Somtochukwu C Osigwe Daniel Lab 4 sco46

1. The expected output value of result is 6.

   The integers x and y are explicitly cast to floating-point numbers using (float), ensuring that the subsequent division operation results in a floating-point value, preserving the fractional part.

   The division x / y is performed as a floating-point division due to the explicit casting, resulting in 10.0 / 3.0, which evaluates to approximately 3.333333.

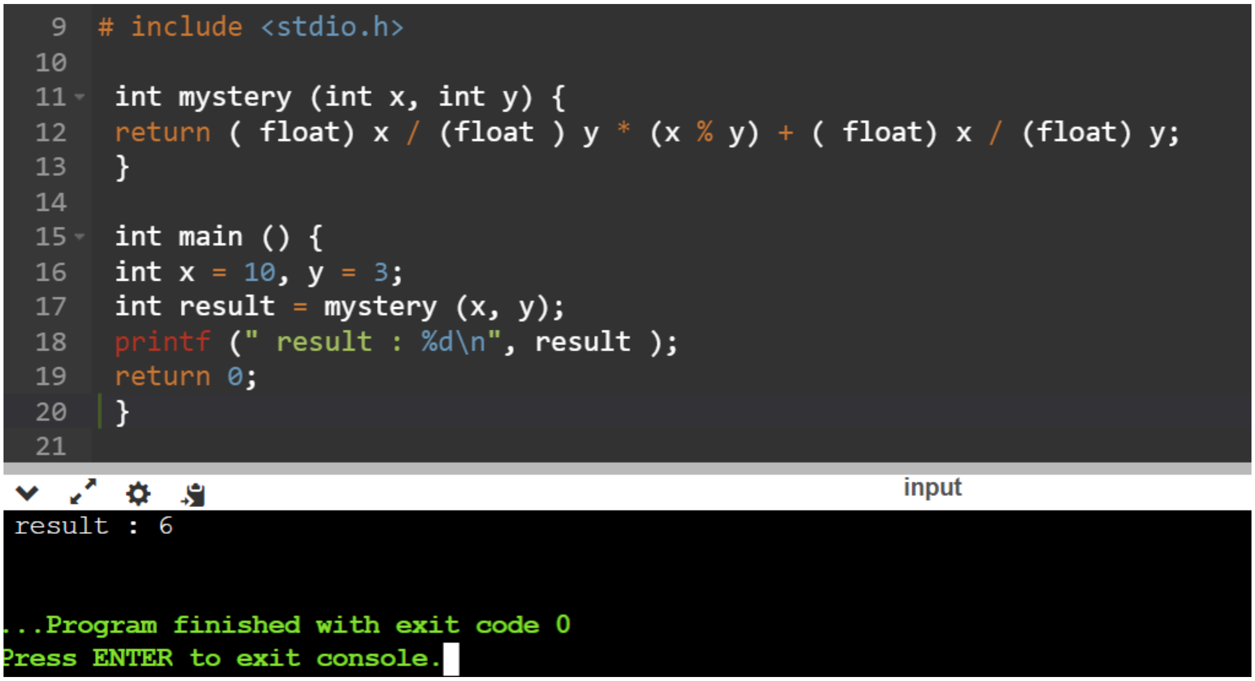
   The expression x%/y calculates the remainder of x divided by y, resulting in 10 %/ 3 = 1.

   The result of the division (`3.333333`) is multiplied by the result of the modulus operation (`1`), giving 3.333333 * 1 = 3.333333.

   Addition: Finally, the result of the multiplication is added to the result of the division again, resulting in 3.333333 + 3.333333 = 6.666666.

   Since the return type of the mystery function is int, the final floating-point result of 6.666666 is implicitly cast back to an integer. This cast removes the fractional part without rounding, resulting in the final output being 6.

