Alden Ardiwinata Putra
06 – 2501977286 – LF01

STUDY CASE COMP6047001

ALGORITHM AND PROGRAMMING

Problem Number 1

Narrative explanation:

To determine which same value element(s) is the closest to the centroid, we must search the surrounding coordinates that may contain the same value element. I wrote my code to search all same value elements in the array, calculate their distance to the centroid, find the shortest distance, then display the coordinates based on the shortest distance calculated before.

```
int N;
do //validating input, array size must be gr
{
    printf("Matrix Size (N): ");
    scanf("%d", &N);
    getchar();

    if(N <= 2)
    {
        printf("Invalid! Note: (N > 2)\n");
    }
    printf("\n");
}
while(N <= 2);
```

In this section, I use the *Do-While* loop to make sure that the matrix's size is greater than 2 as stated in the question. If the *'scanf'* function reads a value less than 2, then the program will display an invalid message and ask the user to re-input the matrix's size. The loop ends if a user inputs the desirable value (*while N > 2*).

```
int array[N][N];
int i, j;
printf("Insert matrix's elements:\n");
for(i = 0; i < N; i++) //inputting arra
{
    for(j = 0; j < N; j++)
    {
        scanf("%d", &array[i][j]);
    }
}
```

After the size value is validated, the program will ask the user to fill in the array. We will fill the array later based on the given sample input. Integer i represents the row and integer j represents the column. Iteratively they read user input into 'array[i][j].'

```
int C; //calculating centroid coordina
if(N % 2 == 0) //if size is an even nu
{
    C = N/2; //formula: (N / 2) + 1 -
}
else //if size is an odd number
{
    C = ((N + 1)/2) - 1; //subtract by
}
printf("Centroid at (%d,%d)\n", C, C);
int centroid = array[C][C]; //assignin
```

Centroid is the center coordinate. We need to divide the row and column length or simply the matrix's size by half. Thus, the formula differs by odd and even numbers. Because the index started from 0,0 at the top leftmost coordinates, we need to subtract by 1.

Alden Ardiwinata Putra
06 – 2501977286 – LF01

```
int count = 0;
int rekamBaris[N*N + 1], rekamKolom[N*N + 1];
for(i = 0; i < N; i++)
{
    for(j = 0; j < N; j++)
    {
        if(array[i][j] == centroid && (i != C || j != C))
        {
            rekamBaris[count] = i;
            rekamKolom[count] = j;
            count++;
            //printf("Same value coordinate(s): %d,%d\n",
        }
    }
}
```

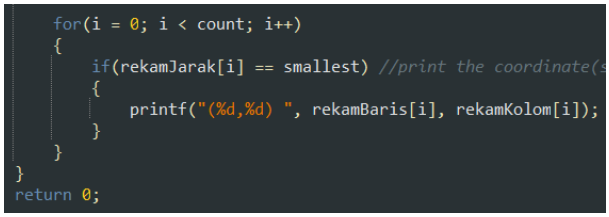
In this section, the program will search the same value element, the amount of same element (*int count*), and saving its coordinates into two arrays: *rekamBaris*, *rekamKolom* (excluding the centroid), each for row and column (I set the arrays sizes to N*N+1, because that is the maximum possible number of all points in a 2d array with a size of N). I set the if condition to be true if *array[i][j] == centroid && (i != C || j != C).* The OR statement will only be false if both conditions are false. With this precaution, the if statement will be not executed when the for loop reach the centroid position (both i and j cannot be the same as C at a single time). If the OR statement true, the AND statement will compare if such a position contains the same value as centroid's value.

```
printf("\nNearest same elements is at: ");
if(count == 0) //if there is no same value
{
    printf("no nearest element.\n");
}
```

If *count* is zero, it means there is no same value element detected, then the program should display "no nearest element."

```
else
{
    //oversimplified distance math for
    int rekamJarak[N*N + 1];
    for(i = 0; i < count; i++)
    {
        int A = C - rekamBaris[i];
        int B = C - rekamKolom[i];

        int jarak = A*A + B*B; //befor
        rekamJarak[i] = jarak; //save
    }

    int smallest = rekamJarak[0];
    for(i = 0; i < count; i++) //searc
    {
        if(rekamJarak[i] < smallest)
        {
            smallest = rekamJarak[i];
        }
    }
```

