Exam Imperative Programming

Friday, November 1, 2019, 15:00 h.

- You can earn 90 points. You will get 10 points for free. So, you can obtain 100 points in total, and your exam grade is calculated by dividing your score by 10.
- This exam consists of 5 problems. The first two problems are multiple choice questions. The problems 3, 4, and 5 are made using a computer. All problems are assessed by the Themis judging system. For each of the problems 3, 4 and 5, there are 10 test cases. Each test case is worth 10% of the points.
- Note that manual checking is performed after the exam. For example, if a recursive solution is requested then a correct iterative solution will be rejected after manual checking, even though Themis accepted it. Also, precomputed answers will be rejected.
- This is an open book exam! You are allowed to use the pdf of the reader (which is available in Themis), pdfs of the lecture slides (also available in Themis), the prescribed ANSI C book (hard copy) and a dictionary. Any other documents are not allowed. You are allowed to use previous submissions that you made to Themis.

Problem 1: Assignments (20 points)
For each of the following annotations determine which choice fits on the empty line (.....). The variables `x`, and `y` are of type `int`. Note that `A` and `B` (uppercase letters!) are specification constants (so not program variables).

1.1
```
/* x == A, y == B */
.....
/* x == A + B, y == A - B */

(a) x = x + y; y = x - y;
(b) y = x - y; x = x + y;
(c) x = x + y; y = x - 2*y;
```

1.2
```
/* x == A + B, y = A - B */
.....
/* x == B, y == A */

(a) x = (x - y)/2; y = x + y;
(b) y = x - y; x = x/2;
(c) x = x/2; y = x - y;
```

1.3
```
/* x + y == A, 2*x + y == B */
.....
/* x + y == A, 2*x + y - 1 == B */
(a) x++; y = y - 2;
(b) x--; y++;
(c) x++; y--;
```

1.4
```
/* x == 2*A - B, y == A + B */
y = x + y; x = x + y;
.....

(a) /* x == 3*A, y == 4*A + B */
(b) /* x == 5*A - B, y == 3*A */
(c) /* x == 3*A - 3*B, y == 2*B - A */
```

1.5
```
/* x == 2*A - B, y == A + B */
x = x + y; y = x + y;
.....

(a) /* x == 3*A, y == 4*A + B */
(b) /* x == 5*A - B, y == 3*A */
(c) /* x == A - 2*B, y == 3*B */
```

1.6
```
/* x == A, y == B */
x = x + 2*y; y = x - y; x = x - y;
.....
(a) /* x == A + B, y == A */
(b) /* x == B, y == A + B */
(c) /* x == B, y == A */
```

Problem 2: Time complexity (20 points)
In this problem the specification constant `N` is a positive integer (i.e. `N>0`). Determine for each of the following program fragments the *sharpest upper limit* for the number of calculation steps that the fragment performs in terms of `N`. For a fragment that needs `N` steps, the correct answer is therefore $O(N)$ and not $O(N^2)$ as $O(N)$ is the sharpest upper limit.

1. ```
   int s = 0;
   for (int i=0; i < N; i += s) {
     s++;
   }
   ```
   (a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

2. ```
   int s = 0;
   for (int i=1; i < N*N; i*=2) {
     s = s + i;
   }
   ```
   (a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

3. ```
   int s = 0;
   for (int i=0; i < 2*N; i+=2) {
     for (int j=N; j > 0; j-= 3) {
       s += i + j;
     }
   }
   ```
   (a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

4. ```
   int i = 0, s = 0;
   while (s < N) {
     s = (i % 2 == 0 ? 2*s : s+1);
     i++;
   }
   ```
   (a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

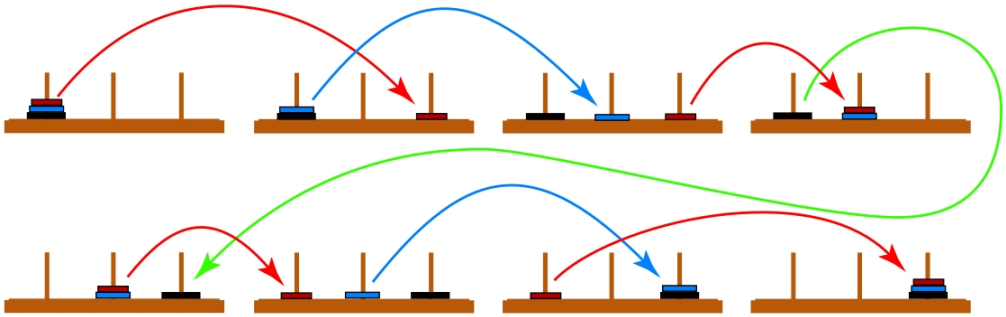
5. ```
   int s = 0;
   for (int i=N*N; i > 0; i = i/2) {
     for (int j=0; j < N; j++) {
       s += i + j;
     }
   }
   ```
   (a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

6. ```
   int k = 0, s = 0;
   for (int i=0; i < N; i+=k) {
     k++;
     for (int j=0; j*j < N; j++) {
       s = s + j;
     }
   }
   ```
   (a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