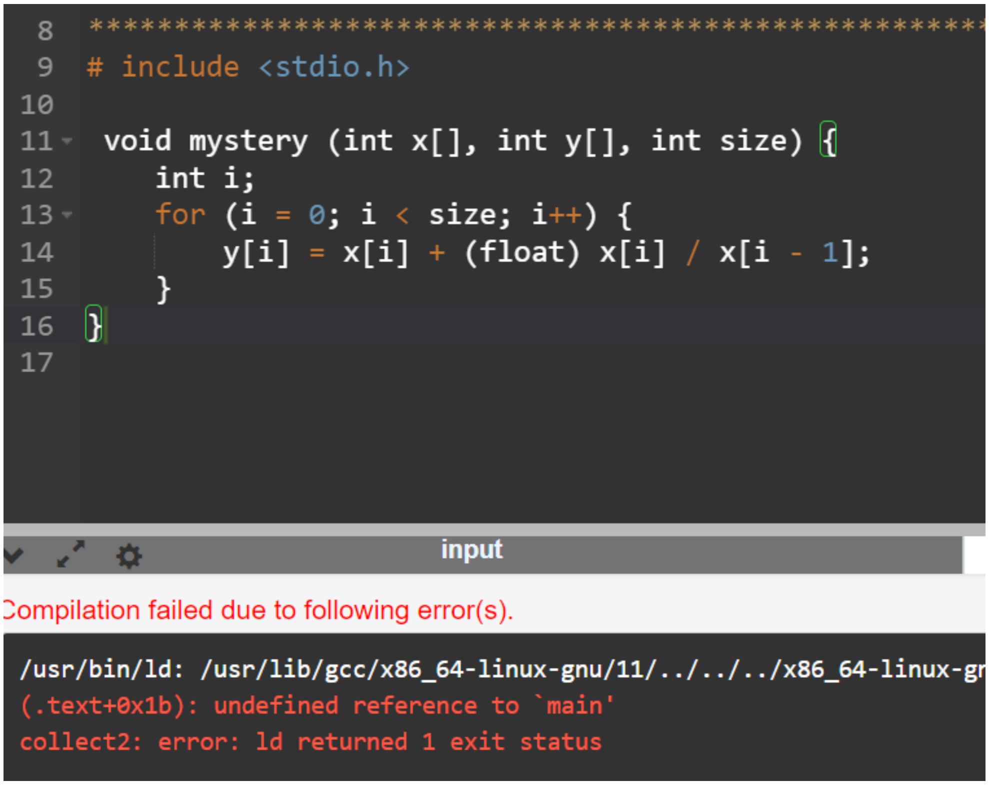
   Therefore, the expected output value of result is 6.



```c
# include <stdio.h>

int mystery (int x, int y) {
return ( float) x / (float ) y * (x % y) + ( float) x / (float) y;
}

int main () {
int x = 10, y = 3;
int result = mystery (x, y);
printf (" result : %d\n", result );
return 0;
}

```

```
result : 6


...Program finished with exit code 0
Press ENTER to exit console.
```

2. .



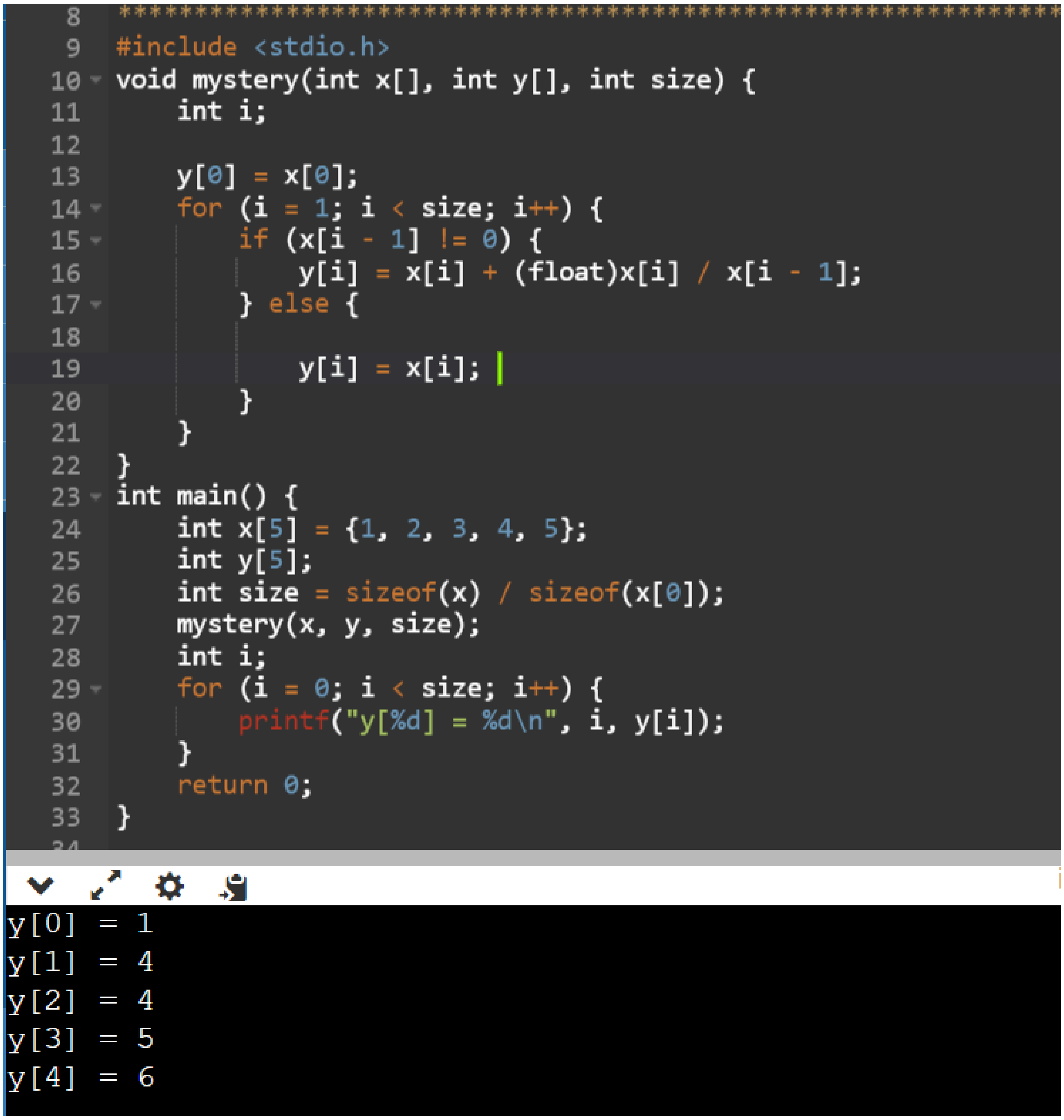
```
*****************************************************
# include <stdio.h>

 void mystery (int x[], int y[], int size) {
     int i;
     for (i = 0; i < size; i++) {
         y[i] = x[i] + (float) x[i] / x[i - 1];
     }
}

