Students:

This content is controlled by your instructor, and is not zyBooks content. Direct questions or concerns about this content to your instructor. If you have any technical issues with the zyLab submission system, use the **Trouble with lab** button at the bottom of the lab.

12.1 The Subset-Sum Problem (Part A - For ints)

Parts A and B are required. Part C is optional and is worth 2 points extra credit (but must be submitted in addition to Part A and Part B). Make sure you have read and understood both modules A and B of this week before submitting this assignment.

Part A (**required**) - The Subset Sum Problem For Ints

Create a simple **main()** and complete other needed globe functions and class methods to solve the subset sum problem for any vector of **int**s. Here is an example of the set-up. You would fill in the actual code that makes it happen.

```
int main()
{
   int target = 0;
   vector<int> dataSet;
   vector<Sublist> choices;
   int k, j, numSets, max, masterSum, newSum, bestSublist;
   bool foundPerfect, needAlgorithm;

   dataSet.push_back(20); dataSet.push_back(12);
dataSet.push_back(22);
   dataSet.push_back(15); dataSet.push_back(25);
   dataSet.push_back(19); dataSet.push_back(29);
   dataSet.push_back(18);
   dataSet.push_back(11); dataSet.push_back(13);
dataSet.push_back(17);

   choices.clear();

   cin >> target;
   cout << "Target time: " << target << endl;
```

```
   // code provided by student
   //
   //

   choices[bestSublist].showSublist();

   return 0;
}
```

The local variables you see above are not required; they come from my solution. If you want to use different local variables, feel free.

Your output style may be different from mine. However, in order to receive the credit points, you need to modify your code to match the output of the Auto Grader. Run the exact case as above first in the Develop mode and show enough other runs to prove your algorithm works, and also show at least one run that does not meet target, exactly. Also, provide a special test to quickly dispose of the case in which the target is larger than the sum of all elements in the master set. This way, any target that is too large will not have to sit through a long and painful algorithm. Demonstrate that this works by providing at least one run that seeks a target larger than the sum of all master set elements.

You are not finding all solutions, just one representative solution. There may be other subsets that do the job, but our algorithm only finds the one. Please do implement the exact algorithm presented in our Canvas modules to generate the same output as the Auto Grader.

Ex. For the target input:

```
180
```

The output will be:

```
Target time: 180
Sublist ----------------------------
  sum: 179
  array[0] = 20,   array[1] = 12,   array[3] = 15,   array[4] = 25,
array[5] = 19,   array[6] = 29,   array[7] = 18,   array[8] = 11,
array[9] = 13, array[10] = 17
```

Note the target is not precisely met in this case. Usually it is, so if you vary the target, you'll find that you can get a perfectly matching sum. Also note this solution doesn't contain array[2]. Please make sure to implement the algorithm presented in our modules (not your own algorithm) so your code will produce exactly the same sublists as in the Auto Grader's solutions.

Once you are satisfied with your runs in the the Develop mode, switch to Submit mode and click "Submit for grading". Note: for the first couple of weeks in this course, you would have max of 10 tries. This limit may be reduced over time to encourage you to do enough test in the Develop

mode. Please don't over use the Submit mode unless you are ready to submit. Your final score will be recorded and sent back to Canvas based on your last submission.

`class Sublist`

I have prompted you with all the essential details of the class **Sublist** that will support your algorithm. I'll add a couple more hints:

- To create an empty **Sublist** (which is the first thing you need to do in order to prime the pump) just instantiate one **Sublist** object. By definition, it represents an empty set, since it will have no elements in its internal **indices array list**.
- The **addItem()** member function is clearly the most interesting of the outstanding instance methods. In that method you will have to create a new **Sublist** object whose sum is the sum of the calling object plus the value of the new item added. This will require a slightly different syntax for an **int** data set than it will for an **iTunesEntry** data set. In the latter case, you'll need to get specific inside **addItem()** about what value within **iTunesEntry** you are adding. This means **addItem()** is going to have syntax that ties it firmly to the underlying data set. That's okay for this part and the next, but not for option C.

Part A run output (test cases) requirement

Required Targets: 1, 67, 180, 200, 1000

Other Targets will be checked by the Auto Grader.

376726.1650026.qx3zqy7

LAB ACTIVITY 12.1.1: The Subset-Sum Problem (Part A - For ints) 0 / 10

Downloadable files

`main.cpp` **Download**

main.cpp Load default template...

```
2  // Assignment #1 The Subset-Sum Problem Part A - int version
3
4  #include <iostream>
5  #include <string>
6  #include <vector>
7  using namespace std;
8
9  // global scope helpers
10 double computeMasterSum( vector<int> data );
11 void showEntireVector( vector<int> data );
12
13 // --------------- Sublist Prototype ---------------
14 class Sublist
15 {
16 public:
```

```
17    Sublist(vector<int> *orig = NULL)
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → main.cpp (Your program) →

Program output displayed here

Coding trail of your work What is this?

History of your effort will appear here once you begin working on this zyLab.

Trouble with lab?