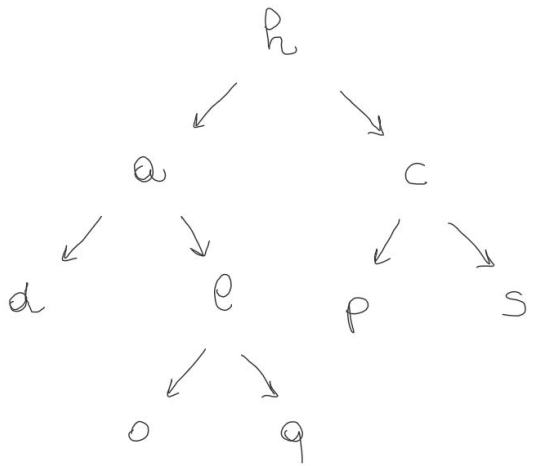
2. Si implementi un algoritmo ricorsivo simile al punto 1, ma per l'ADT BST, in modo che produca un output ordinato e che funzioni in maniera efficiente.
   a. Approfondimento: realizzare una versione **iterativa**.

3. Si implementi una procedura ricorsiva che inserisca in una lista solo il contenuto dei nodi foglia dell'albero.
   a. Approfondimento: realizzare una versione **iterativa**.

4. Realizzare l'ordinamento di una lista impiegando un albero binario di ricerca.
   a. Nota: utilizzare l'ADT **BST**.

Esercizio 1

Input → Albero binario t, Item max = “p”, char* sep = ‘, ’



Output → h, a, d, l, o, c

Esercizio 1

Soluzione ricorsiva

```c
// funzione di servizio
void doRecConcatLessThen(BTree t, Item it, char* separator, char* out) {
    if(!isEmptyTree(t)){
        if(cmpItem(t->value, it) < 0) {
            if(strlen(out) > 0) {
                strcat(out, separator);
            }
            strcat(out, toString(t->value));
        }
        doRecConcatLessThen(t->left, it, separator, out);
        doRecConcatLessThen(t->right, it, separator, out);
    }
}

// versione ricorsiva
char* recConcatLessThen(BTree t, Item it, char* separator) {
    char* str = malloc(sizeof(char) * 100);
    str[0] = '\0';
    doRecConcatLessThen(t, it, separator, str);
    return str;
}
```

Esercizio 1

Soluzione iterativa

```
// versione iterativa
char* itConcatLessThen(BTree t, Item it, char* separator) {
    char* str = malloc(sizeof(char) * 100);
    str[0] = '\0';
    BTree left, right;
    Stack s = newStack();
    push(s, t);
    while (!isEmptyStack(s)){
        BTree node = top(s);
        if(cmpItem(node->value, it) < 0) {
            if(strlen(str) > 0) {
                strcat(str, separator);
            }
            strcat(str, toString(node->value));
        }
        pop(s);
        if ((right = getRight(node))!=NULL)
            push(s, right);
        if ((left = getLeft(node))!=NULL)
            push(s, left);
    }
    return str;
}
```

Esercizi su alberi

1. Si implementi un algoritmo che, dato un albero binario, restituisca, per tutti i nodi con chiave **minore** ad una data chiave **k**, una stringa data dalla concatenazione di tali chiavi (ed un **separatore** specificato dall'utente).
    a. Realizzare prima una versione **ricorsiva** e poi una **iterativa**.
    b. **Nota**: aggiungere una funzione toString(Item) ad Item.h (l'algoritmo deve funzionare per qualsiasi tipo di Item, non solo gli Item stringa).

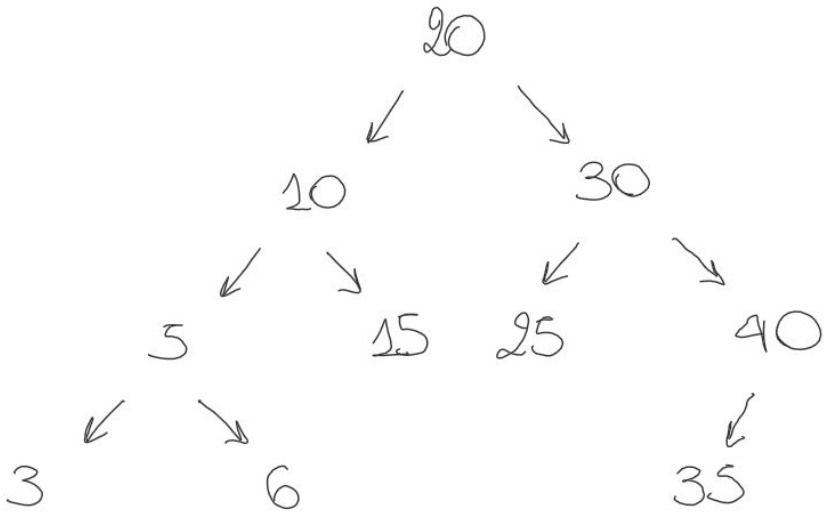
2. Si implementi un algoritmo ricorsivo simile al punto 1, ma per l'ADT BST, in modo che produca un output ordinato e che funzioni in maniera efficiente.
    a. Approfondimento: realizzare una versione **iterativa**.

