Programmazione II - dal 2024

Navigazione quiz

1 ✓ 2 ✓ 3 ✓ 4 ✓ 5 ✓ 6 ✓

Visualizza una pagina alla volta

Fine revisione

Iniziato	giovedì, 12 settembre 2024, 09:04
Stato	Completato
Terminato	giovedì, 12 settembre 2024, 10:53
Tempo impiegato	1 ora 49 min.
Valutazione	**30,00** su un massimo di 30,00 (**100**%)

Domanda **1**
Risposta corretta
Punteggio ottenuto 4,00 su 4,00
Contrassegna domanda

Realizzare un'implementazione RICORSIVA della seguente specifica di funzione:

/** @brief Verifica se una porzione di stringa è palindroma. Palindromo significa che o la stringe e' vuota oppure
* la meta' sinistra e' lo specchio della meta' destra.
* Esempi: kayak, anna sono palindrome. Invece Kayak non è palindroma perche' le 'k' sono una maiuscola e l'altra minuscola.

* P-IN(s,first,last): s è una stringa, first e last sono indici validi di caratteri in s
* P-OUT(s,first,last,result): result è il valore di verità di “la sequenza di caratteri contenuti in s[first...last] è palindroma”
*/
_Bool isPalindrome(const char *s, int first, int last);

Answer: (penalty regime: 0 %)

Reset answer

```c
/** @brief Verifica se una porzione di stringa è palindroma.
*    Palindromo significa che o la stringe e' vuota oppure la
*    Esempi: kayak, anna sono palindrome. Invece Kayak non è p
*
*    P-IN(s,first,last): s è una stringa, first e last sono in
*    P-OUT(s,first,last,result): result è il valore di verità
*/
_Bool isPalindrome(const char *s, int first, int last){
    //caso di puntatore non valido
    if(s == NULL) return false;
    //caso base della ricorsione con ritorno 1
    if(first>last){
        return 1;
    }
    //se le lettere non sono uguali, la stringa non palindroma
    //ritorno 0
    if(s[first] != s[last]){
        return 0;
    }
    //proseguo con la ricorsione aggiornando gli indici
    return isPalindrome(s, first+1, last-1);
}
```

	Test	Expected	Got
✓	`/** TEST 1: stringa vuota **/`	test passed!	test passed!
✓	`/** TEST 2: singolo carattere **/`	test passed!	test passed!
✓	`/** TEST 3: stringa palindroma corta **/`	test passed!	test passed!
✓	`/** TEST 4: stringa non palindroma corta **/`	test passed!	test passed!
✓	`/** TEST 5: stringa palindroma lunga **/`	test passed!	test passed!
✓	`/** TEST 6: stringa non palindroma lunga **/`	test passed!	test passed!

Passed all tests! ✓

Risposta corretta
Punteggio di questo invio: 4,00/4,00.

Domanda **2**
Risposta corretta
Punteggio ottenuto 7,00 su 7,00
Contrassegna domanda

Date le dichiarazioni:

typedef struct node IntNode, *IntList;

struct node {
int data;
IntList next;
}

e la specifica di funzione:

/** @brief Date due liste ordinate *lsPtr1 e *lsPtr2, restituisce una lista ordinata contenente tutti i nodi di *lsPtr1 e *lsPtr2.
* I nodi vanno posizionati nella lista risultato a seconda del loro valore, in modo tale da renderla ordinata.
* Non bisogna usare la malloc bensì bisogna togliere i nodi dalle liste di input *lsPtr1 e *lsPtr2, che alla fine conterranno

* entrambi NULL (in altri termini non si alloca nuova memoria).

* Ad es. date [1, 5, 9] e [0, 2, 4, 6, 8] restituisce [0, 1, 2, 4, 5, 6, 8, 9].

*/
IntList merge(IntList *lsPtr1, IntList *lsPtr2);
realizzarne un'implementazione ITERATIVA.

Answer: (penalty regime: 0 %)

Reset answer

```c
/** @brief Restituisce la lista alternata dei nodi di *lsPtr1
*   che alla fine conterranno entrambi NULL (non alloca nuova
*   Ad es. date [1, 5, 9] e [0, 2, 4, 6, 8] restituisce [1, 0,
*/
IntList merge(IntList *lsPtr1, IntList *lsPtr2){
    //caso base in cui non posso generare una lista
    if(*lsPtr1 == NULL && *lsPtr2){
        return NULL;
    }

    IntList curr1 = *lsPtr1; //copia lista 1
    IntList curr2 = *lsPtr2; //copia lista 2
    IntList mergeList = NULL; // lista risultato che viene res
    IntList *risPtr = &mergeList; //puntatore alla lista per m

    //scorro finche entrambe le liste hanno valori
    while(curr1 && curr2){
        //se curr1 < curr2 lo aggiungo per primo alla lista
        if(curr1->data < curr2->data){
            *risPtr = curr1;      //collego il nodo alla lista
            curr1 = curr1->next; //passo al nodo successivo de
        //se curr1 > curr2 aggiungo prima curr2
        }else{
           *risPtr = curr2;  //collego il nodo alla lista risu
            curr2 = curr2->next; //passo al nodo successivo de
        }
        //aggiorno il puntatore della lista risultato
        risPtr = &((*risPtr)->next);
    }
    // collego la parte restante della lista 1 se ha ancora el
    if(curr1){
        *risPtr = curr1;
    }
    //altrimenti collego la parte restante della lista 2
    if(curr2){
        *risPtr = curr2;
    }
    //setto le due liste di partenza a NULL come da consegna
    *lsPtr1 = NULL;
    *lsPtr2 = NULL;
    return mergeList;
}
```

