Programming Assignment 1.

Brute Force, *Backtracking*, *Recursive*, *Subset Sum*

Purpose of the exercise

This exercise will help you do the following:

1. Practice developing Brute Force based program that is used to solve Subset Sum problem.
2. Practice implementing backing tracking algorithm by writing recursive function.

Overview

This assignment is light workload, you are required to complete a C++ program by writing a recursive function that is an implementation of backtracking algorithm.

Backtracking

Some problems can be solved, by *exhaustive search*. The exhaustive-search technique suggests generating all candidate solutions and then identifying the one (or the ones) with a desired property.

Backtracking is a more intelligent variation of this approach. The principal idea is to construct solutions one component at a time and evaluate such partially constructed candidates as follows.

- If a partially constructed solution can be developed further without violating the problem's constraints, it is done by taking the first remaining legitimate option for the next component.
- If there is no legitimate option for the next component, and no alternatives for any remaining component need to be considered. In this case, the algorithm backtracks to replace the last component of the partially constructed solution with its next option.

It is convenient to implement this kind of processing by constructing a tree of choices being made, called the **state-space tree**. Its root represents an initial state before the search for a solution begins. The nodes of the first level in the tree represent the choices made for the first component of a solution; the nodes of the second level represent the choices for the second component, and so on.

In the majority of cases, a state space tree for a backtracking algorithm is constructed in the manner of depth-first search. The general algorithm for backtracking that find a single solution is as follows:

```
bool findSolutions(n, other params) {
  if (found a solution) {
     displaySolution();
     return true;
  }

  for (val = first to last) {
    if (isValid(val, n)) {
       applyValue(val, n);
       if (findSolutions(n + 1, other params))
           return true;
```

```
        removeValue(val, n);
    }
  }
   return false;
}
```

Task

The subset sum problem is an important and classic problem in computer theory. Given a set of integers and a target number, your goal is to find a subset of those numbers that sum to that target number. For example, given the numbers {3, 7, 1, 8, -3} and the target sum 4, the subset {3, 1} sums to 4. On the other hand, if the target value were 2, the result is False since there is no subset that sums to 2.

Your task is to complete the following function in source file `q.cpp`.

```
bool subset_rec(std::vector<int> const& set, int sum, std::vector<int> & subset,
unsigned long index) {

    int curr_sum = std::accumulate(subset.begin(), subset.end(), 0);

    /*----- Success: find a solution -------*/
    if (curr_sum == sum) {
        for (unsigned long i = 0; i < subset.size(); ++i) {
            std::cout << subset[i] << " "; }
        std::cout << "  sum " << curr_sum << std::endl;
        return true;
    }

    /*----- Hit the bottom: start backtracking -------*/

    // Add code here
    // Your code should print out the candidate solution before return false
    // The output should be in format (# represents a number):
    // # # # #   sum #

    /*----- Backtracking -------*/

    // ADD code here.
    // You backtracking should stop as soon as the first solution is found.

    // left: add nothing {} to the solution

    // right: add the element set[index] to the solution


    return false; // continue backtracking
}
```

Compile and link the completed source file *q.cpp* using the required g++ options:

```
g++ -std=c++17 -pedantic-errors -Wall -Wextra -Werror q.cpp qdriver.cpp -o q.out
```

Then run the executable with input 0, 1, 2 respectively ：

```
./q.out >> myout0.txt
0
./q.out >> myout1.txt
1
./q.out >> myout2.txt
2
```

Use the *diff* command in the Linux bash shell to compare the output files:

```
diff --strip-trailing-cr -y --suppress-common-lines myout0.txt out0.txt
diff --strip-trailing-cr -y --suppress-common-lines myout1.txt out1.txt
diff --strip-trailing-cr -y --suppress-common-lines myout2.txt out2.txt
```

If *diff* completes the comparison without generating any output, then the contents of the two files are an exact match.

Submission

Once your implementation of *q.cpp* is complete, again ensure that the program works and that it contains updated **file-level** documentation comments.

```
/*!*****************************************************************************
\file    q.cpp
\author  ABC
\par     DP email: ABC@digipen.edu
\par     Course: CS330
\par     Section: A
\par     Programming Assignment #1
\date    28-06-2021

\brief

*******************************************************************************/
```

Then upload the file `q.cpp` to the submission page in Moodle and the system will auto-grade your submission.