Računske vježbe 9

Programiranje I

Elementi liste su sortirani cijeli brojevi u rastućem poretku. Napisati sljedeće funkcije:

a) `insert_node` koja vrši umetanje elementa u listu tako da se ne naruši poredak,

b) `reverse` koja okreće elemente liste,

c) `remove_node` koja vrši brisanje prvog elementa čija je vrijednost jednaka proslijeđenom argumentu,

d) `free_list` koja oslobađa memoriju zauzetu za listu.

Glavni program kreira listu i poziva funkcije napisane pod a, b, c i d.

```c
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int number;
    struct node* next;
};

void print_list(struct node*);
struct node* insert_node(struct node*, int);
struct node* reverse(struct node*);
struct node* remove_node(struct node*, int);
void free_list(struct node*);

int main()
{
    struct node* current, * previous, * head, * result;
    int i, n, insertNumber;
    puts("Unesite broj elemenata liste");
    scanf("%d", &n);
    puts("Unesite elemente liste");
    for (i = 0; i < n; i++)
    {
        current = (struct node*)malloc(sizeof(struct node));
        scanf("%d", &(current->number));
        current->next = NULL;
        if (i == 0) head = current;
        else previous->next = current;
        previous = current;
    }
    printf("Unesite trazeni broj:");
    scanf("%d", &insertNumber);
    printf("Vrsimo umetanje broja: %d\n", insertNumber);
    head = insert_node(head, insertNumber);
    print_list(head);
```

```
    puts("Okrecemo listu");
    head = reverse(head);
    print_list(head);
    printf("Vrsimo brisanje broja: %d\n", insertNumber);
    head = remove_node(head, insertNumber);
    print_list(head);
    puts("Oslobadjamo memoriju");
    free_list(head);
    head = NULL; // ukazujemo da glava ni na sta ne pokazuje
}

void print_list(struct node* head)
{
    printf("Elementi liste su: ");
    while (head != NULL)
    {
        printf("%d ", head->number);
        head = head->next;
    }
    printf("\n");
}

struct node* insert_node(struct node* head, int number)
{
    struct node* newNode, * start, * previous;
    //u promjenljivoj start pamtimo glavu liste
    start = head;
    newNode = (struct node*)malloc(sizeof(struct node));
    newNode->number = number;
    //dio programa gdje se vrsi provjera da li broj treba umetnuti na pocetku liste
    if (number <= head->number)
    {
        newNode->next = head;
        return newNode;
    }
    //dio programa gdje se vrsi provjera da li broj treba umetnuti unutar liste
    previous = head;
    head = head->next;
    while (head != NULL)
    {
        if (number <= head->number)
        {
            previous->next = newNode;
            newNode->next = head;
            return start;
        }
        else
        {
            previous = head;
            head = head->next;
        }
    }
    //ako program stigne do ove linije onda broj treba umetnuti na kraju liste
    previous->next = newNode;
    newNode->next = NULL;
    return start;
}
```

```c
struct node* reverse(struct node* head)
{
    struct node* previous = NULL;
    struct node* current = head;
    struct node* next = NULL;
    while (current != NULL) {
        // Cuvamo next
        next = current->next;
        // Okrecemo next za tekuci cvor
        current->next = previous;
        // Idemo naprijed
        previous = current;
        current = next;
    }
    return previous;
}

struct node* remove_node(struct node* head, int value)
{
    struct node* temp = head, * previous;
    // dio programa gdje se vrsi provjera da li broj treba obrisati na pocetku liste
    if (temp != NULL && temp->number == value) {
        head = temp->next; // glava se mijenja
        free(temp); // brisemo staru glavu
        return head;
    }
    // Vrsimo pretragu po vrijednosti, vodimo racuna o
    // prethodnom cvoru jer moramo promijeniti 'previous->next'
    while (temp != NULL && temp->number != value) {
        previous = temp;
        temp = temp->next;
    }
    // ukoliko vrijednost nije pronadjena u listi
    if (temp == NULL)
        return head;
    // preskacemo element koji treba obrisati
    previous->next = temp->next;

    free(temp); //oslobadjamo memoriju

    return head;

}

void free_list(struct node* head)
{
    struct node* temp;

    while (head != NULL)
    {
        temp = head;
        head = head->next;
        free(temp);
    }
}
```