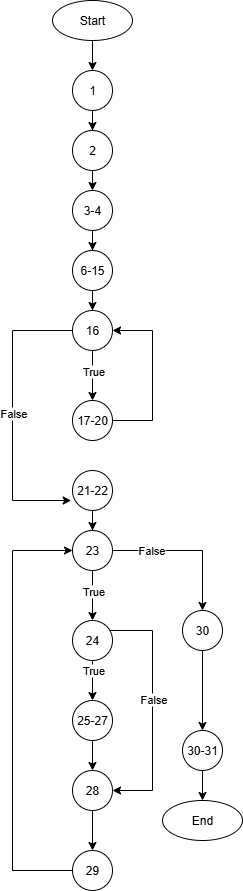
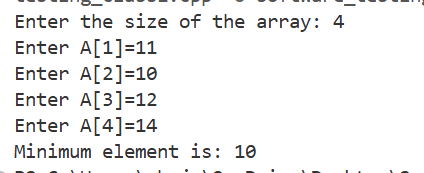
**LAB 1: Refer the following program to find the smallest integer from an array of numbers entered by the user.**

1.1: **Prepare the CFG of the minimum.c Program**

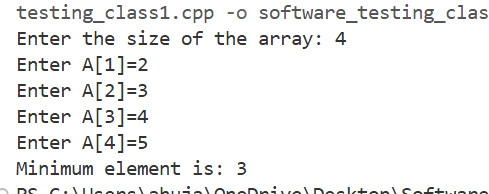


1.2: **Write the test cases to find the errors in the Program:**

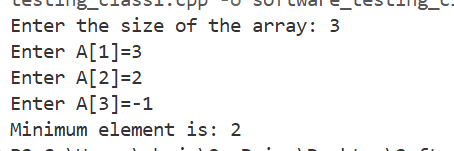
1. Test Case1:



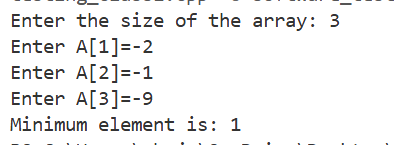
1. Test Case2:



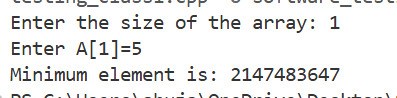
1. Test Case3:



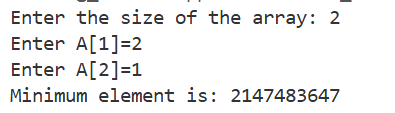
1. Test Case4:



1. Test Case5:



1. Test Case6:



TEST CASES

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.no | Input | Expected Output | Actual Output | Result |
| 1. | Size=4 [11, 10, 12, 14] | 10 | 10 | Pass |
| 2. | Size=4 [2, 3, 4, 5] | 2 | 3 | Fail |
| 3. | Size=3 [3, 2, -1] | -1 | 2 | Fail |
| 4. | Size=3 [-2, -1, -9] | -9 | 1 | Fail |
| 5. | Size=1  [5] | 5 | INT\_MAX | Fail |
| 6. | Size=2  [2, 1] | 1 | INT\_MAX | Fail |

1.3: **Summarize the errors to facilitate the user to debug the program**

The code is not able to handle:

* The last element of the array for checking the minimum integer.
* The first element of the array.
* Not able to handle real numbers; only integers can be handled.
* Not able to handle a negative integer value to determine the minimum integer.

1.4: **Debug the Program for the identified error and execute the same**

#include <stdio.h>

#include <limits.h>

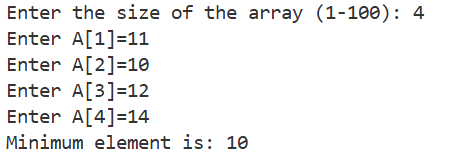
1. void MinMax();
2. int main(){
3. MinMax();
4. return 0;
5. }
6. void Minimum(){
7. int array[100];
8. int min = INT\_MAX;
9. int Number;
10. int i;
11. int tmpData;
12. printf("Enter the size of the array (1-100): ");
13. scanf("%d", &Number);
14. for (i = 0; i < Number; i++) {
15. printf("Enter A[%d]=", i + 1);
16. while(1){
17. char term;
18. int result = scanf("%d%c", &tmpData, &term);
19. if (result == 2 && term == '\n'){

break;