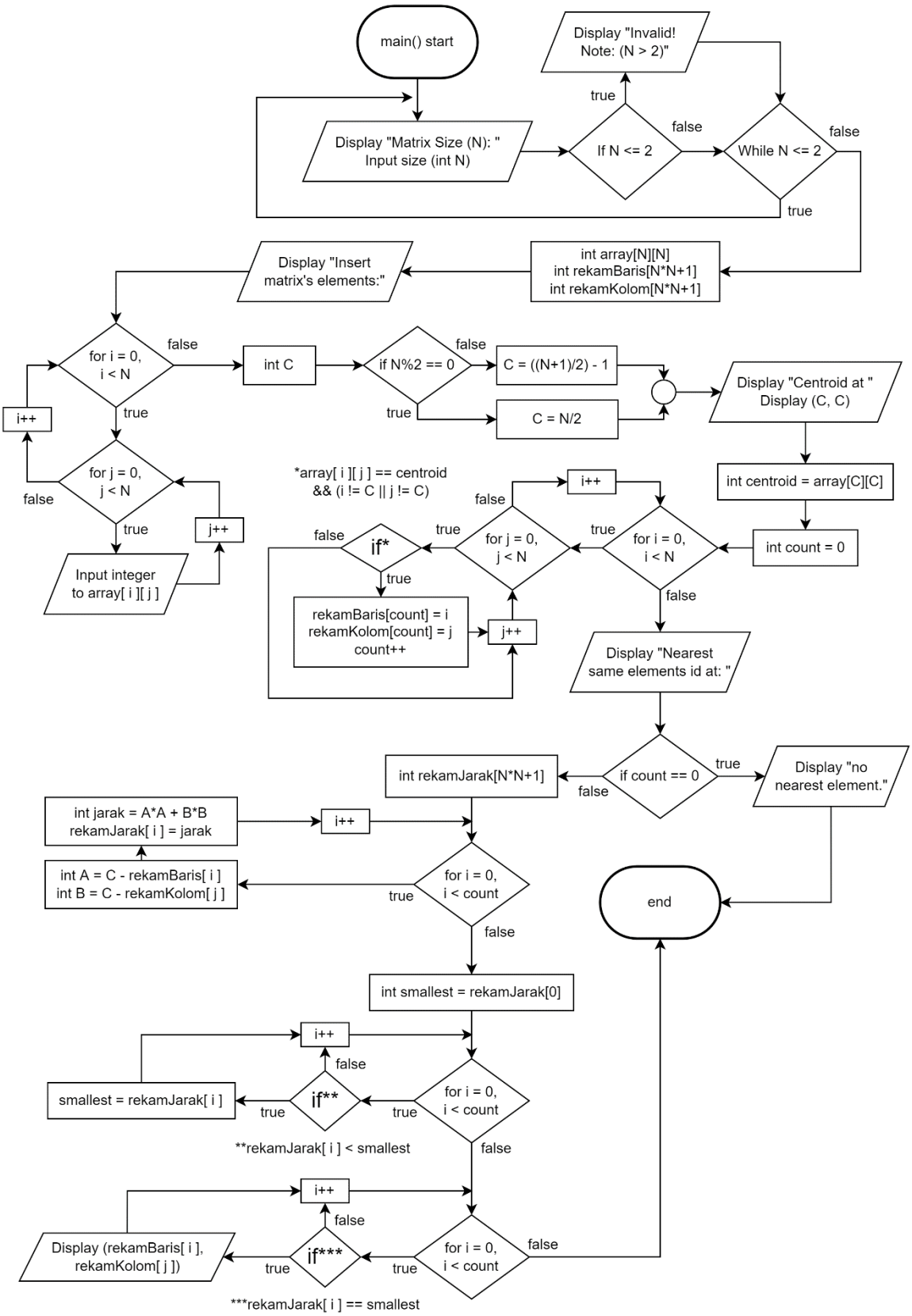
If the previous nested-for-loop statement found the same value element (*count != 0*), now we should calculate their distances to centroid. The mathematical formula for a distance between two points is $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. Without the root square, we still be able to determine which is the closest distance by finding the smallest sum between two squared coordinates. Thus, the formula: $(x_2 - x_1)^2 + (y_2 - y_1)^2$. The program will do the calculations iteratively (*i < count*) for all coordinates, then saving their distances into *rekamJarak[i]*. After all distances calculated, the program will find the smallest distance. By using linear search method, all of array indexes will correlate into same context. Therefore, we don't need to worry about the array's input order (only if we write the iterative parameters correctly).



```
    for(i = 0; i < count; i++)
    {
        if(rekamJarak[i] == smallest) //print the coordinate(s
        {
            printf("(%d,%d) ", rekamBaris[i], rekamKolom[i]);
        }
    }
}
return 0;
```

