REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
ECOLE NATIONALE SUPERIEURE D'INFORMATIQUE

CPI 1

Année 2023 / 2024

TP
EN ALGORITHMIQUE ET STRUCTURES DE DONNEES

DYNAMIQUES (ALSDD)

REALISE PAR : Bendifallah Rami

Bensaha khalil

Patient management in a polyclinic

Section : A

Groupe N° : 02

Summary : page

introduction :

The Patient Management System serves as an essential tool for healthcare facilities, aiming to streamline patient registration, record-keeping, and queue management processes. This project addresses the critical need for efficient patient care by providing a platform that prioritizes patient data organization and timely service delivery. Through this report, we'll delve into the system's design, functionality, and its impact on enhancing healthcare service efficiency and patient satisfaction.

Problematic:

How can a basic patient management system be designed to efficiently handle patient queues and records in a small healthcare facility?

Solution:

Designing a patient management system using queues and linked lists to efficiently manage patient records and prioritize patient care in a healthcare facility.

Data structure:

```
// the structure of the patient .
Patient = record
Begin
Age, priority : integer .
Phone, ss_number : long integer .
Address,bloodgroup,name: string . // which is array of char
Gender : char .
Priority: Boolean .
End .
// Structure for a single node in a queue
typedef struct queuenode
{
   // every single node will have two things
   patient *val;          // first: the patient information
   struct queuenode *addr; // second : a pointer to the next node
} QueueNode;

// Structure for a  the head and the tail of the queue
typedef struct
```

```
{
   QueueNode *head, *tail;
} Queue;

// Structure for a node in the linked list
typedef struct listNode
{
   Queue data;          // each node contain a queue
   struct listNode *next; // pointer to the- next node
} ListNode;
```

Explanation:

- Every patient will be represented by a record.

patient Structure:

Represents information about a patient.

Fields:

name: A character array to store the patient's name.

age: An integer to store the patient's age.

address: A character array to store the patient's address.

phone: A long integer to store the patient's phone number.

bloodgroup: A character array to store the patient's blood group.

gender: A character to store the patient's gender.

ss_number: A long integer to store the patient's social security number.

priority: An integer representing the priority of the patient.

queuenode Structure:

Represents a single node in a queue.

Fields:

val: A pointer to a patient structure containing patient information.

addr: A pointer to the next node in the queue.

Queue Structure:

Represents the head and tail of a queue.

Fields:

head: A pointer to the first node in the queue.

tail: A pointer to the last node in the queue.

:

ListNode Structure:

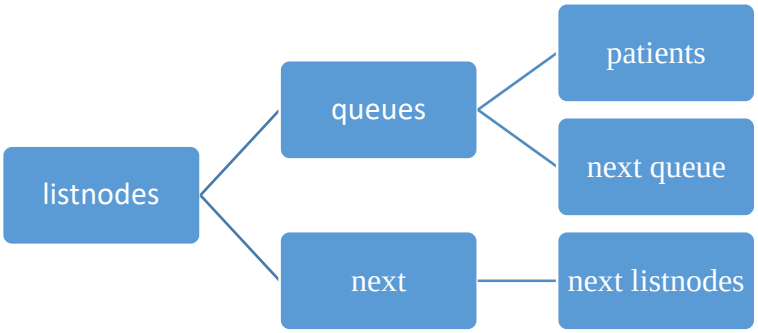
Represents a node in a linked list each node will represent a single department in the hospital.

In our situation It will be a list with 5 nodes.

Fields:

data: Contains a Queue structure, representing a queue of patients in a specific department.

next: A pointer to the next node in the linked list.



Modular division:

The abstract machine: // All the names are descriptive

void ass_first_name(patient *p, char *first_name)

void ass_age(patient *p, int age)

void ass_address(patient *p, char *address)

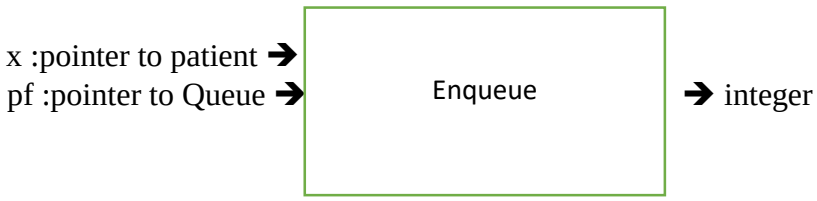
void ass_phone(patient *p, long phone)

void ass_bloodgroup(patient *p, char *bloodgroup)

void ass_gender(patient *p, char gender)

Queues basic operation:

function



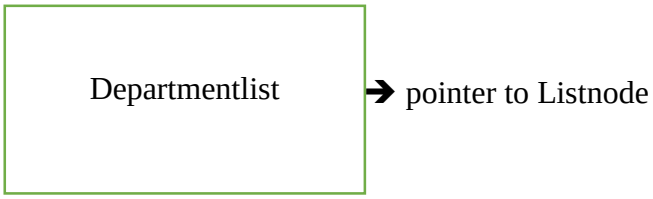
role: The enqueue function inserts a patient into the queue based on their priority level, ensuring patients with higher priority are placed ahead. It returns a boolean indicating successful insertion.