	Test	Expected	Got
✓	`/** TEST 1: liste NULL e NULL **/`	test passed!	test pas
✓	`/** TEST 2: liste [3,4] e NULL **/`	test passed!	test pas
✓	`/** TEST 3: liste [3,4] e [2,5] **/`	test passed!	test pas
✓	`/** TEST 4: liste [1, 5, 9] e [0, 2, 4, 6, 8] **/`	test passed!	test pas
✓	`/** TEST 5: liste [0, 2, 4, 6, 8] e [1, 5, 9] **/`	test passed!	test pas

Passed all tests! ✓

Questo è il feeback generale

Risposta corretta
Punteggio di questo invio: 7,00/7,00.

Domanda **3**
Risposta corretta
Punteggio ottenuto 7,00 su 7,00
Contrassegna domanda

Dato un albero albero binario (non per forza di ricerca binaria) definito da:

typedef struct treeNode IntTreeNode, *IntTree;

struct treeNode {
IntTree left;
int data;
IntTree right;
};

e la specifica di funzione:

/**@brief Trasforma un albero, agendo su ogni nodo con entrambi i rami,
* scambiandoli se le loro radici non sono nell'ordine corretto (ovvero lo
* scambio avviene quando sinistra > destra).
*/
void sort(IntTree tree);

realizzarne una implementazione RICORSIVA.

Ad esempio la versione ordinata di:

```
   2
  / \
 5   1
/ \
9   4
   / \
  3   11
   \
    7
```

è:

```
  2
 / \
1   5
   / \
  4   9
 / \
3   11
 \
  7
```

Answer: (penalty regime: 0 %)

Reset answer

```c
/** @brief Trasforma un albero nella sua versione speculare. *
void sort(IntTree tree){
    //caso base ricorsivo con uscita dalla ricorsione
    if(tree == NULL){
        return;
    }
    //come da consegna, agisco su un nodo solo se ha entrambi
    if(tree->left && tree->right){
        //agisco sul nodo se left > right
        if(tree->left->data > tree->right->data){
            //inverto i figli
            IntTree tempNode = tree->left;
            tree->left = tree->right;
            tree->right = tempNode;
        }
    }
    //chiamo la ricorsione per i figli
    sort(tree->left);
    sort(tree->right);
}
```

	Test	Expected	Got	
✓	`/** TEST 1 **/`	test passed!	test passed!	✓
✓	`/** TEST 2 **/`	test passed!	test passed!	✓
✓	`/** TEST 3 **/`	test passed!	test passed!	✓
✓	`/** TEST 4 **/`	test passed!	test passed!	✓
✓	`/** TEST 5 **/`	test passed!	test passed!	✓
✓	`/** TEST 6 **/`	test passed!	test passed!	✓

Passed all tests! ✓

Questo è il feeback generale

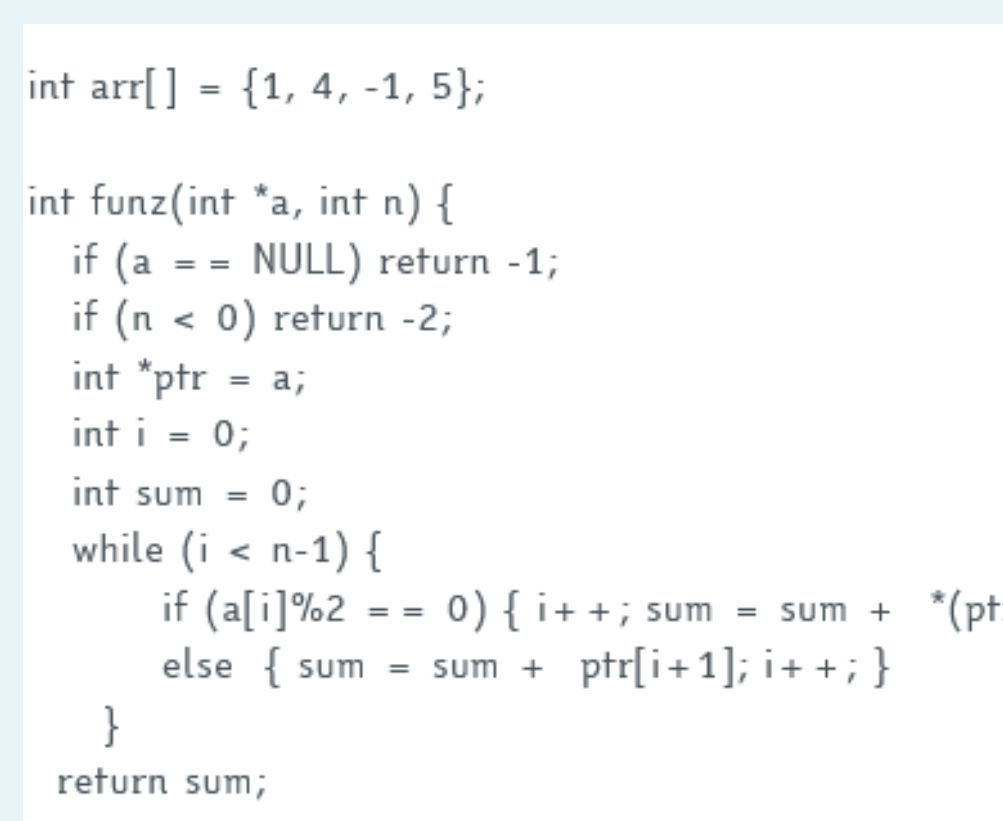
Question author's solution (C):

```c
void sort(IntTree tree){
    if(tree == NULL) return;
    sort(tree->left);
    sort(tree->right);
    if (tree->left != NULL && tree->right != NULL && tree->lef
        IntTree tmp = tree->left;
        tree->left = tree->right;
        tree->right = tmp;
    }
}
```