The last step is to display the closest same value coordinates. If the for loop reach the distance (*rekamJarak[i]*) that equals to the *smallest*, then display the coordinates based on the corresponding index. The flowchart for the whole code is included onto the next page.

Alden Ardiwinata Putra
06 – 2501977286 – LF01



Alden Ardiwinata Putra
06 – 2501977286 – LF01

Problem Number 2

Narrative explanation:

The problem simply asks us to display Riri's friends' data based on name column search key. If the search key does not meet the criteria and the corresponding data is not found, then we should display "Data not exist".

```
struct data
{
    char name[15];
    char phone[14];
    char address[30];
};

struct data teman[8];
```

Based on the description, the data should be structured from name, to phone, and finally address. I set the three data as a char, size of 15 for name, size of 14 for phone, size of 30 for address, and declare an array of struct named 'teman' that has size of 8.

```
void dataArchive()
{
    strcpy(teman[0].name, "Dini Noviani");
    strcpy(teman[0].phone, "081225224336");
    strcpy(teman[0].address, "Pekalongan, Jawa Tengah");

    strcpy(teman[1].name, "Dini Maryati");
    strcpy(teman[1].phone, "081225224337");
    strcpy(teman[1].address, "Semarang, Jawa Tengah");

    strcpy(teman[2].name, "Fahri");
    strcpy(teman[2].phone, "081225224338");
    strcpy(teman[2].address, "Jayapura, Papua");

    strcpy(teman[3].name, "Fahrur");
    strcpy(teman[3].phone, "081227224336");
    strcpy(teman[3].address, "Sintang, Kalimantan Barat");

    strcpy(teman[4].name, "Zafir");
    strcpy(teman[4].phone, "081235294339");
    strcpy(teman[4].address, "Samarinda, Kalimantan Timur");

    strcpy(teman[5].name, "Ruben");
    strcpy(teman[5].phone, "081205220336");
    strcpy(teman[5].address, "Malang, Jawa Timur");

    strcpy(teman[6].name, "Tria");
    strcpy(teman[6].phone, "081225224336");
    strcpy(teman[6].address, "Bandung, Jawa Barat");
}
```

Then we should assign Riri's Friends' data beforehand. I wrote a function, *dataArchive()*, to write all the data mentioned in the question into the struct starting from index 0.

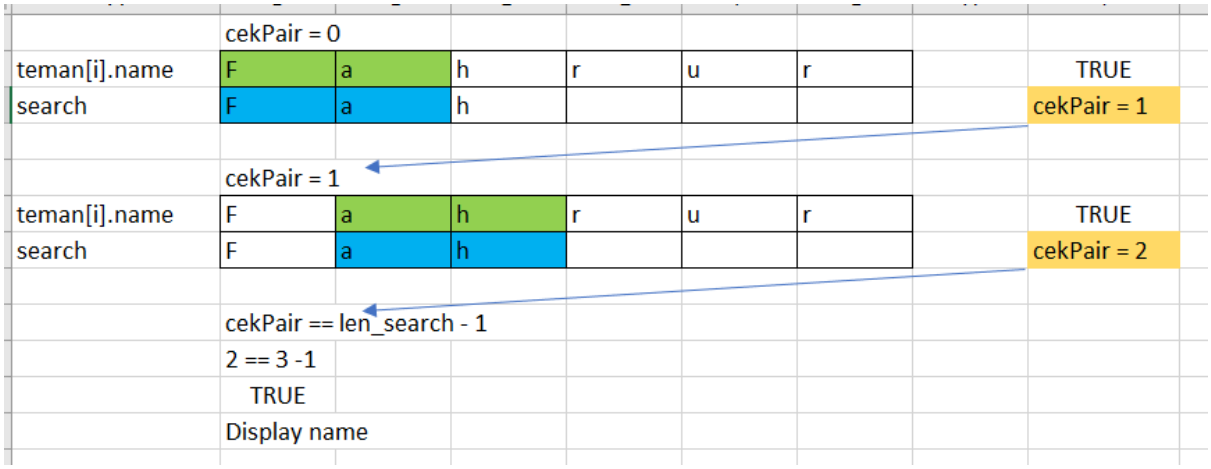
```
dataArchive();

char search[15] = {};
printf("Attention! Please be aware of case sensitivity!\n");
printf("Search Data: ");
scanf("%[^\n]", search);

int len_search = strlen(search);
int i, j;
printf("\n|NAME\t\t|PHONE\t\t|ADDRESS\n");
```

