Tema d'esame (16/02/2024)

Informatica A - 05/12/2024

Esercizio C

Una TODO list è una lista di attività che devono essere svolte. Una TODO list multi-utente è una variante in cui a ogni attività è assegnata una persona che la deve eseguire.

Implementare la TODO list mediante una lista di elementi concatenati. Ogni elemento deve contenere un titolo, un testo, un identificativo numerico della persona incaricata, il costo e la data in cui l'attività è stata creata.

Gli elementi della lista devono essre ordinati prima per identificativo dell'utente (in ordine crescente) e, successivamente (ovvero per elementi che hanno lo stesso identificativo utente), dall'elemento inserito da più tempo a quello inserito più recentemente.

Svolgere l'esercizio attenendosi a quanto richiesto. **NON E' RICHIESTO SCRIVERE IL MAIN**.

1. Si definiscano le strutture dati necessarie allo sviluppo di questo programma. (1 punto)

```c
// Definizione di un elemento della TODO list
typedef struct todoItem {
    char titolo[50];              // Titolo dell'attività
    char testo[200];              // Descrizione dell'attività
    int id_utente;                // Identificativo dell'utente
    double costo;                 // Costo associato all'attività
    time_t data_creazione;        // Data e ora di creazione (formato time_t)
    struct TodoItem *next;        // Puntatore al prossimo elemento nella lista
} TodoItem;

// Definizione della lista TODO
typedef TodoItem *TodoItemPtr;
```

Esercizio C

2. Implementare una funzione **RICORSIVA** *aggiungiAttività* che prende in ingresso i dati di una nuova attività e la aggiunge in modo ordinato nella lista.. (3 punti)

```c
// Funzione ricorsiva per aggiungere un'attività in modo ordinato
TodoItemPtr aggiungiAttivita(TodoItemPtr lista, const char *titolo, const char *testo, int id_utente, double costo, time_t data_creazione)
    // Caso base: inserimento all'inizio o fine della lista
    if (lista == NULL ||
        lista->id_utente > id_utente ||
        (lista->id_utente == id_utente && lista->data_creazione > data_creazione)) {
        TodoItemPtr nuovoNodo = (TodoItem *)malloc(sizeof(TodoItem));
        strncpy(nuovoNodo->titolo, titolo, 50);
        strncpy(nuovoNodo->testo, testo, 200);
        nuovoNodo->id_utente = id_utente;
        nuovoNodo->costo = costo;
        nuovoNodo->data_creazione = data_creazione;
        nuovoNodo->next = lista;
        return nuovoNodo;
    }
    // Chiamata ricorsiva: scorriamo la lista
    lista->next = aggiungiAttivita(lista->next, titolo, testo, id_utente, costo, data_creazione);
    return lista;
}
```

3. Implementare una funzione *rimuoviAttività* che prende in ingresso un identificativo utente e rimuove (il nodo non viene eliminato dalla memoria) l'elemento inserito da più tempo che appartiene all'identificativo utente passato come parametro. La funzione restituisce iniltre un puntatore all'elemento rimosso. (4 punti)

```c
TodoItemPtr rimuoviAttivita(TodoItemPtr *lista, int id_utente) {
    if (lista == NULL || *lista == NULL) {
        // Lista vuota o non valida
        return NULL;}
    TodoItemPtr corrente = *lista;
    TodoItemPtr precedente = NULL;
    // Cerca il primo elemento che corrisponde all'id_utente
    while (corrente != NULL && corrente->id_utente != id_utente) {
        precedente = corrente;
        corrente = corrente->next;}
    // Se nessun elemento corrisponde all'id_utente
    if (corrente == NULL) {
        return NULL;}
    // Rimuovi l'elemento corrente dalla lista
    if (precedente == NULL) {
        // L'elemento da rimuovere è il primo nella lista
        *lista = corrente->next;
    } else {
        // L'elemento da rimuovere è successivo al primo
        precedente->next = corrente->next;}
    // Scollega il nodo dalla lista
    corrente->next = NULL;
    // Restituisci il nodo scollegato
    return corrente;}
```

4. Scrivere una funzione **RICORSIVA** che, presa in ingresso la lista ed un identificativo, calcoli il costo totale delle attività da svolgere presenti nella lista. (3 punti)

```c
double calcolaCostoTotale(TodoItemPtr lista, int id_utente) {
    // Caso base: la lista è vuota
    if (lista == NULL) {
        return 0.0;}
    // Se l'attività appartiene all'utente specificato, somma il costo
    if (lista->id_utente == id_utente) {
        return lista->costo + calcolaCostoTotale(lista->next, id_utente);}
    // Altrimenti, prosegui con il resto della lista
    return calcolaCostoTotale(lista->next, id_utente);}
```