3. Si implementi una procedura ricorsiva che inserisca in una lista solo il contenuto dei nodi foglia dell'albero.
    a. Approfondimento: realizzare una versione **iterativa**.

4. Realizzare l'ordinamento di una lista impiegando un albero binario di ricerca.
    a. Nota: utilizzare l'ADT **BST**.

Esercizio 2

Input → BST bst, Item max = 22, char* sep = ‘, ’



Output → 3, 5, 6, 10, 15, 20

Esercizio 2

Soluzione ricorsiva

```
// funzione di servizio
void doRecConcatLessThen(BST t, Item it, char* separator, char* out) {
    if(!isEmptyBST(t)){
        doRecConcatLessThen(t->left, it, separator, out);
        if(cmpItem(t->value, it) < 0) {
            if(strlen(out) > 0) {
                strcat(out, separator);
            }
            strcat(out, toString(t->value));
            doRecConcatLessThen(t->right, it, separator, out);
        }
    }
}

// versione ricorsiva
char* recConcatLessThen(BST t, Item it, char* separator) {
    char* str = malloc(sizeof(char) * 100);
    str[0] = '\0';
    doRecConcatLessThen(t, it, separator, str);
    return str;
}
```

Esercizi su alberi

1. Si implementi un algoritmo che, dato un albero binario, restituisca, per tutti i nodi con chiave **minore** ad una data chiave **k**, una stringa data dalla concatenazione di tali chiavi (ed un **separatore** specificato dall'utente).
    a. Realizzare prima una versione **ricorsiva** e poi una **iterativa**.
    b. **Nota**: aggiungere una funzione toString(Item) ad Item.h (l'algoritmo deve funzionare per qualsiasi tipo di Item, non solo gli Item stringa).

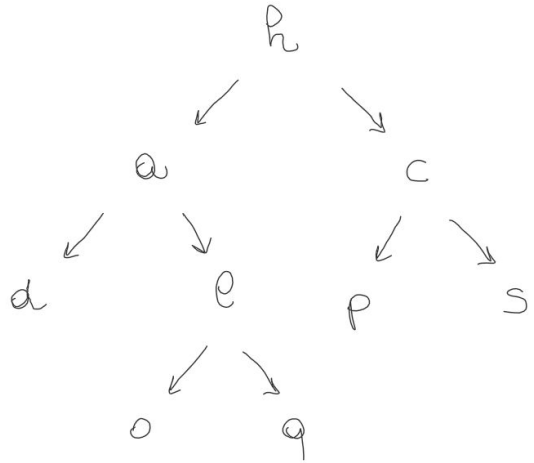
2. Si implementi un algoritmo ricorsivo simile al punto 1, ma per l'ADT BST, in modo che produca un output ordinato e che funzioni in maniera efficiente.
    a. Approfondimento: realizzare una versione **iterativa**.

3. Si implementi una procedura ricorsiva che inserisca in una lista solo il contenuto dei nodi foglia dell'albero.
    a. Approfondimento: realizzare una versione **iterativa**.

4. Realizzare l'ordinamento di una lista impiegando un albero binario di ricerca.
    a. Nota: utilizzare l'ADT **BST**.

Esercizio 3

Input → Albero binario t



Output → s p q o d

Esercizio 3

Soluzione ricorsiva

```
// funzione di servizio
void doLeaves(BTree t, List l) {
    int noLeft, noRight;
    noLeft = isEmptyTree(t->left);
    noRight = isEmptyTree(t->right);
    if(noLeft && noRight) {
        addHead(l, t->value);
    }
    if(!noLeft) {
        doLeaves(t->left, l);
    }
    if(!noRight) {
        doLeaves(t->right, l);
    }
}

List leaves(BTree t) {
    List l = newList();
    if(!isEmptyTree(t)) {
        doLeaves(t, l);
    }
    return l;
}
```

Esercizi su alberi

1. Si implementi un algoritmo che, dato un albero binario, restituisca, per tutti i nodi con chiave **minore** ad una data chiave **k**, una stringa data dalla concatenazione di tali chiavi (ed un **separatore** specificato dall'utente).
    a. Realizzare prima una versione **ricorsiva** e poi una **iterativa**.
    b. **Nota**: aggiungere una funzione toString(Item) ad Item.h (l'algoritmo deve funzionare per qualsiasi tipo di Item, non solo gli Item stringa).

