HOSPITAL MANAGEMENT SYSTEM

DEVELOPED BY L.AYADI AND C.REDOUANI.

SUPERVISED BY LAKHDARI KEIRA.

PROJECT OBJECTIVES

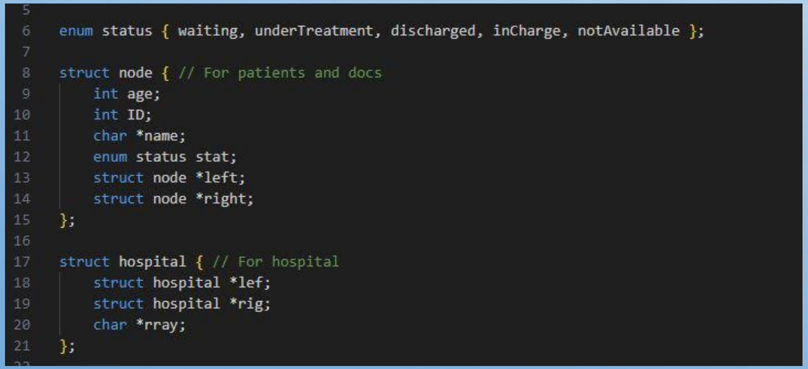
Key Goals :

- Manage patients (add/edit/delete) and track status:
  - ➢ Waiting (Queue)
  - ➢ Under treatment (BST)
  - ➢ Discharged (Stack)
- Handle doctor availability (inchaharge/notAvailable via linked list)
- Data persistence: save/load records to/from .txt files

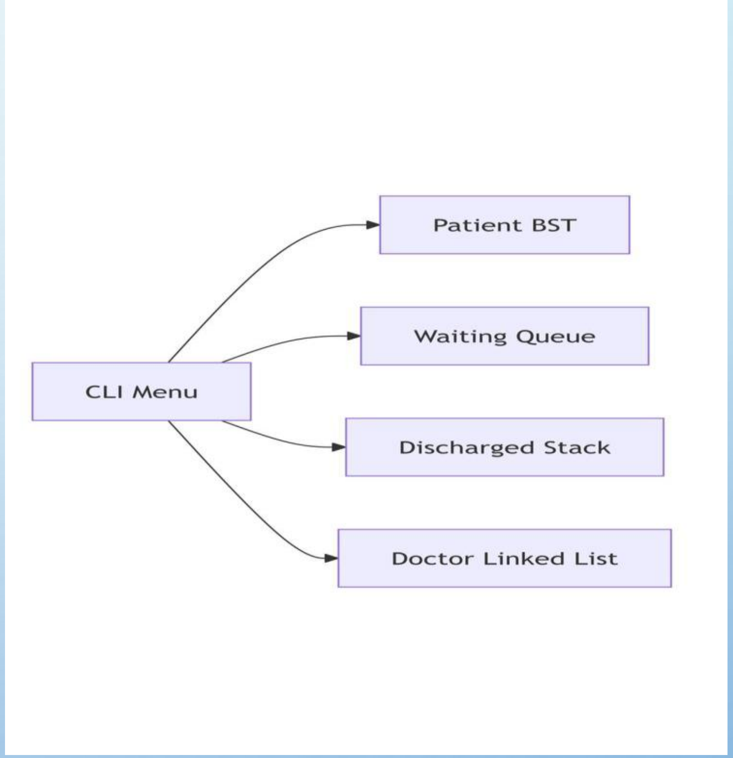
SYSTEM ARCHITECTURE

why these structures ?

