Experiment No-1

Experiment Name: Write a C program to calculate the average of N numbers using arrays.

Objective:

- To write a C program that takes n integer inputs from the user and stores them in an array.
- To calculate the sum of all input numbers using a loop structure.
- To find the average value of the numbers using the sum and count.
- To develop logical thinking for using arrays and loop control structures effectively.
- To improve understanding of type casting in arithmetic expressions in C.

Problem analysis:

The given problem requires calculating the average of a set of numbers provided by the user. First, the program should take the total number of elements (n) as input. Then, it uses a loop to collect n individual numbers into an array. Another loop is used to compute the total sum of all array elements. After collecting all values and calculating the sum, the average is obtained by dividing the total sum by n. To ensure accuracy, type casting is used to convert the result into a float during division. Finally, the average is displayed with two decimal precision. The program should be able to handle a flexible number of inputs and ensure correct computation using simple logic and control statements.

Input variable	Processing variable	Output variable	Header file
arr[n]	Sum n	average(float)	#include<stdio.h>>

Algorithm:

Step1: start

Step2: declare an array and necessary variables (n, sum, average)

Step3: input the number of elements n

Step4: use a loop to input n numbers into the array

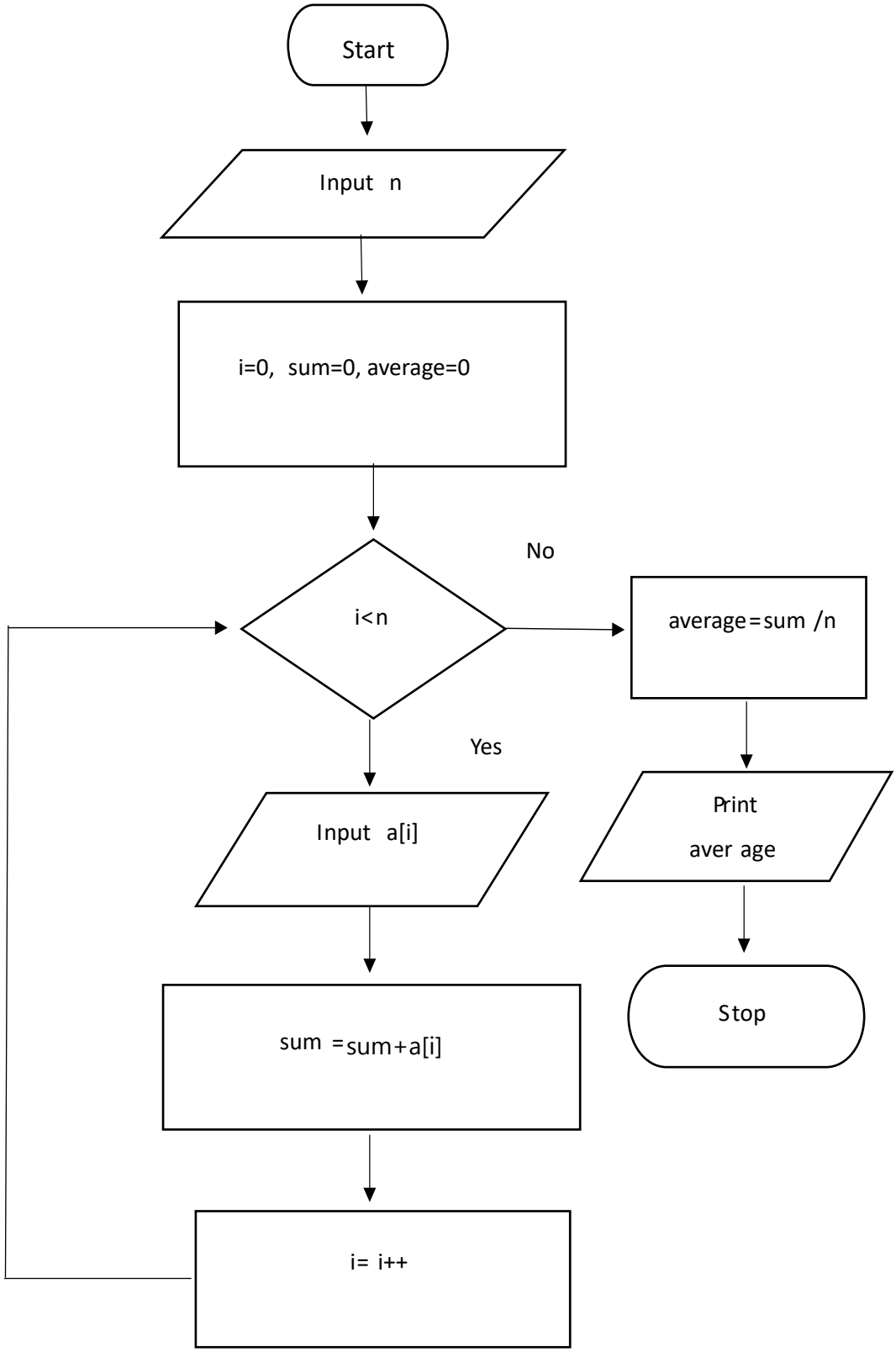
Step5: use another loop to calculate the sum