function

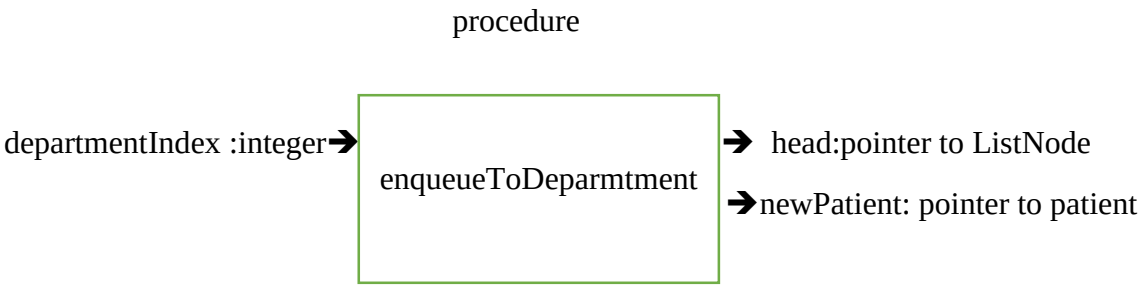
role :
The **dequeue** function removes a patient from the queue based on their priority level, ensuring that patients with the highest priority are dequeued first. It returns a boolean value indicating whether the dequeue operation was successful

department list operations:

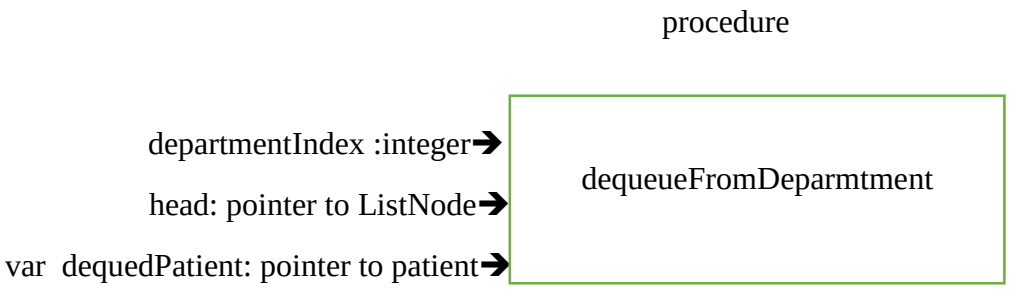
function



Role: The Departmentlist function creates a list of 5 nodes, each representing a hospital department, and initializes a queue within each node for managing patients. It then returns the head of the list.

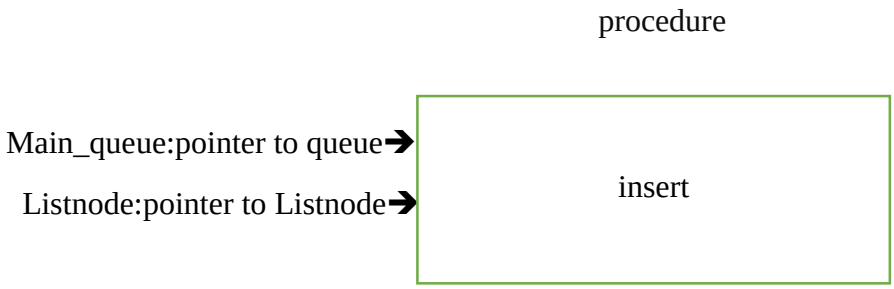


Role: The enqueueToDepartment function adds a patient to a department's service queue by copying their information and enqueuing them into the department's queue. If the department doesn't exist, it prints an error message.



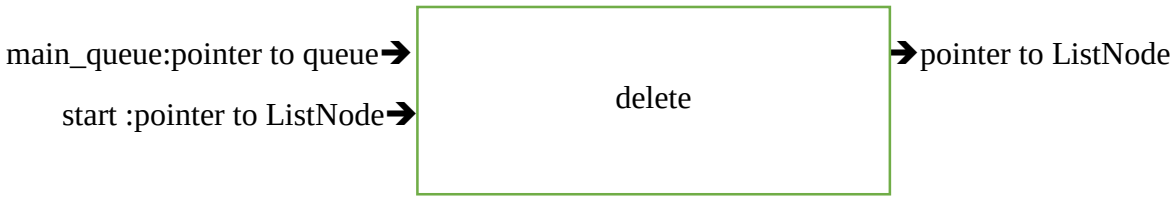
Role: The dequeueFromDepartment function removes a patient from a specified department's queue based on their priority, ensuring higher priority patients are dequeued first. It handles cases where the department is not found or empty. If successful, it dequeues the patient.

Main operation:



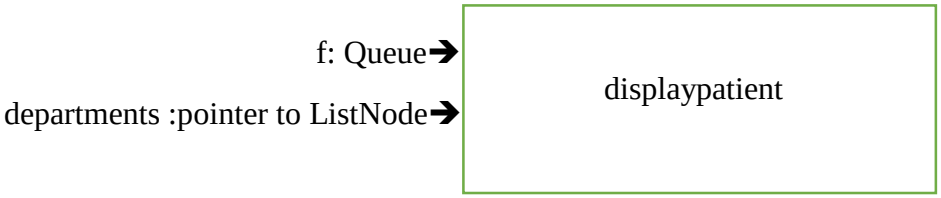
Role: This function collects patient information and enqueues them either into a specific department or the main service queue based on user choice. It also allows assigning a priority level to patients.

function



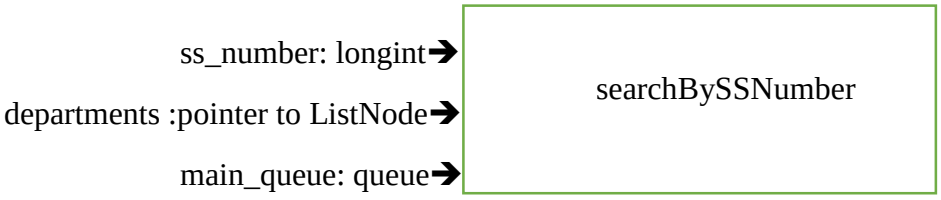
Role: This function facilitates patient deletion from either the main service or a department. It prompts the user for the deletion choice, dequeues the patient, displays their information if successful, and returns the updated start node of the list.