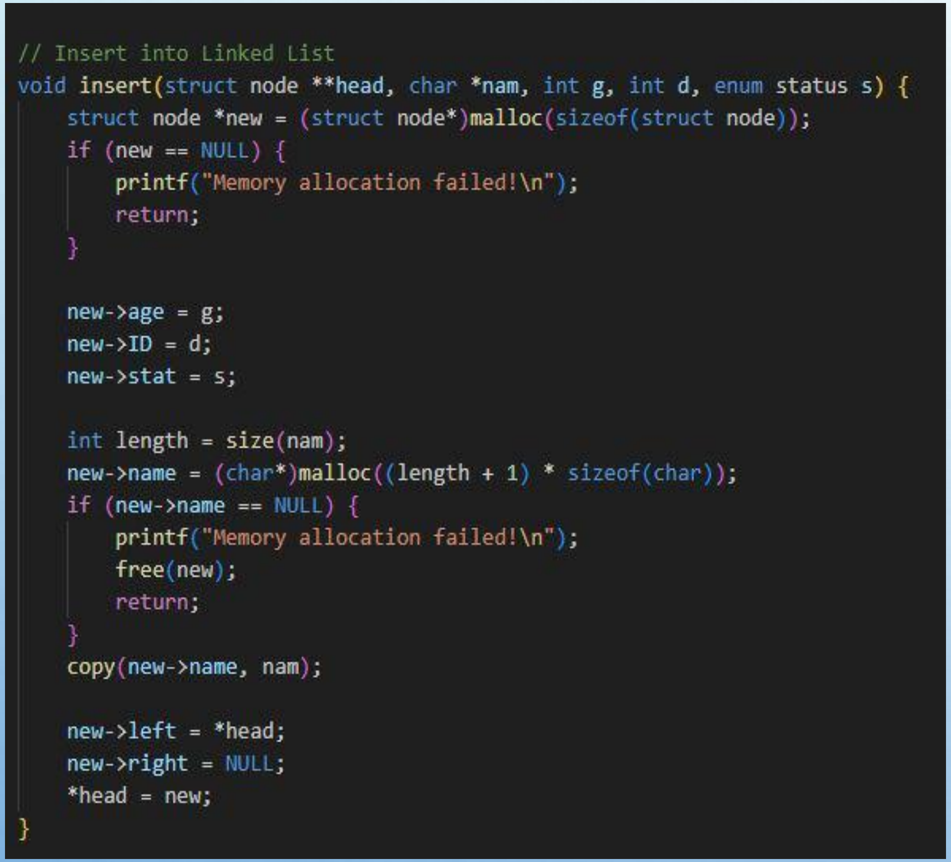
- BST: efficient search/modify for active patients (o(log n)).
- Queue: FIFO fairness for waiting patients.
- Stack: LIFO for discharge undo functionality.



```c
enum status { waiting, underTreatment, discharged, inCharge, notAvailable };

struct node { // For patients and docs
    int age;
    int ID;
    char *name;
    enum status stat;
    struct node *left;
    struct node *right;
};

struct hospital { // For hospital
    struct hospital *lef;
    struct hospital *rig;
    char *rray;
};
```

