

Homework #8

Due date: 18:00, December 19th, Monday, 2016

Coin change

Generate all the ways and count the number of ways to make change of dollars, given several kinds of coins.

For testing purpose, we assume that there are 4 kinds of coins whose face values are declared *globally* by

```
#define kinds 4  
  
int d[kinds] = {1,5,10,50}; // denominations of the 4 kinds of coins
```

However, keep in mind that your program shall be able to work for other data, say

```
#define kinds 5 // 5 kinds of coins  
  
int d[kinds] = {5,20,50,1,10}; // denominations in arbitrary order
```

This problem can be solved in a recursive manner – the change may include or exclude coins of a specific kind.

Let $cc(n, k)$ = the number of ways to change n dollars using $k \geq 0$ kinds of coins.

Then,

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

those that include the k^{th} kind coin

those that don't

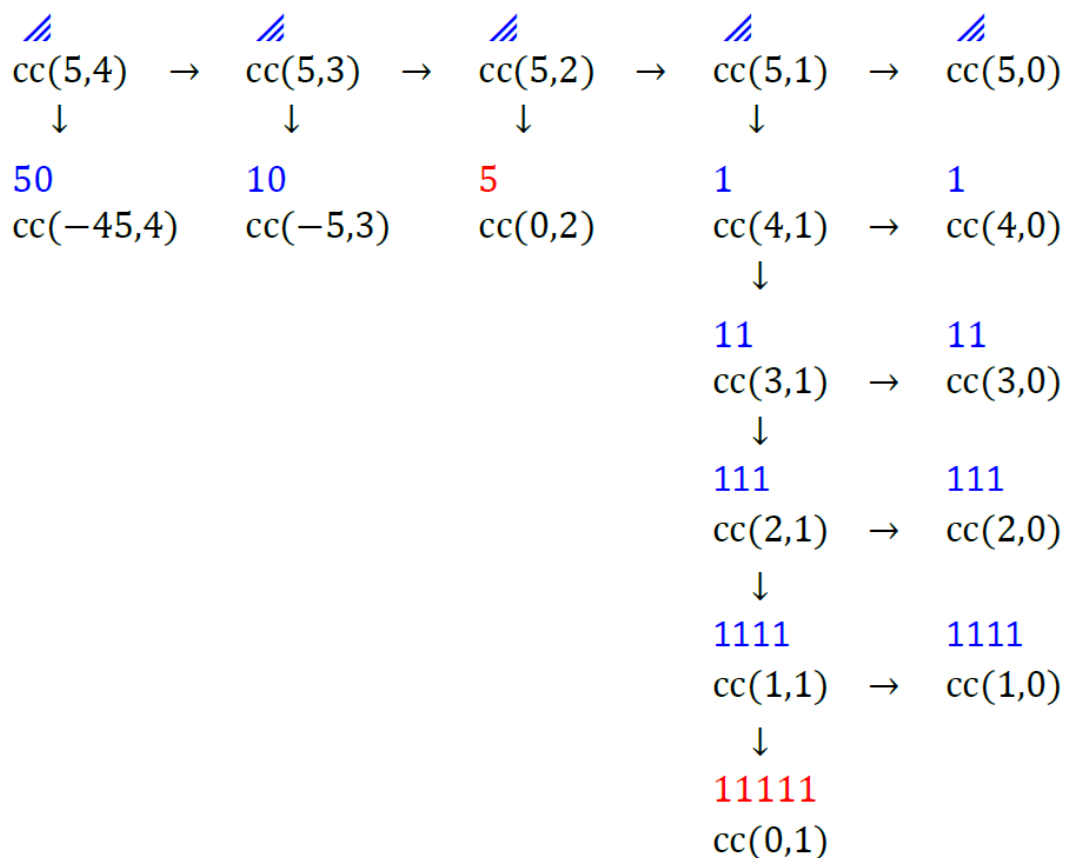
To generate all the ways to make change, we may resort to a stack for recording the coins changed. For the purpose of this homework, declare a [global](#) stack by

```
typedef struct stack {
    int top;

    int stk[100]          // arbitrarily assume that the array size is 100
} stack;
```

Since the array size is arbitrarily set to 100, the amount to be changed shall NOT exceed 100. That is, we shall restrict $n \leq 100$. (Why?)

