

1. Considerate la seguente funzione, che deve restituire il prodotto dei suoi due argomenti

```
int multiply(int x, int y) {  
    if ( AAAAA ) {  
        BBBBB;  
    }  
    else {  
        return multiply(x - 1, y) + y;  
    }  
}
```

x == 0
return 0

Cosa si deve scrivere al posto di AAAAA e di BBBBB?

- ☐ AAAAA x == y
BBBBB return x*y
- ☐ AAAAA x == 1
BBBBB return 1
- ☐ AAAAA x == 0
BBBBB return 1
- ☐ AAAAA x == 0
BBBBB return 0
- ☐ AAAAA x == 1
BBBBB return 0
- ☐ AAAAA x == 1
BBBBB return y

2. Considerate il seguente programma:

```
#include <stdio.h>
#define print(x) printf("%d ", x)
int x;
void Q(int z) {
    z += x;
    print(z);
}

void P(int *y) {
    int x = *y + 2;
    Q(x);
    *y = x - 1;
    print(x);
}

int main(void) {
    x = 5;
    P(&x);
    print(x);
    return 0;
}
```

Quale è l'output del programma?

Risposta:

12, 7, 6

2p

3. Abbiamo i puntatori al primo e all'ultimo elemento di una lista concatenata semplice (con solo puntatore a next). Quali delle seguenti operazioni hanno tempo di esecuzione che dipende dalla lunghezza della lista?

- | | | |
|--------------------------|------------------------------------|---|
| <input type="checkbox"/> | Inserimento alla fine della lista | Si, dato che devo trovare il penultimo elemento e per farlo devo scorrere tutta la lista |
| <input type="checkbox"/> | Cancellazione del primo elemento | NO, costo lineare, basta mettere il puntatore al primo elemento a first -> next |
| <input type="checkbox"/> | Inserimento all'inizio della lista | NO |
| <input type="checkbox"/> | Ricerca di un elemento | Si, dato che devo scorrere dall'inizio alla fine |
| <input type="checkbox"/> | Cancellazione dell'ultimo elemento | Si, come il primo punto, devo trovare il penultimo elemento e identificarlo come nuova tail |

4. Considerate queste porzione di codice.

```
struct item {  
    int data;  
    struct item *next;  
};  
  
int f(struct item *p) {  
    return (  
        (p == NULL) ||  
        (p->next == NULL) ||  
        (( p->data <= p->next->data) && f(p->next))  
    );  
}
```

Sia p l'indirizzo del primo elemento di una lista, la funzione f(p) restituisce 1 se e solo se

- ☐ La lista, a partire dal secondo elemento, è ordinata in maniera non decrescente
- ☐ La lista è ordinata in maniera decrescente
- ☐ La lista è ordinata in maniera crescente
- ☐ Nessuna delle altre risposte
- ☐ La lista, a partire dal secondo elemento, è ordinata in maniera crescente
- ☐ La lista è ordinata in maniera non crescente
- ☐ La lista è ordinata in maniera non decrescente

5. a) In una lista doppiamente concatenata, il numero di puntatori sui quali si interviene per un'operazione di inserimento è:

- ☐ 1
- ☐ nessuna delle altre risposte
- ☐ 2
- ☐ 4
- ☐ 0

1p

b) Giustificate brevemente la risposta.

Risposta:

Dipende se l'inserimento avviene in coda/testa oppure in un altro nodo (che non sia coda o testa).
L'inserimento in testa/coda è il seguente

```
ListNode insert(ListNode l, int v){  
    Node new = new_node (v);  
    new -> next = l -> head;  
    l -> head -> prev = new;  
    new -> prev = NULL;  
    l -> head = new;  
    return l -> head;
```

Nel caso di un inserimento al "centro" della lista

```
Listnode insert(Listnode l, inv v)  
{  
    ...trovo la posizione desiderata nella lista  
    node new = new_node(v);  
    new -> next = curr;  
    new -> prev = prev;  
    prev -> next = new;  
    curr -> prev = new;  
    return l-> head;
```

2p

6. a) Questa funzione dovrebbe incrementare di 1 il valore di ogni nodo di un albero binario, ma non è corretta.

