**CHAPTER 1**

# **INTRODUCTION**

## **1.0 PROBLEM DEFINITION:**

Students will forget their vaccination details like date and name of the vaccine. To keep a track of the data this project will be helpful. This can store multiple Student’s vaccine data. This can help the College to keep a record of the student’s vaccine. When we search the student id it shows a name of student name, vaccine name and date.

**1.1 OBJECTIVES:**

This will reduce the usage of paper and store data digitally in the form of files. The data can also be easily accessed.

## **1.2 METHODOLOGY TO BE FOLLOWED:**

Use of doubly linked lists to store the kid’s vaccination data in the form of files. There is use of files also in this project. Doubly linked list uses self-referential structure.

## **1.3** **EXPECTED OUTCOMES:**

In this project we can:

* Create record
* Add record
* Display record
* Search record
* Delete record

This has details in the form of file. These details include like

* Student Name
* USN\_No
* Department Name
* Phone\_No
* Vaccine Name and
* Date

## **1.4 HARDWARE AND SOFTWARE REQUIREMENTS:**

Hardware Requirements:

* Personal Computer
* Minimum of 2gb RAM
* 64-bit Operating system
* Windows 7 or above

Software requirements:

* Dev C or any other IDEs

**CHAPTER 2**

**2.0 DATA STRUCTURES**

Data Structures are collections of data values that preserve a link between the data, functions, and the data they are applied to. The following are various types of data structures,

* Arrays
* Stacks
* Queues
* Linked List
* Trees
* G

**2.1 Arrays:**

A data structure that stores a finite number of similar items in contiguous memory locations is known as an array. The number of elements to be used is commonly expressed as an index ranging from 0 to N -1, with 0 denoting the bottom limit and N -1 denoting the upper boundary. The arrays can be one-dimensional (fig. 2.1.1), two-dimensional (fig. 2.1.2), or multi-dimensional (fig. 2.1.3). It is very simple to access all of the elements in an array by using the index number.

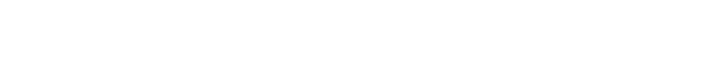
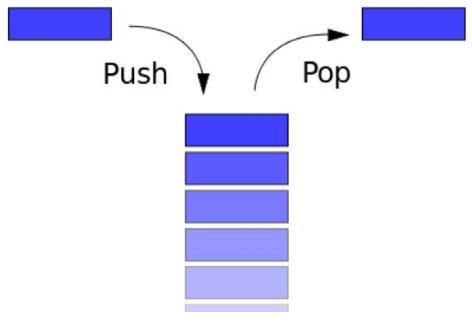
Syntax:

Data – type Array – Name [index]; Example: Int A[20];

**2.2 Stack:**

It's a linear data structure in which components are added and deleted from the same end, according to the LIFO technique (last in, first out)(in fig2.2.1). The data structure can be represented as an array or as a linked list. A variable called top is used to modify the stack.

For example, a stack of papers, books, and so on.



**Fig 2.2.1: Stack operation**

The basic and primitive operations are

* Push.
* Pop.
* Display

1. **Push operation:** Push is the process of adding an element to the top of a stack. Stack overflow occurs when we try to put a large number of elements into a stack that is larger than its maximum capacity.
2. **Pop operation:** Pop is the process of removing an element from the top of a stack. A stack underflow scenario occurs when the stack is fully empty but we continue to try to delete elements.
3. **Display operation:** Printing all of the elements of the stack is called a display.

**2.3 Queue:**

A queue is a linear structure in which all operations are carried out in a predefined order. First in, first out is the order (FIFO)(fig 2.3.1). It is a collection of data types organised in such a way that data is inserted at one end and deleted at the other.E.g. queue in the cafeteria, petrol stations, etc.

The main queue operations are:

* Insertion
* Deletion
* Display

1. **Insertion:** Insertion is the process of adding a new element to the queue. Before inserting a new item, the rear is incremented first, and then the item is inserted. The condition to check in insertion is queue overflow, which occurs when the back of the queue reaches its maximum size.
2. **Deletion:** The elimination of the front element from the queue is referred to as deletion, and following deletion, the front pointer must be incremented. The criterion to verify in deletion is queue under flow, which means there are no items in the queue.
3. **Display:** Printing all the elements from front to the rear is called a display.