2. Si implementi un algoritmo ricorsivo simile al punto 1, ma per l'ADT BST, in modo che produca un output ordinato e che funzioni in maniera efficiente.
    a. Approfondimento: realizzare una versione **iterativa**.

3. Si implementi una procedura ricorsiva che inserisca in una lista solo il contenuto dei nodi foglia dell'albero.
    a. Approfondimento: realizzare una versione **iterativa**.

4. Realizzare l'ordinamento di una lista impiegando un albero binario di ricerca.
    a. Nota: utilizzare l'ADT **BST**.

Esercizio 4

Input → Array/List

```
int arr[]={82, 61, 62, 1, 38, 2, 40, 83, 37, 5};
```

Output →1 2 5 37 38 40 61 62 82 83

Esercizio 4

Soluzione

```
void treeSort(Item items[], int n)
{
    BST t = NULL;

    for (int i=0; i<n; i++)
        t = insert(t, items[i]);

    int i = 0;
    in_order(t, items, &i);
}
```

Parte III - Ricorsione

Esercizi su ricorsione

1. Scrivere una funzione che calcoli il quoziente della divisione tra interi. Svolgere l'esercizio nelle due versioni **ricorsiva** e **tail ricorsiva**.
    a. Suggerimento: $x/y = (x - y + y)/y = 1 + (x - y)/y$.

2. Scrivere una funzione C che calcola, dati due numeri interi M ed N, la potenza M^N. Si progettino le versioni ricorsiva, ricorsiva tail e iterativa.

3. Scrivere una funzione che, data una **stringa** s, stampi tutte le stringhe ottenute **permutando** i caratteri di s.
    a. Ad esempio, l'invocazione permutazioni("abc") deve effettuare la seguente stampa su standard output
        i. abc/acb/bac/bca/cab/cba

4. La **Torre di Hanoi**. Sono date tre torri (sinistra, centrale, e destra) e un certo numero N di dischi forati.
    a. I dischi hanno diametro diverso gli uni dagli altri, e inizialmente sono infilati uno sull'altro (dal basso in alto) dal più grande al più piccolo sulla torre di sinistra.
    b. Scopo del gioco è portarli tutti sulla torre destra, rispettando due regole:
        i. si può muovere un solo disco alla volta;
        ii. un disco grande non può mai stare sopra un disco più piccolo.

Esercizi su ricorsione

1. Scrivere una funzione che calcoli il quoziente della divisione tra interi. Svolgere l'esercizio nelle due versioni **ricorsiva** e **tail ricorsiva**.
    a. Suggerimento: $x/y = (x - y + y)/y = 1 + (x - y)/y$.

2. Scrivere una funzione C che calcola, dati due numeri interi M ed N, la potenza M^N. Si progettino le versioni ricorsiva, ricorsiva tail e iterativa.

3. Scrivere una funzione che, data una **stringa** s, stampi tutte le stringhe ottenute **permutando** i caratteri di s.
    a. Ad esempio, l'invocazione permutazioni("abc") deve effettuare la seguente stampa su standard output
        i. abc/acb/bac/bca/cab/cba

4. La **Torre di Hanoi**. Sono date tre torri (sinistra, centrale, e destra) e un certo numero N di dischi forati.
    a. I dischi hanno diametro diverso gli uni dagli altri, e inizialmente sono infilati uno sull'altro (dal basso in alto) dal più grande al più piccolo sulla torre di sinistra.
    b. Scopo del gioco è portarli tutti sulla torre destra, rispettando due regole:
        i. si può muovere un solo disco alla volta;
        ii. un disco grande non può mai stare sopra un disco più piccolo.

