Exam Imperative Programming

Tuesday 1 December 2015, 18:30-21:30h

- 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 (partly) assessed by the Justitia system. If Justitia accepts a solution, then half of the points for the corresponding problem are awarded automatically. The other half of the points are awarded by manual grading after the exam. The manual grading focuses on efficiency, style, and correctness. For example, if a recursive solution is requested, then Justitia will accept a correct iterative solution while the manual assessment will refute the solution.
- 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.
- 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 `A`, `B` and `C` (uppercase!) are specification-constants (so not program variables).

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

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

```
1.2 /* 4*x + 5*y == A */
    .....
    /* x + y == A */

    (a) x = 4*(x + y);
    (b) x = 3*(x + y) + y;
    (c) x = -3*(x + y) -y;
```

```
1.3 /* x == A*A*A, y == A*A, z == A */
    .....
    /* x == (A+1)*(A+1)*(A+1) */

    (a) x = 3*x + 3*y + z + 1;
    (b) x = x + 3*y + 3*z + 1;
    (c) x = x + y + 3*z + 3;
```

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

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

```
1.5 /* x == A + 2, y == 2*A */
    x = 3*x - 4; y = x - y;
    .....

    (a) /* x == (A + 6)/3, y == 6 - A */
    (b) /* x == A + 6, y == 2 - A      */
    (c) /* x == 3*A + 2, y == A + 2    */
```

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

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

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

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

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

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: counting *emirps* (10 points)
An *emirp* is a prime number of which its reversal (in decimal representation) is a *different* prime number.

An example of an emirp is 17, since 17 and its reverse 71 are both prime numbers. Another example is the prime 9781, because 1879 is also prime. Note that the prime number 11 is not an emirp, since its reverse is 11 itself.

Write a program that accepts on its input two integers a and b, where $0 \leq \texttt{a} < \texttt{b} \leq 10000$. Its output should be all emirps n, where $a \leq \texttt{n} \leq \texttt{b}$). Note that each emirp is printed on a separate line (without any spaces), and that the emirps must be printed in increasing order.

The following incomplete code is available from Justitia.

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

void printEmirps(int a, int b) {
  /* implement the body of this function (implement helper functions if needed) */
}

int main() {
  int a, b;
  scanf ("%d %d", &a, &b);  /* you may assume that 0<=a<b<=10000 */
  printEmirps(a, b);
  return 0;
}
```

Example 1:
input:
```
0 50
```
output:
```
13
17
31
37
```

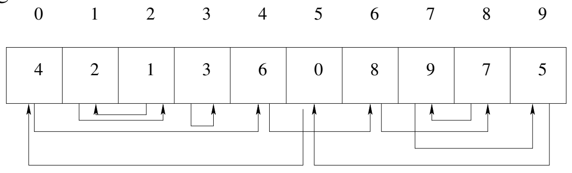
Example 2:
input:
```
1000 1050
```
output:
```
1009
1021
1031
1033
```

Example 3:
input:
```
9900 10000
```
output:
```
9923
9931
9941
9967
```

Problem 4: cycle detection (20 points)
The input of this problem consists of two lines. The first line consists of a positive integer n, where $1 \leq \texttt{n} \leq 100$. The second line consists of a permutation (i.e. rearrangement) of the numbers 0, 1, 2, .., n-1. An example would be the input:

```
10
4 2 1 3 6 0 8 9 7 5
```

This input represents the following figure:



We see that the figure contains 3 cycles. Let a[] be the array containing the number sequence. The first cycle starts at index 0, where we find that a[0]=4. Therefore, we jump to index 4, and detect that a[4]=6. So, the next step is to jump to index 6, and we find a[6]=8. We can repeat this process, until we reached the index 5, where we detect that a[5]=0, i.e. the starting index. This way, we find the cycle [0,4,6,8,7,9,5], which has length 7 (i.e. 7 elements). The other two cycles are [1,2] (with length 2) and the singleton cycle [3] (of length 1).

Write a program that accepts the above mentioned input format, and outputs for each cycle its smallest element and the length of the cycle (one cycle per line). The cycles should be printed in increasing order of the starting index.

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

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

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

Problem 5: recursion (20 points)
Consider the following sequence of positive integers: 8 7 2 3 1 4 5
We can construct from the sequence an arithmetic expression by replacing the spaces *between* two consecutive integers by a '+' or a '-' and evaluate the corresponding expression. For the given sequence, there exist precisely 3 expressions of this type that evaluate to 20:

$$8+7+2+3+1+4-5 \quad 8+7+2+3-1-4+5 \quad 8+7-2-3+1+4+5$$

In this problem, the input consist of two lines. The first line consists of 2 positive integers `n` and `g`. The number `n` is the length of the sequence (in this example 7). The number `g` denotes the goal (in this example 20). The second line of the input consist of `n` positive integers: the input sequence. Your program should output the number of expressions that evaluate to `g` (in this example 3). You may assume that $2 \leq \mathbf{n} \leq 20$.

The following incomplete code fragment is available in Justitia. Download it and complete the code. You are asked to implement the body of the function `plusmin`. 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. Nor are you allowed to introduce *global variables*.

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

int plusmin(int length, int a[], int n) {
  /* Implement the body of this function.
   * Call a recursive helper function that solves the problem.
   */
}

int main() {
  int len, n, i, a[100];
  scanf ("%d %d", &len, &n);
  for (i=0; i < len; i++) {
    scanf("%d", &a[i]);
  }
  printf("%d\n", plusmin(len, a, n));
  return 0;
}
```

Example 1:
input:
```
7 20
8 7 2 3 1 4 5
```
output:
```
3
```

Example 2:
input:
```
2 10
1 9
```
output:
```
1
```

Example 3:
input:
```
9 20
1 2 3 4 5 6 7 8 9
```
output:
```
0
```