Solutions and Evaluation Criteria for Exam #2 (11/4/2016)

Professors of PRO1

April 12th, 2016

1 General comments

As a general rule, for the exercises with no manual evaluation, the submission will be valid unless the program deliberately "cheats" or violates the requirements given in the statement (e.g., the input or substantial parts of it are stored in internal memory, includes libraries other than those specifically allowed, etc.)

2 Exercises

2.1 X90336: ITERATIVE Write rule

```
#include <iostream>
#include <string>
using namespace std;

void write_rule(const string& s, int W) {
  int len = s.length();
  for (int k = 1; k <= W/len; ++k)
    cout << s;
}

// example of alternative solution:
// void write_rule(const string& s, int W) {
//   int len = s.length();
//   while (len <= W) {
//     cout << s;
//     W -= len;
//   }
// }

int main() {
```

```
  string s; int W;
  while (cin >> s >> W) {
    write_rule(s, W);
    cout << endl;
  }
}
```

2.2 X68139: RECURSIVE Write rule

```
#include <iostream>
#include <string>
using namespace std;

void write_rule(const string& s, int W) {
  int len = s.length();
  if (len <= W) {
    cout << s;
    write_rule(s, W-len);
  }
}

int main() {
  string s; int W;
  while (cin >> s >> W) {
    write_rule(s, W);
    cout << endl;
  }
}
```

2.3 X58941: RECURSIVE Write each digit of a number as many times as the digit value plus 1

- ▷ Both procedures must be recursive, otherwise the submission will be invalidated (final grade = 0)
- ▷ Procedure `write_expanded` has weight 70%, procedure `write_digit` has weight 30%
- ▷ The recursive call to `write_expanded(n / 10)` must precede the call to `write_digit`; doing the calls in reverse order results in an incorrect program and is a very serious error, with penalty factor that leads to a manual grade between 0 and 1 (out of 10)
- ▷ Wrong conditions for the basic cases (in both procedures) are a serious error, the penalty factor is ≤ 0.3
- ▷ Unnecesarily complicated solutions, even if correct, will be penalized by a penalty factor between 0.3 and 0.5.

▷ If `write_digit` writes `endl` to `cout`, this is a mild error with penalty factor 0.7

```
#include <iostream>
using namespace std;

// Pre: 0<=d<=9 and 0<=x.
void write_digit(int d,int x) {
  if (x > 0) {
    cout << d;
    write_digit(d, x-1);
  }
}

// Pre: 0<n.
void write_expanded(int n) {
  if (n <= 9) {
    write_digit(n, n + 1);
  } else {
    write_expanded(n / 10);
    int last_digit = n % 10;
    write_digit(last_digit, last_digit + 1);
  }
}

// the following code is also correct with
// precondition n >= 0, but must do nothing if n == 0
//
// void write_expanded(int n) {
//    if (n != 0) {
//      write_expanded(n / 10);
//      int last_digit = n % 10;
//      write_digit(last_digit, last_digit + 1);
//    }
// }


int main() {
  int n;
  while (cin >> n) {
    write_expanded(n);
    cout << endl;
  }
}
```

2.4 X11093: ITERATIVE Write each digit of a number as many times as the digit value plus 1

In order to solve the problem iteratively, it is necessary to reverse the representation base 10 of n, but keeping track of the trailing 0's (the rightmost zeros of n become zeros to the left when reversing n. We give a first solution where the process of counting the number of rightmost/trailing zeros (and removing them) and reversing the representation of n is split in two parts. We then give another similar solution tha uses no additional functions/procedures. The third solution where both tasks (counting trailing zeros and reversing the remaining digits) are combined in a single procedure.

- ▷ The requested procedures must be iterative, otherwise the submission will be invalidated (final grade = 0)
- ▷ Procedure `write_expanded` has weight 80%, procedure `write_digit` has weight 20%
- ▷ The reversal of the representation of n base 10 and counting the number of trailing zeros can be done using one or more additional functions or procedures. A solution that does not use additional subprograms can be more complicated to understand, reasoning about correctness becomes more difficult, etc. A penalty factor between 0.3 and 0.6 will be applied to the score of the procedure `write_expanded` if it is unnecesarily complex, difficult to understand and messy. Not defining additional subprograms is fine as long as the code for `write_expanded` is short, easy to understand and clear, and well documented (with concise and explicit comments and/or well choosen variable names).
- ▷ Solutions that do not dealt with trailing zeros correctly, everything else being correct, will be penalized by a factor of 0.3
- ▷ If `write_digit` writes `endl` to `cout`, this is a mild error with penalty factor 0.7