procedure



Role: This function displays patient information from either the main queue or a specific department's queue based on user input.

procedure



Role: The function **searchBySSNumber** searches for a patient using their social security number in both the main service queue and department queues. If the patient is found, it displays their information.

we use also the menu in every step to give the user choices such as:
void deparmenu () // to display the available department .

void delete_menu() // to chose a department for the operation of removing a patient .

```
void display_menu() // to give the user choice from where he want to display the patient informations.
void displayintro() // to display a beautiful introduction
```

IV. The source codes of the main operations :

1.The procedure enqueueToDepartment :

```
void enqueueToDepartment(ListNode *head, int departmentIndex, patient *newPatient)
int currentIndex = 1;
   ListNode *current = head;

   // Traverse to the specified department
   while (current != NULL && currentIndex < departmentIndex) {
      current = current->next;
      currentIndex++;
   }

   if (current == NULL) {
      printf("Department not found.\n");
      return;
   }

   // Enqueue the patient to the department's queue
   if (!Enqueue(newPatient, &(current->data))) {
      fprintf(stderr, "Failed to enqueue patient to department %d.\n", departmentIndex);
      exit(EXIT_FAILURE);
   }

   printf("Patient enqueued into department %d.\n", departmentIndex);
}
**************************************************************************
```

2.The procedure dequeueFromDepartmen:

```
void dequeueFromDepartment(ListNode *head, int departmentIndex, patient **dequeuedPatient) {

   int currentIndex = 1;

   ListNode *current  = head;

   // Traverse to the specified department
```

```
    while (current != NULL && currentIndex < departmentIndex) {
        current = current->next;
        currentIndex++;
    }
    if (current == NULL) {
        printf("Department not found.\n");
        return; // stop progressing
    }
    // Dequeue a patient from the department's queue
    if (!Dequeue(dequeuedPatient, &(current->data))) {
        printf("Department %d is empty.\n", departmentIndex);
        return;
    }
    printf("Patient dequeued from department %d:\n", departmentIndex);
    // Print patient information if needed
}
```

****

3.The procedure insert:

```
void *insert(Queue *main_queue, ListNode *listnode)   //A Procedure to get patient information and enqueue it into the main queue or a special department
{
    patient *ptr;   // create a pointer to a patient
    allocate(&ptr); // starting by allocating it

printf("*----------enter patient information----------*\n");
    printf("Enter patient Name: ");   // After we collect the information of the patient and assign it
    char name[100];
    scanf("%99s", name);
    ass_first_name(ptr, name); // assign the name
```

```
printf("Enter the patient's age: ");
int age;
scanf("%d", &age);
ass_age(ptr, age); // assign the age

printf("Enter your home address: ");
char address[100];
scanf("%99s", address);
ass_address(ptr, address); // assign the address

printf("Enter your phone number: ");
long phone;
scanf("%ld", &phone);
ass_phone(ptr, phone); // assign the phone number
printf("Enter the blood group of Patient: ");
char bloodgroup[100];
scanf("%99s", bloodgroup);
ass_bloodgroup(ptr, bloodgroup); // assign the blood group of the patient

printf("Enter the gender(m for male and f for female): ");
char gender;
scanf(" %c", &gender);
ass_gender(ptr, gender); // assign the gender
printf("enter the social security number: ");
long socialsecurity;
scanf("%d", &socialsecurity);
ass_ss_number(ptr, socialsecurity); // assign the social security number .
 // Prompt user for priority
```

```
printf("Enter patient priority (1 to 10, where 10 is highest priority): ");
scanf("%d", &(ptr->priority));
printf("*--------------------------------------------------------------------------------*\n");
printf("Do you want to choose a department or stay in the main service?\n");
printf("****************************************\n");
printf("            1: Choose department\n");
printf("            2: Stay in main list\n");
printf("****************************************\n");
int choice;
scanf("%d", &choice);
// after the user choose we deal with his choice
switch (choice)
{
case 1: // choosing a departement
  deparmenu();
   int dept_choice;
   do{
      printf("enter the department(between 1 and 5): ");
      scanf("%d", &dept_choice);}
   while (dept_choice <= 0 || dept_choice > 5);

      enqueueToDepartment(listnode, dept_choice, ptr); // enqeue the patient
      printf("Patient enqueued into department %d.\n", dept_choice);
   break;
case 2:  // if the user choose the main service
   Enqueue(ptr, main_queue);
   printf("Patient enqueued into main list.\n"); // to inform the user by that
   break;
```

```
    default:
        // if the user choose invalid choice
        printf("Invalid choice.\n");
        break;
    }
}
```

