Exam Imperative Programming

Friday November 4th 2016

- 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, and must be answered on the (separate) answer sheet. The problems 3, 4, and 5 are made using a computer.
- The problems 3, 4, and 5 are assessed by the Justitia system. For each problem, 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 Justitia accepted it. Also, precomputed answers will be rejected.
- This is an open book exam! You are allowed to use the reader of the course, and the prescribed ANSI C book. Any other documents are not allowed. You are allowed to use previous submissions that you made to Justitia.
- Do not forget to hand in the answer sheet for the multiple choice questions. You are allowed to take the exam text home!

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

1.1
```
/* 6 < x + 2*y < 11 */
.....
/* 5 < x < 10 */
```

(a) `x = x - 2*y - 1;`
(b) `y = y/2 + x - 1;`
(c) `x = x + 2*y - 1;`

1.2
```
/* 2*x + 3*y == X, 2*y == Y */
.....
/* z == X + Y */
```

(a) `z = X + Y;`
(b) `z = 2*x + 5*y;`
(c) `z = x/2 - y;`

1.3
```
/* x == X + Y, y == 2*X - 7 */
.....
/* x == X + Y, y == Y*/
```

(a) `y = (2*x-y-7)/2;`
(b) `y = (y+7-2*x)/2;`
(c) `y = (y+7)/2 - x;`

1.4
```
/* x == X, y == Y */
y = x + y; x = x*(y-x);
.....
```

(a) `/* x == X*Y, y == X + Y */`
(b) `/* y == X*Y, x == X + Y */`
(c) `/* x + y == Y, x*(x-y) == X */`

1.5
```
/* x == X, y == Y */
x = y; y = x;
.....
```

(a) `/* x == Y, y == X */`
(b) `/* x == Y, y == Y */`
(c) `/* x == X, y == X */`

1.6
```
/* x == X, y == Y */
x = x + y; y = x - y;
.....
```

(a) `/* x == Y, y == X */`
(b) `/* x == X + Y, y == X - Y */`
(c) `/* x == X + Y, y == X */`

Problem 2: Time complexity (20 points)
In this problem the specification constant `N` is a non-zero natural number (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++) {
  for (int j=i; j > 0; j--) {
    s += i + j;
  }
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

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

3.
```
int n=N, a=2, p=1;
while (n > 0) {
  if (n%2 == 1) {
    p = p*a;
    n--;
  } else {
    a = a*a;
    n = n/2;
  }
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

4.
```
int i, j=0, s=0;
for (i=0; i < N; i++) {
  s += i;
}
for (i=0; i < s; i+=2) {
  j += i;
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

