AIM

1. Implement a C program to display Fibonacci series using recursion.  
2. Implement a C program to display factorial of a number using recursion.  
3. Implement a C program to display array in reverse and calculate the items in an array using recursion.  
4. Implement a C program to assign members to a structure and display it.(Use example of your choice).  
5. Display shopping list using structures accessed with its pointer.

THEORY

**RECURSIVE FUCNTION:** A function that calls itself is called a recursive function.

Recursive call always leads to an infinite loop. So, provision must be mase to get outside this infinite loop.

Recursion are mainly of two types depending on weather a function calls itself from within itself weather two function call one another mutually. The former is called direct recursion and t latter is called indirect recursion. Thus, the two types of recursion are:

1. Direct recursion
2. Indirect recursion

|  |  |
| --- | --- |
| DIRECT RECURSION | INDIRECT RECURSION |
| In the direct recursion, only one function is called by itself. | In indirect recursion more than one function are by the other function and number of times. |
| Int num()  {  Num()  } | Int num(){  Sum()  }  Int sum()  {  Num()  } |

Linear recursion

It is the most commonly used recursion, where a function calls itself in simple manner and a terminating condition is used to terminate the recursion. Forwarding recursion is called winding and getting the control back to the caller is called unwinding.

### Tail recursion

It is the recursion where recursive function is called at the end of recursive function.

### Mutual recursion:

Calling two or more functions mutual is called mutual recursion. Say for example, if function A is calling B and function B is calling A recursively then it is said that, they are in mutual recursion.

### Nested recursion:

When a recursive method has a parameter defined in terms of itself then it is called nested recursion

**STRUCTURE** is a collection of dissimilar elements(usually) stored in adjacent locations. They are also known as User-Defined data types.

Syntax

**struct structure\_name**

**{**

**int a;**

**char b;**

**float c;**

**} e1, e2;**

Where struct is a keyword,

a, b, c are the structure elements

e1, e2 are the structure variables

Uses of Structures

1. Database Management
2. Interaction with Mouse, etc.

To access structure elements using structure pointer, use -> operator.

Struct emp e;

Struct emp \*p;

p=&e;

printf(“%%s %d %f”,p->name,p->age,p->salary);

PSEUDO CODE

1)

1. START
2. INPUT N
3. FOR i=0…..N
4. J= FIBO(N)
5. PRINT J
6. END

INT FIBO(INT N)

1. IF N==0
2. RETURN(0)

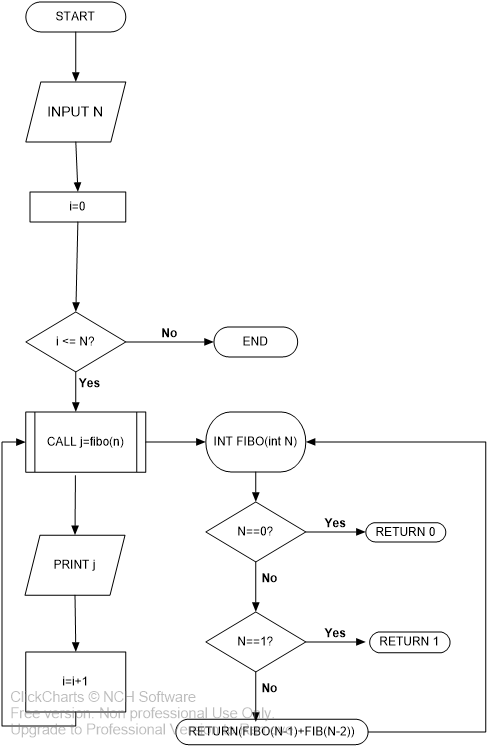
2. ELSE if N==1

1.RETURN(1)