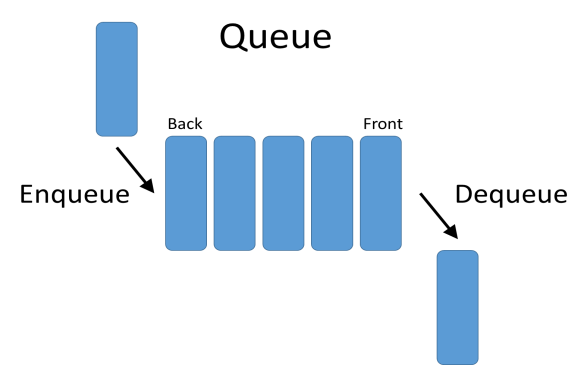


Fig 2.3.1: Queue representation

**Types of queues:**

* Linear Queue (or) ordinary queue
* Circular queue
* Double ended queue
* Priority queue

1. **Linear Queue**

The linear queue is ordered list in which all the data elements are organized in sequential order. In contrast, circular queue stores all the data in circular fashion. The linear queue follows the FIFO order for completing the task (the element which is added in the first position is going to be deleted in the first position).

## **Circular queue**

The linear queue is a sequentially ordered list of data components. A circular queue, on the other hand, stores all data in a round pattern. The linear queue completes the task in the first-in, first-out (FIFO) order (Fig 2.3.2) (the element which is added in the first position is going to be deleted in the first position).

1. **Double ended queue**

Deque, also known as a Double Ended Queue, is generalised queue data structure that allows for insertion and deletion at both ends.

## **Priority queue**

The Priority Queue is a queue extension that has the qualities shown below.

* Each item has a priority number assigned to it.
* A higher priority element is dequeued first, followed by a lower priority element.
* If two components have same priority, they are served in the order in which they appear in the queue.