Step6: calculate the average = sum/n

Step7: display the average

Step8: stop

Flowchart:


Start
Input n
i=0, sum=0, average=0
No
i<n
average=sum /n
Yes
Print
aver age
Input a[i]
Stop
sum =sum+a[i]
i= i++

-

Source code:

```c
#include <stdio.h>

int main() {
    int n, i;
    float sum = 0.0, average;
    float arr[100];

    printf("Enter the number of elements: ");
    scanf("%d", &n);

    printf("Enter %d numbers:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%f", &arr[i]);
        sum += arr[i];
    }

    average = sum / n;
    printf("Average = %.2f\n", average);

    return 0;
}
```

```
D:\lab6\bin\Debug\lab6.exe
Enter the number of elements: 5
Enter 5 numbers:
10
20
30
40
50
Average = 30.00

Process returned 0 (0x0)   execution time : 22.516 s
Press any key to continue.
```

Discussion:

In this program, array indexing and loops are used efficiently to manage multiple inputs from the user. The use of a for loop ensures that all numbers are collected and added to the total sum sequentially. Type casting to float while calculating the average prevents loss of decimal precision. The code is simple, clear, and avoids unnecessary complexity, making it ideal for beginners in C programming. From this lab task, it is observed that arrays and loops are powerful tools for handling repetitive input/output and arithmetic operations. Overall, the program successfully demonstrates the method of calculating the average of user-defined numbers, and it reinforces important concepts such as arrays, loops, and arithmetic operations in C.

Experiment No-2

Experiment Name: Write a C program to sort an array in ascending order using bubble sort algorithm.

Objective:

- To write a C program that sorts a given array in ascending order using the bubble sort algorithm.

Problem analysis:

The goal of this program is to sort a list of numbers in ascending order using the Bubble Sort algorithm. Bubble Sort repeatedly compares adjacent elements and swaps them if they are in the wrong order. This process continues through multiple passes until the entire array is sorted. It uses two loops: the outer loop controls the number of passes, and the inner loop handles the comparisons and swaps. Though not efficient for large inputs, this algorithm is simple and useful for understanding the basic concept of sorting through iterations and conditions.

Input variable	Processing variable	Output variable	Header file
arr[n]	I, j=loop counter temp=temporary	arr[]	<stdio.h>

Algorithm:

Step1: Start

Step:2 Input the number of elements n

Step:3 Input n elements into an array

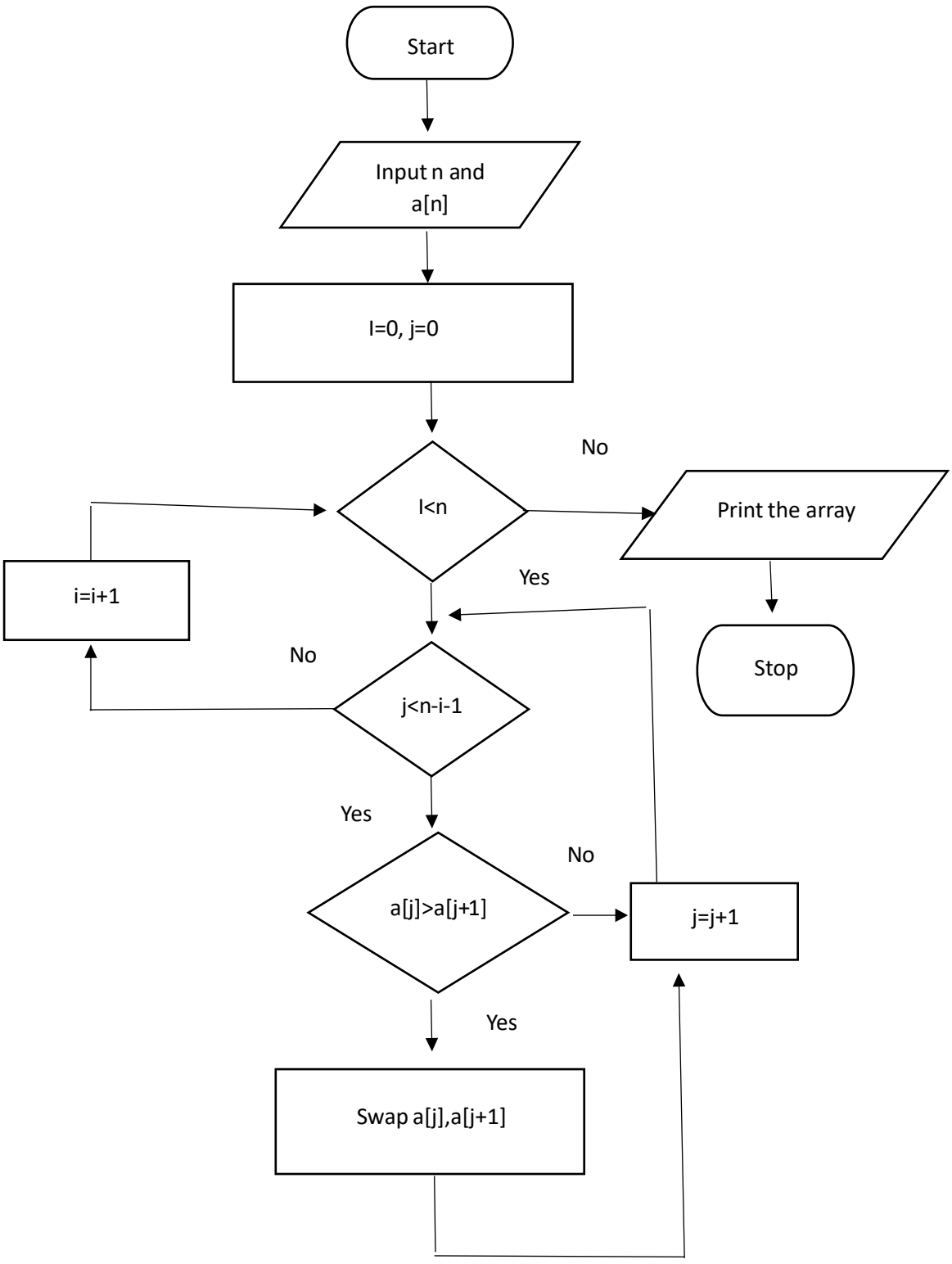
Step4: Use two nested loops:

- Outer loop from I = 0 to n-1
- Inner loop from j = 0 to n-i-1
- If arr[j] > arr[j+1], swap the value

Step5: After sorting, print the array

Step6: End

Flowchart:


Start
Input n and a[n]
I=0, j=0
No
I<n
Print the array
Yes
i=i+1
Stop
No
j<n-i-1
Yes
No
a[j]>a[j+1]
j=j+1
Yes
Swap a[j],a[j+1]

Source code:

```c
#include <stdio.h>
int main() {
    int arr[100], n, i, j, temp;

    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter %d elements:\n", n);
    for(i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    for(i = 0; i < n-1; i++) {
        for(j = 0; j < n-i-1; j++) {
            if(arr[j] > arr[j+1]) {
                temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }

    printf("Sorted array in ascending order:\n");
    for(i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

Output:

```
D:\lab6\bin\Debug\lab6.exe
Enter number of elements: 6
Enter 6 elements:
10
5
20
15
30
25
Sorted array in ascending order:
5 10 15 20 25 30
Process returned 0 (0x0)   execution time : 26.898 s
Press any key to continue.
```

Discussion:

In this program, the Bubble Sort algorithm was applied to sort an array in ascending order. The logic was implemented using nested loops and conditional checks to perform necessary swaps. The flowchart and algorithm helped visualize the step-by-step process. The output confirmed that the array was sorted correctly. While Bubble Sort is not the fastest algorithm, it is easy to understand and ideal for learning how sorting works. This exercise improved understanding of loop structures and array handling in C programming.