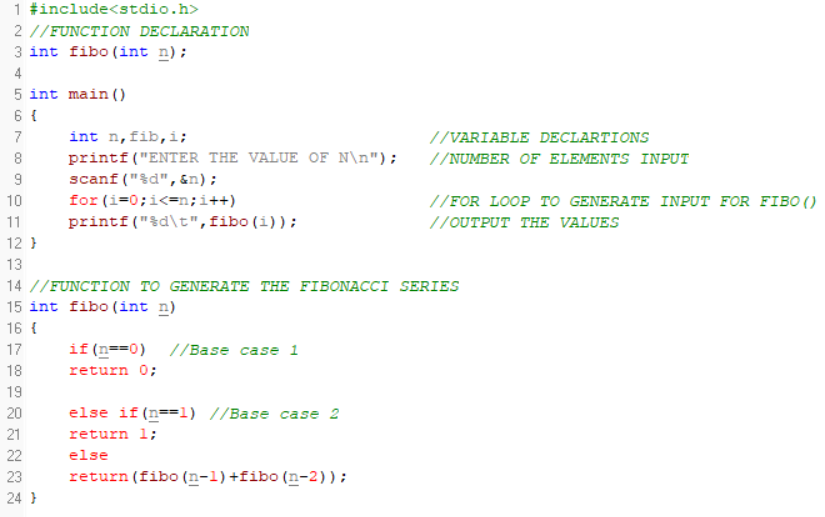
3. ELSE

1. RETURN(FIBO(N-1)+FIBO(N-2))

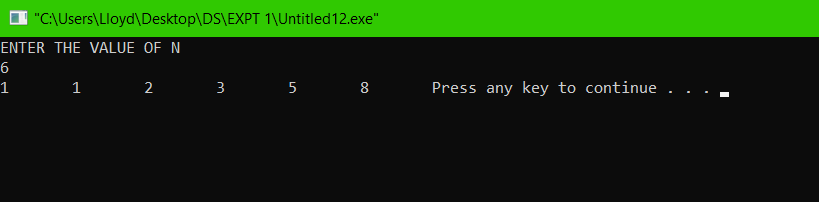
FLOW CHART



SOURCE CODE



OUTPUT



2)

PSEUDO CODE

1. START

2. INPUT N

3. PRINT FACT(N)

4. END

INT FACT(X)

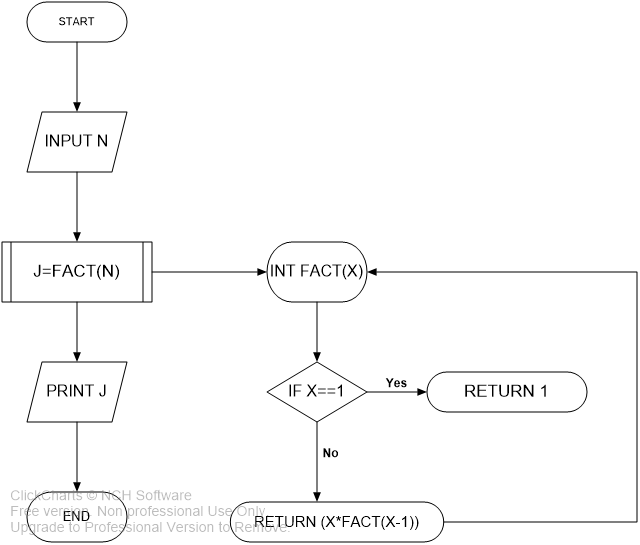
1. IF X==1

1. RETURN 1

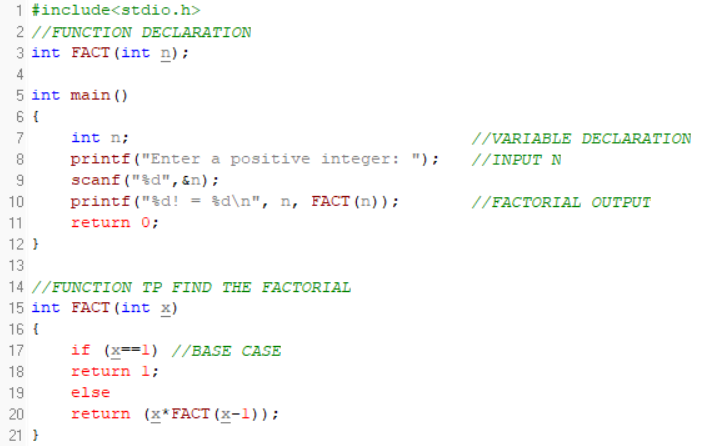
2. ELSE

1. RETURN(X\*FACT(X-1))

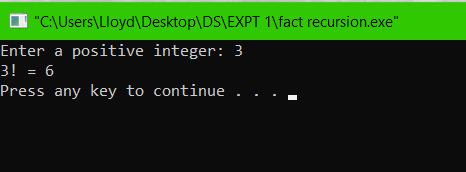
FLOWCHART



SOURCE CODE



OUTPUT:



3)

1. START

2. INPUT NUMBER OF ELEMENTS

3. INPUT ELEMEMTS INTO THE ARRAY

4. PRINT THE ARRAY BEOFRE REVERSING

5. CALL reverse(array,0,n-1)

6. PRINT THE ARRAY AFTER REVERSING

7. PRINT THE SUM

Void reverse(int arr[], int a, int b)