```

Compilation failed due to following error(s).

```
/usr/bin/ld: /usr/lib/gcc/x86_64-linux-gnu/11/../../../x86_64-linux-gn
(.text+0x1b): undefined reference to `main'
collect2: error: ld returned 1 exit status
```

Division by Zero and Access Violation: In the first iteration of the loop (i = 0), the expression x[i - 1] attempts to access x[-1], which is out of bounds of the array and results in undefined behavior. Additionally, if x[i - 1] is 0, it will cause a division by zero error.
Correction: The loop should start from i = 1 to avoid accessing x[-1], and there should be a check to ensure x[i - 1] is not zero before performing the division.
Type Casting: The code casts x[i] to float before division. This ensures floating-point division is used, which is correct if the intention is to preserve the fractional part of the division result. However, the result of the entire expression is assigned to y[i], which is an integer. This means the fractional part will be truncated.
Uninitialized Elements in y[]: For i = 0, y is not assigned any value, which might lead to y containing garbage value if y[] is not initialized elsewhere.
Correction: Initialize y before the loop or within the loop when handling the case of i = 0.

Corrected Code:



```
*********************************************************
#include <stdio.h>
void mystery(int x[], int y[], int size) {
    int i;

    y[0] = x[0];
    for (i = 1; i < size; i++) {
        if (x[i - 1] != 0) {
            y[i] = x[i] + (float)x[i] / x[i - 1];
        } else {

            y[i] = x[i];
        }
    }
}
int main() {
    int x[5] = {1, 2, 3, 4, 5};
    int y[5];
    int size = sizeof(x) / sizeof(x[0]);
    mystery(x, y, size);
    int i;
    for (i = 0; i < size; i++) {
        printf("y[%d] = %d\n", i, y[i]);
    }
    return 0;
}
```

```
y[0] = 1
y[1] = 4
y[2] = 4
y[3] = 5
y[4] = 6
```

Division by Zero and Access Violation: The corrected code initializes y explicitly to avoid using an uninitialized value and starts the loop from i = 1. It also checks if x[i - 1] is zero before performing the division, addressing both issues.

Type Casting: The casting is preserved in the corrected code. However, the result of the entire expression is assigned to y[i], which is an integer, leading to truncation of the fractional part. This behavior is expected and correct as per the problem statement.