Esercizio 1

Soluzione

```c
int div(int x, int y){
    if (x>=y) return 1+div(x-y,y);
    else return 0;
}


int main() {
    int x,y,ris;

    do{
        printf("Inserire dividendo: ");
        scanf("%d",&x);
    }while (x<0);

    do{
        printf("Inserire divisore: ");
        scanf("%d",&y);
    }while (y<1);

    ris = div(x,y);
    printf("%d/%d = %d\n",x,y,ris);
}
```

Esercizio 1

Soluzione - tail recursive

```
int div_tail(int x, int y, int v, int k){
     if (k<=x){
          v = v + 1;
          k = k + y;
          return div_tail(x,y,v,k);
     }
     else return v;
}

int div(int x, int y){
     return div_tail(x,y,0,y);
}
```

Esercizi su ricorsione

1. Scrivere una funzione che calcoli il quoziente della divisione tra interi. Svolgere l'esercizio nelle due versioni **ricorsiva** e **tail ricorsiva**.
   a. Suggerimento: x/y = (x - y + y)/y = 1 + (x - y)/y.

2. Scrivere una funzione C che calcola, dati due numeri interi M ed N, la potenza M^N. Si progettino le versioni ricorsiva, ricorsiva tail e iterativa.

3. Scrivere una funzione che, data una **stringa** s, stampi tutte le stringhe ottenute **permutando** i caratteri di s.
   a. Ad esempio, l'invocazione permutazioni("abc") deve effettuare la seguente stampa su standard output
      i. abc/acb/bac/bca/cab/cba

4. La **Torre di Hanoi**. Sono date tre torri (sinistra, centrale, e destra) e un certo numero N di dischi forati.
   a. I dischi hanno diametro diverso gli uni dagli altri, e inizialmente sono infilati uno sull'altro (dal basso in alto) dal più grande al più piccolo sulla torre di sinistra.
   b. Scopo del gioco è portarli tutti sulla torre destra, rispettando due regole:
      i. si può muovere un solo disco alla volta;
      ii. un disco grande non può mai stare sopra un disco più piccolo.

Esercizio 2

Soluzione

```c
int pot(int m, int n){
    if (n>0) return m*pot(m, n-1);
    else return 1;
}

int pot_tail(int m, int n, int v, int k)
{
    if (k<=n){
     v = v * m;
     k++;
     return pot_tail(m,n,v,k);
    }
    else return v;
}


int pot1(int m, int n){
    return pot_tail(m,n,1,1);
}
```

Esercizi su ricorsione

1. Scrivere una funzione che calcoli il quoziente della divisione tra interi. Svolgere l'esercizio nelle due versioni **ricorsiva** e **tail ricorsiva**.
   a. Suggerimento: $x/y = (x - y + y)/y = 1 + (x - y)/y$.

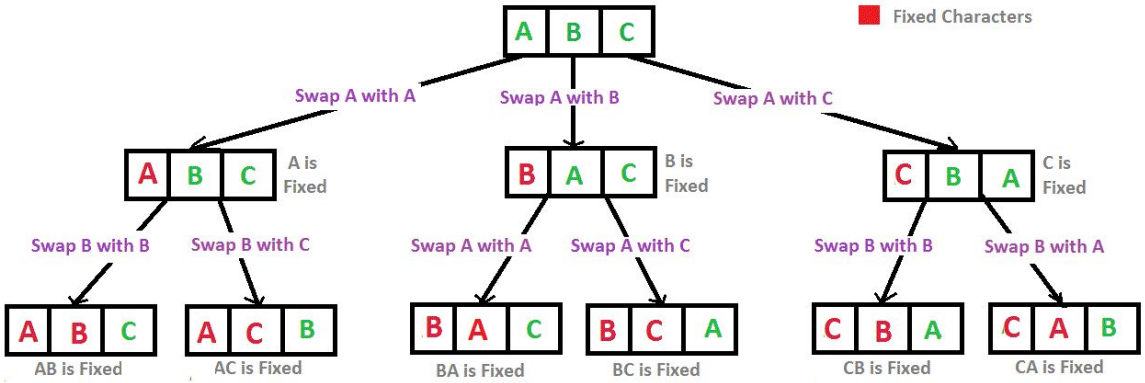
2. Scrivere una funzione C che calcola, dati due numeri interi M ed N, la potenza M^N. Si progettino le versioni ricorsiva, ricorsiva tail e iterativa.