First solution

```
#include <iostream>
using namespace std;

// Pre: 0<=d<=9 and 0<=x.
void write_digit(int d,int x) {
  for (int i = 1; i <= x; ++i)
    cout << d;
}

// Pre: n = n0 * 10^k > 0 for some k >= 0 and n0 mod 10 != 0
```

```
void count_and_remove_trailing_zeros(int n, int& tz) {
  tz = 0;
  while (n % 10 == 0) {
    n /= 10;
    ++tz;
  }
}
// Post: tz = k and n = n0, i.e., all trailing zeros
// have been removed from the original value of n

// Pre: n > 0, n = (d_r d_{r-1} ... d_0) in base 10
int reverse(int n) {
  int result = 0;
  while (n != 0) {
    result = result * 10 + (n % 10);
    n /= 10;
  }
  return result;
}
// Post: reverse(n) = (d_0 d_1 ... d_r) in base 10

// Pre: 0<n.
void write_expanded(int n) {
  int tz;
  count_and_remove_trailing_zeros(n, tz);
  n = reverse(n);
  while (n != 0) {
    int last_digit = n % 10;
    write_digit(last_digit, last_digit + 1);
    n /= 10;
  }
  write_digit(0, tz);
}

int main() {
  int n;
  while (cin >> n) {
    write_expanded(n);
    cout << endl;
  }
}
```

Second solution

```
#include <iostream>
using namespace std;
```

```
// Pre: 0<=d<=9 and 0<=x.
void write_digit(int d,int x) {
  for (int i = 1; i <= x; ++i)
    cout << d;
}

// Pre: 0<n.
void write_expanded(int n) {

  int trailing_zeros = 0;
  for(int nn = n; nn % 10 == 0; nn /= 10)
    ++trailing_zeros;

  int reverse = 0;
  for (int nn = n; nn != 0; nn /= 10)
    reverse = reverse * 10 + nn % 10;

  while (reverse != 0) {
    write_digit(reverse % 10, (reverse % 10) + 1);
    reverse = reverse / 10;
  }
  write_digit(0, trailing_zeros);
}

int main() {
  int n;
  while (cin >> n) {
    write_expanded(n);
    cout << endl;
  }
}
```

Third solution

```
#include <iostream>
using namespace std;

// Pre: 0<=d<=9 and 0<=x.
void write_digit(int d,int x) {
  for (int i = 1; i <= x; ++i)
    cout << d;
}

// Pre: n > 0, n = (d_r d_{r-1} ... d_0) in base 10
// let k be the largest index <= r such that
```

```
// d_k > 0, d_{k-1} = d_{k-2} = ... = d0 = 0
void reverse_and_count_zeros(int& n, int& tz) {
  int result = 0;
  tz = 0;
  // let j = the number of digits of the original n examined so far
  // Invariant: n = (d_r ... d_j) in base 10 and result = (d_0 ... d_{j-1})
  // tz is the largest value <= j-1 such that
  // d_0 = d_1 = ... = d_{tz-1} = 0
  while (n != 0) {
    result = result * 10 + (n % 10);
    if (n % 10 == 0 and result == 0)
      ++tz;
    n /= 10;
  }
  n = result;
}
// Post: n = (d_k ... d_r) in base 10, d_k > 0 and tz = k

// Pre: 0<n.
void write_expanded(int n) {
  int tz;
  reverse_and_count_zeros(n, tz);
  while (n != 0) {
    int last_digit = n % 10;
    write_digit(last_digit, last_digit + 1);
    n /= 10;
  }
  write_digit(0, tz);
}

int main() {
  int n;
  while (cin >> n) {
    write_expanded(n);
    cout << endl;
  }
}
```