Output Explanation:

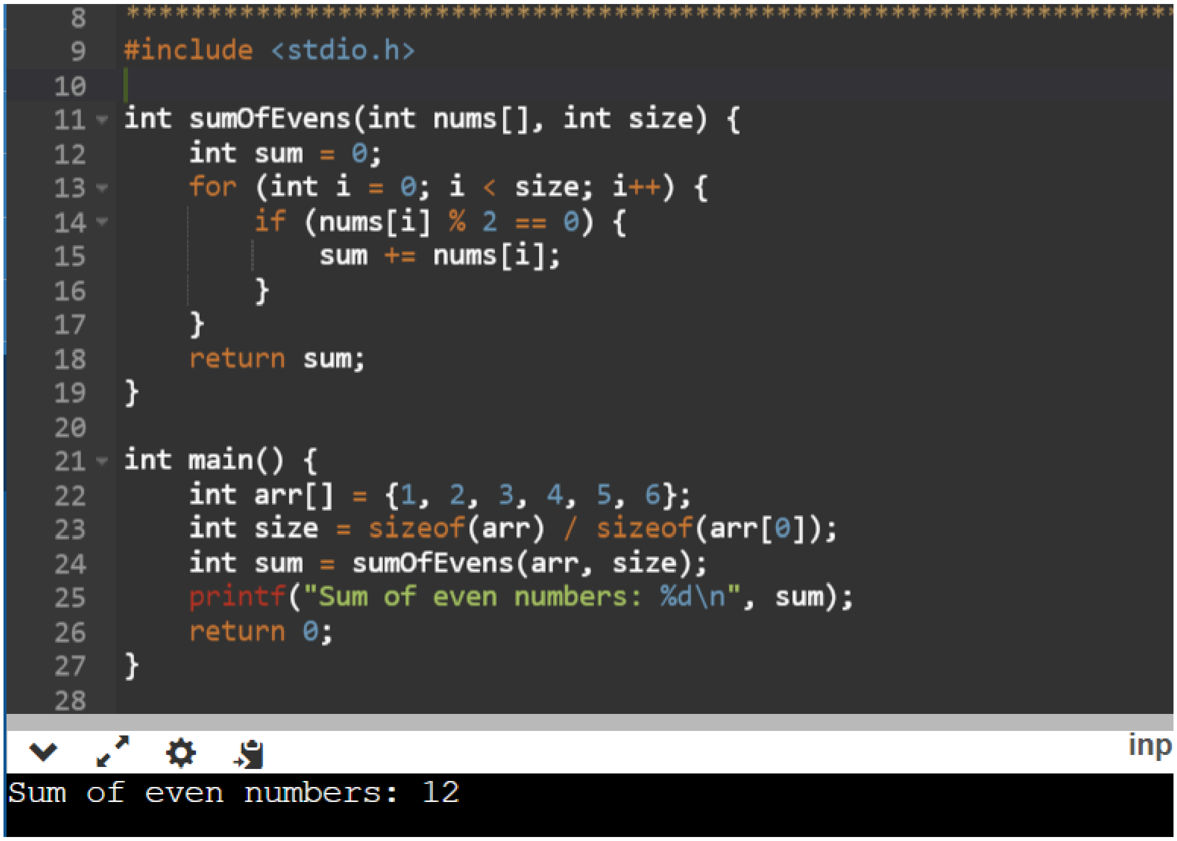
For i = 1: y[1] = 2 + (float)2 / 1 = 2 + 2.0 = 4.0 (truncated to 4 when stored in y[1]).

For i = 2: y[2] = 3 + (float)3 / 2 = 3 + 1.5 = 4.5 (truncated to 4 when stored in y[2]).

For i = 3: y[3] = 4 + (float)4 / 3 = 4 + 1.333333 = 5.333333 (truncated to 5 when stored in y[3]).

For i = 4: y[4] = 5 + (float)5 / 4 = 5 + 1.25 = 6.25 (truncated to 6 when stored in y[4]).