Problem 3: Towers of Hanoi (15 points)
In one of the lectures we discussed the *Towers of Hanoi* puzzle. It consists of three rods and a number of disks of different sizes, which can slide onto any rod. The rods are numbered 0 (left rod), 1 (middle rod), and 2 (right rod). The puzzle starts with the disks (on rod 0) in a neat stack in ascending order of size, the smallest at the top, thus making a conical shape. The objective of the puzzle is to move the entire stack to another rod (either rod 1 or rod 2), obeying the following simple rules:

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
- No larger disk may be placed on top of a smaller disk.

The following figure shows that the puzzle with 3 disks can be solved in 7 moves.



The input of this problem consists of two positive integers n (where $1 \leq n < 10$) and m, where n denotes the number of disks in the stack at the start of the game, and m the number of moves that follow on the remainder of the input. A move in which a disk is moved from rod `x` to rod `y` is denoted by `x->y`. Your program should output `SOLVED` if the m moves are according to the rules of the game, and the stack of disks has been moved from rod 0 to either rod 1 or rod 2 by performing the m moves. It should output `UNSOLVED` if the m moves are according to the rules of the game, however in the final position the stack of disks in not on rod 1 or rod 2. The program should output `ILLEGAL MOVE X`, where `X` is a number, if the `X`th move is illegal. A move is illegal when it places a larger disk on top of a smaller one (see example 3), or when it tries to move a disk from an empty pile (see example 4).

Example 1:

input:
```
3 7 0->2 0->1 2->1 0->2 1->0 1->2 0->2
```
output:
```
SOLVED
```

Example 2:

input:
```
3 5 0->2 0->1 2->1 0->2 1->0
```
output:
```
UNSOLVED
```

Example 3:

input:
```
3 7 0->2 0->2 2->1 0->2 1->0 1->2 0->2
```
output:
```
ILLEGAL MOVE 2
```

Example 4:

input:
```
3 7 0->2 0->1 2->1 2->0 1->0 1->2 0->2
```
output:
```
ILLEGAL MOVE 4
```

Problem 4: Removing Too Many Occurrences (15 points)
The input of this problem consists of two positive integers n and k, followed by a series of n non-negative integers (see examples). The output should be the same series of numbers in the same order as the input. However, if a number occurs at least k times in the input, then it should not occur in the output.

Example 1:
input:
```
8 3:1,2,4,3,2,1,7,8
```
output:
```
1,2,4,3,2,1,7,8
```

Example 2:
input:
```
8 2:1,2,4,3,2,1,7,8
```
output:
```
4,3,7,8
```

Example 3:
input:
```
13 3:1,2,3,4,5,5,4,3,2,1,1,2,3
```
output:
```
4,5,5,4
```

Problem 5: Increasing Fibonacci Sums (20 points)
In this problem we consider the following Fibonacci like series of numbers:

$$F(0) = 1, \;\; F(1) = 2, \;\; F(n) = F(n-1) + F(n-2) \;\; (\text{for } n > 1).$$

Note that this recurrence is the standard Fibonacci series shifted by two positions (i.e. the base cases differ from the standard Fibonacci series).

It is well known that every integer $n > 2$ can be written as a sum of increasing Fibonacci numbers. For example, the number 16 can be written as a sum of increasing Fibonacci numbers in 4 different ways:

$$\begin{aligned} F(0) + F(1) + F(3) + F(4) &= 1 + 2 + 5 + 8 = 16 \\ F(0) + F(1) + F(5) &= 1 + 2 + 13 = 16 \\ F(2) + F(3) + F(4) &= 3 + 5 + 8 = 16 \\ F(2) + F(5) &= 3 + 13 = 16 \end{aligned}$$

Note that 16 can also be written as $16 = F(4)+F(4) = 8+8$, however this is not a sum of increasing numbers. Moreover, note that a sum contains at least two terms, so $13 = F(5)$ is not a Fibonacci sum, while $13 = F(3) + F(4) = 5 + 8$ is a Fibonacci sum.
Write a program that reads from the input an integer n (where $2 < n \leq 500000$) and outputs the number of increasing Fibonacci sums that evaluate to n.

Example 1:
input:
```
16
```
output:
```
4
```

Example 2:
input:
```
42
```
output:
```
6
```

Example 3:
input:
```
100
```
output:
```
9
```