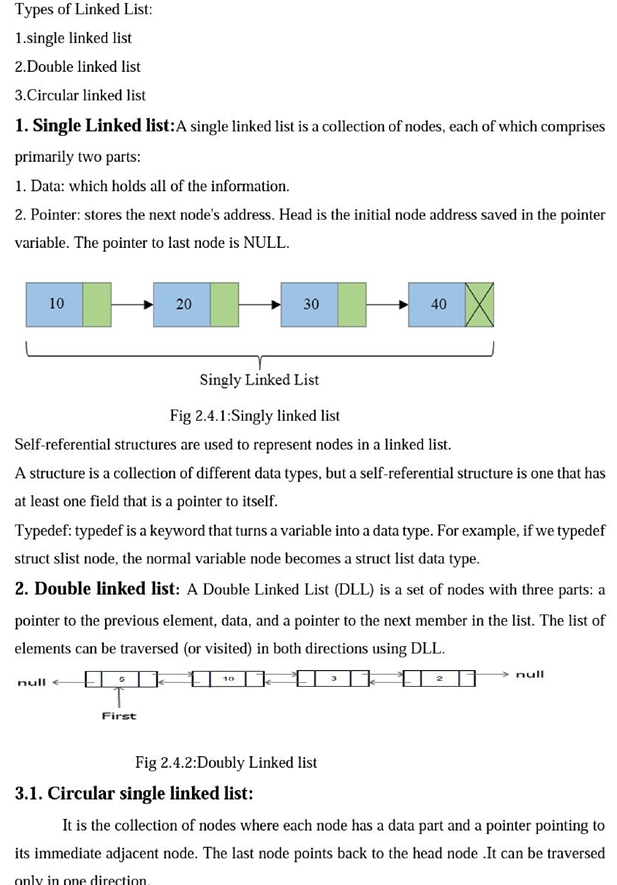
**2.4 Linked List:**

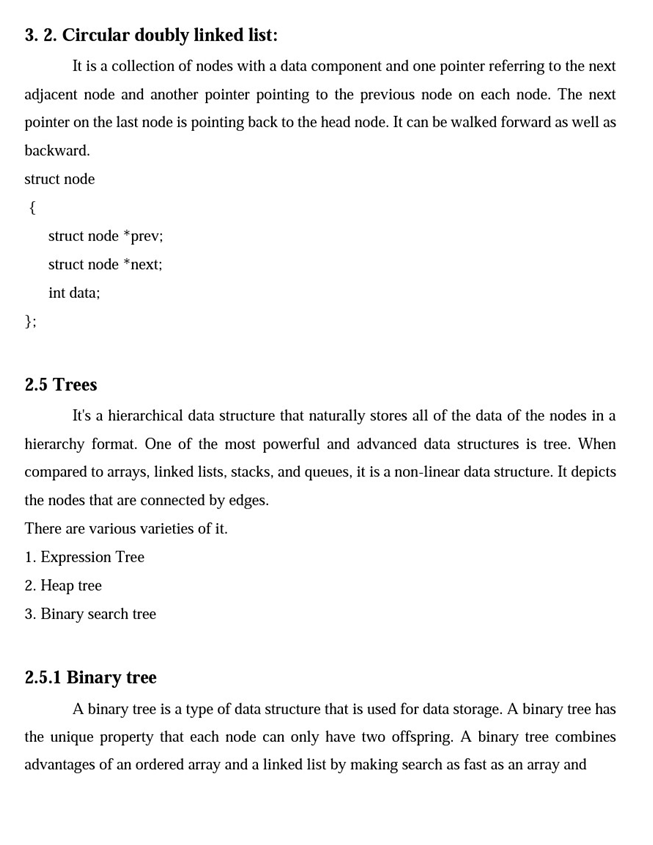
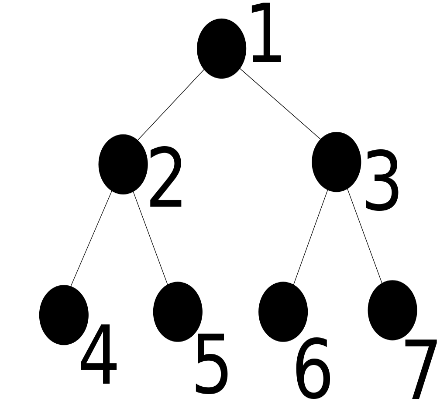
We must use all dynamic memory allocation functions such as malloc, calloc, realloc, and deallocation function free in linked lists (). The heap is used by all dynamic memory allocation functions to allocate a block of memory.

A linked list is a dynamic data structure made up of non-sequential groups of data items. Because of its appearance, a linked list is referred to as a linear data structure. A linked list, on the other hand, is a collection of items, each of which contains data and a pointer (or) link to the next item.

**The advantages of linked lists:**

1. Linked lists are dynamic, which means they can expand or contract during the execution of a programme.
2. Because memory is not pre-allocated in linked lists, it encourages efficient memory usage. 3. In linked lists, insertion and deletion might be easily carried over. 4. Linked lists can be used to create many complex applications. The disadvantages of a linked list are as follows:
3. It uses more memory since each node requires an extra pointer to store the following node's address.
4. Finding a certain item takes a significant amount of time.



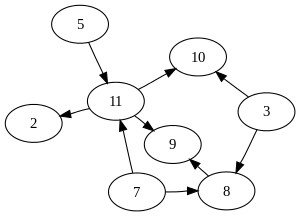


insertion and deletion operations as fast as a linked list

It is self-balancing Binary Search Tree where the difference of heights of left and right subtrees cannot be more than one for all the nodes. The below figure shows an example Tree that is an AVL Tree.

**2.6 Graphs:**

A graph is non-linear data structure consisting of both nodes and edges. The edges are lines or arcs that connect any of two nodes in the graph, and the nodes are also known as vertices. A Graph is described more precisely as: A Graph is made up of a finite number of vertices (or nodes) and a set of Edges that connect two nodes.



**Fig 2.6.1: Graph** example

**CHAPTER 3**

# **DESIGN:**

## **3.1 DESIGN GOALS**

The project is designed in such a way that all the fields are connected with each other and all the functions are called by main function. This makes it easier to store data.