1. IF a>b

1.RETURN 0;

2. ELSE

1. temp=arr[a]

2. arr[a]=arr[b]

3. arr[b]=temp

4. reverse(arr,a+1,b-1)

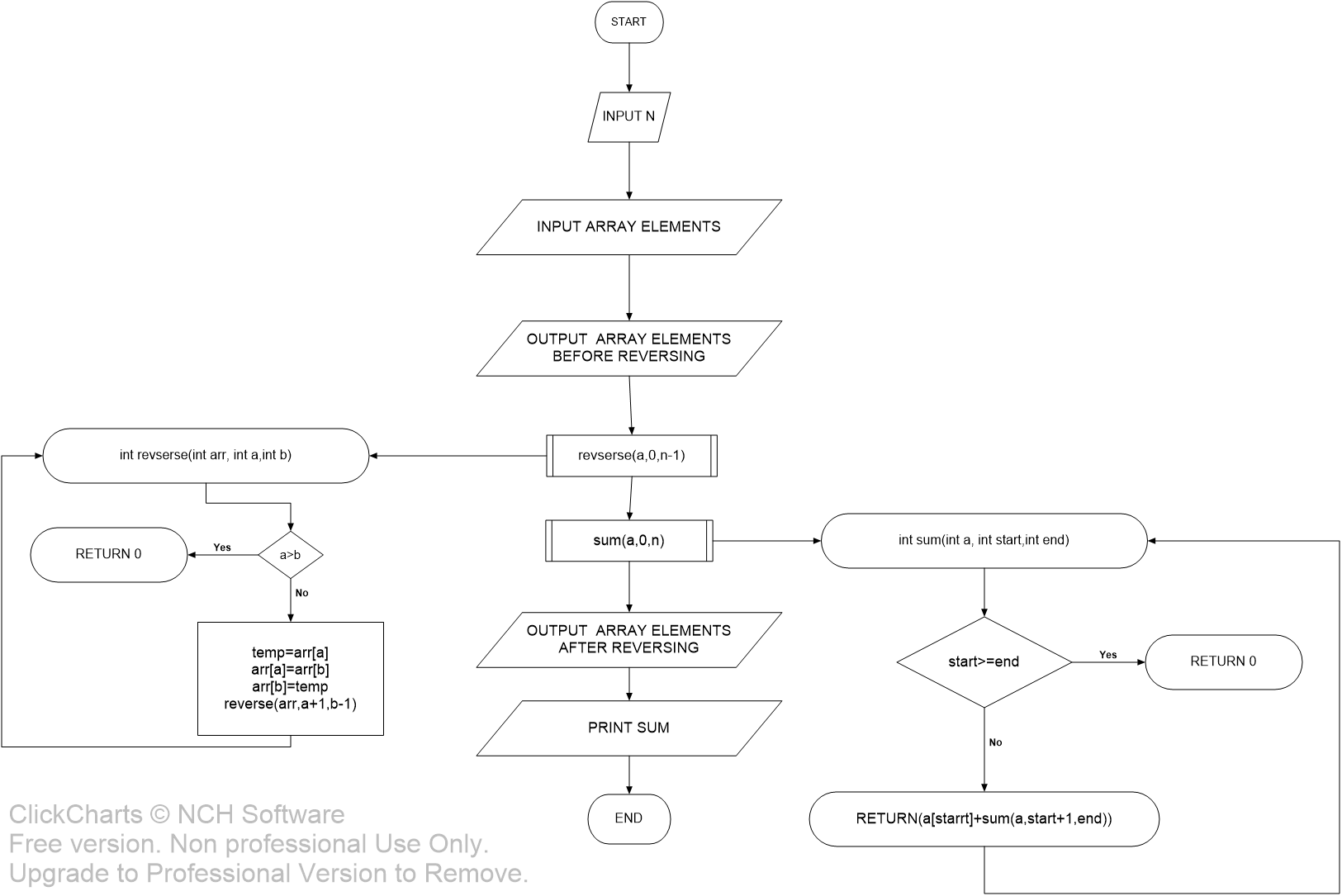
Void sum(int a[], int start, int end)

