Exam Imperative Programming

Friday, November 2, 2018, 14:00 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.
- 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` and `B` (uppercase letters!) are specification constants (so not program variables).

1.1
```
/* 10 < 2*x + 6*y < 20 */
.....
/* 10 < x < 15 */
```

(a)
```
x = 2*x + 6*y - 5;
```
(b)
```
x = 2*x + 6*y + 5;
```
(c)
```
x = x + 3*y + 5;
```

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

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

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

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

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

(a)
```
/* x == 4*A + 4*B, y == 2*A + 2*B */
```
(b)
```
/* x == 6*A + 4*B, y == 2*A + 2*B */
```
(c)
```
/* x == 2*A + 2*B, y == 2*A + 2*B */
```

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

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

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

(a)
```
/* x == 3*B, y == B - A, z == A + 2*B */
```
(b)
```
/* x == 2*B, y == B - 2*A, z == B - A */
```
(c)
```
/* x == A + 2*B, y == B - A, z == B - 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 j = 0, s = 0;
   for (int i = N; i > 0; i--) {
     j = j + i;
   }
   while (j > 5) {
     s = s + j%7;
     j--;
   }
   ```
   (a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

2. ```
   int i = 1, s = 0;
   while (i < N*N) {
     s = s + 3*i;
     i = 2*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 = 42; i < 7*N; i+=3) {
     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 = 0, s = 0;
   while (s < N) {
     s = i*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) $O(N^2)$

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

Problem 3: Sum of Pairs (15 points)
The input of this problem consists of two lines. The first line contains a positive integer n, and the second line consists of a series of positive integers terminated by a zero. The numbers in the series are guaranteed to be less than 1000.

The output of your program must be all unique pairs a and b drawn from the series such that $a + b = n$ and $a < b$. Your program should print these combinations in increasing order for a. If no such combination exists, then your program should print `NONE`.

Example 1:

input:
```
8
1 1 2 3 4 5 6 7 8 0
```
output:
```
1+7
2+6
3+5
```

Example 2:

input:
```
5
1 2 8 6 0
```
output:
```
NONE
```

Example 3:

input:
```
42
10 11 12 20 21 22 0
```
output:
```
20+22
```

Problem 4: Split3 (15 points)
Consider the series $[2, 5, 4, 1, 2]$. It has the sum $2+5+4+1+2 = 14$. We can split this series in 3 non-empty consecutive subseries in 6 ways:

- $[2, 5, 4]$, $[1]$, and $[2]$: the largest of the three sums, called the *maxsum*, is $2 + 5 + 4 = 11$.
- $[2, 5]$, $[4]$, and $[1, 2]$: the largest of the three sums is $2 + 5 = 7$.
- $[2, 5]$, $[4, 1]$, $[2]$: the largest of the three sums is $2 + 5 = 7$.
- $[2]$, $[5, 4, 1]$, and $[2]$: the largest of the three sums is $5 + 4 + 1 = 10$.
- $[2]$, $[5, 4]$, and $[1, 2]$: the largest of the three sums is $5 + 4 = 9$.
- $[2]$, $[5]$, and $[4, 1, 2]$: the largest of the three sums is $4 + 1 + 2 = 7$.

The objective of this problem is to split a series of positive integers into 3 non-empty subseries such that the maxsum is minimized. For the above example, this means that an optimal split has the minimal maxsum 7.

The input of this problem consists of two lines. The first line contains a positive integer n, and the second line consists of a series of n positive integers. The output of your program should be the minimal maxsum of the possible splits.

Example 1:

input:
```
5
2 5 4 1 2
```
output:
```
7
```

Example 2:

input:
```
5
1 2 8 6 1
```
output:
```
8
```

Example 3:

input:
```
10
1 1 1 1 1 1 1 1 1 1
```
output:
```
4
```

Problem 5: Pattern Matching (20 points)
This problem is about pattern matching. A *pattern* is a string that specifies (describes) the strings that match the pattern. A pattern may only consist of lower case letters from the alphabet (i.e. `a..z`), and question marks (i.e. the `?` character). A question mark may only follow a letter and indicates zero or one occurrence of the preceding character. For example, `colou?r` matches both `color` and `colour`.

The input of this problem is a line containing two non-empty strings: a pattern (with less than 30 characters) followed by a string (with less than 30 characters). The output of your program should be `MATCH` if the string can be produced by the pattern, otherwise the output should be `NO MATCH`.

The following incomplete code fragment is available in Themis (file `match.c`). Download it and complete the code. You are asked to implement the body of the function `isMatch` that returns 1 if there is a match, and 0 otherwise. The function `isMatch` must be a *recursive* function, or it should call a *recursive* helper function with suitably chosen parameters/arguments. You are not allowed to make changes in the `main` function.

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

int isMatch(char *pattern, char *string) {
  /* Implement the body of this function.
   * Moreover, this function must be recursive, or
   * it should call a recursive helper function.
   */
}

int main(int argc, char *argv[]) {
  char pattern[30], string[30];
  scanf("%s %s", pattern, string);
  if (isMatch(pattern, string) == 0) {
    printf("NO ");
  }
  printf("MATCH\n");
  return 0;
}
```

Example 1:
input:
`hello hello`
output:
`MATCH`

Example 2:
input:
`hello hi`
output:
`NO MATCH`

Example 3:
input:
`hell?o helo`
output:
`MATCH`

Example 4:
input:
`hell?o helllo`
output:
`NO MATCH`

Example 5:
input:
`h?e?l?i?o? hi`
output:
`MATCH`