BitNode è un tipo che rappresenta un nodo di albero binario: si tratta di un puntatore a una struttura con 2 membri left e right che puntano rispettivamente ai figli sinistro e destro, e un membro val di tipo intero.

```
void f(BitNode root) {
    if (root != NULL) {
        root -> val++;
        if (root -> left != NULL) {
            root -> left -> val++;
            f(root -> left -> left);
        }
        if (root -> right != NULL) {
            root -> right -> val++;
            f(root -> right -> right);
        }
    }
}
```

Spiegate cosa fa invece la funzione, poi individuate la causa di questo errore e descrivetela.

Risposta:




La funzione incrementa solamente i nodi "esterni", infatti viene richiamata solo sui nodi sx e dx più esterni. Per evitare questo problema si può semplicemente realizzare questa funzione

```
void incrementa(Bit_node b){
    if (b != NULL){
        b -> value ++;
        if (b -> l != NULL)
            incrementa(b -> l);
        if (b -> r != NULL)
            incrementa(b -> r);
    } return;
}
```

b) Correggete la funzione.

Risposta:



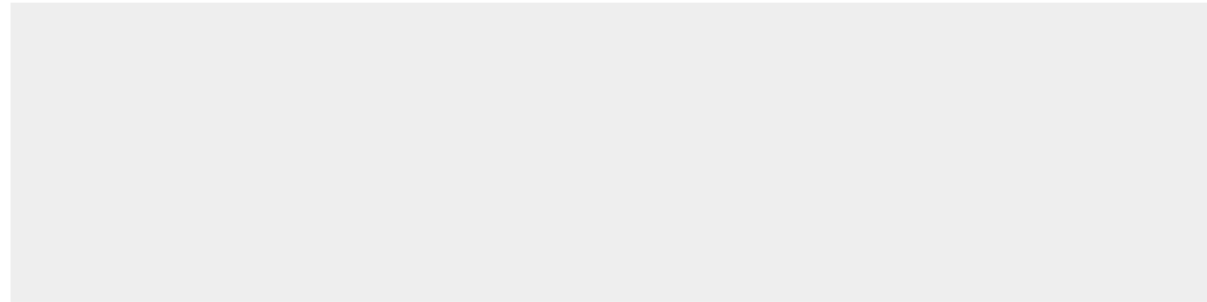
I

2p

7. Considerate la seguente funzione, che riceve un vettore A di n interi e un vettore B di m interi.

```
1 int f (int A[], int B[], int n, int m) {  
2     for ( int i= 0; i < n; i++) {  
3         int found = 0;  
4         for (int j = 0; j < m; j++)  
5             if (A[i] == B[j])  
6                 found = 1;  
7         if (!found) return 0;  
8     }  
9     return 1;  
10 }
```

- a) Sia $A = \{2, 4, 5\}$. Selezionate tutti e soli i B per cui la funzione restituisce 1.



2p

- b) Cosa fa la funzione f?

- ☐ Decide se A e B hanno i primi due elementi in comune
- ☐ Decide se A e B sono uguali
- ☐ Decide se ogni valore di A si trova anche in B
- ☐ Decide se A e B hanno almeno un elemento in comune
- ☐ Decide se ogni valore di B si trova anche in A
- ☐ Decide se A e B hanno lo stesso numero di elementi
- ☐ Decide se A e B contengono gli stessi valori

2p

c) Sia N la lunghezza di A e M la lunghezza di B . Quale è il tempo di esecuzione nel caso pessimo, in funzione di N e M ?


- ☐ $O(N^2)$
- ☐ $O(NM)$
- ☐ $O(M^2)$
- ☐ $O(N+M)$
- ☐ $O(N \log N)$

1p

d) Riprogettate la funzione f in modo che sia asintoticamente più veloce. Potete usare strutture dati di supporto.

Potete spiegare la vostra soluzione a parole (siate brevi), con pseudocodice, con un disegno, oppure potete scriverla in C. Indicate il tempo di esecuzione; più la funzione è veloce, meglio è.

Risposta:



-uso una hash map:Inserisco tutti gli elementi di A in una hashmap (costo medio: $O(1)$), poi ricerco ogni elemento di B nella hasmap (costo medio: $O(1)$)complessità: $O(n+m)$

-ordino i due array (costo: $O(n \log n + m \log m)$), dopodiché per controllare se un elemento di A è presente anche in B basta una ricerca binaria (quindi costo $\log M$), in totale gli elementi di A sono n quindi il costo di questa parte è $O(n \log m)$.

5p