1. IF start>=end

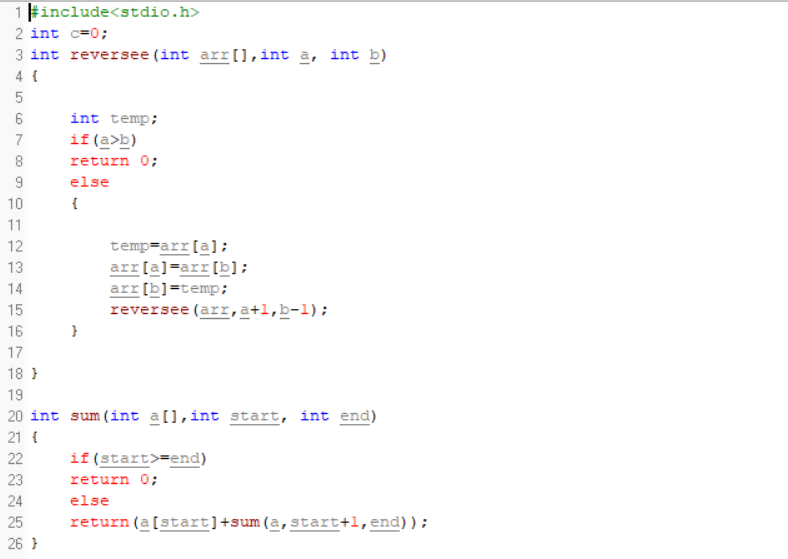
RETURN 0

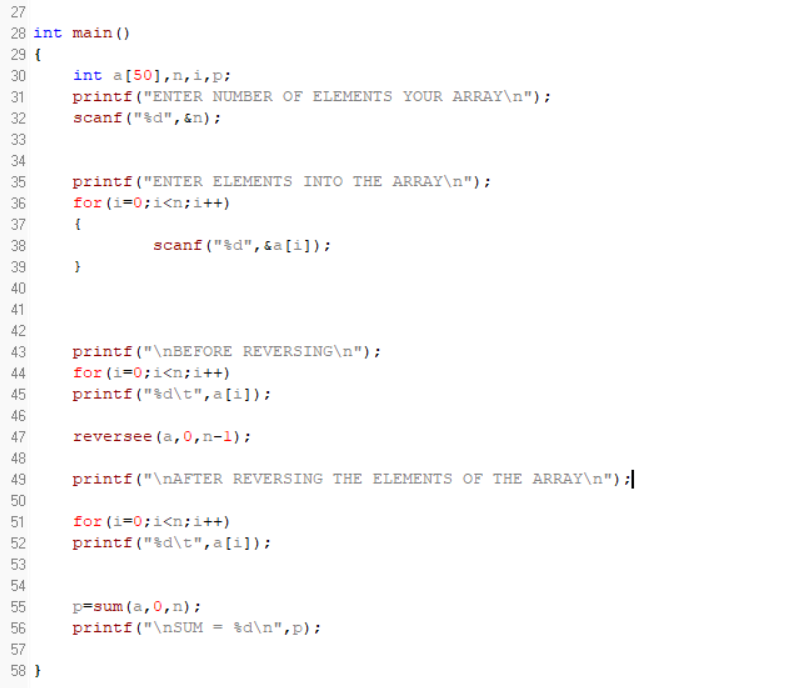
2. ELSE

RETURN(a[start]+sum(a,start+1,end)



SOURCE CODE





4)

PSEUDO CODE

1. START

2. DEFINE A STRUCTURE VACCIINE

1. char name[30]

2. int age

3. char gender[6]

4. char vaccineName[20]

5. int d1,m1,y1

6. int d2,m2,y2

3. INPUT N

4. DECLARE A POINTER p1 OF TYPE VACCINE AND DYNAMICALLY RESERVE MEMORY FOR N PERSONS

5. CALL input(p1)

6. CALL output(p1)

7.END

void output(vaccine p[])

1. i=0

2. FOR i < N

1. INPUT p1[i].name, p1[i].age, p1[i].gender, p1[i].vaccineName ,p1[i].d1,p1[i].m1,p1[i].y1, p1[i].d2,p1[i].m2,p1[i].y2

2. i++

void output(vaccine p[])