ITERATIVE Write rule — X90336_en

Write an **ITERATIVE** procedure `write_rule(s, W)` that receives a string s and an integer $W \geq 0$ as parameters, and prints in the standard output (`cout`) the string s as many times as possible so that the total number of written characters does not exceed W. In other words, if the lenght of s is ℓ, then the number of times that `write_rule` prints s is the maximum k satisfying $\ell \cdot k \leq W$. The procedure does not print blank spaces nor an end of line. It just prints k times the string s.
For instance, if $s =$ `"Hello"`, then the length of s is $\ell = 5$. In such a case, if $W \leq 4$, then `write_rule(s, W)` prints nothing. Otherwise, if $5 \leq W \leq 9$ then `write_rule(s, W)` prints `"Hello"`. Otherwise, if $10 \leq W \leq 14$ then `write_rule(s, W)` prints `"HelloHello"`, etc.
Your solution must be **ITERATIVE** and use the following C++ code, modifying only the parts that are indicated.

```
#include <iostream>
#include <string>
using namespace std;

void write_rule(const string& s, int W) {
    // insert here your code. Your implementation must be ITERATIVE.
}

int main() {
    string s; int W;
    while (cin >> s >> W) {
        write_rule(s, W);
        cout << endl;
    }
}
```

==========
CATALÀ
==========
Escriu un procediment **ITERATIU** `write_rule(s, W)` que rep un string s i un enter $W \geq 0$ com a paràmetres, i imprimeix a la sortida estàndar (`cout`) l'string s tantes vegades com sigui possible de manera que el nombre total de caràcters escrits no excedeixi W. Dit d'altra manera, si la longitud d's és ℓ, llavors el nombre de vegades que `write_rule` imprimeix s és el màxim k tal que $\ell \cdot k \leq W$. El procediment no escriu cap espai en blanc ni cap salt de línia. Només escriu k vegades l'string s.
Per exemple, si $s =$ `"Hello"`, llavors la longitud d's és $\ell = 5$. En aquest cas, si $W \leq 4$ llavors `write_rule(s, W)` no escriu res. Si $5 \leq W \leq 9$ llavors `write_rule(s, W)` escriu `"Hello"`. Si $10 \leq W \leq 14$ llavors `write_rule(s, W)` escriu `"HelloHello"`, etc.
La teva solució ha de ser **ITERATIVA** i utilitzar el codi C++ donat a dalt, modificant exclusivament les parts indicades.
Exam score: 2.5 **Automatic part:** 100%

Input

The input is a sequence of pairs (s, W), where s is a string and W a non-negative integer.

==========

CATALÀ

==========

L'entrada és una seqüència de parells (s, W), on s és un string i W un enter no negatiu.

Output

For each pair (s, W) the program outputs a sequence consisting of as many repetitions of the string s as possible whose total length does not exceed W. Each sequence is followed by an end-of-line.

==========

CATALÀ

==========

Per a cada parell (s, W) el programa escriu una seqüència consistent en tantes repeticions de l'string s com sigui possible de manera que la longitud total no excedeix W. Cada seqüència està finalitzada amb un salt de línia.

Sample input

```
= 40
*-*- 55
..-- 13
x 30
//-// 21
```

Sample output

```
==========
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
..-..-..-..--
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
//-////-////-////-//
```

Problem information

Author : PRO1
Generation : 2016-04-08 14:51:08

RECURSIVE Write rule — X68139_en

Write a **RECURSIVE** procedure `write_rule(s, W)` that receives a string s and an integer $W \geq 0$ as parameters, and prints in the standard output (`cout`) the string s as many times as possible so that the total number of written characters does not exceed W. In other words, if the lenght of s is ℓ, then the number of times that `write_rule` prints s is the maximum k satisfying $\ell \cdot k \leq W$. The procedure does not print blank spaces nor an end of line. It just prints k times the string s.
For instance, if $s =$ `"Hello"`, then the length of s is $\ell = 5$. In such a case, if $W \leq 4$, then `write_rule(s, W)` prints nothing. Otherwise, if $5 \leq W \leq 9$ then `write_rule(s, W)` prints `"Hello"`. Otherwise, if $10 \leq W \leq 14$ then `write_rule(s, W)` prints `"HelloHello"`, etc.
Your solution must be **RECURSIVE** and use the following C++ code, modifying only the parts that are indicated.

```
#include <iostream>
#include <string>
using namespace std;

void write_rule(const string& s, int W) {
   // insert here your implementation, it  must be RECURSIVE.
}

int main() {
    string s; int W;
    while (cin >> s >> W) {
        write_rule(s, W);
        cout << endl;
    }
}
```