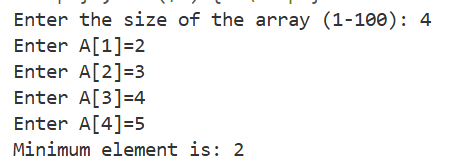
1. }
2. else printf("Invalid input! Enter an integer: ");
3. while (getchar() != '\n');
4. }
5. array[i] = tmpData;
6. }
7. i=0;
8. while(i < Number){
9. if(min > array[i]){
10. min = array[i];
11. }
12. i++;
13. }
14. printf("Minimum element is: %d\n", min);
15. }

1.5: **Prepare the set of Test Cases to Re-test the Program**

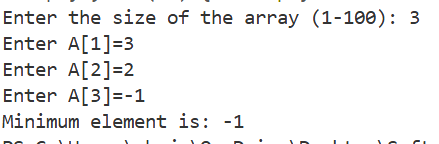
1. Test Case1:



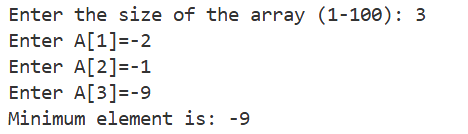
1. Test Case2:



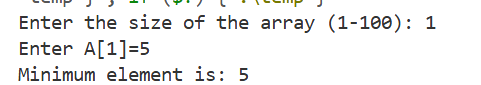
1. Test Case3:



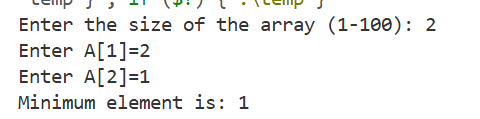
1. Test Case4:



1. Test Case5:



1. Test Case6:



TEST CASES:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.no | Input | Expected Output | Actual Output | Result |
| 1. | Size=4 [11, 10, 12, 14] | 10 | 10 | Pass |
| 2. | Size=4 [2, 3, 4, 5] | 2 | 2 | Pass |
| 3. | Size=3 [3, 2, -1] | -1 | -1 | Pass |
| 4. | Size=3 [-2, -1, -9] | -9 | -9 | Pass |
| 5. | Size=1  [5] | 5 | 5 | Pass |
| 6. | Size=2  [2, 1] | 1 | 1 | Pass |

**LAB 2: Problem Statement: Re-write and extend the minimum.c program (exercise 1.4) with the name Min\_Max\_Safe\_Input for the following additional conditions:**

1. Array size entered by the user will be whole number between 1 to 100
2. The number entered by the user in the Array cannot be a character but can be any real number.
3. The program shall also find minimum and maximum value entered in the array.

#include <stdio.h>

#include <float.h>

void MinMax();

int main() {

    MinMax();

    return 0;

}

void MinMax() {

    double array[100];

    double min = DBL\_MAX, max = -DBL\_MAX;

    int Number;

    int i;

    double tmpData;

    printf("Enter the size of the array (1-100): ");

    while (scanf("%d", &Number) != 1 || Number < 1 || Number > 100) {

        printf("Invalid size! Enter a whole number between 1 and 100: ");

        while (getchar() != '\n');

    }

    for (i = 0; i < Number; i++) {

        printf("Enter A[%d]: ", i + 1);

        while(1){

            char term;

            int result = scanf("%lf%c", &tmpData, &term);

            if (result == 2 && term == '\n') {

                break;

            }

            else {

                printf("Invalid input! Enter a real number: ");

            }

        }

        array[i] = tmpData;

        if (tmpData < min){

            min = tmpData;

        }

        if (tmpData > max){

            max = tmpData;

        }

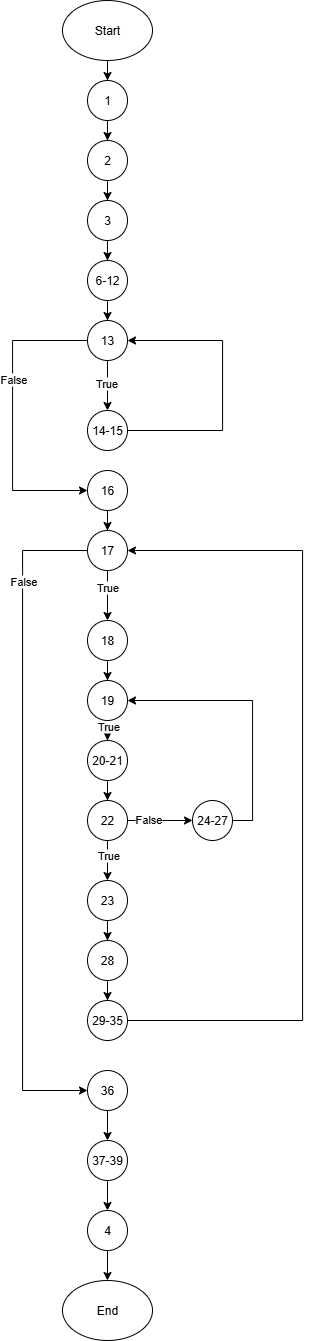
    }

    printf("Minimum value: %.6lf\n", min);

    printf("Maximum value: %.6lf\n", max);

