British Informatics Olympiad

Time allowed: 3 hours

The 2006 British Informatics Olympiad

Instructions

You should write a program for part (a) of each question, and produce written answers to the remaining parts. Programs may be used to help produce the answers to these written questions but are not always necessary.

You may use a calculator and the on-line help that your programming language provides. You should have a pen, some blank paper, and a blank floppy disk (or other storage device) on which to save your programs. You must not use any other material such as disks, files on a computer network, books or other written information. You may not communicate with anyone, other than the person invigilating this paper.

Mark the first page of your written answers with your name, age in years and school/college. Number all pages in order if you use more than one sheet. All of your computer programs should display your name and school/college when they are run, and the floppy disk you use to submit the programs should also show your name and school/college.

For your programs to be marked, the source code must be saved, along with executables if your language includes a compiler; this includes programs used to help answer written questions. You must clearly indicate the name given to each program on your answer sheet(s).

Sample runs are given for parts 1(a), 2(a) and 3(a). **`Bold text`** indicates output from the program, and `normal text` shows data that has been entered. The output format of your programs should follow the 'sample run' examples. Your programs should take less than 10 seconds of processing time for each test.

Attempt as many questions as you can. Do not worry if you are unable to finish this paper in the time available. Marks allocated to each part of a question are shown in square brackets next to the questions. Partial solutions (such as programs that only get some of the test cases correct within the time limit, or partly completed written answers) may get partial marks. Questions can be answered in any order, and you may answer the written questions without attempting the programming parts.

Hints

- If you can only see how to solve part of a problem it is worth writing a program that solves that part. We want to give you marks and questions are evaluated using multiple tests of differing difficulty. Remember, partial solutions may get partial marks.
- Question 2 is an implementation challenge and question 3 is a problem solving challenge.
- Most written questions can be solved by hand without solving the programming parts.
- Do not forget to indicate the name given to your programs on your answer sheet(s).

Question 1: ***Anagrams***

Two words are anagrams of each other if they can both be formed by rearranging the same combination of letters. For example, GADGET and TAGGED are anagrams because they both contain one occurrence of each of the letters A, D, E and T, and two occurrences of the letter G.

1(a) [24 marks]

Write a program which inputs two words (each of which will contain fewer than 10 uppercase letters) and then prints **`Anagrams`** if they are anagrams of each other, or prints **`Not anagrams`** otherwise. Your program should then terminate.

Sample run

```
SUMMER
RESUME
Not anagrams
```

1(b) [2 marks]

Suppose that any combination of the letters A, B and C makes a valid word. How many anagrams are there of the word ABC (including the word ABC itself)?

1(c) [4 marks]

Suppose that the only valid letters are A, B and C, and that any combination of these letters makes a valid word. A list is created that contains words that are 5 letters long; no two words on the list are anagrams of each other. What is the maximum number of words on list?

Question 2: *Rules*

Some computer programs try to prevent easily guessable passwords being used, by rejecting those that do not meet some simple rule. Your task for this question is to determine if a password is accepted by a rule.

The only characters that can occur in a password are the digits 0 to 9.

A rule is made up from a sequence of symbols, each with a special meaning. Symbols can be combined to make more complicated rules, and are written in order. Rules are described as follows:

- The symbol `x` means any digit is acceptable.

  For example: The rule `x` means the password has to be a single digit. The rule `xxx` means the password can be any combination of three digits.

- The symbol `u` means any digit that is higher than the previous digit.
- The symbol `d` means any digit that is lower than the previous digit.

  For example: The rule `xu` means the password has exactly two digits, the second of which is higher than the first; 46 would be accepted, 64 or 66 would be rejected. The rule `xud` means the password has exactly three digits, the second of which is higher than the first, and the third of which is lower than the second; 040 would be accepted.

Combinations of the `x`, `u` and `d` symbols (and only these symbols) can be placed inside brackets without changing their meaning; brackets must contain some symbols. The following two symbols can only follow an `x`, `u`, `d` or bracketed expression. If it is an `x`, `u` or `d` they change the meaning of the previous symbol. If they follow a bracketed expression, they change the meaning of everything inside the brackets.