After the program call the function *dataArchive()*, the code will warn the user about case sensitivity and asks to input the keyword to *search*. Then we calculate the length of the *search* word.

The next part is roughly divided into two conditions, the first one is when the length of the search keyword is more than 1 (a word), or the search keyboard is a single alphabet.

```
int cekPair, flag, not_exist = 0;
int last = 0;
for(i = 0; i < 7; i++)
{
    flag = 0;
    cekPair = 0;
    if(len_search > 1)
    {
        flag = 1;
        for(j = 0; j < strlen(teman[i].name) - 1; j++)
        {
            if(teman[i].name[j] == search[cekPair] && teman[i].name[j + 1] == search[cekPair + 1])
            {
                cekPair++;
            }
        }

        if(cekPair == len_search - 1)
        {
            printf("|%-15s|%-15s|%s\n", teman[i].name, teman[i].phone, teman[i].address);
        }
        else
        {
            not_exist++;
        }
    }
```

This first condition is executed if the search key length is more than 1 (len_search > 1). Using for loop, this program will check each name data (for i = 0; i < 7; i++) and screening it (for j = 0; j < strlen(teman[i].name) - 1; j++) comparing the name with the search key that has inputted. Instead of checking it one character per one character, I wrote the program to check the name by a pair of character obtained from the search key. Illustration provided.



Because the program checks the search key in pair, so the maximum *cekPair* counted must be the same as the length of the search key subtracted by 1 (*len_search -1*). If this condition is true, then display the current name. If not, then the data didn't match the search key, count it as *not_exist*. The variable *flag* is automatically set to 1 in this condition (necessary due to condition checking for not existent data).