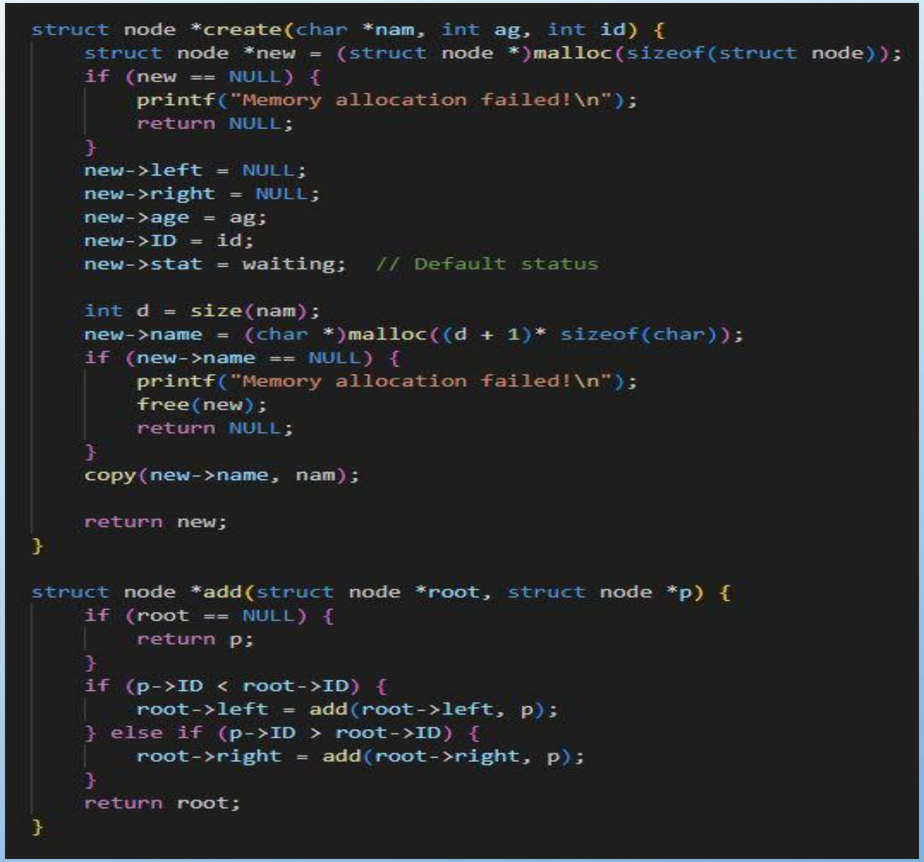


KEY CODE SNIPPETS



```c
// Insert into Linked List
void insert(struct node **head, char *nam, int g, int d, enum status s) {
    struct node *new = (struct node*)malloc(sizeof(struct node));
    if (new == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }

    new->age = g;
    new->ID = d;
    new->stat = s;

    int length = size(nam);
    new->name = (char*)malloc((length + 1) * sizeof(char));
    if (new->name == NULL) {
        printf("Memory allocation failed!\n");
        free(new);
        return;
    }
    copy(new->name, nam);

    new->left = *head;
    new->right = NULL;
    *head = new;
}
```

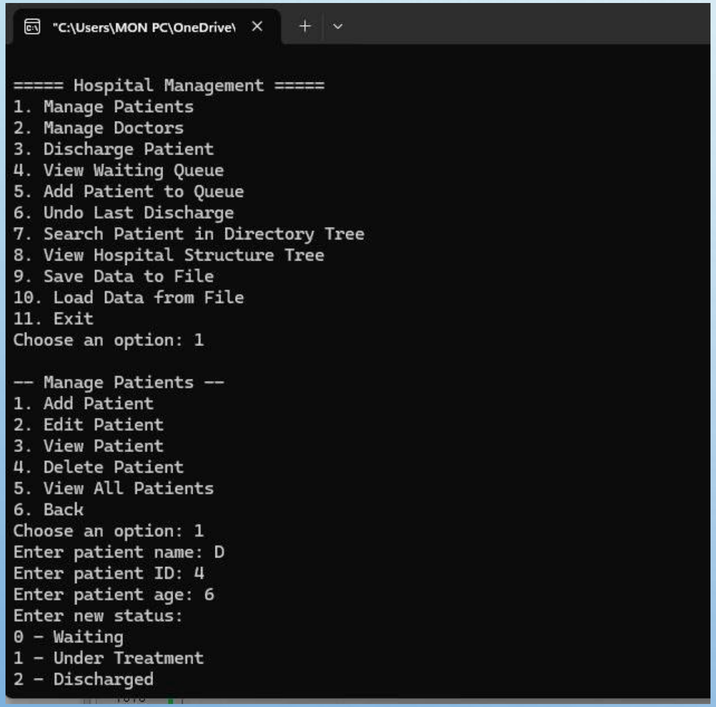
Insert function inserts a node onto the linked list and left pointer used as next .



```c
struct node *create(char *nam, int ag, int id) {
    struct node *new = (struct node *)malloc(sizeof(struct node));
    if (new == NULL) {
        printf("Memory allocation failed!\n");
        return NULL;
    }
    new->left = NULL;
    new->right = NULL;
    new->age = ag;
    new->ID = id;
    new->stat = waiting;  // Default status

    int d = size(nam);
    new->name = (char *)malloc((d + 1)* sizeof(char));
    if (new->name == NULL) {
        printf("Memory allocation failed!\n");
        free(new);
        return NULL;
    }
    copy(new->name, nam);

    return new;
}

struct node *add(struct node *root, struct node *p) {
    if (root == NULL) {
        return p;
    }
    if (p->ID < root->ID) {
        root->left = add(root->left, p);
    } else if (p->ID > root->ID) {
        root->right = add(root->right, p);
    }
    return root;
}
```