- The symbol `?` means ignore or apply once.

  For example: The rule `xxx?` means the password must match either the rule `xx` or the rule `xxx`. The rule `x(xx)?` means the password must match either the rule `x` or the rule `xxx`.

- The symbol `*` means ignore, or repeat any number of times.

  For example: The rule `x(xu)*` means the password must match one of the rules `x`, `xxu`, `xxuxu`, `xxuxuxu`, etc...

Some combinations of these symbols are invalid, such as those that break any of the above conditions or, if for some interpretation of their `*` and `?` symbols, are equivalent to an invalid combination. For example, the rule `u` is invalid since there is no previous digit. The rule `x?u` is invalid, since it means match either the rule `xu` or the rule `u`, and the rule `u` is invalid. The rule `(x(ud)*)*` is invalid because the only valid symbols within a pair of brackets are `x`, `u` and `d`.

You will not be given any invalid rules.

2(a) [24 marks]

Write a program to test passwords.

Your program should first read in a single line, containing the rule; this rule will have between 1 and 12 symbols (inclusive, and including any brackets). You should then read in two more lines, each of which will contain a single password; these passwords will contain between 1 and 12 digits (inclusive).

No rule will contain more than two `?` or `*` symbols (combined).

You should output two lines, the first line indicating whether the first password is accepted by the rule, and the second line indicating whether the second password is accepted by the rule. Your should output **`Yes`** if a password is accepted and **`No`** if the password is rejected.

Sample run

```
xu*
02468
4688

Yes
No
```

2(b) [3 marks]

Consider the rule `x(xd)?(xx)?`. How many different passwords will be accepted this rule?

2(c) [3 marks]

A zig-zag password is one where the digits alternatively rise and fall, always rising initially if the password has more than one digit. For example, 5 and 08362 are zig-zag passwords. Find the rule that only matches zig-zag passwords and uses the smallest number of symbols. (No justification is necessary.)

2(d) [5 marks]

Is there a rule, containing exactly 11 symbols, that can only match a single password? Justify your answer.

Question 3: *Drats*

The game of *drats* is played by throwing drats (which are small and dart-like) at a long board which is split into 20 adjacent segments, numbered from 1 to 20. The first drat thrown scores twice the value of the segment it lands in, and all subsequent drats score the value of the segment they land in.

Drats that miss the board are thrown again, so every drat must score some points. Several drats are allowed to land in the same segment.

For example, in a game with three drats, if the first drat lands in segment 4, the second in segment 10 and the third in segment 4, the total score is 8 + 10 + 4 = 22.

Some scores can be obtained in multiple ways. For example, in a game with three drats, the score 6 can be obtained in four ways: double-1 + 1 + 3, double-1 + 2 + 2, double-1 + 3 + 1 and double-2 + 1 + 1.

3(a) [25 marks]

Write a program to determine how many different ways a score can be obtained with a given number of drats.

Your program should input a single line containing two integers. The first integer s ($1 \le s \le 200$) will indicate the required score. The second integer d ($1 \le d \le 8$) will indicate the number of drats.

Sample run

```
7 3
6
```

3(b) [2 marks]

What is the next lowest score, after 2, that cannot be obtained with two drats? What is the next lowest score, after 3, that cannot be obtained with three drats?

3(c) [4 marks]

Two games of drats are played. In each game three drats are thrown. In how many different ways can the drats be thrown if the total score in both games is equal and all six drats land in different segments?

3(d) [4 marks]

On a regulation drat board the segments do not appear in numerical order. The segments 1 to 10 count as low-scores and the segments 11 to 20 count as high-scores. Is it possible for the segments to be arranged in such an order that high and low scores alternate and, at the same time, odd and even segments alternate? Justify your answer.

Total Marks: 100 End of BIO 2006 Round One paper