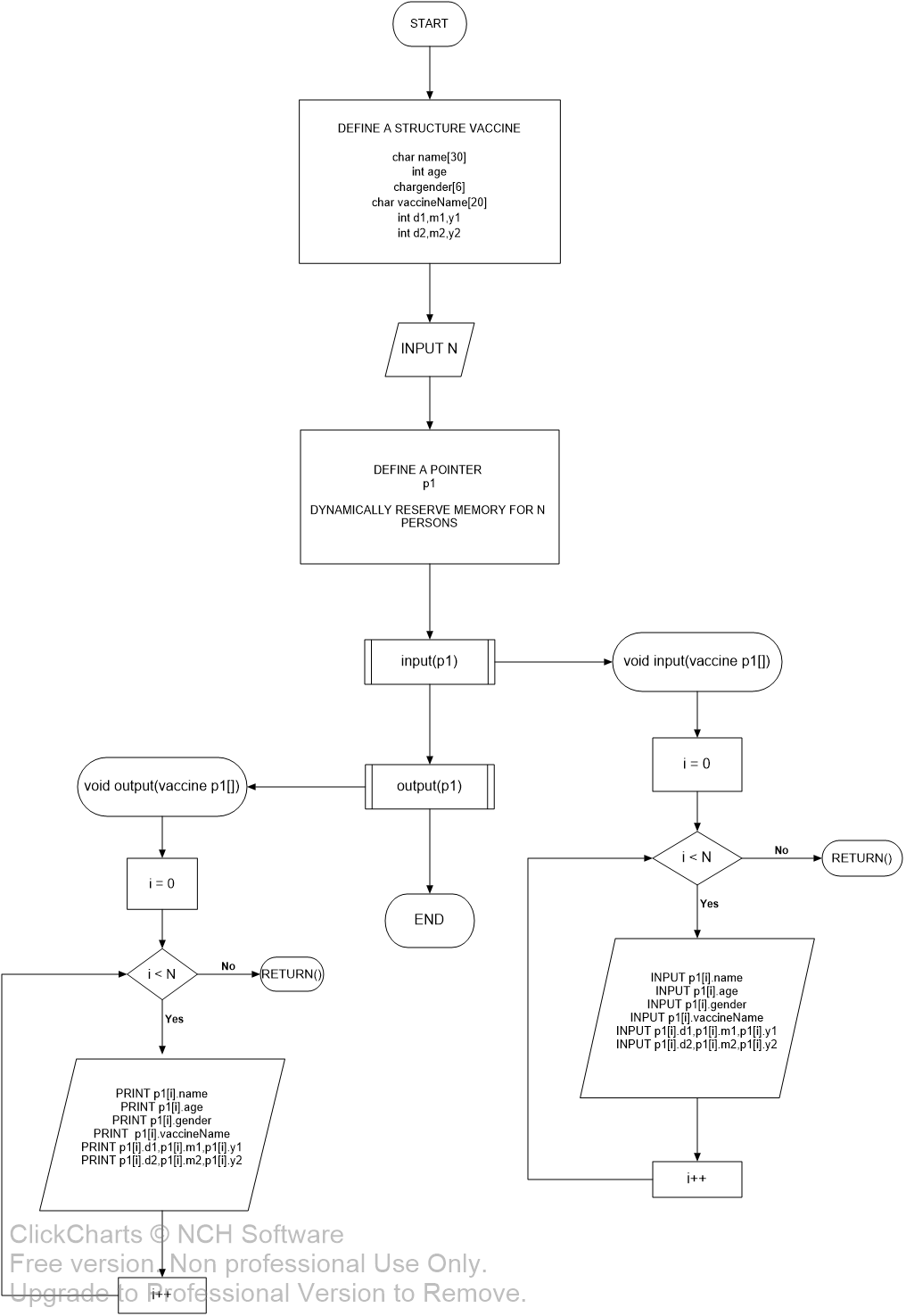
1. i=0

2. FOR i<N

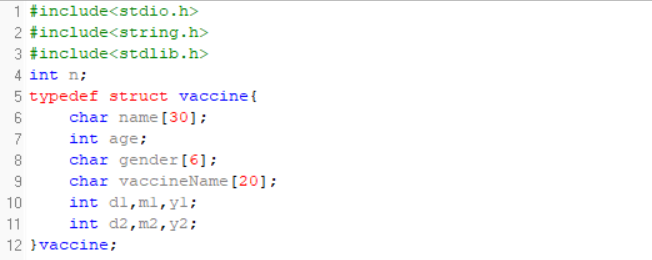
1. PRINT p1[i].name, p1[i].age, p1[i].gender, p1[i].vaccineName ,p1[i].d1,p1[i].m1,p1[i].y1, p1[i].d2,p1[i].m2,p1[i].y2

