Homework #7

Due date: 12/22

Integer partitions

Given a positive integer n, generate all the ways and count the number of ways to partition n as a sum of positive integers (that are less than or equal to n, of course). For example, $n = 5$ can be partitioned in seven ways:

$$
\begin{aligned}
5 &= 5 \\
&= 4 + 1 \\
&= 3 + 2 \\
&= 3 + 1 + 1 \\
&= 2 + 2 + 1 \\
&= 2 + 1 + 1 + 1 \\
&= 1 + 1 + 1 + 1 + 1
\end{aligned}
$$

Note that the order of the addends is considered insignificant. Thus, $2 + 2 + 1$, $2 + 1 + 2$, and $1 + 2 + 2$ are all the same partition. By convention, partitions are normally written from largest to smallest addends.

This problem can be solved in a manner similar to the recursive algorithm for the k-combination problem given in the lecture.

Let

$p(n) =$ the number of ways to partition n into positive integers

$q(n, k) =$ the number of ways to partition n into positive integers $\leq k$

Then,

$$
\begin{aligned}
p(n) &= q(n, n), \\
q(n, k) &= 0, && \text{if } n < 0 \text{ or } k = 0 \\
&= 1, && \text{if } n = 0 \text{ (and } k > 0) \\
&= q(n - k, k) + q(n, k - 1), && \text{if } n > 0 \text{ (and } k > 0)
\end{aligned}
$$

k is an addend k isn't an addend

Requirements

1 The number of partitions grows exponentially with n. For example, $n = 200$ can be partitioned in $3{,}972{,}999{,}029{,}388$ ways.
To make life easier, you may assume that $n \leq 20$ and declare

```
struct stack {
    int top;
    int stk[20];        // why 20?
};
```

2 All functions in your program shall be recursive. In particular, each partition shall be output recursively (see point 3 below) and the user interface shall also be produced recursively (see point 4 below).

3 You shall write a recursive function to print out the contents of the stack. How to write this recursive function depends on the declaration of the stack. Which method to use is up to you.

Method 1 – Global stack

Let **s** be the global stack

a) You shall write the recursive function
`void show(int i);`
to print out the array elements from **s.stk[0]** to **s.stk[i]**.

b) Invoke it by the call **show(s.top)**.

Method 2 – Local static stack

Let **s** be the local static stack

a) You shall write the recursive function
`void show(int stk[],int i);`
to print out the array elements from **stk[0]** to **stk[i]**.

b) Invoke it by the call **show(s.stk,s.top)**.

Q: Which method is better?

A: This is an efficiency-security trade-off. Method 1 is more efficient but less secure than Method 2.

Q: Why we pass the array **s.stk**, rather than the stack **s** itself, in Method 2?

A: Well… I shall tell you why later on.

4 You shall write the recursive function

`void ui(void);`

to produce the user interface (see sample output for the required interface). Essentially, this function has to handle multiple inputs $n_1, n_2, \cdots, n_k,\ k \geq 0$ by means of divide-and-conquer:

$$\begin{aligned} \text{ui}() &= \text{do nothing}, && \text{if } k = 0 \text{ (i.e. EOF)} \\ &= \text{read in } n_1, \text{partition it, and} \\ &\quad \text{call ui() recursively to handle } n_2, \cdots, n_k, && \text{if } k > 0 \end{aligned}$$

This function is invoked by **`main`**.

Q: Why do we need **`ui`**? Why don't we simply make **`main`** recursive?

A: The function **`main`** can only be recursive in C – it can't be recursive in C++.

Sample run

```
Enter a positive integer <= 20: 5
  5
  4  1
  3  2
  3  1  1
  2  2  1
  2  1  1  1
  1  1  1  1  1
There are 7 ways to partition 5.

Enter a positive integer <= 20: 7
  7
  6  1
  5  2
  5  1  1
  4  3
  4  2  1
  4  1  1  1
  3  3  1
  3  2  2
  3  2  1  1
  3  1  1  1  1
  2  2  2  1
```

```
  2  2  1  1  1
  2  1  1  1  1  1
  1  1  1  1  1  1  1
There are 15 ways to partition 7.

Enter a positive integer <= 20: 8
  8
  7  1
  6  2
  6  1  1
  5  3
  5  2  1
  5  1  1  1
  4  4
  4  3  1
  4  2  2
  4  2  1  1
  4  1  1  1  1
  3  3  2
  3  3  1  1
  3  2  2  1
  3  2  1  1  1
  3  1  1  1  1  1
  2  2  2  2
  2  2  2  1  1
  2  2  1  1  1  1
  2  1  1  1  1  1  1
  1  1  1  1  1  1  1  1
There are 22 ways to partition 8.

Enter a positive integer <= 20: ^Z
```

Other values

$p(10) = 42$ $p(15) = 176$ $p(18) = 385$
$p(12) = 77$ $p(16) = 231$ $p(19) = 490$
$p(14) = 135$ $p(17) = 297$ $p(20) = 627$