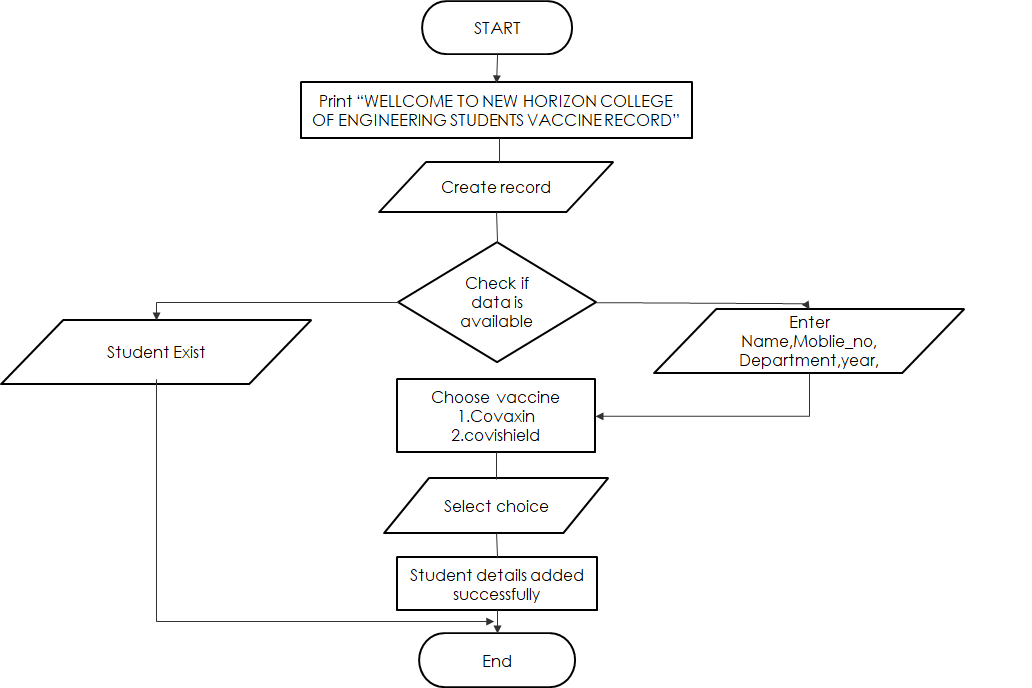
## **3.2 ALGORITHM / PSEUDOCODE**

1. Start
2. Input the number for case statement
   1. Case 1: create a node
   2. Case 2: add node
      1. Case 1:in the first
      2. Case 2:in the last
      3. Case 3:in the middle
   3. Case 3: display
      1. Case 1: Display from forward
      2. Case 2: Display from backward
   4. Case 4: search

This will print a message about the next vaccine

* 1. Case 5: Delete
     1. Case 1:in the first
     2. Case 2:in the last
     3. Case 3:in the middle
  2. End the main function

* 1. FLOWCHART



**CHAPTER 4**

**IMPLEMENTATION:**

**4.1 MODULE 1 CREATE A STRUCTURE**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

struct node

{

char student\_name[50];

int USN\_no;

char branch[20];

long long int student\_number;

char vaccine\_name[10];

char dose\_one\_date[20];

char dose\_two\_date[20];

struct node \*next;

struct node \*previous;

} \*start = NULL, \*end = NULL;

FILE \*file;

**4.2 MODULE 2 – CREATE\_STUDENTS\_RECORD()**

void create\_students\_record()

{

struct node \*new\_node, \*current;

int i, number\_of\_node;

printf("\n\n\n\n\n\n\t\t\tEnter Number of students's for Record: ");

scanf("%d", &number\_of\_node);

cls();

for(i = 1; i <= number\_of\_node; i++)

{

new\_node = (struct node \*)malloc(sizeof(struct node));

if(new\_node == NULL)

{

printf("\nMemory Does Not Created.\n");

exit(0);

}

else

{

file = fopen("student\_Record.txt","a+");

if(file == NULL)

{

printf("File does not create.\n");

}

else

{

printf("\n\n\t\t\tstudent's Details\n");

fprintf(file,"\n\n\t\t\tstudent's Details\n");

printf("\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

fprintf(file,"\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\n\n\tEnter student Name: ");

fflush(stdin);

gets(new\_node->student\_name);

fprintf(file,"\n\tstudent Name: %s", new\_node->student\_name);

printf("\n\t USN\_Number: ");

scanf("%d", &new\_node->USN\_no);

fprintf(file,"\n\tUSN\_no: %d",new\_node->USN\_no);

printf("\n\tEnter department name: ");

fflush(stdin)

gets(new\_node->branch);

fprintf(file,"\n\tdepartment Name: %s",new\_node->branch);

printf("\n\tStudent Phone Number: ");

scanf("%lld", &new\_node->student\_number);

fprintf(file,"\n\tStudent Phone Number: %lld",new\_node->student\_number);