****

4.The procedure delete:

```
ListNode *delete(ListNode *start, Queue *main_queue) // to delete a patient from the main service or one of . the departments
{
    int choix;
    patient *patient = NULL; // Initialize patient pointer
    delete_menu();          // Added function call
    scanf("%d", &choix);
    switch (choix)
    {
    case 1:
        if (Dequeue(&patient, main_queue)) // Check if Dequeue operation is successful
        {
            printf("Patient dequeued from main service:\n"); // display the patient information for the last time
            printf("*****************************************************\n");
            printf("*                Patient Information                *\n");
            printf("*****************************************************\n");
            printf("* %-14s: %-30s *\n", "Name", patient->name);
            printf("* %-14s: %-30d *\n", "Age", patient->age);
            printf("* %-14s: %-30s *\n", "Address", patient->address);
            printf("* %-14s: %-30ld *\n", "Phone", patient->phone);
            printf("* %-14s: %-30s *\n", "Blood Group", patient->bloodgroup);
            printf("* %-14s: %-30c *\n", "Gender", patient->gender);
            printf("* %-14s: %-30ld *\n", "Social Security Number", patient->ss_number);
            printf("*****************************************************\n\n");
            free(patient); // Free memory allocated for the patient
        }
        else
        { // we inform the user that the main service is empty
            printf("Main service is empty.\n");
        }
        break;
```

```
        case 2:
        {
            int index;
            deparmenu();
            printf("Choose the department: \n");
            scanf("%d", &index);
            dequeueFromDepartment(start, index, &patient);  // the same thing for the delete from a department
            break;
        }

        default: // when the user's choice doesn't exist .
            printf("Invalid choice.\n");
            break;
        }

        return start; // Added return statement
    }
    ********************************************************************************
```

The deplaypatinet code:

```
void displaypatient(Queue f, ListNode *departments) {
    QueueNode *p;
    int choice;
    display_menu();
    printf("Enter your choice: ");
    scanf("%d", &choice);
    switch (choice) {
        case 1:
            if (EmptyQueue(f)) {
                printf("Queue is empty.\n");
                return;
            }
            p = f.head;
            printf("Main Queue elements:\n");
            while (p != NULL) {
                printf("*****************************************************\n");
                printf("*                Patient Information              *\n");
```

```
        printf("*****************************************************\n");
        printf("* %-14s: %-30s *\n", "Name", p->val->name);
        printf("* %-14s: %-30d *\n", "Age", p->val->age);
        printf("* %-14s: %-30s *\n", "Address", p->val->address);
        printf("* %-14s: %-30ld *\n", "Phone", p->val->phone);
        printf("* %-14s: %-30s *\n", "Blood Group", p->val->bloodgroup);
        printf("* %-14s: %-30c *\n", "Gender", p->val->gender);
        printf("* %-14s: %-30ld *\n", "Social Security Number", p->val->ss_number);
        printf("* %-14s: %-30d *\n", "Priority", p->val->priority); // Display priority
        printf("*****************************************************\n\n");
        p = p->addr;
    }
    break;

case 2:
    if (departments == NULL) {
        printf("No departments found.\n");
        return;
    }
    int departmentIndex;
    printf("Enter department index (1 to 5): ");
    deparmenu();
    scanf("%d", &departmentIndex);

    if (departmentIndex < 1 || departmentIndex > 5) {
        printf("Invalid department index.\n");
        return;
    }

    ListNode *current = departments;
    int currentIndex = 1;
```

```
    while (current != NULL && currentIndex < departmentIndex) {
        current = current->next;
        currentIndex++;
    }

    if (current == NULL) {
        printf("Department not found.\n");
        return;
    }

    p = current->data.head;
    printf("Department %d elements:\n", departmentIndex);
    while (p != NULL) {
        printf("*****************************************************\n");
        printf("*                Patient Information                *\n");
        printf("*****************************************************\n");
        printf("* %-14s: %-30s *\n", "Name", p->val->name);
        printf("* %-14s: %-30d *\n", "Age", p->val->age);
        printf("* %-14s: %-30s *\n", "Address", p->val->address);
        printf("* %-14s: %-30ld *\n", "Phone", p->val->phone);
        printf("* %-14s: %-30s *\n", "Blood Group", p->val->bloodgroup);
        printf("* %-14s: %-30c *\n", "Gender", p->val->gender);
        printf("* %-14s: %-30d *\n", "Social Security Number", p->val->ss_number);
        printf("* %-14s: %-30d *\n", "Priority", p->val->priority); // Display priority
        printf("*****************************************************\n\n");
        p = p->addr;
    }
    break;

default:
    printf("Invalid choice.\n");
```

```
        break;
    }
}
```

The search procedure:

```
void searchBySSNumber(long ss_number, Queue main_queue, ListNode *departments)  // procedure to search for a patient using social security number in the hospital
{
    QueueNode *p;
    int found = 0; // a boolean variable initialized by zero

    // Search in the main queue
    p = main_queue.head; // to start manipulating the list
    while (p != NULL)
    {
        if (p->val->ss_number == ss_number)
        { // verify the condition
            // we display his informations
            printf("Patient found in the main queue:\n");
            printf("*****************************************************\n");
            printf("*                Patient Information              *\n");
            printf("*****************************************************\n");
            printf("* %-14s: %-30s *\n", "Name", p->val->name);
            printf("* %-14s: %-30d *\n", "Age", p->val->age);
            printf("* %-14s: %-30s *\n", "Address", p->val->address);
            printf("* %-14s: %-30ld *\n", "Phone", p->val->phone);
            printf("* %-14s: %-30s *\n", "Blood Group", p->val->bloodgroup);
            printf("* %-14s: %-30c *\n", "Gender", p->val->gender);
            printf("* %-14s: %-30ld *\n", "Social Security Number", p->val->ss_number);
            printf("*****************************************************\n");
            found = 1; // we set the variable in one to ensure that we found the patient
            break;
```

```
        }
        p = p->addr; // go to the next node
    }

    // Search in department queues
    ListNode *current = departments; // Initialize current to departments
    int i = 1;                       // Initialize i to 1
    char *departmentNames[] = {"Cardiology", "Chirurgical (Surgery)", "Orthopedics", "Internal Medicine", "Otorhinolaryngology (ENT)"};

    while (current != NULL)
    {                        // to manipulate the list
        p = current->data.head; // Initialize p inside the loop
        while (p != NULL)
        {
            if (p->val->ss_number == ss_number)
            { // that mean the condition is verified
                // we display the patient's information .
                printf("Patient found in department queue of department %s:\n", departmentNames[i - 1]); // Adjust index for department name
                printf("*****************************************************\n");
                printf("*                Patient Information                *\n");
                printf("*****************************************************\n");
                printf("* %-14s: %-30s *\n", "Name", p->val->name);
                printf("* %-14s: %-30d *\n", "Age", p->val->age);
                printf("* %-14s: %-30s *\n", "Address", p->val->address);
                printf("* %-14s: %-30ld *\n", "Phone", p->val->phone);
                printf("* %-14s: %-30s *\n", "Blood Group", p->val->bloodgroup);
                printf("* %-14s: %-30c *\n", "Gender", p->val->gender);
                printf("* %-14s: %-30ld *\n", "Social Security Number", p->val->ss_number);
                printf("*****************************************************\n");
```

```
            found = 1; // set the variable in one to ensure that we found him .
            break;
        }
        p = p->addr; // to the next .
    }
    if (found) // the boolean that we se before .
        break; // we stop progressing
    current = current->next;
    i++; // Increment i inside the outer loop to show the department name
}

if (!found)
{
    // we didn't find him
    printf("Patient with social security number '%ld' not found.\n", ss_number);
}
}
```