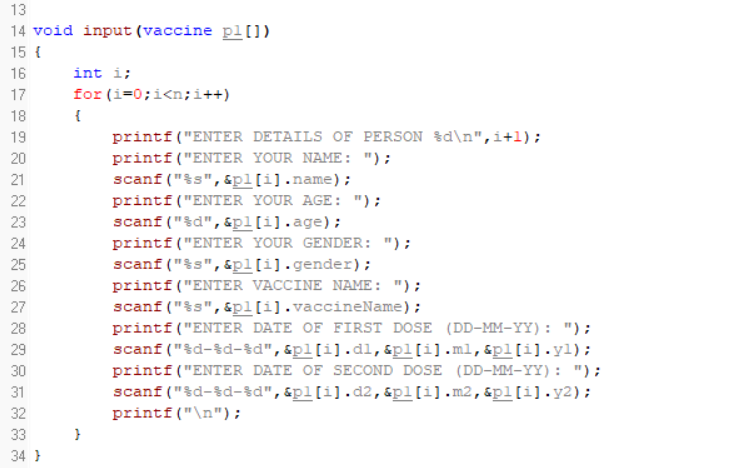
2. i++

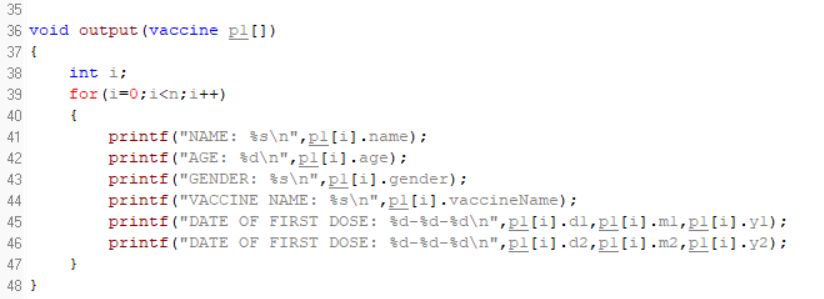
FLOWCHART

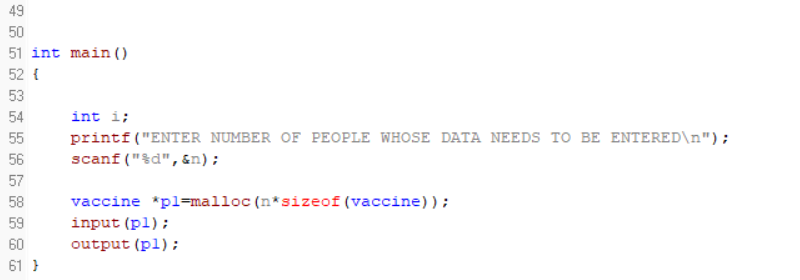


SOURCE CODE

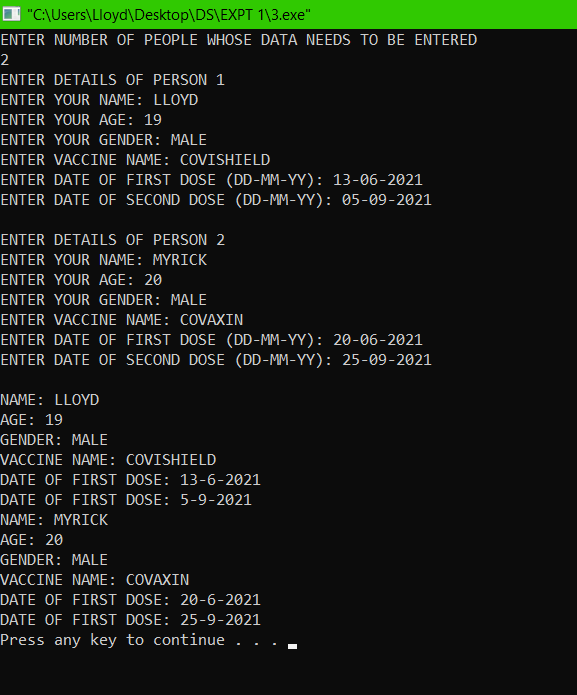








OUTPUT:



5)

1. START

2. DEFINE A STRUCTURE VACCIINE

1. char item[

2. int quantity

3. char price

3. INPUT N

4. DECLARE A POINTER p1 OF TYPE VACCINE AND DYNAMICALLY RESERVE MEMORY FOR N PERSONS

5. CALL inputElements(shpList \*list)

6. CALL outputList(shpList \*list)

7.END

Void inputElements(shpList \*list)

1. for i=0,1,2,…n

1.INPUT list->item, list->quantity, list->price

2. total=total+( (list->quantity) \* (list->price) )