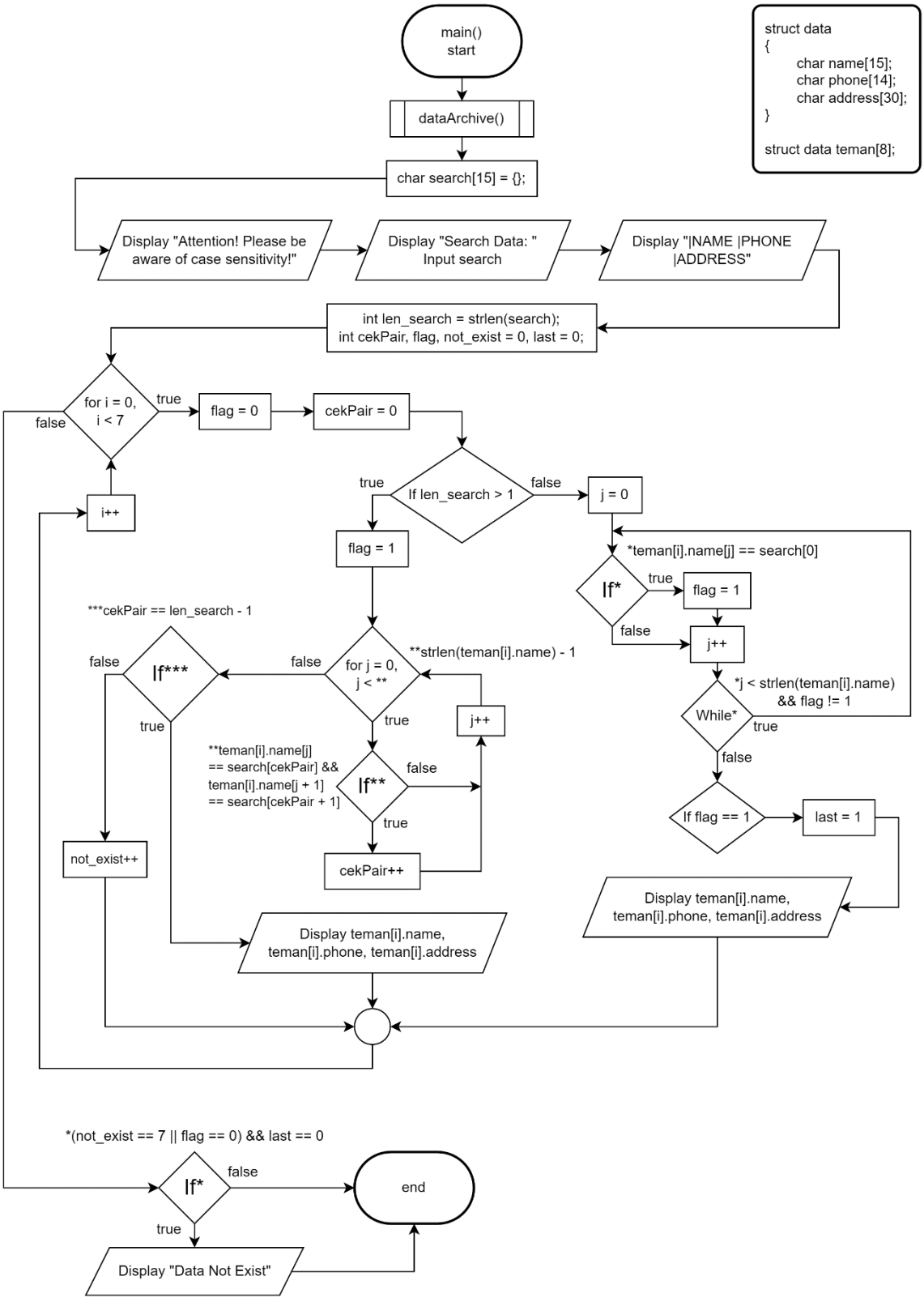
```
    else
    {
        j = 0;
        do
        {
            if(teman[i].name[j] == search[0])
            {
                flag = 1;
            }
            j++;
        }
        while(j < strlen(teman[i].name) && flag != 1);

        if(flag == 1)
        {
            last = 1;
            printf("|%-15s|%-15s|%s\n", teman[i].name, teman[i].phone, teman[i].address);
        }
    }
}
```

This second condition is executed if we only input a single character into the *search* key. Basically, the program is only needed to do linear search, if the first same character (*search[0]*) is found in the current name (*teman[i].name[j]*), then the loop is ended, and the program will have to display the current name. If there is no same character found, then variable *flag* and *last* will be remain 0 as a parameter of non-existent data.

