2005 Mid-Atlantic Regional Programming Contest
Practice Round

Welcome to the practice round for the 2005 Programming Contest. Before you start the contest, please be aware of the following notes:

1. There is one (1) practice problem. Please submit solutions or request clarifications **for this problem only.** Unless you have a real question about the problem, please submit at most one clarification request, and at most two runs. It is important that everyone have a chance to see how the system works.

2. After (or even before) completing the practice problem, please read all of the notes listed here. They are designed to help you solve the problems during the contest.

3. All solutions must read from standard input and write to standard output. In C this is scanf/printf, in C++ this is cin/cout, and in Java this is System.in/System.out. The judges will ignore all output sent to standard error. (You may wish to use standard error to output debugging information.) From your workstation you may test your program with an input file by redirecting input from a file:

   ```
   program < file.in
   ```

4. Solutions for problems submitted for judging are called runs. Each run will be judged. Runs for each particular problem will be judged in the order they are received. However, it is possible that runs for different problems (during the actual contest) may be judged out of order. For example, you may submit a run for B followed by a run for C, but receive the response for C first. DO NOT request clarifications on when a response will be returned. If you have not received a response for a run within 60 minutes of submitting it, **you may have a runner ask the site judge to determine the cause of the delay. Under no circumstances should you ever submit a clarification request about a submission for which you have not received a judgment.**

   The judges will respond to your submission with one of the following responses. In the event that more than one response is applicable, the judges may respond with any of the applicable responses.

Response	Explanation
Correct	Your submission has been judged correct.
Wrong Answer	Your submission generated output that is not correct or is incomplete.
Output Format Error	Your submission's output is not in the correct format or is misspelled.
Excessive Output	Your submission generated output in addition to or instead of what is required.
Compilation Error	Your submission failed to compile.
Run-Time Error	Your submission experienced a run-time error.
Time-Limit Exceeded	Your submission did not solve the judges' test data within 30 seconds.

5. A team's score is based on the number of problems they solve and penalty points, which reflect the amount of time and incorrect submissions made before the problem is solved. For each problem solved correctly, penalty points are charged equal to the time at which the problem was solved plus 20 minutes for each incorrect submission. No penalty points are added for problems that are never solved. Teams are ranked first by the number of problems solved and second by the fewest penalty points.

6. This problem set contains sample input and output for each problem. However, you may be assured that the judges will test your submission against several other more complex datasets, which will not be revealed until after the contest. Your major challenge is designing other input sets for yourself so that you may fully test your program before submitting your run. Should you receive an incorrect judgment, you should consider what other datasets you could design to further evaluate your program.

7. In the event that you feel a problem statement is ambiguous, you may request a clarification. Read the problem carefully before requesting a clarification. If the judges believe that the problem statement is sufficiently clear, you will receive the response, "The problem statement is sufficient, no clarification is necessary." If you receive this response, you should read the problem description more carefully. If you still feel there is an ambiguity, you will have to be more specific or descriptive of the ambiguity you have found. If the problem statement is ambiguous in specifying the correct output for particular input, please include that input data in the clarification request.

   Additionally, you may submit a clarification request asking for the correct output for input you provide. The judges will seek to respond to these requests with the correct output. These clarification requests will be answered only when no clarifications regarding ambiguity are pending. The judges reserve the right to suspend responding to these requests during the contest. If the provided input does not meet the specifications of the problem, or contains real numbers that are not representable using the available programming languages, the response "illegal input" will be returned.

   If a clarification, including output for a given input, is issued during the contest, it will be broadcast to all teams.

8. The submission of abusive programs or clarification requests to the judges will be considered grounds for immediate disqualification.

9. All lines of program input and output should end with a newline character (`\n`, `endl`, or `println()`).

10. Unless otherwise specified, all lines of program output should be left justified, with no leading blank spaces prior to the first non-blank character on that line.

11. All input sets used by the judges will follow the input format specification found in the problem description.

12. Unless otherwise specified, all numbers will appear in the input and should be presented in the output beginning with the `-` if negative, followed immediately by 1 or more decimal digits. If it is a real number, then the decimal point appears, followed by any number of decimal digits (for output of real number the number of decimal digits will be specified in the problem description as the "precision" in the problem description). All real numbers printed to a given precision should be rounded to the nearest value.

    In simpler terms, neither scientific notation nor commas will be used in numbers.

13. Good luck, and HAVE FUN!!!

Practice Problem: Making Change

Grocery stores have long struggled with how to avoid long checkout lines that leave customers frustrated. The "10 item or less" express line has been a common technique, but many stores a now trying to do even better by featuring self-service checkout lines. Such a system needs to have a mechanism to give correct change to the customer at the end of the transaction.

Write a program that, given the amount of change in the machine, can determine the quantities of each type of coins to return to the customer while minimizing the total number of coins dispersed.

Input

Input consists of one or more lines, each of the form:

```
Q D N P C
```

where Q is the number of quarters in the dispenser, D is the number of dimes, N the number of nickels, P the number of pennies, and C is the number of cents (0...99) owed to the customer.

End of the input is signaled by a line of 5 zeros.

Output

For each line of input data, your program should output either:

```
Dispense # quarters, # dimes, # nickels, and # pennies.
```

or

```
Cannot dispense the desired amount.
```

if it is not possible to dispense the exact amount.

Example

Input:

```
5 9 9 9 37
0 9 9 9 37
10 10 10 0 37
1 3 0 10 30
1 3 6 10 30
0 0 0 0 0
```

Output:

```
Dispense 1 quarters, 1 dimes, 0 nickels, and 2 pennies.
Dispense 0 quarters, 3 dimes, 1 nickels, and 2 pennies.
Cannot dispense the desired amount.
Dispense 0 quarters, 3 dimes, 0 nickels, and 0 pennies.
Dispense 1 quarters, 0 dimes, 1 nickels, and 0 pennies.
```