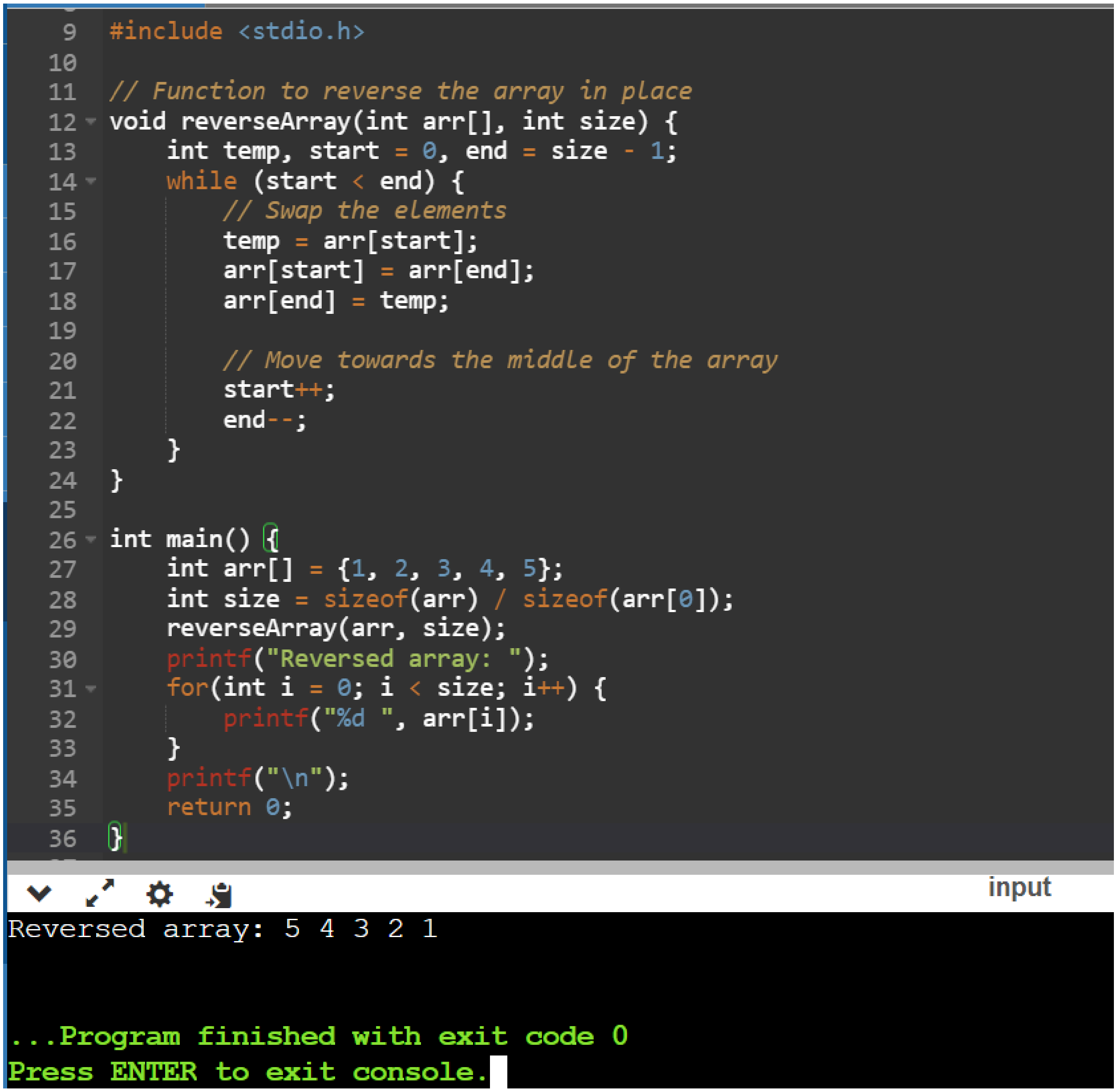
3. .



```
*******************************************************************
#include <stdio.h>

int sumOfEvens(int nums[], int size) {
    int sum = 0;
    for (int i = 0; i < size; i++) {
        if (nums[i] % 2 == 0) {
            sum += nums[i];
        }
    }
    return sum;
}

int main() {
    int arr[] = {1, 2, 3, 4, 5, 6};
    int size = sizeof(arr) / sizeof(arr[0]);
    int sum = sumOfEvens(arr, size);
    printf("Sum of even numbers: %d\n", sum);
    return 0;
}

```

```
Sum of even numbers: 12
```

4. .



```c
#include <stdio.h>

// Function to reverse the array in place
void reverseArray(int arr[], int size) {
    int temp, start = 0, end = size - 1;
    while (start < end) {
        // Swap the elements
        temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;

        // Move towards the middle of the array
        start++;
        end--;
    }
}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int size = sizeof(arr) / sizeof(arr[0]);
    reverseArray(arr, size);
    printf("Reversed array: ");
    for(int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

```
Reversed array: 5 4 3 2 1


...Program finished with exit code 0
Press ENTER to exit console.
```