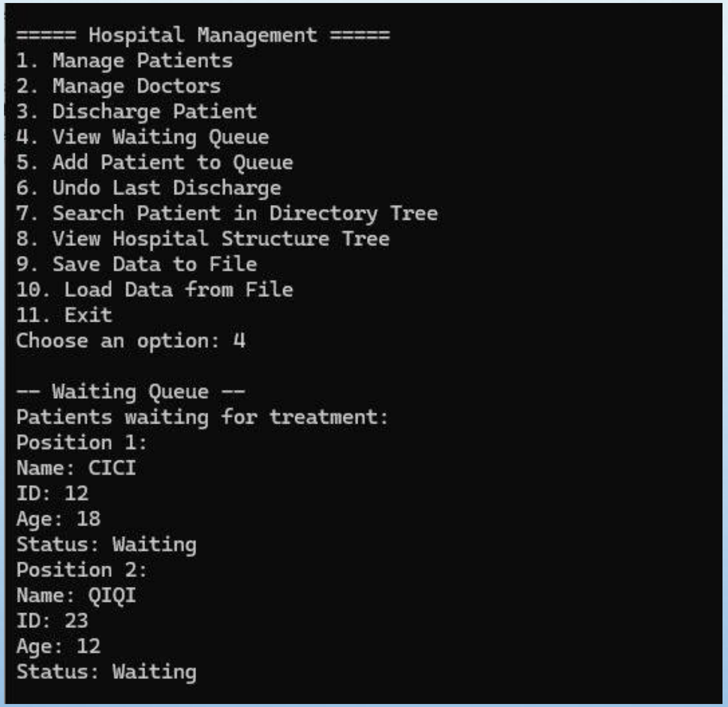
The function add insert a node onto the under treatment patient tree.

USER INTERFACE DEMO



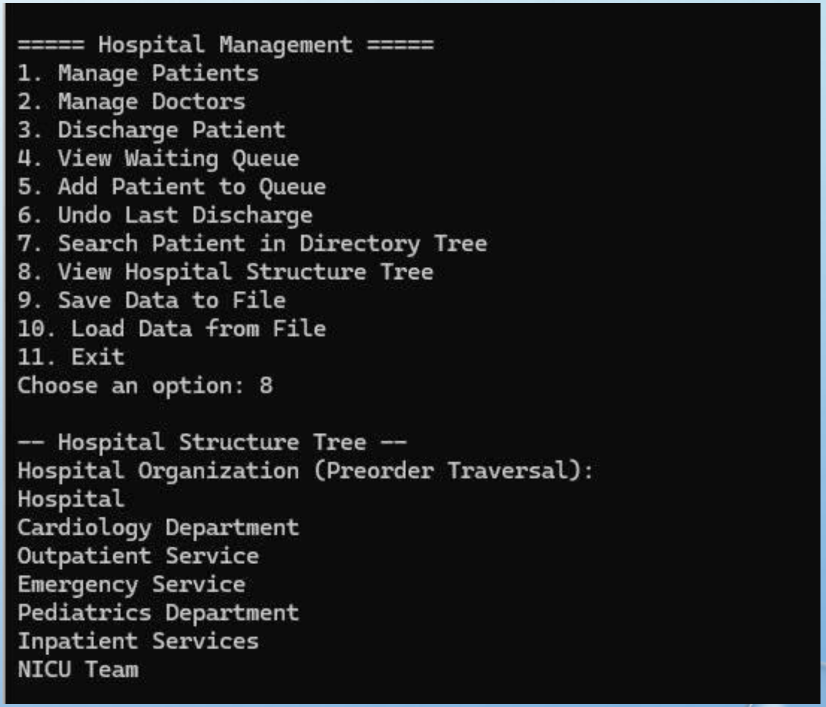
```
===== Hospital Management =====
1. Manage Patients
2. Manage Doctors
3. Discharge Patient
4. View Waiting Queue
5. Add Patient to Queue
6. Undo Last Discharge
7. Search Patient in Directory Tree
8. View Hospital Structure Tree
9. Save Data to File
10. Load Data from File
11. Exit
Choose an option: 1

-- Manage Patients --
1. Add Patient
2. Edit Patient
3. View Patient
4. Delete Patient
5. View All Patients
6. Back
Choose an option: 1
Enter patient name: D
Enter patient ID: 4
Enter patient age: 6
Enter new status:
0 - Waiting
1 - Under Treatment
2 - Discharged
```

