Exam Imperative Programming

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 `A` and `B` (uppercase letters!) are specification-constants (so not program variables).

1.1
```
/* x + 2*y == A, x + y == 2*A */
.....
/* x == -y */

(a) x = A/3;
(b) y = y/2;
(c) x = x/3;
```

1.2
```
/* 2*x + 3*y == A, x + y == B */
.....
/* x == B, y == A */

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

1.3
```
/* x + A == B, y + B == 2*A */
.....
/* x == B, y == A */

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

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

(a) /* x == B, y == A */
(b) /* x == A, y == B */
(c) /* x == y == B */
```

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

(a) /* x == A, z == A, x == B */
(b) /* x == A, y == A, z == A */
(c) /* x == B, y == B, z == A */
```

1.6
```
/* x == A, y == B */
x = x - y; z = x - y; y = z - y;
.....

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

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+=3) {
  for (int j=3*N; j > 0; j/=2) {
    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 s = 0;
for (int i=1; i < N*N; i*=3) {
  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 i=0, s=0;
while (s < N) {
  s += i;
  i++;
}
while (i > 0) {
  s += i;
  i--;
}
```
(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++) {
  for (j=0; j < 5*i; j += 2) {
    s += i + j;
  }
}
```
(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; i > 0; i--) {
  int d = 2 + i%5;
  for (int j=1; j < N; j*=d) {
    s += j*s;
  }
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

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

Problem 3: Missing number (15 points)
The input of this problem consists of a positive integer n (where $0 < n \leq 35000$), followed by n unique integers from the range $[0..n]$. Note that this range contains $n + 1$ numbers (since 0 and n are included in this interval). This means that exactly one integer from this range is missing on the input. Your program should determine this number, and output it.

Example 1:
input:
```
2
0 2
```
output:
```
1
```

Example 2:
input:
```
3
0 3 1
```
output:
```
2
```

Example 3:
input:
```
10
10 9 8 7 6 5 3 2 1 0
```
output:
```
4
```

Problem 4: Transposition cipher (15 points)
A very simple technique for encrypting a sentence is the so-called *transposition cipher*. A sentence with k characters is written row-wise in a 2-dimensional matrix of size m rows and n columns, where $m^2 \leq k < (m + 1)^2$ and n is the smallest integer such that $m \cdot n \geq k$. For example, the string `"PROGRAMMING IN C IS FUN"` has length $k = 23$, which means that $m = 4$ and $n = 6$. Spaces are replaced by the `'#'` character, and remaining entries of the square are also filled with the charcater `'#'`. The encoded string is simply the string that is obtained by reading the matrix column-wise (i.e. the transposition of the matrix). Note that the length of the encoded string is $m \cdot n$, which might be greater than k.
The example sentence is encoded as `"PMISRMN#OI#FGNCURG#NA#I#"`, which is clear from the following matrix.

P	R	O	G	R	A
M	M	I	N	G	#
I	N	#	C	#	I
S	#	F	U	N	#

The input of this problem consists of a line containing a sentence that is terminated by a period (i.e. the `'.'` character). The sentence consists of letters and spaces. The terminating period is not considered to be part of the sentence. The sentence is at most 400 characters long. The output of your program should be the encryption of the input sentence that is obtained using the transposition cipher technique. Make sure that the encrypted sentence is terminated by a newline (i.e. the `'\n'` character).

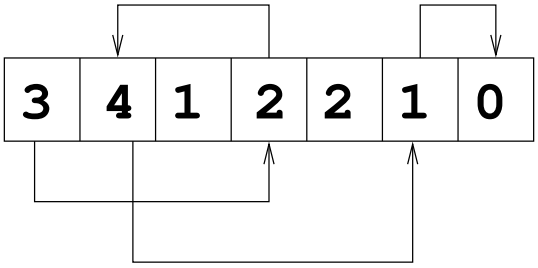
Example 1:
input:
```
PROGRAMMING IN C IS FUN.
```
output:
```
PMISRMN#OI#FGNCURG#NA#I#
```

Example 2:
input:
```
Hello world.
```
output:
```
Hore#llwdlo#
```

Example 3:
input:
```
ABCD.
```
output:
```
ACBD
```

Problem 5: Puzzle (20 points)
The input of this problem is a series of at most 20 non-negative integers, of which only the last one is a zero. This series represents actually a puzzle. Think of the series as a row of tiles with a number on each tile. On the first tile a coin is placed. Let the number on the tile on which the coin is placed be `n`. Now you are allowed to move the coin `n` positions to the right or to the left, but it may not move past either end of the row. For example, the only legal first move is to move the marker to the right because there is no room to move to the left. Given the new position of the coin, you can repeat this moving process. The objective is to find out whether you can move the coin to the last position in the row (i.e. the tile with the number zero).

For example, consider the series [3,4,1,2,2,1,0] which is depicted in the following figure.



As indicated by the arrows, it is indeed possible to find a series of movements that ends at the last position.
Another example would be the series [3,1,2,3,0]. In this series it is clearly not possible to reach the last position.

The following incomplete code fragment is available in Themis (file `puzzle.c`). Download it and complete the code. You are asked to implement the body of the function `isSolvable` that returns 1 if the puzzle can be solved, and 0 otherwise. The 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 isSolvable(int len, int series[]) {
  /* 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 len=0, series[20];
  do {
    scanf("%d", &series[len]);
    len++;
  } while (series[len-1] != 0);
  printf("%d\n", isSolvable(len, series));
  return 0;
}
```

Example 1:
input:
```
3 4 1 2 2 1 0
```
output:
```
1
```

Example 2:
input:
```
3 1 2 3 0
```
output:
```
0
```

Example 3:
input:
```
1 2 3 4 5 6 7 0
```
output:
```
1
```