3. Scrivere una funzione che, data una **stringa** s, stampi tutte le stringhe ottenute **permutando** i caratteri di s.
   a. Ad esempio, l'invocazione permutazioni("abc") deve effettuare la seguente stampa su standard output
      i. abc/acb/bac/bca/cab/cba

4. La **Torre di Hanoi**. Sono date tre torri (sinistra, centrale, e destra) e un certo numero N di dischi forati.
   a. I dischi hanno diametro diverso gli uni dagli altri, e inizialmente sono infilati uno sull'altro (dal basso in alto) dal più grande al più piccolo sulla torre di sinistra.
   b. Scopo del gioco è portarli tutti sulla torre destra, rispettando due regole:
      i. si può muovere un solo disco alla volta;
      ii. un disco grande non può mai stare sopra un disco più piccolo.

Soluzione - Idea

Esercizio 3



Recursion Tree for Permutations of String "ABC"

Esercizio 3

Soluzione

```c
void permute(char *a, int l, int r) {
    int i;
    if (l == r)
        printf("%s\n", a);
    else
    {
        for (i = l; i <= r; i++)
        {
            swap((a+l), (a+i));
            permute(a, l+1, r);
            swap((a+l), (a+i)); //backtrack
        }
    }
}
```

Esercizi su ricorsione

1. Scrivere una funzione che calcoli il quoziente della divisione tra interi. Svolgere l'esercizio nelle due versioni **ricorsiva** e **tail ricorsiva**.
   a. Suggerimento: $x/y = (x - y + y)/y = 1 + (x - y)/y$.

2. Scrivere una funzione C che calcola, dati due numeri interi M ed N, la potenza M^N. Si progettino le versioni ricorsiva, ricorsiva tail e iterativa.

3. Scrivere una funzione che, data una **stringa** s, stampi tutte le stringhe ottenute **permutando** i caratteri di s.
   a. Ad esempio, l'invocazione permutazioni("abc") deve effettuare la seguente stampa su standard output
      i. abc/acb/bac/bca/cab/cba

4. La **Torre di Hanoi**. Sono date tre torri (sinistra, centrale, e destra) e un certo numero N di dischi forati.
   a. I dischi hanno diametro diverso gli uni dagli altri, e inizialmente sono infilati uno sull'altro (dal basso in alto) dal più grande al più piccolo sulla torre di sinistra.
   b. Scopo del gioco è portarli tutti sulla torre destra, rispettando due regole:
      i. si può muovere un solo disco alla volta;
      ii. un disco grande non può mai stare sopra un disco più piccolo.

Soluzione

Esercizio 4

```c
/* Questa funzione (ricorsiva) descrive le mosse da fare per poter
    trasferire n dischi dal piolo ORIGINE al piolo DESTINAZIONE,
    usando eventualmente APPOGGIO come piolo per gli spostamenti intermedi.
    Ovviamente, questi spostamenti vengono fatti tenendo conto del
    vincolo espresso nel problema delle "torri di Hanoi": ciascun
    disco non puo' mai essere poggiato su un disco piu' piccolo. */
void hanoi(int n, int origine, int destinazione, int appoggio){

    if (n==1)
        printf("Muovo da %d a %d\n", origine,destinazione);
    else{
        hanoi(n-1, origine, destinazione, appoggio);
        printf("Muovo da %d a %d\n", origine, appoggio);
        hanoi(n-1, appoggio, destinazione, origine);
    }
}
```

Esercizi su ricorsione

1. Scrivere una funzione che calcoli il quoziente della divisione tra interi. Svolgere l'esercizio nelle due versioni **ricorsiva** e **tail ricorsiva**.
    a. Suggerimento: $x/y = (x - y + y)/y = 1 + (x - y)/y$.

2. Scrivere una funzione C che calcola, dati due numeri interi M ed N, la potenza M^N. Si progettino le versioni ricorsiva, ricorsiva tail e iterativa.

3. Scrivere una funzione che, data una **stringa** s, stampi tutte le stringhe ottenute **permutando** i caratteri di s.
    a. Ad esempio, l'invocazione permutazioni("abc") deve effettuare la seguente stampa su standard output
        i. abc/acb/bac/bca/cab/cba

4. Si progetti la funzione ricorsiva che svolge il compito seguente. Siano dati due vettori V_1 e V_2, di dimensione N_1 e N_2, rispettivamente (con $1 \leq N_2 \leq N1$). La funzione restituisce il valore 1 se tutti gli elementi del vettore V_2 si trovano nel vettore V_1 nell'ordine inverso rispetto a quello in cui essi figurano in V_2, ma non necessariamente in posizioni immediatamente consecutive; altrimenti, la funzione restituisce valore 0.
    a. Ad esempio, $V_1 = [1\ 2\ 3]$ e $V_2 = [3\ 1]$ restituisce true.