printf("\n\tenter the vaccine name:");

printf("\n\t You can Enter :");

printf("\n\t 1. Covaxin ");

printf("\n\t 2. Covishield ");

fflush(stdin);

gets(new\_node->vaccine\_name);

fprintf(file,"\n\tvaccine name is %s",new\_node->vaccine\_name);

printf("\n\tenter the date of 1st vaccine:");

fflush(stdin);

gets(new\_node->dose\_one\_date);

fprintf(file,"\n\tDate is %s",new\_node->dose\_one\_date);

printf("\n\tenter the date of 2nd vaccine:");

fflush(stdin);

gets(new\_node->dose\_two\_date);

fprintf(file,"\n\tDate is %s",new\_node->dose\_two\_date);

fclose(file);

fopen("student\_Record.txt","a+");

}

new\_node->next = NULL;

new\_node->previous = NULL;

if(start == NULL && end == NULL)

{

start = new\_node;

end = new\_node;

current = new\_node;

}

else

{

current->next = new\_node;

new\_node->previous = current;

current = new\_node;

end = new\_node;

}

}

}

}

**4.3 MODULE 3 - ADD\_STUDENTS\_RECORD\_AT\_FIRST()**

void add\_students\_record\_at\_first()

{

struct node \*new\_node, \*current;

new\_node = (struct node \*)malloc(sizeof(struct node));

if(new\_node == NULL)

{

printf("\nMemory Does Not Created.\n");

exit(0);

}

else

{

file = fopen("student\_Record.txt","a+");

if(file == NULL)

{

printf("File does not create.\n");

}

else

{

printf("\n\n\t\t\tstudent's Details\n");

fprintf(file,"\n\n\t\t\tstudent's Details\n");

printf("\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

fprintf(file,"\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\n\n\tEnter student Name: ");

fflush(stdin);

gets(new\_node->student\_name);

fprintf(file,"\n\tstudent Name: %s", new\_node->student\_name);

printf("\n\t USN\_Number: ");

scanf("%d", &new\_node->USN\_no);

fprintf(file,"\n\tUSN\_no: %d",new\_node->USN\_no);

printf("\n\tEnter department name: ");

fflush(stdin);

gets(new\_node->branch);

fprintf(file,"\n\tdepartment Name: %s",new\_node->branch);

printf("\n\tStudent Phone Number: ");

scanf("%lld", &new\_node->student\_number);

fprintf(file,"\n\tStudent Phone Number: %lld",new\_node->student\_number);

printf("\n\tenter the vaccine name:");

printf("\n\t You can Enter :");

printf("\n\t 1. Covaxin ");

printf("\n\t 2. Covishield ");

fflush(stdin);

gets(new\_node->vaccine\_name);

fprintf(file,"\n\tvaccine name is %s",new\_node->vaccine\_name);

printf("\n\tenter the date of 1st vaccine:");

fflush(stdin);

gets(new\_node->dose\_one\_date);

fprintf(file,"\n\tDate is %s",new\_node->dose\_one\_date);

printf("\n\tenter the date of 2nd vaccine:");

fflush(stdin);

gets(new\_node->dose\_two\_date);

fprintf(file,"\n\tDate is %s",new\_node->dose\_two\_date);

fclose(file);

fopen("student\_Record.txt","a+");

}

}

new\_node->next = NULL;

new\_node->previous = NULL;

current = start;

new\_node->next = current;

current->previous = new\_node;

start = new\_node;

}

**4.4 MODULE 4 - ADD\_STUDENTS\_RECORD\_AT\_LAST()**

void add\_students\_record\_at\_last()

{

struct node \*new\_node, \*current;

new\_node = (struct node \*)malloc(sizeof(struct node));

if(new\_node == NULL)

{

printf("\nMemory Does Not Created.\n");

exit(0);

}

else

{

file = fopen("student\_Record.txt","a+");

if(file == NULL)

{

printf("File does not create.\n");