3. i = i + 1

4. list++

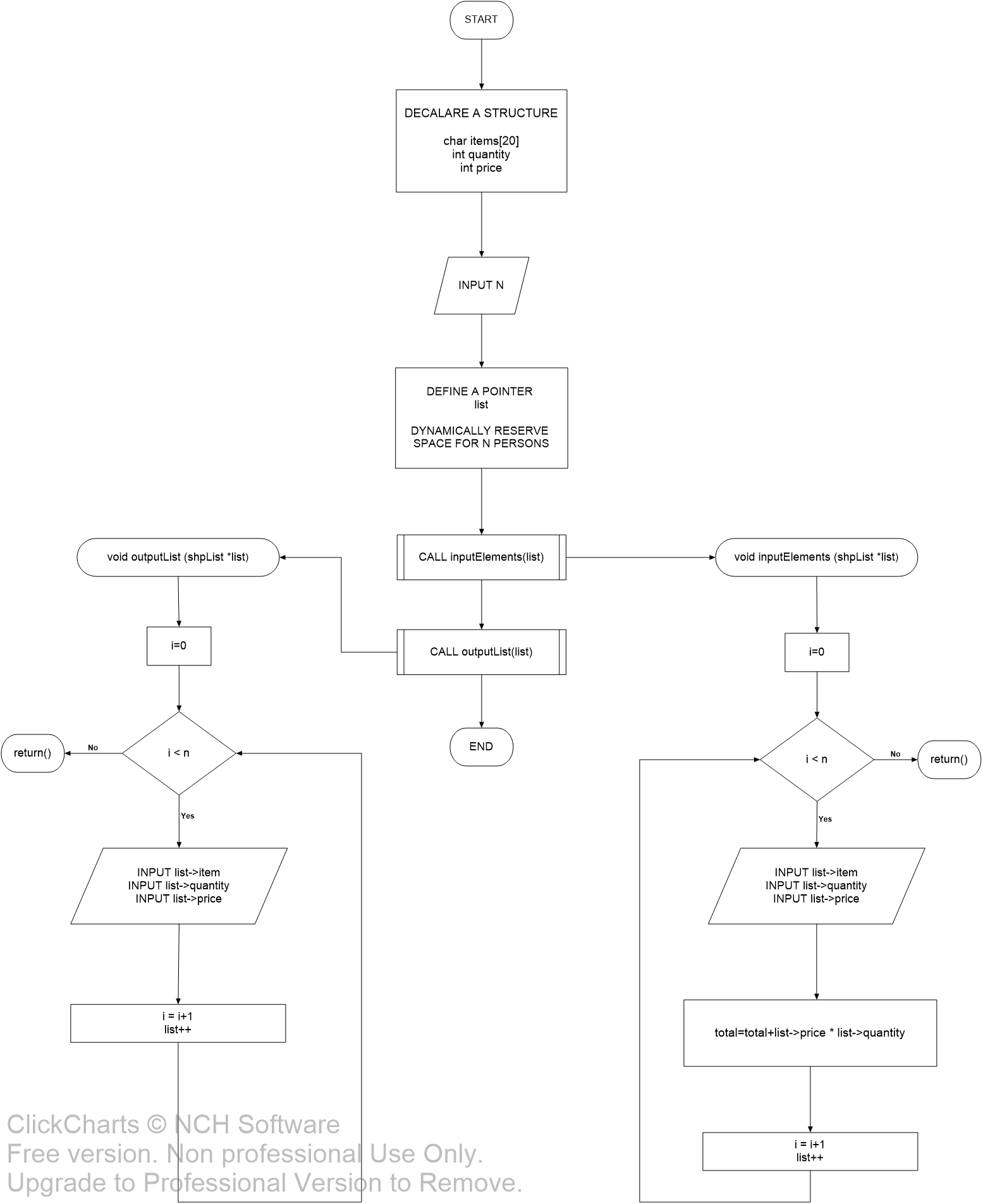
Void outputList(shpList \*list)

1. for i=0,1,2,…n

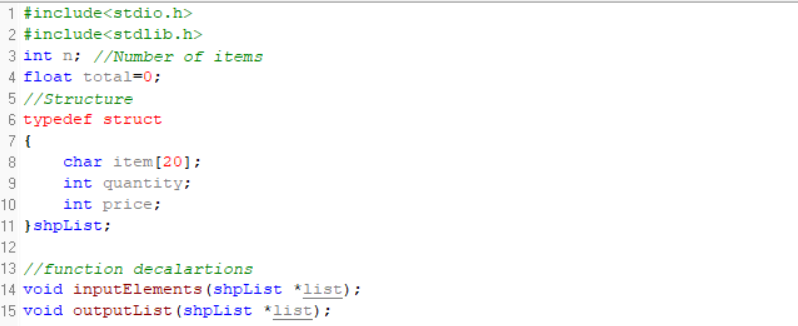
1.OUTPUT list->item, list->quantity, list->price