Createqueue procedure:

```
void CreateQueue(Queue *pf)  // procedure_create_queue
{

    pf->head = NULL;     // to create a queue we need to :
    pf->tail = NULL;     // initilise the head and the tail by NULL
}
/*************************************************************************
*************************************************/
int EmptyQueue(Queue f)   // The_function_Empty_queue
{                         // it verify if the queue is empty or not
    return (f.head == NULL); // it will give 1 if true and 0 if else
}
```

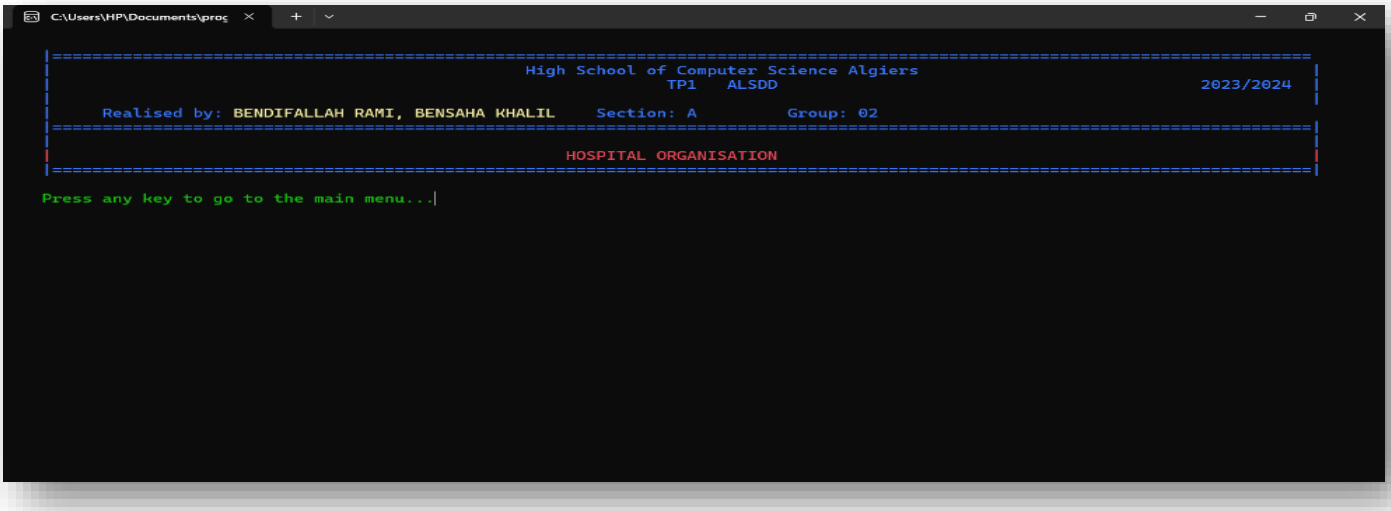
Enqueuer and dequeuer:

```c
int Enqueue(patient *x, Queue *pf) {
    // Allocate memory for a new queue node
    QueueNode *p = (QueueNode *)malloc(sizeof(QueueNode));
    if (p == NULL)
        return 0; // Return 0 if memory allocation fails
    p->val = x;
 // Assign patient pointer to the new node's value
    p->addr = NULL;
 // Initialize the next pointer of the new node to NULL

    // Enqueue based on priority
    if (EmptyQueue(*pf) || x->priority > pf->tail->val->priority) {

// If the queue is empty or the new patient has higher priority than the last patient in the queue
        if (!EmptyQueue(*pf))
            pf->tail->addr = p;
 // If not empty, link the previous tail to the new node
        pf->tail = p;
 // Update the tail of the queue to the new node
        if (EmptyQueue(*pf))
            pf->head = p;
 // If the queue was empty, update the head to the new node
    } else {

// If the new patient has lower or equal priority compared to some existing patients in the queue
        QueueNode *current = pf->head;
        while (current->addr != NULL && current->addr->val->priority >= x->priority)
            current = current->addr;
 // Traverse the queue until finding the right position
        p->addr = current->addr;
 // Link the new node to the next node in the queue
        current->addr = p;
 // Link the previous node to the new node
    }

    return 1;
 // Return 1 to indicate successful enqueue operation
}

