7ο Φροντιστήριο Δομημένου Προγραμματισμού

Άσκηση 1: Ορίστε ένα struct Student με πεδία: id, name, grade.

Λύση

```
typedef struct {
    int id;
    char name[50];
    float grade;
} Student;
```

Επεξήγηση

- Το struct ομαδοποιεί διαφορετικούς τύπους δεδομένων.
- Το typedef μας επιτρέπει να χρησιμοποιούμε το Student χωρίς τη λέξη struct.

Άσκηση 2: Δημιουργήστε έναν φοιτητή και έναν δείκτη που δείχνει σε αυτόν.

Λύση

```
Student s = {1, "Maria", 8.5};
Student *ptr = &s;
```

Επεξήγηση

- Ο δείκτης ptr αποθηκεύει τη διεύθυνση του s.
- Η πρόσβαση στα πεδία γίνεται με ptr->grade.

Άσκηση 3: Γράψτε συνάρτηση που εκτυπώνει φοιτητή μέσω δείκτη.

Λύση

```
void printStudent(Student *s) {
    printf("%d %s %.2f\n", s->id, s->name, s->grade);
}
```

Επεξήγηση

- Χρησιμοποιούμε -> γιατί έχουμε δείκτη.
- Αποφεύγεται η αντιγραφή δεδομένων.

Άσκηση 4: Ορίστε κόμβο λίστας φοιτητών.

Λύση

```
typedef struct Node {
    Student data;
    struct Node *next;
} Node;
```

Επεξήγηση

- Ο δείκτης next δείχνει στον επόμενο κόμβο.
- struct Node * απαιτείται γιατί το struct δεν έχει ολοκληρωθεί ακόμα.

Άσκηση 5: Δημιουργήστε συνάρτηση που δεσμεύει νέο κόμβο.

Λύση

```
Node *createNode(Student s) {
    Node *newNode = malloc(sizeof(Node));
    newNode->data = s;
    newNode->next = NULL;
    return newNode;
}
```

Επεξήγηση

- Το malloc δεσμεύει δυναμική μνήμη.
- Το next = NULL δηλώνει τέλος λίστας.

Άσκηση 6: Προσθέστε φοιτητή στην αρχή της λίστας.

Λύση

```
void insertFront(Node **head, Student s) {
    Node *newNode = createNode(s);
    newNode->next = *head;
    *head = newNode;
}
```

Επεξήγηση

- Χρησιμοποιούμε Node ** για να αλλάξουμε το head.
- Η λίστα ενημερώνεται σωστά.

Άσκηση 7: Εκτυπώστε όλους τους φοιτητές της λίστας.

Λύση

```
void printList(Node *head) {
     while (head != NULL) {
          printStudent(&head->data);
          head = head->next;
     }
}
```

Επεξήγηση

- Η λίστα διασχίζεται μέχρι το NULL.
- Δεν τροποποιούμε το αρχικό head.

Άσκηση 8: Βρείτε φοιτητή με συγκεκριμένο id.

Λύση

```
Node *findStudent(Node *head, int id) {
     while (head != NULL) {
          if (head->data.id == id)
               return head;
               head = head->next;
          }
     return NULL;
}
```

Επεξήγηση

- Επιστρέφεται δείκτης στον κόμβο.
- Αν δεν βρεθεί, επιστρέφεται NULL.

Άσκηση 9: Διαγράψτε φοιτητή με συγκεκριμένο ID.

Λύση

```
void deleteStudent(Node **head, int id) {
    Node *curr = *head, *prev = NULL;

    while (curr != NULL && curr->data.id != id) {
        prev = curr;
        curr = curr->next;
    }

    if (curr == NULL) return;

    if (prev == NULL)
        *head = curr->next;
    else
        prev->next = curr->next;

    free(curr);
}
```

Επεξήγηση

- Ελέγχουμε αν ο κόμβος είναι πρώτος.
- Απελευθερώνουμε μνήμη με free.

Άσκηση 10: Απελευθερώστε όλη τη λίστα από τη μνήμη.

Λύση

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

Επεξήγηση

- Αποφεύγουμε memory leaks.
- Κάθε κόμβος απελευθερώνεται ξεχωριστά.

Το παρακάτω πρόγραμμα: Δημιουργεί μια κενή λίστα φοιτητών, Εισάγει φοιτητές στην αρχή της λίστας, Εκτυπώνει όλους τους φοιτητές, Αναζητά φοιτητή με βάση το ID, Διαγράφει φοιτητή, Απελευθερώνει τη μνήμη

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// Ορισμός struct Student
typedef struct {
    int id;
    char name[50];
    float grade;
} Student;
// Ορισμός κόμβου λίστας
typedef struct Node {
    Student data;
    struct Node *next;
} Node;
// Συνάρτηση εκτύπωσης φοιτητή
void printStudent(Student *s) {
    printf("ID: %d | Όνομα: %s | Βαθμός: %.2f\n",
           s->id, s->name, s->grade);
}
// Δημιουργία νέου κόμβου
Node *createNode(Student s) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    if (newNode == NULL) {
        printf("Σφάλμα δέσμευσης μνήμης\n");
        exit(1);
    }
    newNode->data = s;
    newNode->next = NULL;
    return newNode;
}
// Εισαγωγή στην αρχή της λίστας
void insertFront(Node **head, Student s) {
    Node *newNode = createNode(s);
    newNode->next = *head;
    *head = newNode;
}
// Εκτύπωση ολόκληρης λίστας
void printList(Node *head) {
    printf("\n--- Λίστα Φοιτητών ---\n");
    while (head != NULL) {
        printStudent(&head->data);
        head = head->next;
    }
}
// Αναζήτηση φοιτητή με ID
Node *findStudent(Node *head, int id) {
    while (head != NULL) {
        if (head->data.id == id)
            return head;
        head = head->next;
    }
    return NULL;
}
```

```c
// Διαγραφή φοιτητή με ID
void deleteStudent(Node **head, int id) {
    Node *curr = *head;
    Node *prev = NULL;
    while (curr != NULL && curr->data.id != id) {
        prev = curr;
        curr = curr->next;
    }
    if (curr == NULL) {
        printf("Ο φοιτητής με ID %d δεν βρέθηκε.\n", id);
        return;
    }
    if (prev == NULL) {
        *head = curr->next;  // διαγραφή πρώτου κόμβου
    } else {
        prev->next = curr->next;
    }
    free(curr);
    printf("Ο φοιτητής με ID %d διαγράφηκε.\n", id);
}
// Απελευθέρωση λίστας
void freeList(Node *head) {
    while (head != NULL) {
        Node *temp = head;
        head = head->next;
        free(temp);
    }
}
// main
int main() {
    Node *head = NULL; // κενή λίστα
    Student s1 = {1, "Maria", 8.5};
    Student s2 = {2, "Nikos", 7.2};
    Student s3 = {3, "Eleni", 9.1};
    insertFront(&head, s1);
    insertFront(&head, s2);
    insertFront(&head, s3);
    printList(head);
    // Αναζήτηση φοιτητή
    int searchId = 2;
    Node *found = findStudent(head, searchId);
    if (found != NULL) {
        printf("\nΒρέθηκε φοιτητής:\n");
        printStudent(&found->data);
    } else {
        printf("\nΟ φοιτητής με ID %d δεν βρέθηκε.\n", searchId);
    }
    // Διαγραφή φοιτητή
    deleteStudent(&head, 1);
    printList(head);
    // Καθαρισμός μνήμης
    freeList(head);
    return 0;
}
```