5. Scrivere una funzione **RICORSIVA** che presa in ingresso la lista, ritorni l'identifiativo utente e la descrizione dell'attività più costosa. La funzione non deve ritornare una struct o un nodo di una lista, ma le due variabili. (3 punti)

```c
void attivitaPiuCostosa(TodoItemPtr lista, int *id_utente_max, char *testo_max, double *costo_max) {
    // Caso base: lista vuota
    if (lista == NULL) {
        return;}
    // Controlla se l'attività corrente ha un costo maggiore di quello massimo trovato finora
    if (lista->costo > *costo_max) {
        *costo_max = lista->costo;                // Aggiorna il massimo costo
        *id_utente_max = lista->id_utente;        // Aggiorna l'ID dell'utente
        strcpy(testo_max, lista->testo);}         // Copia la descrizione dell'attività
    // Passa al nodo successivo nella lista
    attivitaPiuCostosa(lista->next, id_utente_max, testo_max, costo_max);}
```

6. *RECUPERO LABORATORIO* Scrivere una funzione che preso in ingresso un identificativo, elimini dalla lista tutti le attività legate all'identificativo inserito.(3 punti)

```c
void eliminaAttivita(TodoItemPtr *lista, int id_utente) {
    if (lista == NULL || *lista == NULL) {
        // Lista vuota o non valida
        return;}
    TodoItemPtr corrente = *lista;
    TodoItemPtr precedente = NULL;
    while (corrente != NULL) {
        if (corrente->id_utente == id_utente) {
            // Rimuovi il nodo corrente
            if (precedente == NULL) {
                // Il nodo corrente è la testa della lista
                *lista = corrente->next;
            } else {
                // Il nodo corrente è in mezzo o in fondo
                precedente->next = corrente->next;}
            // Conserva il nodo da eliminare e passa al successivo
            TodoItemPtr da_eliminare = corrente;
            corrente = corrente->next;
            // Libera il nodo dalla memoria
            free(da_eliminare);
        } else {
            // Passa al nodo successivo
            precedente = corrente;
            corrente = corrente->next;}}}
```

Esercizio Matlab

Data la matrice A:

```
A =

    32    76    36   -97    75
    16    56    12   -32   -12
    78    -6   -23    55    74
    14   -21    43    83    31
```

1. Moltiplicare tutti i valori della matrice A per un numero casuale tra -1 e 100 (1 riga)(1 punto)

```
A = A*randi([-1 100])
```

2. Cancellare la terza colonna (1 riga)(1 punto)

```
A(:, 3) = []
```

3. Inserire dopo la seconda colonna una colonna contenente valori casuali tra -11 e 37 (1 riga)(1 punto)

```
A = [A(:, 1:2) randi([-11 37], [4 1]) A(:,3:end)]
```

Esercizio Matlab

4. Crea la matrice B contenente solo i numeri della matrice A multipli di un numero causale tra 2 e 5 (1 riga)(1 punto)

```
B = A(rem(A, randi([2 5]))==0)
```

5. Calcolare la media dei soli valori dispari della matrica A (1 riga - 1 punto)

```
mean(A(rem(A,2)!=0))
```

6. Scrivere una funzione Matlab che calcoli la somma dei valori interi da 1 fino a n, con n parametro formale della funzione. Stampare a video solamente il valore della somma (max 4 righe)(2 punti).

```
function s = somma(n)
    s = sum(1:n); % Calcola la somma dei numeri interi da 1 a n
    disp(s);      % Stampa il risultato
end
```

Algebra booleana

(a) Si costruisca la tabella di verità della seguente espressione booleana in tre variabili, badando alla precedenza tra gli operatori logici. Eventualmente si aggiungano parentesi.
Scrivere l'espressione semplificata. Non si accetteranno soluzioni senza il procedimento. (2 punti)

NOT (A OR NOT B) AND (B OR NOT C)

NOT(A OR NOT B) AND (B OR NOT C)

(NOT A AND B) AND (B OR NOT C) → De Morgan

(NOT A AND B AND B) OR (NOT A AND B AND NOT C) → Proprietà distributiva

(NOT A AND B) OR (NOT A AND B AND NOT C)

(NOT A AND B) AND (1 OR NOT C) → Fattorizzazione

(NOT A AND B) AND 1

(NOT A AND B)

Algebra booleana

(a) Si costruisca la tabella di verità della seguente espressione booleana in tre variabili, badando alla precedenza tra gli operatori logici. Eventualmente si aggiungano parentesi.
Scrivere l'espressione semplificata. Non si accetteranno soluzioni senza il procedimento. (2 punti)

NOT (A OR NOT B) AND (B OR NOT C)

NOT A AND B

A	B	NOT(A)	NOT(A) AND B
0	0	1	0
0	1	1	1
1	0	0	0
1	1	0	0