Homework #8

Due date: 12/23

Coin change

Generate all the ways and count the number of ways to make change of n 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

```
const int 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

```
const int 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 manner similar to the recursive algorithm for the k-combination problem given in the lecture – the change may include or exclude coins of a specific kind.

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

Then,

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

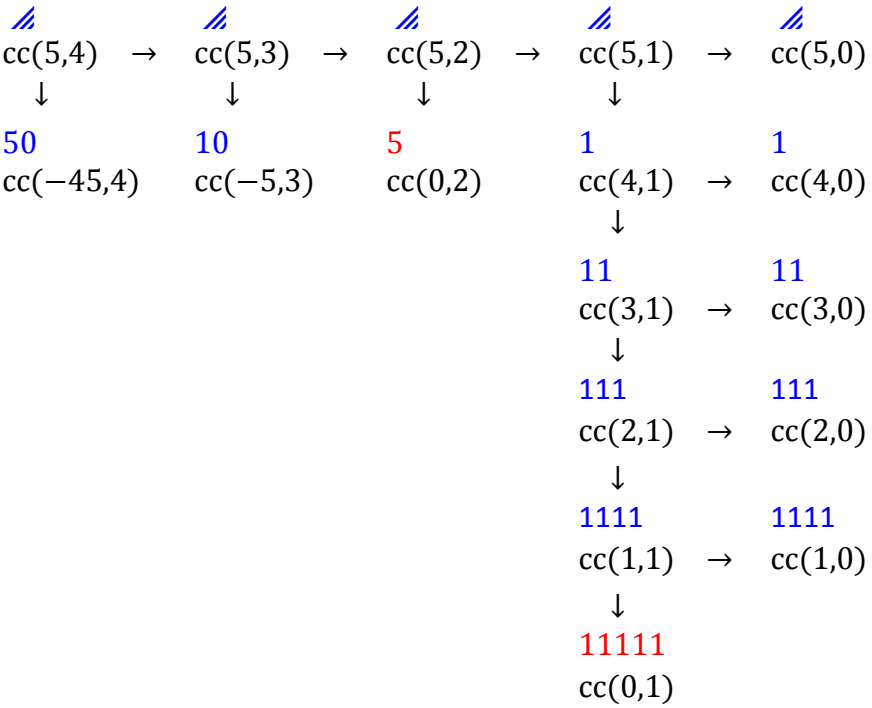
those that include coins of the kth kind — those that don't

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

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

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

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



Each time the function cc is called with $n = 0$, 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 by the following sample run.

```
Enter amount of money: 10          ··· by function ui
   $1   $5  $10  $50               ··· by function facevalue
    0    0    1    0
    0    2    0    0              ··· by function cc
    5    1    0    0
   10    0    0    0
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"

1 This function handles multiple inputs $n_1, n_2, \cdots, n_k, k \geq 0$ by means of divide-and-conquer:
ui() = do nothing, if $k = 0$
= read in n_1, process it, and call ui() recursively to handle $n_2, \cdots, n_k$, if $k > 0$

2 It is invoked from the function **main** by the call
ui();

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++.

void facevalue(int k);

1 This function displays the denominations $\mathrm{d}[0], \mathrm{d}[1], \ldots, \mathrm{d}[k-1]$ of the k kinds of coins in a line, prefixing each face value with a $ sign.
Again, by the technique of divide-and-conquer, it may be defined as follows:
facevalue(k) = do nothing, if $k = 0$
= call facevalue($k-1$) recursively to display $\mathrm{d}[0], \ldots, \mathrm{d}[k-2]$, and then display $\mathrm{d}[k-1]$, 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.

Bonus (20%)

As mentioned before, within function **cc**, whenever one way of making change is found, the contents of the stack have to be transformed to the required output format. You may choose to transform the stack contents *iteratively* or *recursively*. However, you are encouraged to do *both*.

If you do so, 20 bonus points are ready for you. In that case, test them as follows:

```
if (rand()%2==0)
```
    call your own *iterative* transformation function

```
else
```
    call your own *recursive* transformation function

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
```