```

```c
int Dequeue(patient **x, Queue *pf) {
    // Declare necessary pointers for traversal and tracking
    QueueNode *p, *prev, *highestPriorityNode, *prevHighestPriorityNode;

    // Check if the queue is empty
    if (EmptyQueue(*pf))
        return 0; // Return 0 if the queue is empty


// Initialize variables to track the highest priority node and its previous node
    highestPriorityNode = prevHighestPriorityNode = pf->head;
    prev = pf->head;
    p = pf->head->addr;

    // Traverse the queue to find the node with the highest priority
    while (p != NULL) {
        if (p->val->priority > highestPriorityNode->val->priority) {
            highestPriorityNode = p;
 // Update highest priority node if found
            prevHighestPriorityNode = prev; // Update its previous node
        }
        prev = p; // Move to the next node
        p = p->addr;
    }

    // Dequeue the highest priority node
    *x = highestPriorityNode->val; // Retrieve the patient information
    if (highestPriorityNode == pf->head)
        pf->head = highestPriorityNode->addr;
 // Update head if dequeued node is the first node
    else
        prevHighestPriorityNode->addr = highestPriorityNode->addr;
 // Update previous node's link

    if (highestPriorityNode == pf->tail)
        pf->tail = prevHighestPriorityNode;
 // Update tail if dequeued node is the last node

    free(highestPriorityNode);
 // Free memory allocated for the dequeued node

    return 1; // Return 1 to indicate successful dequeue operation
}
```

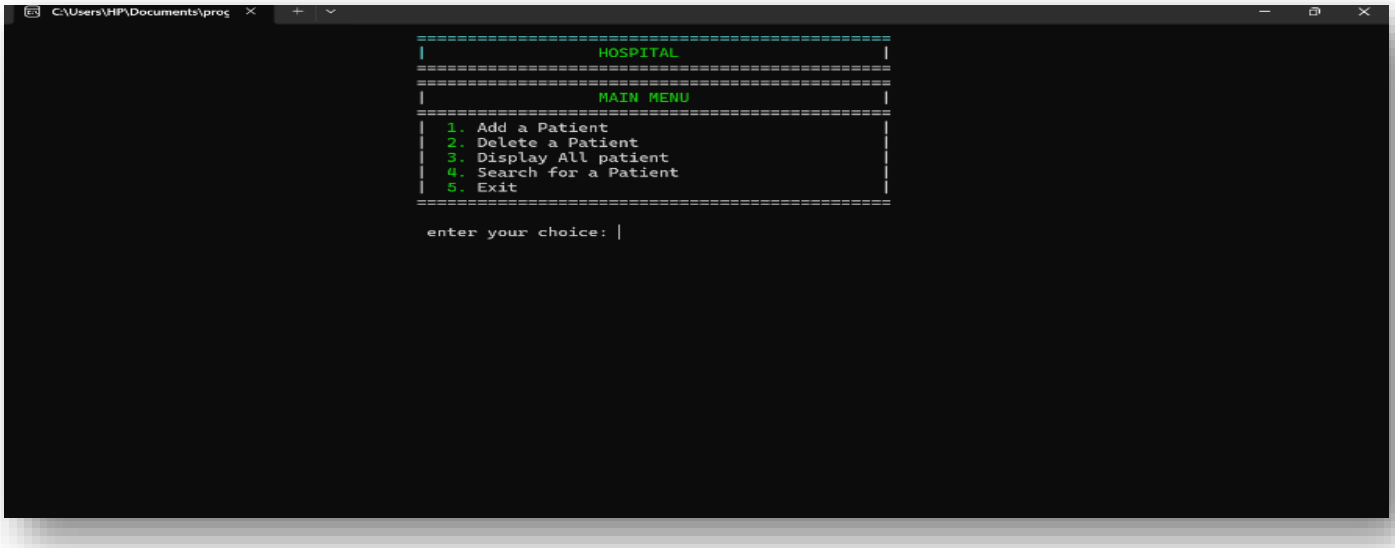
IV. Screen impressions of a test game:

1.The introduction:



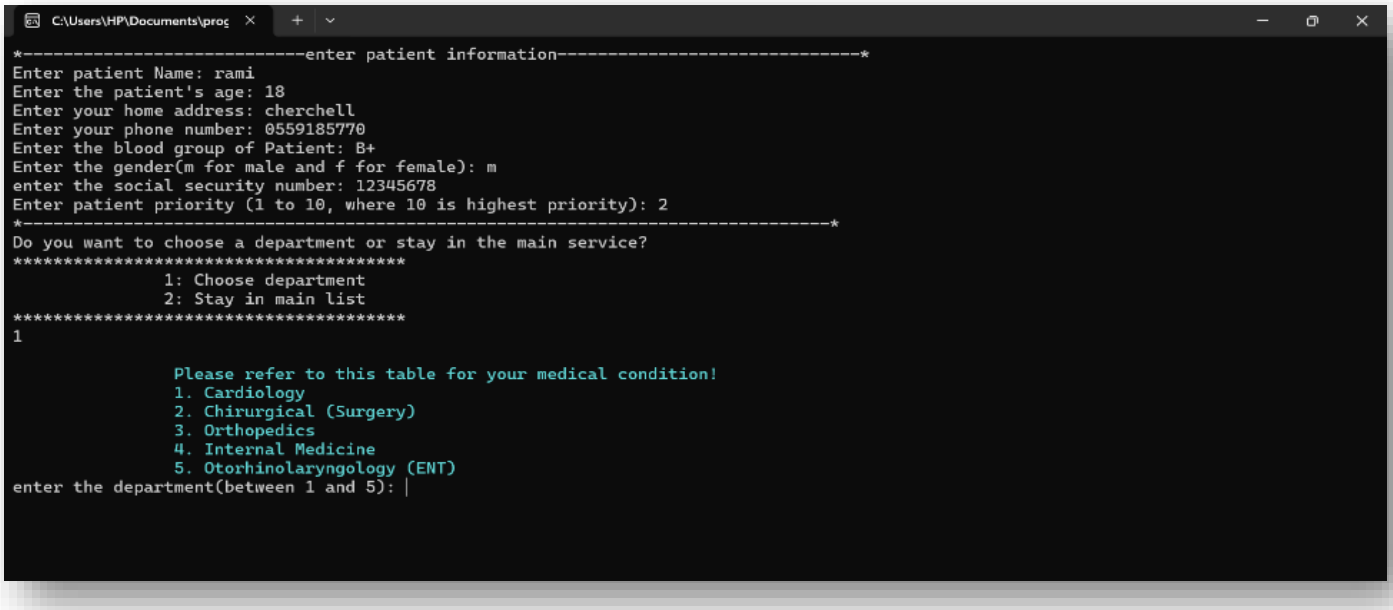
Which is the first what does the user face when he uses the program .
In our example it is a presentation of the work " realized by "
And the other informations .
When the user click in any key he will go to the main menu.