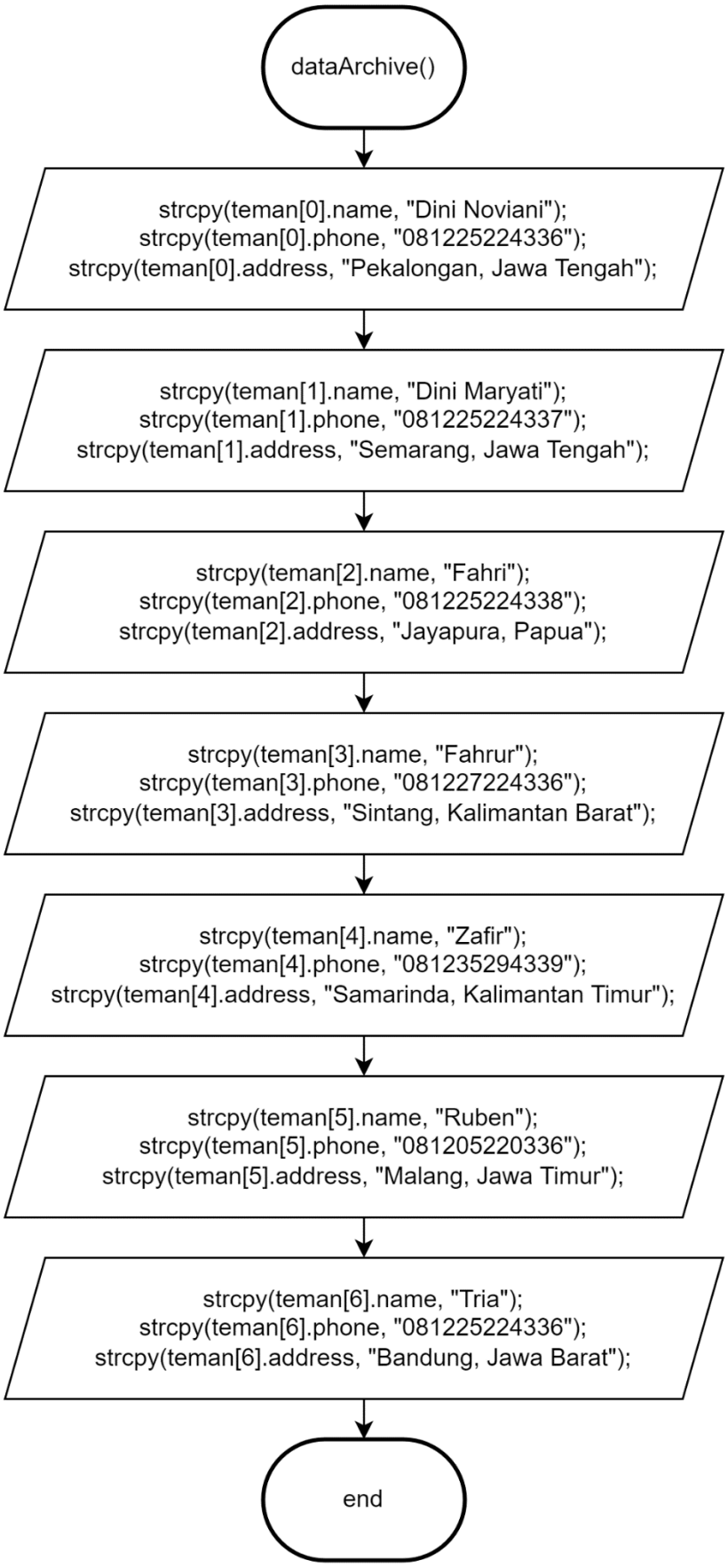
```
if((not_exist == 7 || flag == 0) && last == 0)
{
    printf("|Data Not Exist\n");
}
return 0;
```

Finally, to determine if such data do not exist based on the search key, there are three parameters that I set. The variable *not_exist* affected from the first condition, if the search key didn't find any match pair, *not_exist* will be added by 1 until all data checked, which is 7 times. The variable *flag* and *last* is affected from the second condition. Variable *flag* is used to tell whether a data is existed or not for the second condition, if the data searched not exist, then *flag* will be remained 0. But in some cases, because the *flag* variable is always re-assigned to 0 after each loop, the program may misinterpret the parameter and displaying "data not exist" at the end of loop after the name searched actually exist and displayed. This problem can be handled by adding another variable, *last*, to mark the end of the loop if a data is existed or vice versa. The flowchart for the whole code is included onto the next page.

Alden Ardiwinata Putra
06 – 2501977286 – LF01



Alden Ardiwinata Putra
06 – 2501977286 – LF01



Alden Ardiwinata Putra
06 – 2501977286 – LF01

Problem Number 3

Narrative explanation:

There are some complications in understanding statements in problem number 3. Problem number 3 tells us to check which object inserted into the 2D array has the larger area, but then it is stated that we do not check the area by circumference. This statement contradicts the metric measurement approach because circumference and area are two inseparable aspects that affect each other's value. Thus, problem number 3 can't be solved using metric measurement by calculating the object's length and width then multiplying both variables using a specific formula.

We can assume that the object's area can be measured by counting how many zeros are there which is enclosed by another distinct value (such as 1). With this understanding, I designed my algorithm to act similarly to on-off switch. The program will only count the number of zeros (horizontally) when it encountered the first distinct value (switch on) until it reaches the second distinct value (switch off) that will enclose the zeros. This process will be repeated start from the first row until the last row (vertically). Because an object is considered as an enclosed 0 by another distinct value, then we must validate if a specific row is enclosed by a distinct value if the program is about to switch to the next row.

```
int N;
printf("Matrix size: ");
scanf("%d", &N);
getchar();

int array_one[N][N], array_two[N][N];
int i, j;
```

The program will ask and read the size of array (*int N*) that we want to input. From the size, the program will set the array's size to *N*N*. *array_one* is for object 1, *array_two* is for object 2.