}

else

{

printf("\n\n\t\t\tstudent's Details\n");

fprintf(file,"\n\n\t\t\tstudent's Details\n");

printf("\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

fprintf(file,"\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\n\n\tEnter student Name: ");

fflush(stdin);

gets(new\_node->student\_name);

fprintf(file,"\n\tstudent Name: %s", new\_node->student\_name);

printf("\n\t USN\_Number: ");

scanf("%d", &new\_node->USN\_no);

fprintf(file,"\n\tUSN\_no: %d",new\_node->USN\_no);

printf("\n\tEnter department name: ");

fflush(stdin);

gets(new\_node->branch);

fprintf(file,"\n\tdepartment Name: %s",new\_node->branch);

printf("\n\tStudent Phone Number: ");

scanf("%lld", &new\_node->student\_number);

fprintf(file,"\n\tStudent Phone Number: %lld",new\_node->student\_number);

printf("\n\tenter the vaccine name:");

printf("\n\t You can Enter :");

printf("\n\t 1. Covaxin ");

printf("\n\t 2. Covishield ");

fflush(stdin);

gets(new\_node->vaccine\_name);

fprintf(file,"\n\tvaccine name is %s",new\_node->vaccine\_name);

printf("\n\tenter the date of 1st vaccine:");

fflush(stdin);

gets(new\_node->dose\_one\_date);

fprintf(file,"\n\tDate is %s",new\_node->dose\_one\_date);

printf("\n\tenter the date of 2nd vaccine:");

fflush(stdin);

gets(new\_node->dose\_two\_date);

fprintf(file,"\n\tDate is %s",new\_node->dose\_two\_date);

fclose(file);

fopen("student\_Record","a+");

}

}

new\_node->next = NULL;

new\_node->previous = NULL;

current = end;

current->next = new\_node;

new\_node->previous = current;

end = new\_node;

}

**4.5 MODULE 5 - ADD\_STUDENTS\_RECORD\_AT\_MIDDLE()**

void add\_students\_record\_at\_middle()

{

struct node \*new\_node, \*current, \*temp1, \*temp2;

int i,USN\_no ,position;

new\_node = (struct node \*)malloc(sizeof(struct node));

if(new\_node == NULL)

{

printf("\nMemory Does Not Created.\n");

exit(0);

}

else

{

file = fopen("student\_Record.txt","a+");

if(file == NULL)

{

printf("File does not create.\n");

}

else

{

printf("\n\n\t\t\tstudent's Details\n");

fprintf(file,"\n\n\t\t\tstudent's Details\n");

printf("\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

fprintf(file,"\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\n\n\tEnter student Name: ");

fflush(stdin);

gets(new\_node->student\_name);

fprintf(file,"\n\tstudent Name: %s", new\_node->student\_name);

printf("\n\t USN\_Number: ");

scanf("%d", &new\_node->USN\_no);

fprintf(file,"\n\tUSN\_no: %d",new\_node->USN\_no);

printf("\n\tEnter department name: ");

fflush(stdin);

gets(new\_node->branch);

fprintf(file,"\n\tdepartment Name: %s",new\_node->branch);

printf("\n\tStudent Phone Number: ");

scanf("%lld", &new\_node->student\_number);

fprintf(file,"\n\tStudent Phone Number: %lld",new\_node->student\_number);

printf("\n\tenter the vaccine name:");

printf("\n\t You can Enter :");

printf("\n\t 1. Covaxin ");

printf("\n\t 2. Covishield ");

fflush(stdin);

gets(new\_node->vaccine\_name);

fprintf(file,"\n\tvaccine name is %s",new\_node->vaccine\_name);

printf("\n\tenter the date of 1st vaccine:");

fflush(stdin);

gets(new\_node->dose\_one\_date);

fprintf(file,"\n\tDate is %s",new\_node->dose\_one\_date);

printf("\n\tenter the date of 2nd vaccine:");

fflush(stdin);

gets(new\_node->dose\_two\_date);

fprintf(file,"\n\tDate is %s",new\_node->dose\_two\_date);

fclose(file);

fopen("student\_Record.txt","a+");

}

}

new\_node->next = NULL;

new\_node->previous = NULL;

cls();

printf("Enter USN\_no before Insert Record: ");

scanf("%d", &USN\_no);

position = pos(USN\_no);

current = start;

for(i = 1; i <= (position - 1); i++)

{

current = current->next;

}

temp2 = current;

temp1 = current->previous;

temp1->next = new\_node;

new\_node->previous = temp1;

new\_node->next = temp2;

temp2->previous = new\_node;

}

**4.6 MODULE 6- DISPLAY\_STUDENTS\_RECORD\_FROM\_FORWARD()**

void display\_students\_record\_from\_forward()

{

struct node \*current;

current = start;

if(current == NULL)