}

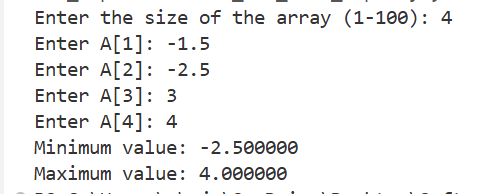
2.1: **Prepare the CFG of the minimum.c Program**



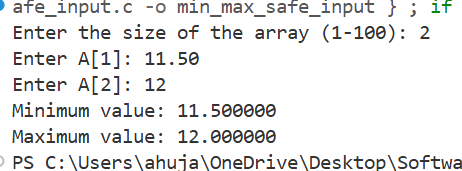
2.2 **Write the test cases to find the errors in the Program**.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S. No | Input | Expected Output | Actual Input | Result | Comments |
| 1. | Size=4  [-1.5, -2.5, 3, 4] | -2.5, 4 | -2.5, 4 | Pass | Handles negative real numbers |
| 2. | Size=2  [11.50, 12] | 11.5, 12 | 11.5, 12 | Pass | Handles all real numbers |
| 3. | Size=1  [c] | Error | Error | Pass | Input data type check |
| 4. | Size=-1 | Invalid | Invalid | Pass | Size input type check |

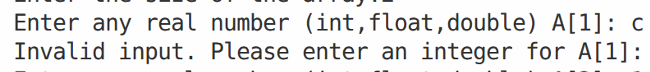
1. Test Case1:



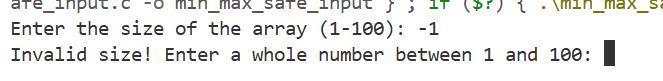
1. Test Case2:



1. Test Case3:



1. Test Case4:



2.3: **Summarize the errors to facilitate the user to debug the program**

There is no error in the program; it is able to handle all edge cases.

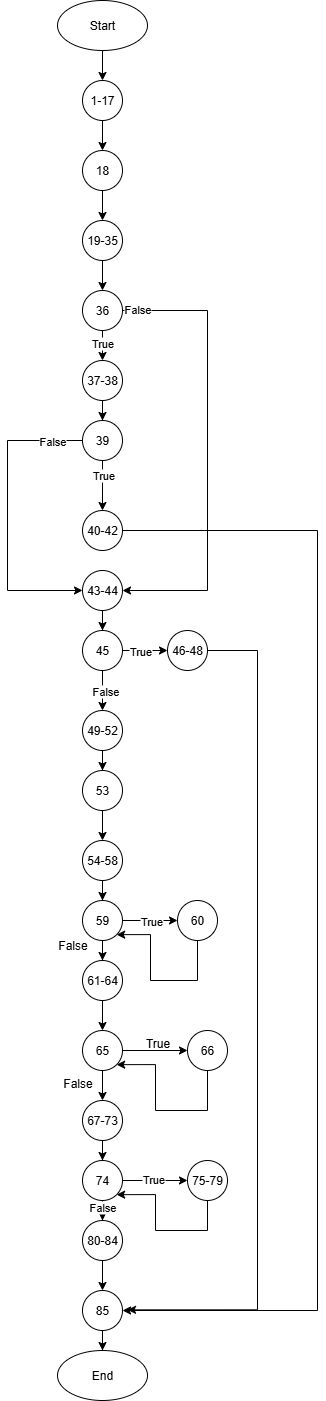
2.4: **Debug the Program for the identified error**

The code is correct, no debugging is needed.

2.5: **Prepare the set of Test Cases to re-test the Program**.