==========
CATALÀ
==========
Escriu un procediment **RECURSIU** `write_rule(s, W)` que rep un string s i un enter $W \geq 0$ com a paràmetres, i imprimeix a la sortida estàndar (`cout`) l'string s tantes vegades com sigui possible de manera que el nombre total de caràcters escrits no excedeixi W. Dit d'altra manera, si la longitud d's és ℓ, llavors el nombre de vegades que `write_rule` imprimeix s és el màxim k tal que $\ell \cdot k \leq W$. El procediment no escriu cap espai en blanc ni cap salt de línia. Només escriu k vegades l'string s.
Per exemple, si $s =$ `"Hello"`, llavors la longitud d's és $\ell = 5$. En aquest cas, si $W \leq 4$ llavors `write_rule(s, W)` no escriu res. Si $5 \leq W \leq 9$ llavors `write_rule(s, W)` escriu `"Hello"`. Si $10 \leq W \leq 14$ llavors `write_rule(s, W)` escriu `"HelloHello"`, etc.
La teva solució ha de ser **RECURSIVA** i utilitzar el codi C++ donat a dalt, modificant exclusivament les parts indicades.
Exam score: 2.5 **Automatic part:** 100%

Input

The input is a sequence of pairs (s, W), where s is a string and W a non-negative integer.
===
CATALÀ
===
L'entrada és una seqüència de parells (s, W), on s és un string i W un enter no negatiu.

Output

For each pair (s, W) the program outputs a sequence consisting of as many repetitions of the string s as possible whose total length does not exceed W. Each sequence is followed by an end-of-line.
===
CATALÀ
===
Per a cada parell (s, W) el programa escriu una seqüència consistent en tantes repeticions de l'string s com sigui possible de manera que la longitud total no excedeix W. Cada seqüència està finalitzada amb un salt de línia.

Sample input

```
= 40
*-*- 55
..-- 13
x 30
//-// 21
```

Sample output

```
========================================
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
..--..--..--
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
//-////-////-////-//
```

Problem information

Author : PRO1
Generation : 2016-04-08 14:52:49

RECURSIVE Write each digit of a number as many times as the digit value plus 1 — X58941_en

You have to write a **RECURSIVE** procedure `write_digit(d,x)` that receives a digit d and a natural number x, and writes x times the digit d in the standard output (`cout`). For example, the call `write_digit(3,5)` writes `33333`, whereas the call `write_digit(5,3)` writes `555`.
You have also to write a **RECURSIVE** procedure `write_expanded(n)` that receives a natural number $n > 0$, and writes in the standard output (`cout`) each digit of n as many times as the digit value plus 1. For instance, `write_expanded(315)` writes `333311555555` in the `cout`, whereas `write_expanded(204)` writes `222044444`. Your implementation of `write_expanded(n)` must conveniently use the function `write_digit(d,x)`.
Your implementation of both procedures must be **RECURSIVE** and use the following C++ code, modifying only the parts that are indicated. Notice that the main procedure below writes an end-of-line after each call to `write_expanded`. Thus, the procedures must not write the end-of-line.

```
#include <iostream>
using namespace std;

// Pre: 0<=d<=9 and 0<=x.
void write_digit(int d,int x) {
    // insert here your (RECURSIVE) code
}

// Pre: 0<n.
void write_expanded(int n) {
    // insert here your (RECURSIVE) code
}

int main() {
    int n;
    while (cin >> n) {
        write_expanded(n);
        cout << endl;
    }
}
```

==========
CATALÀ
==========
Escriu un procediment **RECURSIU** `write_digit(d,x)` que rep un dígit d i un número natural x, i escriu x vegades el dígit d en la sortida estàndard (`cout`). Per exemple, la crida `write_digit(3,5)` escriu `33333`, mentres que la crida `write_digit(5,3)` escriu `555`.
Escriu també un procediment **RECURSIU** `write_expanded(n)` que rep un número natural $n > 0$, i escriu a la sortida estàndar (`cout`) cada dígit d'n tantes vegades com el valor del dígit més 1. Per exemple, `write_expanded(315)` escriu 333311555555 en el

`cout`, mentre que `write_expanded(204)` escriu 222044444. La teva implementació de `write_expanded` ha de fer un ús convenient de la funció `write_digit(d,x)`.
La teva implementació dels dos procediments ha de ser **RECURSIVA** i utilitzar el codi C++ donat més amunt, modificant exclusivament les parts indicades. Fixa't que la funció `main` escriu un salt de línia després de cada crida a `write_expanded`, i per tant els teus procediments no han d'escriure salts de línia.
Exam score: 2.5 **Automatic part:** 50%

Input

The input is a sequence of strictly positive integers without leading zeros.
==========
CATALÀ
==========
L'entrada és una seqüència de enters estrictament positius sense zeros a l'esquerra.

