Gebze Technical University

CSE 331

Computer Organizations

HW2 Report

Yusuf Can Kan
161044007

1 Program Explanation

1.1 Part 1 (C++)

In this part I implemented 1 recursive function. The function "CheckSumPossibility" which takes 3 variable as input. As first variable it takes the number that we are looking in the array(num). Second it takes the array that we will search on(arr[]). The third argumet is the size of the second argument(size). If process is successful it gives 3 output. First is success or not (1-0), second is array that contains success elements and third is size of second output. Second and third output defined as global and maximum size of this array is 256.
The program calls itself in 2 way. The first way is it updates the first variable with substituting with array[size]. The other way, it does not touch num. Both of the cases it decreases size element before call itself. So in this way it call itself with including array[size] element and does not including it.
Note:
Since it including array[size] element it looks for count from last element. That is why result may be different from homework pdf file. The reason why I implemented the function this way is, in my implementation if I start to look sums from beginning I should store another memory for it. It is more simple and clean in this way.

There are 3 base case;

- If num(first variable) is zero it means we found the result in the array. Result is successful so we don't need to search rest of the array. Return success.This case satisfies the optimization given in homework pdf file.
- If num smaller than 0 it means condition does not satisfies. So don't need to call function again and don't need to look rest of the array in this condition. Return fail.
- If size is -1 it means we are iterated all the array. Condition does not satisfied. Return fail.

For the bonus part; In first recursive call (the call with we included the array[size] element.), if it returns success we add the array[size] element in result array. At the end function return success an in this way it applies the same thing recursively.

1.2 Part2 - MISP

In this part main function first takes array size, takes num and takes array elements. "read_array" labes helps to takes array elements one by one. Then it calls the "CheckSumPossibility" procedure/function. It takes 3 variable as input and gives 3 output as same with c++ part. For input and output array size is limited with 1026 byte(so size is 256).
Every time function call itself first it stores the current arguments in stack pointer. It controls the base cases and call itself recursively.
Before it call itself first time it decreases the array[size] from num and after recursive call is ended it increases it directly.
I implemented 3 label for this function.

- First label is "return_fail". This label sets "$v0" (return value) to 0(fail) and restores the stack pointer and jumps back to last call.
- Second label is "return_success". It does the same thing as "return_fail" but it sets the return value to 1(success).
- The third label is "return_success_add". This label does the same thing with "return_succes" but before jumps back to last call it adds the array[size] element to result element and increases size. This part is implemented for bonus part.

At the end "CheckSumPossibility" function finishes its job and returns to main. If result is success, main jumps to "print_success" label and prints the result array elements with using loop label. If result is fail it jumps to "print_fail" label and prints fail.

2 Test Cases

```
yck@ubuntu:~/Desktop/CSE331-HW2$ ./hw2
Provide size of array:4

Provide target number:25

Provide your array values:
0
9
1
15
Possible!
Result array:
15 1 9
yck@ubuntu:~/Desktop/CSE331-HW2$
```

Figure 1: Test1 - C++

```
Provide size of array:4
Provide target number:25
Provide your array values:
0
9
1
15
Possible!
Result Array: 15 1 9
-- program is finished running --
```

Figure 2: Test1 - MISP

```
yck@ubuntu:~/Desktop/CSE331-HW2$ ./hw2
Provide size of array:8

Provide target number:30

Provide your array values:
9
1
5
90
123
15
10
2
Possible!
Result array:
10 15 5
yck@ubuntu:~/Desktop/CSE331-HW2$
```

Figure 3: Test2 - C++

```
Provide size of array:8
Provide target number:30
Provide your array values:
9
1
5
90
123
15
10
2
Possible!
Result Array: 10 15 5
-- program is finished running --
```

Figure 4: Test2 - MISP

```
yck@ubuntu:~/Desktop/CSE331-HW2$ ./hw2
Provide size of array:3

Provide target number:10

Provide your array values:
9
11
20
Not possible!
```

Figure 5: Test3 - C++

```
Provide size of array:3
Provide target number:10
Provide your array values:
9
11
20
Not possible!!

-- program is finished running --
```

Figure 6: Test3 - MISP

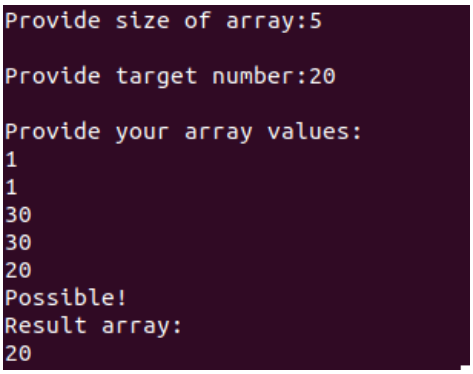

Figure 7: Test4 - C++

```
Provide size of array:5
Provide target number:20
Provide your array values:
1
1
30
30
20
Possible!
Result Array: 20
-- program is finished running --
```