2.The main menu:



The main menu display the available operations
Which are " add a patient , delete a patient , display all patients , and search for patient "
and exit if he want to leave the program .
all these operations are available just by typing the number mentioned in menu .

3.Adding a patient:



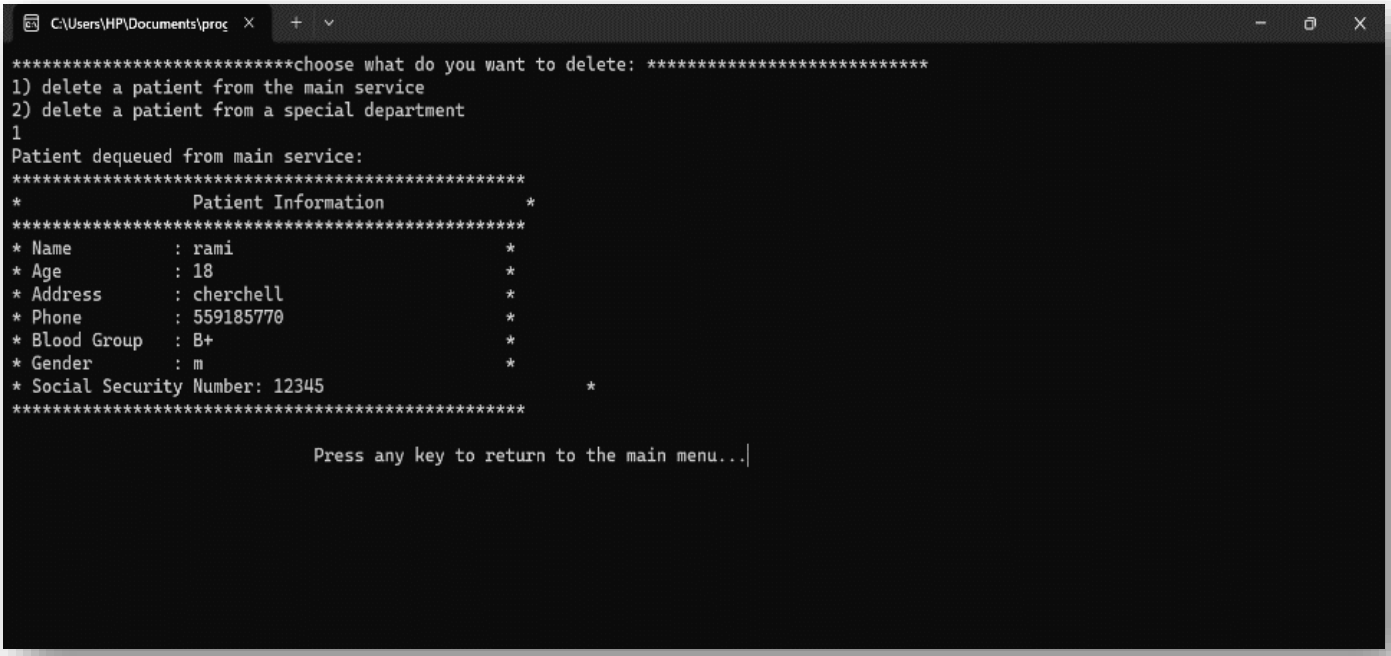
This operation passe by three steps:

1\ Typing the patient ‘s informations and patient priority .

2\ choosing where we want to add this patient ‘ in the main service or in a specific department ’ .