{

printf("\n\n\n\n\n\n\n\t\t\tThere Are No Record In The List.\n");

}

else

{

while(current != NULL)

{

printf("\n\n\t\t\tstudent's Details\n");

printf("\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\n\n\tEnter student Name: %s",current->student\_name);

printf("\n\tUSN\_Number: %d",current->USN\_no);

printf("\n\tEnter department name: %s",current->branch);

printf("\n\tStudent Phone Number: %lld",current->student\_number);

printf("\n\tenter the vaccine name:%s",current->vaccine\_name);

printf("\n\tenter the date of 1st vaccine:%s",current->dose\_one\_date);

printf("\n\tenter the date of 2nd vaccine:%s",current->dose\_two\_date);

current = current->next;

printf("\n");

}

}

}

**4.7 MODULE 7- DISPLAY\_STUDENTS\_RECORD\_FROM\_BACKWARD()**

void display\_students\_record\_from\_backward()

{

struct node \*current;

current = end;

if(current == NULL)

{

printf("\n\n\n\n\n\n\n\t\t\tThere Are No Record In The List.\n");

}

else

{

while(current != NULL)

{

printf("\n\n\t\t\tstudent's Details\n");

printf("\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\n\n\tEnter student Name: %s",current->student\_name);

printf("\n\tUSN\_Number: %d",current->USN\_no);

printf("\n\tEnter department name: %s",current->branch);

printf("\n\tStudent Phone Number: %lld",current->student\_number);

printf("\n\tenter the vaccine name:%s",current->vaccine\_name);

printf("\n\tenter the date of 1st vaccine:%s",current->dose\_one\_date);

printf("\n\tenter the date of 2nd vaccine:%s",current->dose\_two\_date);

current = current->previous;

printf("\n");

}

}

}

**4.8 MODULE 8 - FUNCTIONALITY**

int pos(student\_id)

{

int position = 0;

struct node \*current;

current = start;

while(current != NULL)

{

position++;

if(student\_id == current->USN\_no)

{

return position;

}

current = current->next;

}

return -1;

}

int search\_students\_record(student\_id)

{

int position = 0;

struct node \*present;

present = start;

while(present != NULL)

{

position++;

if(student\_id == present->USN\_no)

{

printf("%s is vaccinated on %s",present->student\_name,present->vaccine\_name,present->dose\_one\_date);

printf("\n\n");

getch();

cls();

return position;

}

present= present->next;

}

return -1;

}

**4.9 MODULE 4 – DELETE\_STUDENTS\_RECORD()**

void delete\_students\_record()

{

struct node \*current, \*temp1, \*temp2;

int i, delet\_student\_id, position;

printf("\n\n\n\n\n\n\n\t\t\tEnter USN\_no for delete: ");

scanf("%d", &delet\_student\_id);

position = pos(delet\_student\_id);

current = start;

for(i = 1; i <= (position - 1); i++)

{

current = current->next;

}

if(current == start && current->previous == NULL)

{

current = current->next;

start = current;

current->previous = NULL;

printf("\n\n\n\n\n\n\n\t\t\t\nFirst USN\_no Delete Successfully.\n");

}

else if(current->next == NULL && current == end)

{

current = current->previous;

end = current;

current->next = NULL;

printf("\nLast USN\_no Delete Successfully.\n");

}

else

{

temp2 = current->next;

temp1 = current->previous;

temp1->next = temp2;

temp2->previous = temp1;

printf("\nDelete Successfully.\n");

}

}

int cls()

{

return system("cls");

}

**4.1.1 MODULE 4 – WelcomeMessage()**

void welcomeMessage()

{

printf("\n\n\n\n\n");

printf("\n\t\t\t \*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*\n");

printf("\n\t\t\t =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=");

printf("\n\t\t\t = WELCOME =");

printf("\n\t\t\t = TO =");

printf("\n\t\t\t = New Horizon College of Engineering =");

printf("\n\t\t\t = Student Vaccine Registration Record =");

printf("\n\t\t\t = MANAGEMENT =");

printf("\n\t\t\t = SYSTEM =");

printf("\n\t\t\t =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=");

printf("\n\t\t\t \*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*\n");

printf("\n\n\n\t\t\t Enter any key to continue.....");

getch();

}

**4.1.2 MODULE 4 – MAIN\_FUNCTION()**

int main()