Figure 8: Test4 - MISP

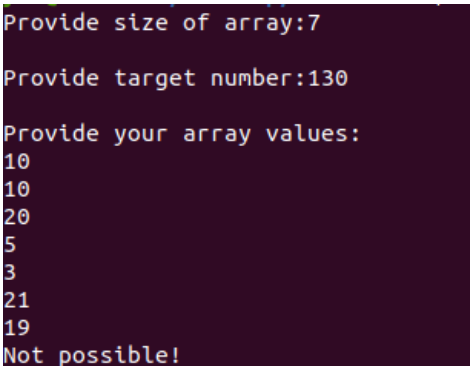

Figure 9: Test5 - C++

```
Provide size of array:7
Provide target number:130
Provide your array values:
10
10
20
5
3
21
19
Not possible!!

-- program is finished running --
```

Figure 10: Test5 - MISP

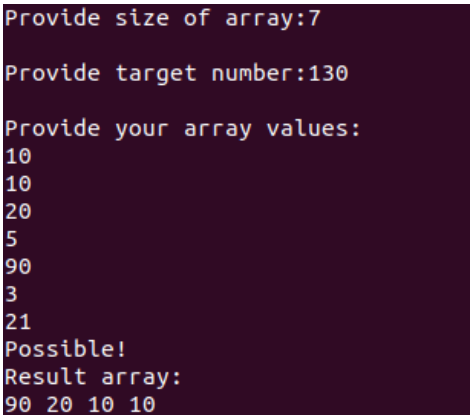

Figure 11: Test6 - C++

```
Provide size of array:7
Provide target number:130
Provide your array values:
10
10
20
5
90
3
21
Possible!
Result Array: 90 20 10 10
-- program is finished running --
```

Figure 12: Test6 - MISP

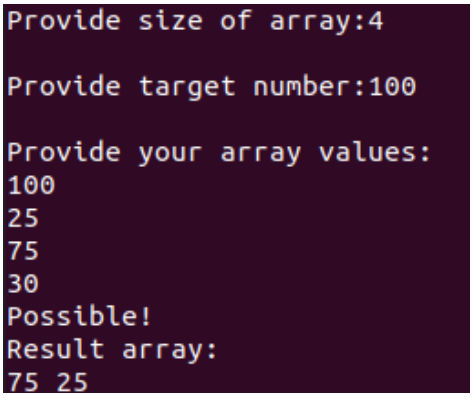

Figure 13: Test7 - C++

```
Provide size of array:4
Provide target number:100
Provide your array values:
100
25
75
30
Possible!
Result Array: 75 25
-- program is finished running --
```

Figure 14: Test7 - MISP

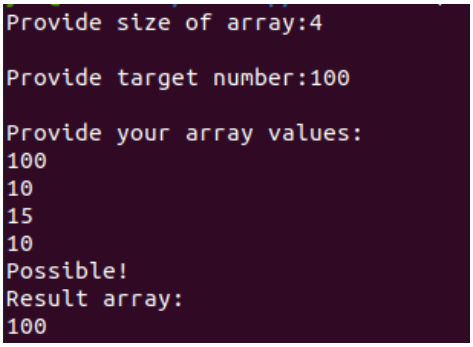

Figure 15: Test8 - C++

```
Provide size of array:4
Provide target number:100
Provide your array values:
100
10
15
10
Possible!
Result Array: 100
-- program is finished running --
```

Figure 16: Test8 - MISP

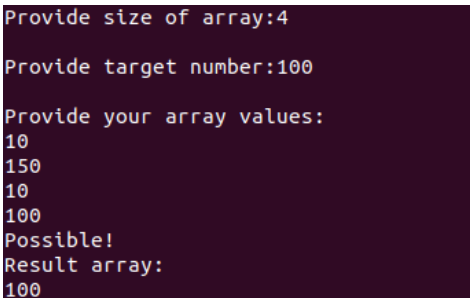

Figure 17: Test9 - C++

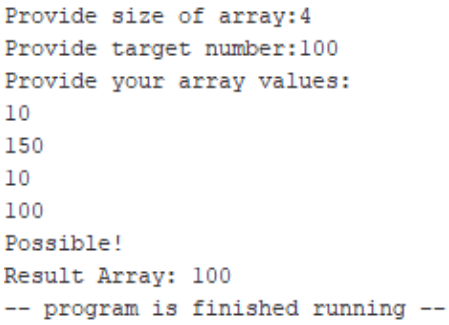

Figure 18: Test9 - MISP