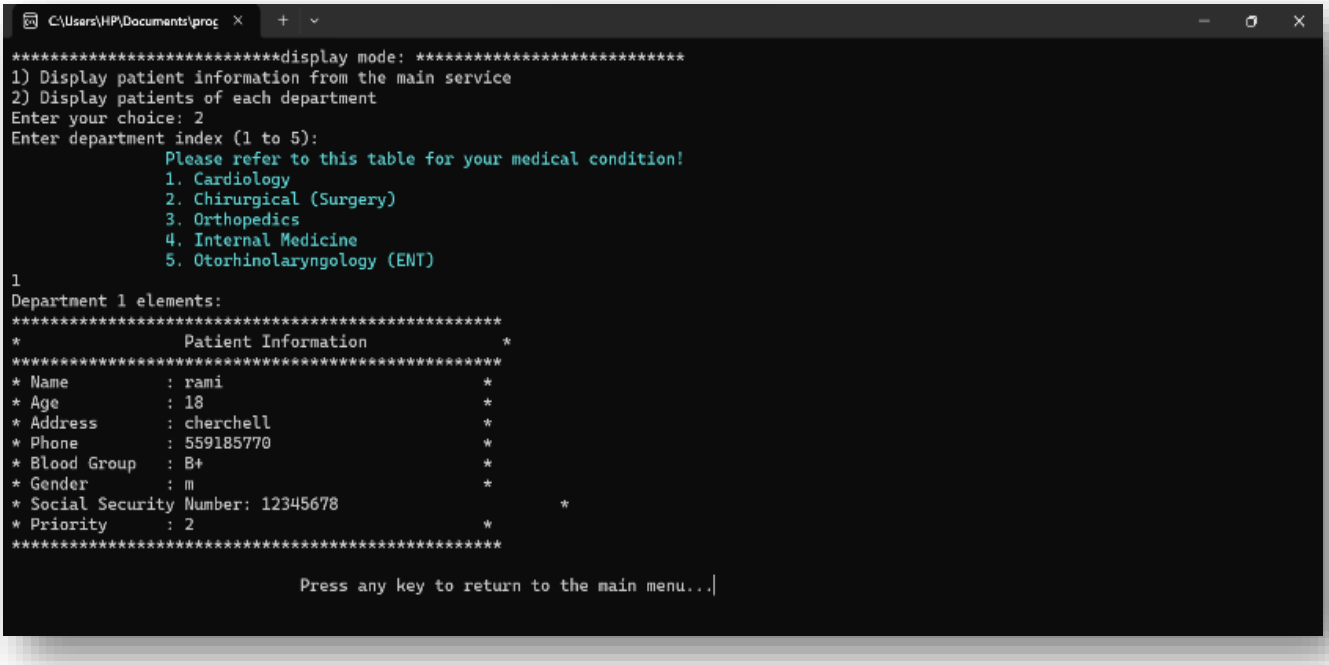
3\ if the user choose In department the program displays the available department and ask the user to choose .

3.Delete a patient



the delete operation passe by two steps:
1\ we ask the user from where he want to delete the patient .
2\we display the information of the patient that we delete .

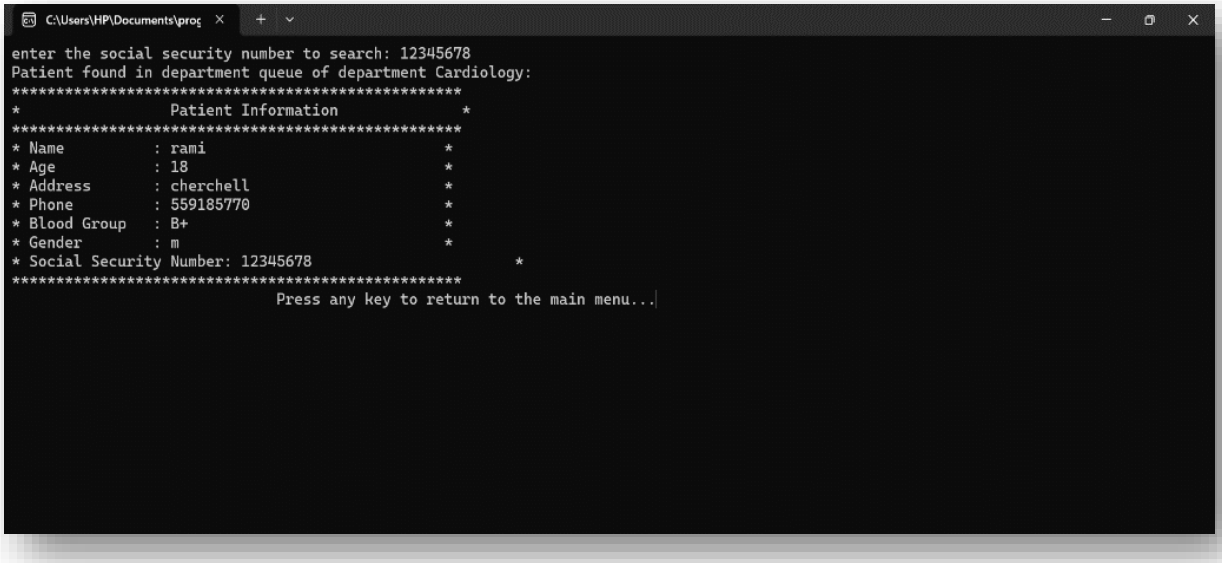
4.Display a patient:



It pass by three steps:
1\ we ask the user from where he want to display the patient informations ‘ from the main service or from a department ’ .
2\ next if the user want to display from a department we ask about the department .
3\ finaly we display the informations of the patient .

5.Saerch by the social security

We ask the user to type the social security number .



We display the patient's informations if the patient exist and where he exist .

-And the the choice five to leave the program.

VI .Conclusion :

Finally , this work present the importance of the linked list and it's adventages over the arrays which are :
1\ The dynamic size : it can easily grow in size during compilating the program .
2\ Efficient insertion and deletion : the operation of insertions or deletion can be performed in constant time O(1) if we have a reference to the node where the operation in performed .
3\ Flexible memory allocation : linked list can be dynamically allocated in memory as long as the nodes created .Allowing for more efficiently memory utilization then the arrays .
4\ Easy to implement priority queue : it allows for implementation of priority queue by inserting elements by their priority .