5.
```
int i=0, s=1;
for (i=0; i < 4*N; i+=2) {
  s = 2*s;
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

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

Problem 3: k**-even sequence** (15 points)
An integer sequence is called k-even if the sum of *any* k consecutive terms of the sequence is even.
For example, the sequence [1, 1, 2, 3, 5, 8] is 3-even, since all sums $1+1+2$, $1+2+3$, $2+3+5$, $3+5+8$ are even. The sequence is clearly not 2-even, since the sums $1+2$, $2+3$, and $5+8$ are odd.

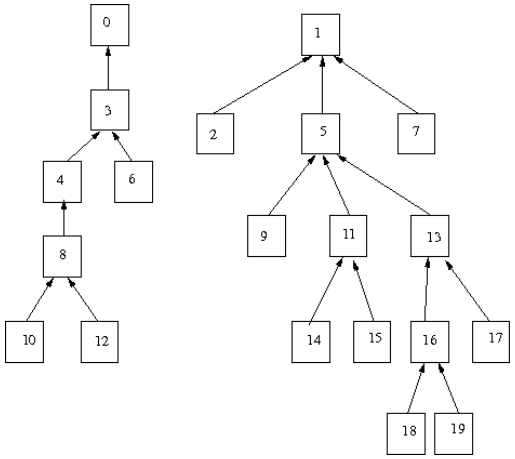
The input of this problem consists of a line containing two positive integers n and k (where $k < n$), followed by a line containing a sequence of n non-negative integers. The output of your program should be `YES` it the sequence is k-even, otherwise the output should be `NO`. The output should not contain any spaces, and must end with a newline (`\n`):

Example 1:
input:
```
6 3
1 1 2 3 5 8
```
output:
```
YES
```

Example 2:
input:
```
6 2
1 1 2 3 5 8
```
output:
```
NO
```

Example 3:
input:
```
10 4
1 3 5 7 9 11 13 15 17 19
```
output:
```
YES
```

Problem 4: Highest Common Ancestor (15 points)
A *rooted tree* is a tree structure in which one node is the root, while other nodes of the tree are descendants of the root. In the following figure, there are two rooted trees: one with root 0, and the other with root 1.



These trees can be represented efficiently with a one-dimensional array `parent[]` , where `parent[i]==j` means that node `j` is the parent of the non-root node `i`. For a root node `r` we have `parent[r]==r`. This representation is used in the following code fragment, which is available as a downloadable file `ancestor.c` from Justitia:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
  int a, b, parent[20] = {0,1,1,0,3,1,3,1,4,5,8,5,8,5,11,11,13,13,16,16};
  scanf("%d %d", &a, &b);
  /* place here your own code */
  return 0;
}
```

Note that `parent[i]<=i` for all indexes `i`. A node n is called a *common ancestor* of a and b if a and b are both descendants of n. Moreover, a node is called a *highest common ancestor* if there exists no other common ancestor with a higher value. For example, the nodes 18 and 9 have two common ancestors (5 and 1), of which 5 is the highest common ancestor. Clearly, the nodes 10 and 18 do not have a (highest) common ancestor.
The input of this problem are two integers `a`, and `b` (where $0 \leq$ `a`, `b` < 20). The output of your program should be the highest common ancestor of `a` and `b`, or `NONE` if no common ancestor exists.

Example 1:
input:
```
18 9
```
output:
```
5
```

Example 2:
input:
```
15 5
```
output:
```
5
```

Example 3:
input:
```
10 18
```
output:
```
NONE
```

Problem 5: Balanced subsequences (20 points)
The increasing sequence [1, 2, 3, 4, 5] has 9 non-empty increasing subsequences that contain as many even numbers as odd numbers. These 9 subsequences are: $[4,5],\ [3,4],\ [2,5],\ [2,3],\ [2,3,4,5],\ [1,4],\ [1,2],\ [1,2,4,5],\ [1,2,3,4]$.

The input of this problem consists of a line containing a positive integer n (where $n \leq 20$), followed by a line containing an increasing sequence of n positive integers. The output of your program should be the number of non-empty subsequences that contain as many even numbers as odd numbers.

The following incomplete code fragment is available in Justitia (file `balsubseq.c`). Download it and complete the code. You are asked to implement the body of the function `numberOfBalancedSubsets`. This function should call a *recursive* helper function (with suitably chosen parameters/arguments) that solves the problem. You are not allowed to make changes in the `main` function.

```
#include <stdio.h>
#include <stdlib.h>

int numberOfBalancedSubsets(int length, int a[]) {
  /* Implement the body of this function.
   * Moreover, this function should call a recursive helper
   * function that solves the problem.
   */
}

int main(int argc, char *argv[]) {
  int n, i, seq[20];
  scanf ("%d\n", &n);
  for (i=0; i < n; i++) {
    scanf("%d", &seq[i]);
  }
  printf("%d\n", numberOfBalancedSubsets(n, seq));
  return 0;
}
```

Example 1:
input:
```
5
1 2 3 4 5
```
output:
```
9
```

Example 2:
input:
```
5
2 4 6 8 10
```
output:
```
0
```

Example 3:
input:
```
9
1 2 3 4 5 6 7 8 9
```
output:
```
125
```