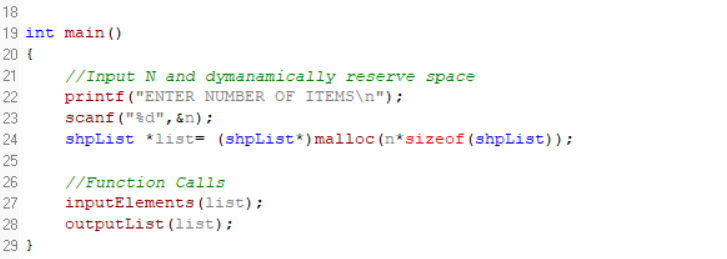
2. i = i + 1

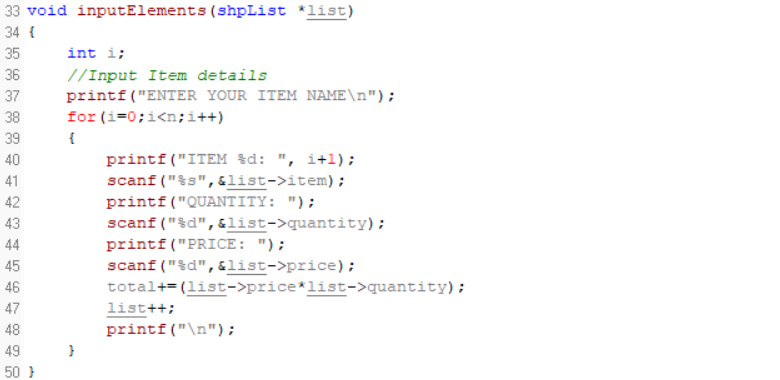
3. list++

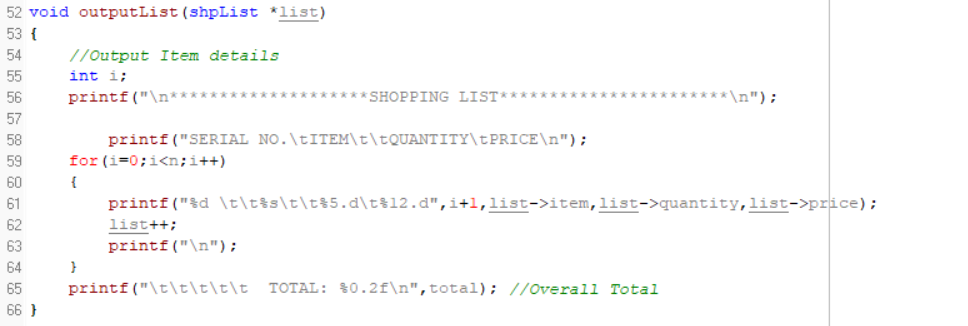


SOURCE CODE

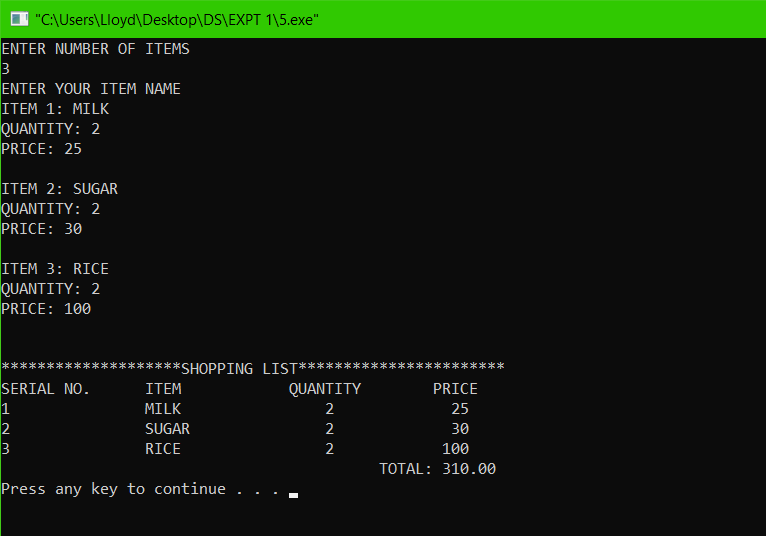








OUTPUT



CONCLUSION AND FINDING

The given problem statements were successfully compiled and executed.