Adding patient in patient management.



```
===== Hospital Management =====
1. Manage Patients
2. Manage Doctors
3. Discharge Patient
4. View Waiting Queue
5. Add Patient to Queue
6. Undo Last Discharge
7. Search Patient in Directory Tree
8. View Hospital Structure Tree
9. Save Data to File
10. Load Data from File
11. Exit
Choose an option: 4

-- Waiting Queue --
Patients waiting for treatment:
Position 1:
Name: CICI
ID: 12
Age: 18
Status: Waiting
Position 2:
Name: QIQI
ID: 23
Age: 12
Status: Waiting
```

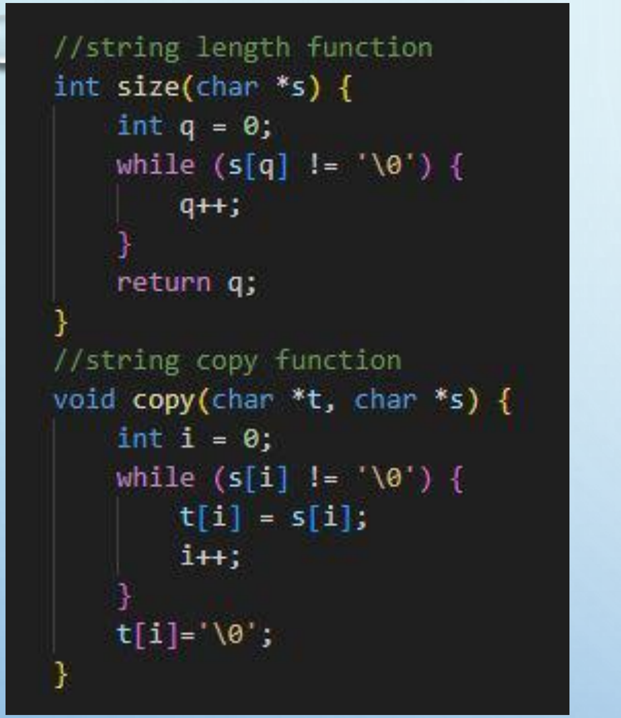
Waiting queue display.



```
===== Hospital Management =====
1. Manage Patients
2. Manage Doctors
3. Discharge Patient
4. View Waiting Queue
5. Add Patient to Queue
6. Undo Last Discharge
7. Search Patient in Directory Tree
8. View Hospital Structure Tree
9. Save Data to File
10. Load Data from File
11. Exit
Choose an option: 8

-- Hospital Structure Tree --
Hospital Organization (Preorder Traversal):
Hospital
Cardiology Department
Outpatient Service
Emergency Service
Pediatrics Department
Inpatient Services
NICU Team
```

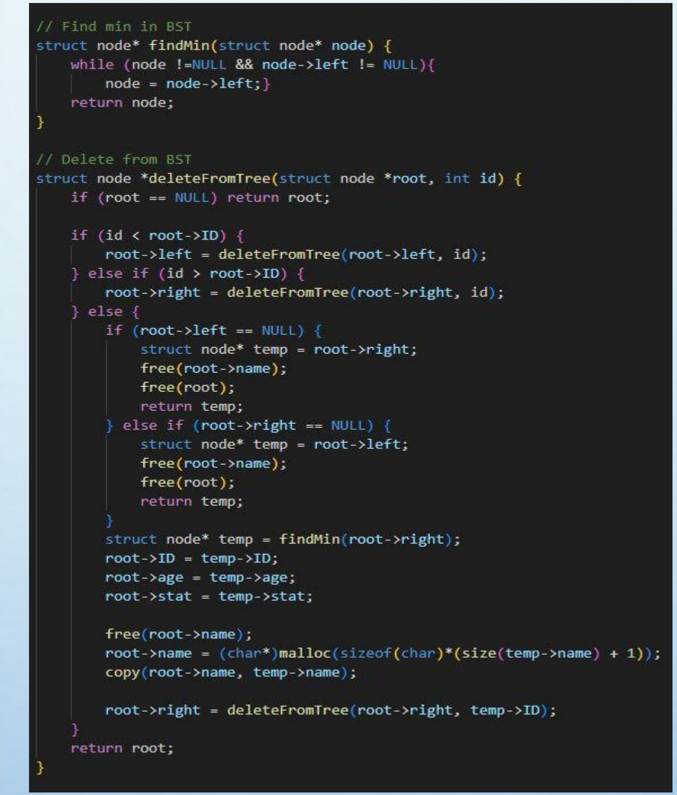
Hospital structure tree display .