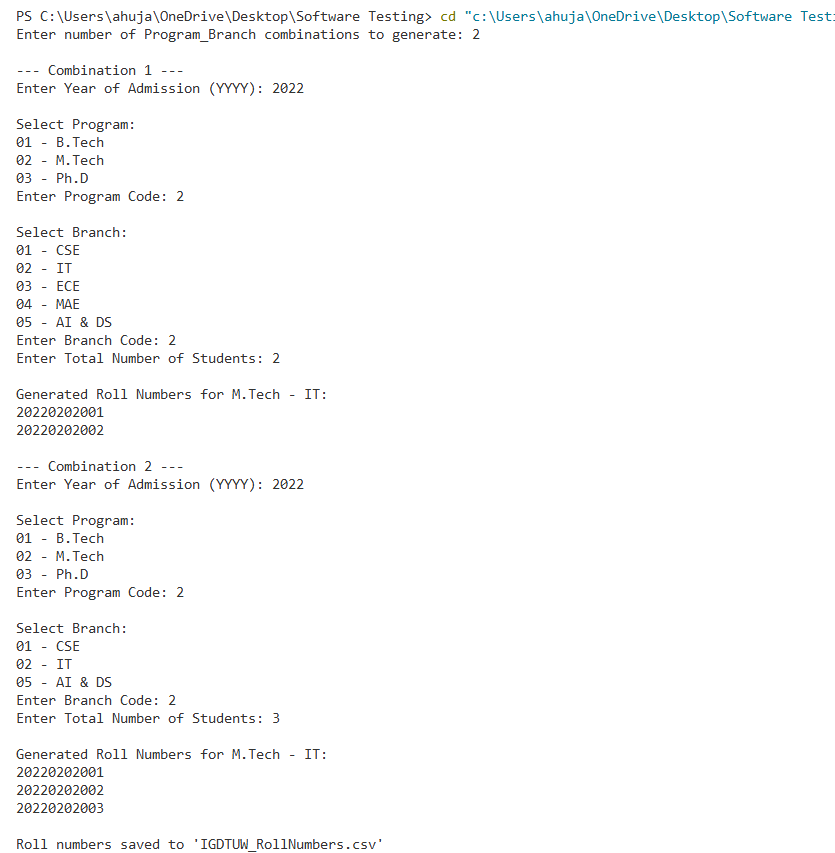
The above code is checked in exercise 2.2.

**LAB 3: Examine the following program for the Generations of Enrollment Number for the University Students admitted in an Academic Year.**



**3.2: Write the test cases to find the errors in the Program**

1. **Test Case1:**



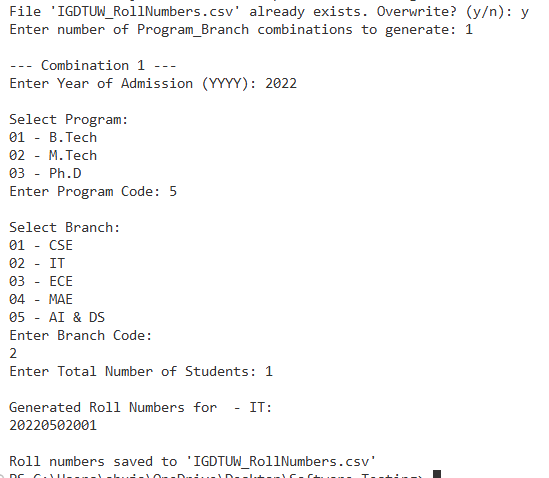
Expected Output: Should not have re-generated the enrolment number

Actual Output: Regenerating the enrolment numbers (Not Unique)

Result: FAIL

Comments: The code is regenerating the enrollment number for the same branch and program in the same academic year. ERROR

1. **Test Case2:**



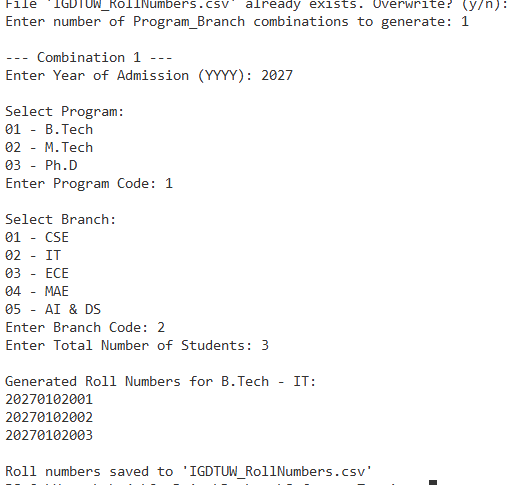
Expected Output: Should not create enrolment numbers

Actual Output: it is creating enrolment numbers for the program which is not specified.