Alden Ardiwinata Putra
06 – 2501977286 – LF01

```c
printf("Matrix 1:\n");
for(i = 0; i < N; i++)
{
    for(j = 0; j < N; j++)
    {
        scanf("%d", &array_one[i][j]);
    }
}
int area_one = hitungZero(N, array_one);
//printf("Area 1: %d\n", area_one);

printf("\nMatrix 2:\n");
for(i = 0; i < N; i++)
{
    for(j = 0; j < N; j++)
    {
        scanf("%d", &array_two[i][j]);
    }
}
int area_two = hitungZero(N, array_two);
//printf("Area 2: %d\n", area_two);
```

The program will ask twice for user to input object 1 and object 2 into each array. Using iterative nested for loop, the program will scan the entire matrix and assigning the objects we would like to compare. As usual, i represents row and j represents column. After each nested for ended, the program will call the same function, *hitungZero()*, with two parameters: array's size (N), the corresponding object's array (*array_one* and *array_two*). The function will return the value to *area_one* and *area_two*.

```c
int hitungZero(int size, int matrix[][size])
{
    int onoffon, count, k, l;
    int save = 0;
    for(k = 0; k < size; k++)
    {
        onoffon = 0, count = 0;
        for(l = 0; l < size; l++)
        {
            if(matrix[k][l] == 1)
            {
                onoffon++;
                //printf("onoff: %d\n", onoffon);
            }
            if(onoffon == 1 && matrix[k][l] == 0)
            {
                count++;
                //printf("Count: %d ", count);
            }
        }

        if(onoffon == 2)
        {
            save = save + count;
        }
    }
    return save;
}
```

Function hitungZero() will count how many zeros are there that enclosed by 1. *Int onoffon* act as a parameter that tells the program when they should count the zeros, when they should stop, and determine whether the counting process is valid or not. Integer *onoffon* and count should be re-assigned to 0 after each outer loop (for loop k) to avoid miscalculation.

If the inner loop (for loop l) encounter value 1, *onoffon* added by 1, if the next column fulfills the condition that the value contained is 0 and *onoffon* is equal to 1, count the zero. After the inner loop ended, validate that *onoffon* == 2 (it means 0 is enclosed by two value of 1). If it doesn't fulfill the condition mentioned, then do not save the zero count. We can infer if *onoffon* is greater than 2, it must be a sequence of 1 that forming a line, if *onoffon* is less than 2, then it's a single integer of 1, or a sequence of all zeros row. I provided a few illustrations for a further understanding.