Output

For each positive integer n in the input, the program writes the digits of n, each digit as many times as its value indicates plus 1, using the procedure `write_expanded`, followed by an end-of-line.
==========
CATALÀ
==========
Per a cada enter positiu n de l'entrada, el programa escriu els dígits d'n, cada dígit repetit tantes vegades com el valor del dígit més 1, usant el procediment `write_expanded`, seguit d'un salt de línia.

Sample input

```
315
204
112233
332211
200000
456789
999999
100000
```

Sample output

```
333311555555
222044444
111122222233333333
333333332222221111
22200000
444445555556666666777777778888888889999999999
999999999999999999999999999999999999999999999999999999999
1100000
```

Problem information

Author : PRO1
Generation : 2016-04-08 14:54:20

ITERATIVE Write each digit of a number as many times as the digit value plus 1 X11093_en

You have to write an **ITERATIVE** procedure `write_digit(d,x)` that receives a digit d and a natural number x, and writes x times the digit d in the standard output (`cout`). For example, the call `write_digit(3,5)` writes `33333`, whereas the call `write_digit(5,3)` writes `555`.
You have also to write an **ITERATIVE** procedure `write_expanded(n)` that receives a natural number $n > 0$, and writes in the standard output (`cout`) each digit of n as many times as the digit value plus 1. For instance, `write_expanded(315)` writes `333311555555` in the `cout`, whereas `write_expanded(204)` writes `222044444`. Your implementation of `write_expanded(n)` must conveniently use the function `write_digit(d,x)`.
Your implementation of both procedures must be **ITERATIVE** and use the following C++ code, modifying only the parts that are indicated. Notice that the main procedure below writes an end-of-line after each call to `write_expanded`. Thus, the procedures must not write the end-of-line.

```
#include <iostream>
using namespace std;

// define here additional functions and/or procedures
// if you need them

// Pre: 0<=d<=9 and 0<=x.
void write_digit(int d,int x) {
    // insert here your (ITERATIVE) code
}

// Pre: 0<n.
void write_expanded(int n) {
    // insert here your (ITERATIVE) code
}

int main() {
    int n;
    while (cin >> n) {
        write_expanded(n);
        cout << endl;
    }
}
```

==========
CATALÀ
==========
Escriu un procediment **ITERATIU** `write_digit(d,x)` que rep un dígit d i un número natural x, i escriu x vegades el dígit d en la sortida estàndard (`cout`). Per exemple, la crida `write_digit(3,5)` escriu `33333`, mentres que la crida `write_digit(5,3)` escriu `555`.

Escriu també un procediment **ITERATIU** `write_expanded(n)` que rep un número natural $n > 0$, i escriu a la sortida estàndar (`cout`) cada dígit d'n tantes vegades com el valor del dígit més 1. Per exemple, `write_expanded(315)` escriu 333311555555 en el `cout`, mentre que `write_expanded(204)` escriu 222044444. La teva implementació de `write_expanded` ha de fer un ús convenient de la funció `write_digit(d,x)`.
La teva implementació dels dos procediments ha de ser **ITERATIVA** i utilitzar el codi C++ donat més amunt, modificant exclusivament les parts indicades. Fixa't que la funció `main` escriu un salt de línia després de cada crida a `write_expanded`, i per tant els teus procediments no han d'escriure salts de línia.
Exam score: 2.5 **Automatic part:** 50%

Input

The input is a sequence of strictly positive integers without leading zeros.

==========

CATALÀ

==========

L'entrada és una seqüència de enters estrictament positius sense zeros a l'esquerra.

Output

For each positive integer n in the input, the program writes the digits of n, each digit as many times as its value indicates plus 1, using the procedure `write_expanded`, followed by an end-of-line.

Sample input

```
315
204
112233
332211
200000
456789
999999
100000
```

Sample output

```
333311555555
222044444
111122222233333333
333333332222221111
22200000
444445555556666666777777788888888999999999
999999999999999999999999999999999999999999999999999999
1100000
```

Problem information

Author : PRO1
Generation : 2016-04-08 14:55:27