UNIQUE FEATURES

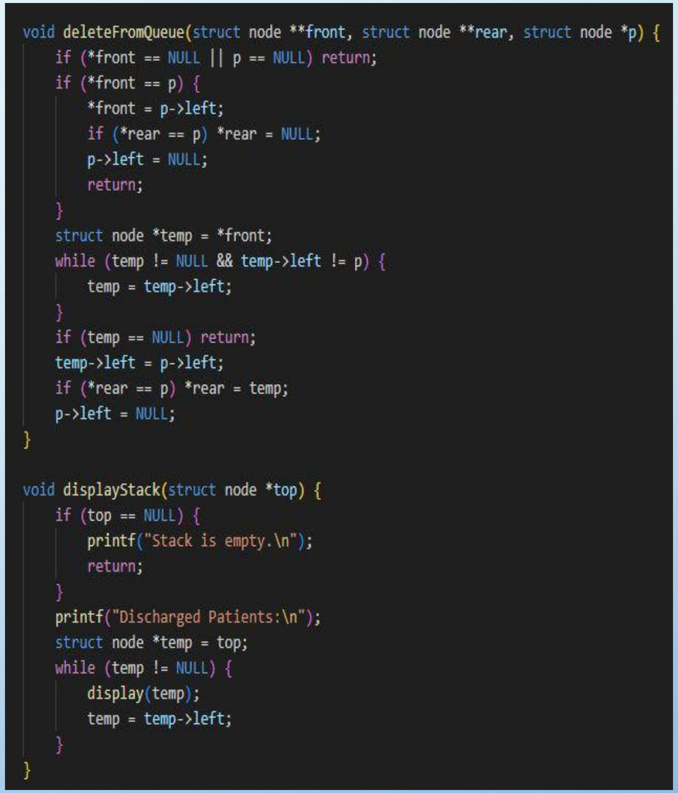


```c
//string length function
int size(char *s) {
    int q = 0;
    while (s[q] != '\0') {
        q++;
    }
    return q;
}
//string copy function
void copy(char *t, char *s) {
    int i = 0;
    while (s[i] != '\0') {
        t[i] = s[i];
        i++;
    }
    t[i]='\0';
}
```

We use size and copy functions to manipulate the name strings of patients making it easier to manipulate and transfer



```c
// Find min in BST
struct node* findMin(struct node* node) {
    while (node !=NULL && node->left != NULL){
        node = node->left;}
    return node;
}

// Delete from BST
struct node *deleteFromTree(struct node *root, int id) {
    if (root == NULL) return root;

    if (id < root->ID) {
        root->left = deleteFromTree(root->left, id);
    } else if (id > root->ID) {
        root->right = deleteFromTree(root->right, id);
    } else {
        if (root->left == NULL) {
            struct node* temp = root->right;
            free(root->name);
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct node* temp = root->left;
            free(root->name);
            free(root);
            return temp;
        }
        struct node* temp = findMin(root->right);
        root->ID = temp->ID;
        root->age = temp->age;
        root->stat = temp->stat;

        free(root->name);
        root->name = (char*)malloc(sizeof(char)*(size(temp->name) + 1));
        copy(root->name, temp->name);

        root->right = deleteFromTree(root->right, temp->ID);
    }
    return root;
}
```



```c
void deleteFromQueue(struct node **front, struct node **rear, struct node *p) {
    if (*front == NULL || p == NULL) return;
    if (*front == p) {
        *front = p->left;
        if (*rear == p) *rear = NULL;
        p->left = NULL;
        return;
    }
    struct node *temp = *front;
    while (temp != NULL && temp->left != p) {
        temp = temp->left;
    }
    if (temp == NULL) return;
    temp->left = p->left;
    if (*rear == p) *rear = temp;
    p->left = NULL;
}

void displayStack(struct node *top) {
    if (top == NULL) {
        printf("Stack is empty.\n");
        return;
    }
    printf("Discharged Patients:\n");
    struct node *temp = top;
    while (temp != NULL) {
        display(temp);
        temp = temp->left;
    }
}
```

In transferring patients between structures ,we remove the free from the delete functions to keep the founded pointer to change its place.