{

system("color B");

welcomeMessage();

getch();

int c, position, student\_id, insert\_choice, display\_choice;

while(1)

{

printf("\n\t\t\t\t\tStudent's Vaccine record\n");

printf("\t\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

printf("\n\n");

printf("------------------------------------------------------------------------------------------");

printf("\n\n");

printf("\t\t\t\t1. Create Record.\n");

printf("\t\t\t\t2. Add Record.\n");

printf("\t\t\t\t3. Display Record.\n");

printf("\t\t\t\t4. Search Record.\n");

printf("\t\t\t\t5. Delete Record.\n");

printf("\t\t\t\t6. Exit.\n\n");

printf("------------------------------------------------------------------------------------------");

printf("\n\t\t\tPlease Enter your choice: ");

scanf("%d", &c);

printf("\n");

cls();

switch(c)

{

case 1: create\_students\_record();

cls();

printf("\n\n\n\n\n\n\n\t\t\tRecord Created Successfully.");

getch();

cls();

break;

case 2: while(1)

{

printf("\n\n\n\n\n");

printf("---------------------------------------------------------------------------------");

printf("\n\n");

printf("\t\t\t\t1. Add Record At First.\n");

printf("\t\t\t\t2. Add Record At Last.\n");

printf("\t\t\t\t3. Add Record At Middle.\n");

printf("\t\t\t\t4. Back To Main Program.\n\n");

printf("---------------------------------------------------------------------------------");

printf("\n\n\n\t\t\tPlease Enter your choice: ");

scanf("%d", &insert\_choice);

printf("\n\n");

cls();

switch(insert\_choice)

{

case 1: add\_students\_record\_at\_first();

cls();

printf("\n\n\n\n\n\n\n\t\t\tRecord Added successfully.");

getch();

cls();

break;

case 2: add\_students\_record\_at\_last();

cls();

printf("\n\n\n\n\n\n\n\t\t\tRecord Added successfully.");

getch();

cls();

break;

case 3: add\_students\_record\_at\_middle();

cls();

printf("\n\n\n\n\n\n\n\t\t\tRecord Added successfully.");

getch();

cls();

break;

case 4: main();

break;

default:printf("\n\n\n\n\n\n\n\t\t\t\tIncorrect input!!!");

printf("\n\t\t\tPlease Enter Correct Key.");

printf("\n\t\t\t\tOr Enter 4 to Main menu.\n");

getch();

cls();

}

}

getch();

cls();

break;

case 3: while(1)

{

printf("\n\n\n\n\n");

printf("----------------------------------------------------------------------------------");

printf("\n\n");

printf("\t\t\t\t1. Display From Forward.\n");

printf("\t\t\t\t2. Display From Backward.\n");

printf("\t\t\t\t3. Back To Main Program.\n\n");

printf("---------------------------------------------------------------------------------");

printf("\n\n\n\t\t\tPlease Enter your choice: ");

scanf("%d", &display\_choice);

printf("\n\n");

cls();

switch(display\_choice)

{

case 1: display\_students\_record\_from\_forward();

getch();

cls();

break;

case 2: display\_students\_record\_from\_backward();

getch();

cls();

break;

case 3: main();

break;

default:printf("\n\n\n\n\n\n\n\t\t\t\tIncorrect input!!!");

printf("\n\t\t\tPlease Enter Correct Key.");

printf("\n\t\t\t\tOr Enter 3 to Main menu.\n");

getch();

cls();

}

}

getch();

cls();

break;

case 4: printf("\n\n\n\n\n\n\t\t\tEnter USN\_no: ");

scanf("%d",&student\_id);

cls();

position = search\_students\_record(student\_id);

if(position == -1)

{

printf("\n\n\n\n\n\n\t\t\tThis USN\_no is not in the Record.\n");

getch();

cls();

}

else

{

printf("\n\n\n\n\n\n\t\t\tThe Position of this Record is at Number %d.\n", position);

getch();

cls();

}

break;

case 5: delete\_students\_record();

getch();

cls();

break;

case 6: exit(1);

break;

default:printf("\n\n\n\n\n\n\n\t\t\t\tIncorrect input!!!");

printf("\n\t\t\tenter correct key");

printf("\n\t\t\t\telse Enter 6 to Exit.\n");

getch();

cls();

}

}

}

**CHAPTER 5**

**RESULTS :**