Result: FAIL

Comments: Can’t handle checking whether the Program and branch asked is Valid or not.

1. **Test Case3**:



Expected Output: ERROR

Actual Output: It is generating the enrolment numbers

Result: FAIL

Comments: No check for the validity of the academic year.

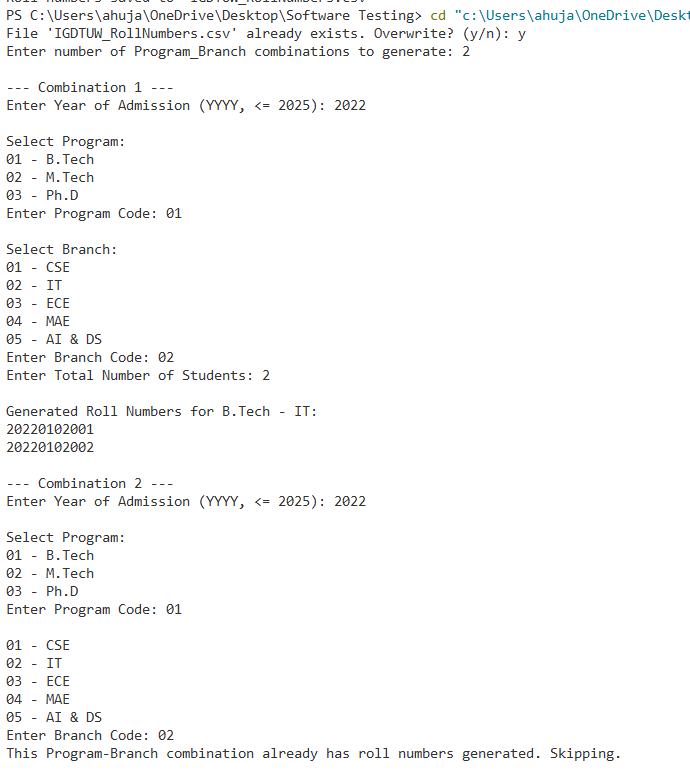
3.3: **Summarize the error to facilitate the user to debug the program**

The code is not able to:

* Check the combination of program and branch code, so that it does not regenerate the enrollment no. for the same combination.
* Keep a check on year before generating enrollment numbers, it should be less or equal to the current year.
* Check if the program, branch code is valid or not.

3.4: **Debug the Program for the identified error**

1. **Test Case1:**

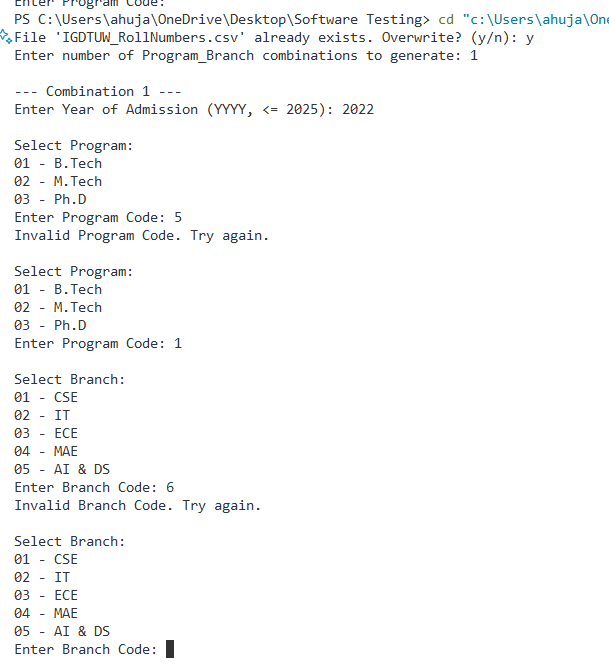


Expected Output: Should not re-generate same enrolment number combinations

Actual Output: Not regenerating the same combinations.

Results: PASS

1. **Test Case2:**

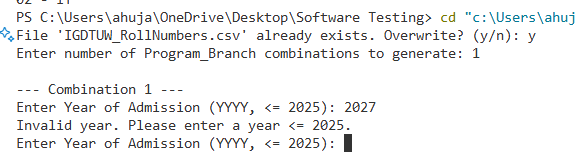


Expected Output: Should verify the Program and Branch Code

Actual Output: It is verifying both.

Results: PASS

1. **Test Case3:**



Expected Output: Should verify the year entered

Actual Output: generates enrolment numbers after validating the year.

Results: PASS