Gebze Technical University
CSE-331
HW2-Report

NEVZAT SEFEROGLU
171044024

➔C++ Version Explanation:

Function takes three different arguments, **int num, int arr, int size.**

num – sum which is queried

arr – array which contains sequence of numbers

size – size of given array

In function body,

There are some escape statements which stop next recursion call.

```
if(num == 0) {
        returnVal = 1;
```

If num == 0, returnVal equals 1 means that we found the solution after that it will print the numbers, and which comply with that solution.

```
else if(size == 0 || num < 0) {
        returnVal = 0;
```

There is an optimization in this statement, the optimization is when the sum < 0, we do not need to evaluate the ongoing recursion call. Therefore, we go to next recursion call. It will reduce our exploration amount a little bit.

In the statement of else, there are two different recursion call. The first call means the current sum query with the last element and, we must reduce the sum with the last element of the array. In the second recursion call, if we don't find the sum with the current query, we must call for the next last element in the array. We save the return value of these recursion calls to or each other later.

For an extra point, I print the numbers which is find as correct.

```
if(cur) {
            cout << arr[size-1] << " ";
            return returnVal;
}
```

➔Assembly Version Explanation:

Firstly, I decided to write some functions which are used frequently. These functions are,

allocate_int_array , allocate an array and return the allocated memory address
get_int , get an integer as an input
print_char , print character to screen
print_string , print string to screen
print_integer , print integer to screen
exit, change the program counter and exit the program safely

Let's talk about function, which is **CheckSumPossibility**, I directly reflect my C++ code to assembly code so there are some rule that I used for myself. These rules are,

Register **$t0** - **returnVal**
Register **$t1** - **current**
Register **$t2** – **rest**

These rules are making my code clearer and easy to implement. At each recursion call these values are loaded with zero and the rest implementation match directly with C++ version of the algorithm that I explain in the C++ part of this report.

I did not use any global variable other than string for printing something to screen. There more explanation in the comment section of the assembly code.

```
          .data
msg_size_input     :     .asciiz "\nArray size: "
msg_num_input    :      .asciiz "\nNumber: "
msg_seq_input      :     .asciiz "\nSequnce:\n"
msg_possible        :     .asciiz "\nPossible!\n"
msg_not_possible :     .asciiz "\nNot possible!\n"
```

➔ How to run and use it:

When the program run,
The user types the size of the array, the num which will be queried and sequence of positive number for quiring the given num whether can be constitute with sum of other numbers in the sequence.

Basic course of events of program:

```
Array Size: 3
Number: 6
Sequence:
1
2
3
1 2 3                (Values which is ensure the given sum!)
Possible!
```

➔ Tests:

1)

```
Array Size: 8
Number: 129
Sequence:
41
67
34
0
69
24
78
58

Not possible!
```

```
2)
Array Size: 8
Number: 129
Sequence:
62
64
5
45
81
27
61
91

Not possible!
```

```
3)
Array Size: 8
Number: 129
Sequence:
95
42
27
36
91
4
2
53
36 91 2
Possible!
```

4)

```
Array Size: 8
Number: 129
Sequence:
92
82
21
16
18
95
47
26
82 21 26
Possible!
```

5)

```
Array Size: 8
Number: 129
Sequence:
71
38
69
12
67
99
35
94
35 94
Possible!
```

6)

Array Size: 8

Number: 129

Sequence:

3

11

22

33

73

64

41

11

Not possible!

➔Notes:

- Program never ever accepts **negative** numbers.
- For the amount of **CheckSumPossiblity** call, when I compare the number of calls with specific example published on Teams, there are some more calls and also less calls in my function depends on the sequences which are given. The optimization that I made, and examples are different.