Risposta corretta
Punteggio di questo invio: 7,00/7,00.

Domanda **4**
Risposta corretta
Punteggio ottenuto 3,00 su 3,00
Contrassegna domanda

Si consideri il codice riportato nel seguito e si risponda alle domande:



```c
int arr[] = {1, 4, -1, 5};

int funz(int *a, int n) {
  if (a == NULL) return -1;
  if (n < 0) return -2;
  int *ptr = a;
  int i = 0;
  int sum = 0;
  while (i < n-1) {
      if (a[i]%2 == 0) { i++; sum = sum +  *(ptr+i); }
      else  { sum = sum +  ptr[i+1]; i++; }
    }
 return sum;
}
```

la funzione può produrre un segmentation fault VERO ✓

se non produce un segmentation fault, la funzione terminerà sempre VERO ✓

La chiamata di funzione funz(arr,4) restituisce: 8 ✓

Risposta corretta.

La risposta corretta è: la funzione può produrre un segmentation fault → VERO,
se non produce un segmentation fault, la funzione terminerà sempre → VERO,

La chiamata di funzione funz(arr,4) restituisce: → 8

Domanda **5**
Risposta corretta
Punteggio ottenuto 3,00 su 3,00
Contrassegna domanda

Si considerino le seguenti dichiarazioni:

typedef struct _scheda Scheda, *Puntatore;
struct _scheda {
int numero;
char *testo;
}
int funzione(Scheda sc, Puntatore pu);
Scheda x;
Scheda *y;
Puntatore *z;

Quali delle seguenti invocazioni sono staticamente (ovvero per il compilatore) corrette e quali sono invece errate?

funzione(*y, &(**z)); CORRETTA ✓
funzione(y, &y); ERRATA ✓
funzione((**z), &x); CORRETTA ✓

Risposta corretta.

La risposta corretta è: funzione(*y, &(**z)); → CORRETTA,
funzione(y, &y); → ERRATA,

funzione((**z), &x); → CORRETTA

Domanda **6**
Risposta corretta
Punteggio ottenuto 6,00 su 6,00
Contrassegna domanda

Date le seguenti definizioni

enum tag_paese {UK, ITALIA};

typedef struct _indUK {
int number;
char street[N1];
int floor;
char pcode[N2];
char town[N3];
} indUK;

enum tag_strada {VIA, PIAZZA, CORSO};

typedef struct _indITA {
int civico;
enum tag_strada strada;
char nomestrada[N1];
char citta[N3];
} indITA;

struct {
enum tag_paese nazione;
union {
indUK address;
indITA indirizzo;
} n;
} X, X2, *PTRX;

indicare quali delle seguenti espressioni sono staticamente (ovvero per il compilatore) corrette e quali sono errate.

PTRX = malloc(sizeof(indITA)); ERRATA ✓
X->n->address->pcode = PTRX.n.address.pcode; ERRATA ✓
PTRX->indirizzo->civico = X.n.address.number; ERRATA ✓
X.n.indirizzo.citta = X2.n.indirizzo.citta; ERRATA ✓
strcpy(X2.n.indirizzo.città, PTRX->n.address.town); CORRETTA ✓
if (strcmp(X2.n.address.street, X.n.indirizzo.nomestrada)); CORRETTA ✓

Risposta corretta.

La risposta corretta è: PTRX = malloc(sizeof(indITA));
→ ERRATA,
X->n->address->pcode = PTRX.n.address.pcode; → ERRATA,

PTRX->indirizzo->civico = X.n.address.number; → ERRATA,

X.n.indirizzo.citta = X2.n.indirizzo.citta; → ERRATA,

strcpy(X2.n.indirizzo.città, PTRX->n.address.town); → CORRETTA,

if (strcmp(X2.n.address.street, X.n.indirizzo.nomestrada)); → CORRETTA

Fine revisione