Cash Flow

Let's represent cash using a dictionary (**dict**) where the keys are the values of coins or banknotes, and the values are the quantities of coins or banknotes. For example, **{100: 5, 50: 2, 10: 5, 1: 15}** represents 5 one-hundred-baht notes, 2 fifty-baht notes, 5 ten-baht coins, and 15 one-baht coins. Write the following functions (see the example below):

- **total(pocket)** returns the total amount of cash in **pocket**.
- **take(pocket, money)** adds the cash from **money** to **pocket** (both are **dict** containing money).
- **pay(pocket, amt)** deducts the amount **amt** (an integer) from **pocket**. The function will return a **dict** representing the cash paid. If the exact amount cannot be paid, return an empty **dict**.

  When paying cash, prioritize taking the highest value coins or bills from the pocket that can cover the payment. For example, if **pocket = {100: 5, 50: 2, 10: 5, 1: 15}** and the required payment is 57, you would take **one 50-baht bill** and **seven 1-baht coins**.

```
def total(pocket):


def take(pocket, money_in):


def pay(pocket, amt):


# The following command must be included when submiting to Grader
exec(input().strip())
```

Input

Python commands to test function behavior.

Output

Result from executing input Python commands.

Example

Input (from keyboard)	Output (on screen)
`p={100:2, 50:2, 5:2, 1:2};print(total(p))`	`312`
`p={100:5};take(p,{100:2, 1:3});print(p)`	`{100: 7, 1: 3}`
`p={100:5};take(p,{100:0, 1:0});print(p)`	`{100: 5, 1: 0}`
`p={10:5, 1:7};print(pay(p, 12));print(p)`	`{10: 1, 1: 2}` `{10: 4, 1: 5}`
`p={10:5, 1:7};print(pay(p, 18));print(p)`	`{}` `{10: 5, 1: 7}`
`p={10:5, 1:7};print(pay(p, 100));print(p)`	`{}` `{10: 5, 1: 7}`
`p={10:5, 1:7};print(pay(p, 57));print(p)`	`{10: 5, 1: 7}` `{10: 0, 1: 0}`