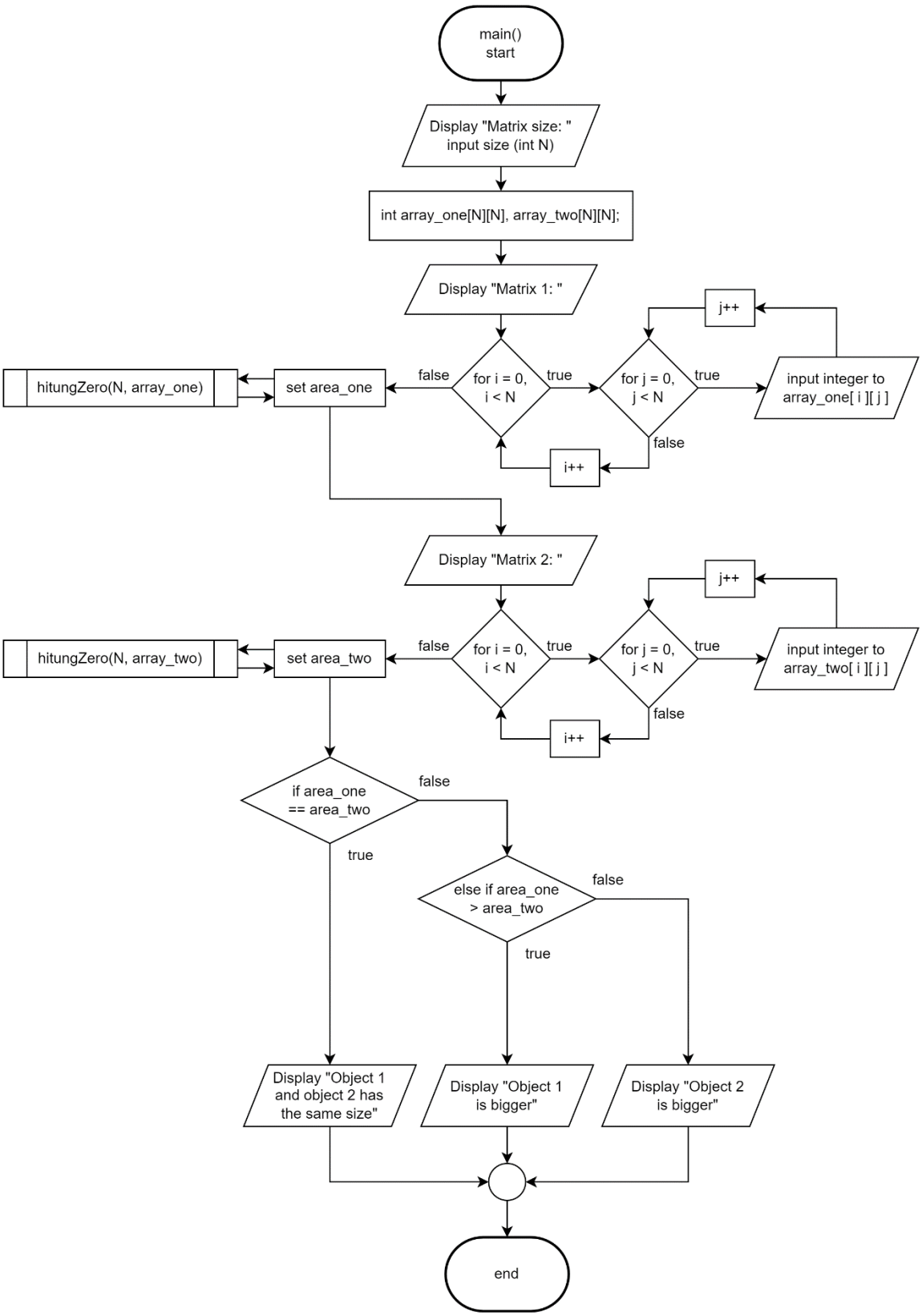
										if onoffon == 2
0	1	1	1	1	onoffon = 0	onoffon = 1	onoffon = 2	onoffon = 3	onoffon = 4	INVALID, DO NOT SAVE *count*
0	1	0	0	1	onoffon = 0	onoffon = 1	count = 1	count = 2	onoffon = 2	VALID, SAVE *count = 2*
0	1	1	1	1	onoffon = 0	onoffon = 1	onoffon = 2	onoffon = 3	onoffon = 4	INVALID, DO NOT SAVE *count*
0	0	0	0	0	onoffon = 0					*save = count*
0	0	0	0	0	onoffon = 0					
										if onoffon == 2
0	0	1	0	0	onoffon = 0	onoffon = 0	onoffon = 1	count = 1	count = 2	INVALID, DO NOT SAVE *count*
0	1	0	1	0	onoffon = 0	onoffon = 1	count = 1	onoffon = 2	onoffon = 2	VALID, SAVE *count = 1*
0	0	1	0	0	onoffon = 0	onoffon = 0	onoffon = 1	count = 1	count = 2	INVALID, DO NOT SAVE *count*
0	0	0	0	0	onoffon = 0	onoffon = 0	onoffon = 0	onoffon = 0	onoffon = 0	*save = count*
0	0	0	0	0	onoffon = 0	onoffon = 0	onoffon = 0	onoffon = 0	onoffon = 0	

As we can see, *onoffon* is used to avoid the program to miscalculate the zeros after value 1. This condition may happen if the object contains a single 1 value in a row. By this logical approach, the program also able to count size of an asymmetrical object. After the iterative process done, the function will return the value of *save* as the value of area for each object.

```
if(area_one == area_two)
{
    printf("\nObject 1 and object 2 has the same size\n");
}
else if(area_one > area_two)
{
    printf("\nObject 1 is bigger\n");
}
else
{
    printf("\nObject 2 is bigger\n");
}
```

From the value of area of each object that we obtained from a function before, now we compare the value to find which object has the greater size, or both are equal. The flowchart for the whole code is included onto the next page.

Alden Ardiwinata Putra
06 – 2501977286 – LF01



Alden Ardiwinata Putra
06 – 2501977286 – LF01