MTH 4300
Spring 2025
E. Fink

Homework 4

1. A *partition* of a positive integer is an expression of that integer as a sum of positive integers. For example, one partition of the integer 10 is 1+1+2+3+3. Also, two partitions are considered to be equivalent if they are permutations of one another; for example, 1+3+1+2+3 would be the same partition of 10, whereas 1+4+5 would be a different partition of 10.

   Write a function named `part()` which receives one positive integer N as an input, and which returns the number of distinct partitions of N. For example, 4 has five different partitions: 4, 3+1, 2+2, 2+1+1, 1+1+1+1. So `part(4)` should return `5`.

   This function can call upon ONE other helper function, which should have two parameters, and work recursively.

2. Write a function called `bool subsum(int arr[], int val, int size)`. `size` should be the length of `arr`. This function should return *whether or not there is a subset of* `arr` *whose sum is equal to* `val`. (Each value in `arr` should be used at most once in the sum.)

   For example, if `arr` is initialized with {`16, 8, 1, 2`}, then `subsum(arr, 19, 4)` would return `true`, since 16 + 1 + 2 = 19; but `subsum(arr, 15, 4)` would return `false`, since no subset of the four elements will equal exactly 15.

   Your function may call one other helper function if you like, although I encourage you to write a SINGLE function. Of course, please present a recursive solution.

3. a. In the code below, identify the dangerous line which can potentially cause a segmentation fault, and remove it.

   b. Once you've removed the dangerous line, what would the following code display?

```
int x = 4, *p, *q;
int y[4] = {10, 30, 40, 50};

p = &x;
*p = 8;
*q = 14;
p = y;
q = p + 2;
*p = *(q + 1);
*q = *p + 1;

cout << x << " " << y[0] << " " << y[1] << " " << y[2] << " " << y[3];
```

4. Complete the code in `main()` in `hw4_q4.cpp` so that it asks the user to store `num_entries` integers and stores them in an appropriately-sized dynamic array. Also, complete the function definition of `print()` so that the final call in `main()` prints the contents of the array all on one line of output.

5. (The following exercise is meant to capture a key idea in the implementation of a very important C++ tool, which we'll introduce later.)

   In `hw4_q5.cpp`, there is code which is supposed to allow the user to enter `strings` until the user enters `STOP`. Each entry (including the final entry `STOP`) is intended to be stored to the dynamically-declared array `arr`.

   However, if the user enters more than 5 entries, the program is at risk of crashing because only 5 entries have been allocated for `arr`.

   Fix the program, as follows. Add code that checks if adding a new entry would exceed the space currently allocated for `arr`. If it does, then:

   - create a new dynamic array, *twice the length* of the current one;
   - copy the current contents of `arr` into the first half of the new dynamic array;
   - delete the old array pointed to by `arr`, and assign the new dynamic array to `arr`.

   You should be able to do this without deleting any of the code – you should only have to add code, perhaps between the lines `++index;` and `cin >> arr[index];`. However, you of course also add in the function you wrote in the previous problem for testing purposes, if you like.

6. Let's try to implement a version of Python's `.split()` function, using the tools we have studied. In this problem, I would personally avoid recursion.

   a. Write a function called `int num_words(const string& total)` which returns the number of "words" contained in `total`. Here, we will assume that `total` consists of only printable characters and spaces; and we will take a "word" to mean any sequence of consecutive non-space characters. For example, the string

   ```
   "   abc    d..e fg&hi   ,  jk "
   ```

   would have 5 words.

   b. Now, write a function called `string* split(const string& total)`. This should return a pointer to an array containing all the words in `total`, defined as in the last problem. For this problem, you may want to review the `.substr()` method. Be sure to be cautious about the cases where the last character is a space or not a space.