For example, in the course of computing $cc(5,4)$, the contents of the stack are shown in blue or red color aside to each node representing a call to function cc .



Each time $n = 0$, one way of making change is found. In the diagram above, each red-colored stack contains the coin(s) changed for 5 dollars.

For this homework, you are asked to write several recursive functions. The task of each function is illustrated in the following sample run.

Enter amount of money: 10	... by function ui																										
<table border="0"> <tr> <td>\$1</td> <td>\$5</td> <td>\$10</td> <td>\$50</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>0</td> <td>2</td> <td>0</td> <td>0</td> </tr> <tr> <td>5</td> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>10</td> <td>0</td> <td>0</td> <td>0</td> </tr> </table>	\$1	\$5	\$10	\$50	0	0	1	0	0	2	0	0	5	1	0	0	10	0	0	0	<table border="0"> <tr> <td>...</td> <td>by function facevalue</td> </tr> <tr> <td rowspan="3">}</td> <td>...</td> </tr> <tr> <td>by function cc</td> </tr> <tr> <td></td> </tr> </table>	...	by function facevalue	}	...	by function cc	
\$1	\$5	\$10	\$50																								
0	0	1	0																								
0	2	0	0																								
5	1	0	0																								
10	0	0	0																								
...	by function facevalue																										
}	...																										
	by function cc																										
There are 4 ways to make change.	... by function ui																										
Enter amount of money:	... by function ui																										

Note that the contents of the stack have to be transformed to the required output format. For example, the preceding four ways to make change of 10 dollars are obtained from the stack contents (with the stack top on the right)

10
5 5
5 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
respectively.

The behaviors of the recursive functions **ui**, **facevalue**, and **cc** are described below.

void ui(void); // **ui** means “user interface”

- This function handles multiple inputs $n_1, n_2, \dots, n_k, k \geq 0$ by means of divide-and-conquer:
 $ui() =$ do nothing, if $k = 0$
 $=$ read in n_1 , process it, and call $ui()$ recursively to handle n_2, \dots, n_k , if $k > 0$
- It is invoked from the function **main** by the call **ui ()** ;
- Why do we need **ui**? Why don't we simply make **main** recursive? It's because the function **main** can only be recursive in C, and can't be recursive in C++.

void facevalue(int k);

1. This function displays the denominations $d[0], d[1], \dots, d[k-1]$ of the k kinds of coins in a line, prefixing each value with a \$ sign.

By the technique of divide-and-conquer, it may be defined as follow:

facevalue(k) = do nothing, if $k = 0$

= call facevalue($k - 1$) recursively to display

$d[0], d[1], \dots, d[k-2]$, and display $d[k-1]$, if $k > 0$

2. It is invoked from the function **ui** by the call

facevalue(kinds);

int cc(int n, int k);

1. This function implements the aforementioned function **cc**.
2. It is invoked from the function **ui** by the call

cc(n, kinds);

where n is the amount of money supplied by the user.

Requirements

1. Implement the three functions **ui**, **facevalue**, **cc** in recursive manner.
2. You may assume the input to be correct. ($n \leq 100$)
3. **Plagiarism is not allowed!**

Sample run

Enter amount of money: 20

\$1	\$5	\$10	\$50
0	0	2	0
0	2	1	0
5	1	1	0
10	0	1	0
0	4	0	0
5	3	0	0
10	2	0	0
15	1	0	0
20	0	0	0

There are 9 ways to make change.

Enter amount of money: 50

\$1	\$5	\$10	\$50
0	0	0	1
0	0	5	0
0	2	4	0
5	1	4	0
10	0	4	0
0	4	3	0
5	3	3	0
10	2	3	0
15	1	3	0
20	0	3	0
0	6	2	0
5	5	2	0
10	4	2	0
15	3	2	0
20	2	2	0
25	1	2	0
30	0	2	0
0	8	1	0
5	7	1	0
10	6	1	0
15	5	1	0
20	4	1	0
25	3	1	0
30	2	1	0
35	1	1	0
40	0	1	0
0	10	0	0
5	9	0	0
10	8	0	0
15	7	0	0
20	6	0	0
25	5	0	0
30	4	0	0
35	3	0	0

40	2	0	0
45	1	0	0
50	0	0	0

There are 37 ways to make change.

Enter amount of money: ^Z