Project 3: Change Maker

Due: Saturday ⇒ May 4th, 2024

This is an individual project and you can only work on it by yourself. Although you may discuss project *concepts* with other students, please keep the **Academic Integrity Policy** in mind---do not show your code to anyone outside of the professors, and do not look at anyone else's code for this project. If you need help, please contact the professor. No extensions will be allowed and no late submissions will be accepted.

The goal of this project — is to use data structures to implement algorithms. Concepts you will be familiar with after this project include:

- Greedy algorithm
- Dynamic programming
- Debugging & testing

In the picturesque tropical haven of Banana Republic, King CocoCrabby IV has once again tasked the Minister of Mint with creating new currency to combat the ever-increasing inflation. The proposed denominations are:

$1, $101, $701, $5001, $32001, $95000

Poor Patrick is facing a hefty tax bill of $96,003, due tomorrow. He and his fellow villagers urgently require assistance.

Can you create a program to help them devise a plan for managing their tax payment with the new currencies?

```
Sample Run: (User input marked in BOLD)
$ ./p3

Welcome to the IR$ of Banana Republic!
Please enter your name:  Poor Patrick
What is your tax $$$ due?  96003



Plan 1: The Greedy Approach
Tax Due:  $96003
                    Bills         Num                           Subtotal
                   $95000    x     1     =  $95000                $95000
                   $32001    x     0     =      $0                $95000
                    $5001    x     0     =      $0                $95000
                     $701    x     1     =    $701                $95701
                     $101    x     2     =    $202                $95903
                       $1    x   100     =    $100                $96003

Total number of bills needed: 1 + 0 + 0 + 1 + 2 + 100 = 104
------------------------------------------------------------------


Plan 2: The Dynamic Programming Approach
Tax Due:  $96003
                        Bills        Num                      Subtotal
                       $95000     x    0    =       $0              $0
                       $32001     x    3    =   $96003          $96003
                        $5001     x    0    =       $0          $96003
                         $701     x    0    =       $0          $96003
                         $101     x    0    =       $0          $96003
                           $1     x    0    =       $0          $96003

Total number of bills needed: 0 + 3 + 0 + 0 + 0 + 0 = 3


------------------------------------------------------------------
Which payment plan do you want to choose (1 or 2)?  2

Dear Poor Patrick, thank you for paying your tax ON TIME!
------------------------------------------------------------------
```

Requirements

Why did Poor Patrick choose the second plan? It is because Plan 2 requires only 3 bills as opposed to 14 in Plan 1. So, in this project, you task is to design a program that calculates the quantity of each bill necessary for a specified amount of money.

The **`main`** function should prompt the user for their name and the tax amount, then display the number of bills for each denomination needed,

1. first by the greedy algorithm,
2. then by the dynamic programming algorithm.

⇒ The output should present all essential information with aligned columns (Bills, Num, and Subtotal) in that order, although it does not have to match the sample output exactly.

1. The Greedy Algorithm

I know we have not covered Greedy Algorithms yet, but in this case, the greedy algorithm happens to be the "natural" algorithm for giving money: fit as many of the **largest** possible denomination into the amount to give, then move to the next largest denomination, until the full amount is given.

Note that while currencies are usually designed so that this greedy algorithm returns an optimal solution, this is not the case here, so expect that the solutions returned by this algorithm will **not** always be optimal.

Write a function **`greedyPlan`** that takes an integer as input (tax amount), that implements the greedy strategy and returns an array of 6 elements containing the number of $1, $101, $701, $5001, $32001, $95000 bills to give (in that order).

2. The Dynamic Programming Algorithm

A better solution to this problem requires dynamic programming. The goal is to minimizing the total number of bills need for a given amount of money.

First, you will need a quick structures to hold all the past solutions. Create a class **`ChangePlan`** that contains only public elements:

- an integer **`totalBills`** that contains the total number of bills of that plan
- an array **`plan`** of 6 integers that contains the number of *$1, $101, $701, $5001, $32001, $95000 bills needed* for the solution (in that exact order).

Write a function **`dynamicPlan`** that takes a large integer as input (tax amount), that implements the dynamic programming approach and returns an array of 6 elements containing the number bills needed.

For each source file, you need to include comments that precede each function and method give you precise information about what the function or method is supposed to do.

Hints:

1. It is probably easiest to implement the function using Bottom-Up Dynamic Programming.
2. You should allocate an array of N+1 **`ChangePlans`** and fill the array from 1 to N.

Deliverables

Submit a .tar file containing the **entire project folder** for the programming assignment using the submission link on Canvas.

- **`main.cpp`**: the main test driver that reads an input of tax due, computes the plans based on the Greedy and Dynamic Programming approaches, and prints out the results.

- **`Makefile`**: The build instructions for your project.

- **README**: description of the project, list of files, dependencies, how to compile & run, development log, honor pledge, and acknowledgments.

  Furthermore, incorporate a discussion section containing a **minimum of 3 test cases** (inputs).

  For each test case, display the change plans (outputs) produced by your program and evaluate the greedy and dynamic approaches according to the following criterion.

  - Time efficiency
  - Space efficiency
  - Which approach is better? Why? Is it always the case?

Grading

Grades will be given according to the following policy.

Requirements	**Points**
`main()`	10
`greedyPlan()`	20
`dynamicPlan()`	40
Formatted output with the correct change plans	15
Documentation (coding style + comments + quality)	5
README (tests + discussion)	10
Total	**100**

Acknowledgment
This project write-up is based on the Project developed by Martin Gagne at Wheaton College.

Commenting and Coding Style Requirements
Please follow Google C++ coding style guidelines for this project.