CSc 600-01 (Section 1)

Homework 2 - Procedural Programming

prepared by Ilya Kopyl

CSc 600 Homework 2 - Procedural Programming

February 28, 2018

Homework is prepared by: Ilya Kopyl.
It is formatted in LaTeX, using TeXShop editor (under GNU GPL license).

1. Plateau program (max sequence length) (a combinatorial algorithm)

The array a(1..n) contains sorted integers. Write a function maxlen(a,n) that returns the length of the longest sequence of identical numbers (for example, if a= 1, 1, 1, 2, 3, 3, 5, 6, 6, 6, 6, 7, 9 then maxlen returns 4 because the longest sequence 6, 6, 6, 6 contains 4 numbers. Write a demo main program for testing the work of maxlen. Explain your solution, and insert comments in your program. The time complexity of the solution should be in O(n).

The answer is listed on the pages 2 through TBD.

A code listing of implementation of maxlen function:

```
unsigned int maxlen(int *a, unsigned int n)
{
    // handling the edge cases - arrays of size 0 and 1:
    if (n < 2)
        return n;

    unsigned int max_count, current_count, i;
    i = max_count = 0;
    current_count = 1;

    printf("    a[%d]=%d; \tcurrent_count=%d; \tmax_count=%d\n",
           i, a[i], current_count, max_count);

    for (i = 1; i < n; ++i)
    {
        if (a[i] == a[i-1])         // counting the current sequence
        {
            current_count++;

            // checking whether the longest sequence is at the end of array
            if(i == n-1 && current_count > max_count)
                max_count = current_count;
        }
        else                        // starting the count of the new sequence
        {
            // before resetting the counter, save it's value if it is above threshold
            if (current_count > max_count)
                max_count = current_count;

            // exit the loop if max_count is sufficiently large
            if (max_count >= n-i)
                break;

            current_count = 1;
        }

        printf("    a[%d]=%d; \tcurrent_count=%d; \tmax_count=%d\n",
               i, a[i], current_count, max_count);
    }
    return max_count;
}
```

The result of the program execution:

```
Array a:    1  1  1  2  3  3  5  6  6  6  6  7  9
    a[0]=1;     current_count=1;        max_count=0
    a[1]=1;     current_count=2;        max_count=0
    a[2]=1;     current_count=3;        max_count=0
    a[3]=2;     current_count=1;        max_count=3
    a[4]=3;     current_count=1;        max_count=3
    a[5]=3;     current_count=2;        max_count=3
    a[6]=5;     current_count=1;        max_count=3
    a[7]=6;     current_count=1;        max_count=3
    a[8]=6;     current_count=2;        max_count=3
    a[9]=6;     current_count=3;        max_count=3
    a[10]=6;    current_count=4;        max_count=3
Max sequence length of array a = 4

Array b:
Max sequence length of array b = 0

Array c:    12
Max sequence length of array c = 1

Array d:    16  16  16  18  18  20
    a[0]=16;    current_count=1;        max_count=0
    a[1]=16;    current_count=2;        max_count=0
    a[2]=16;    current_count=3;        max_count=0
Max sequence length of array d = 3

Array e:    0  0
    a[0]=0;     current_count=1;        max_count=0
    a[1]=0;     current_count=2;        max_count=2
Max sequence length of array e = 2

Array f: 0  1
    a[0]=0;     current_count=1;        max_count=0
Max sequence length of array f = 1

Array g:    1  2  3  3
    a[0]=1;     current_count=1;        max_count=0
    a[1]=2;     current_count=1;        max_count=1
    a[2]=3;     current_count=1;        max_count=1
    a[3]=3;     current_count=2;        max_count=2
Max sequence length of array g = 2
```

2. Integer plot function (find a smart way to code big integers)

Write a program BigInt(n) that displays an arbitrary positive integer n using big characters of size 7x7, as in the following example for BigInt(170):

Write a demo main program that illustrates the work of BigInt(n) and prints the following sequence of big numbers 1, 12, 123, 1234,..., 1234567890, one below the other.

The answer is listed on the pages TBD through TBD.

The code listing of the two-dimensional array that stores bit pattern of each BigInt digit. It is declared in the global space.

```
/**
 * Digits are stored as bit patterns of 8-bit unsigned integer (char) numbers.
 * For convenience of bit extracting & printing, the bit pattern of each char
 * is reversed.
 *
 * Each digit requires just 8 bytes of storage - which is polynomially smaller
 * than the storage in brute-force approach where each digit is represented by
 * a 2D array of 8x8 characters, with 64 bytes of storage per digit.
 */
#define NUMBER_OF_ROWS 8

const unsigned char BIG_DIGITS[NUMBER_OF_ROWS][10] =
{
    {    // row0 of all 10 digits
        0b00000000u, 0b00000000u, 0b00000000u, 0b00000000u, 0b00000000u,
        0b00000000u, 0b00000000u, 0b00000000u, 0b00000000u, 0b00000000u
    },
    {    // row1 of all 10 digits
        0b01111100u, 0b00110000u, 0b01111000u, 0b01111000u, 0b01100000u,
        0b11111100u, 0b01111000u, 0b11111110u, 0b01111000u, 0b01111000u
    },
    {    // row2 of all 10 digits
        0b11000110u, 0b00111000u, 0b11001100u, 0b11001100u, 0b01110000u,
        0b00001100u, 0b11001100u, 0b11000000u, 0b11001100u, 0b11001100u
    },
    {    // row3 of all 10 digits
        0b11000110u, 0b00110000u, 0b11001100u, 0b11000000u, 0b01101000u,
        0b00001100u, 0b00001100u, 0b01100000u, 0b11001100u, 0b11001100u
    },
    {    // row4 of all 10 digits
        0b11000110u, 0b00110000u, 0b01100000u, 0b00110000u, 0b01101100u,
        0b01111100u, 0b01111100u, 0b00110000u, 0b01111000u, 0b11111000u
    },
    {    // row5 of all 10 digits
        0b11000110u, 0b00110000u, 0b00110000u, 0b11000000u, 0b01100110u,
        0b11000000u, 0b11001100u, 0b00011000u, 0b11001100u, 0b11000000u
    },
    {    // row6 of all 10 digits
        0b11000110u, 0b00110000u, 0b00011000u, 0b11001100u, 0b11111110u,
        0b11000000u, 0b11001100u, 0b00001100u, 0b11001100u, 0b11001100u
    },
    {    // row7 of all 10 digits
        0b01111100u, 0b11111100u, 0b11111100u, 0b01111000u, 0b01100000u,
        0b01111100u, 0b01111000u, 0b00000110u, 0b01111000u, 0b01111000u
    }
};
```

Main program, excluding the declaration of BIG_DIGITS array:

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

void BigInt(unsigned int);
unsigned int getNumberOfDigits(unsigned int);

#define NUMBER_OF_ROWS 8
#define NUMBER_OF_BITS 8

/** BIG_DIGITS[][] is declared here; its declaration is listed on the previous page */

int main()
{
    BigInt(1);
    BigInt(12);
    BigInt(123);
    BigInt(1234);
    BigInt(1234567890);
    return 0;
}
void BigInt(unsigned int n)
{
    unsigned int numOfDigits, c;
    c = numOfDigits = getNumberOfDigits(n);
    int decimals[numOfDigits];

    // decomposing the number into an array of decimal digits
    do {
        decimals[c-1] = n % 10;
        c--;
    } while ((n /= 10));

    // printing all digits at once, row by row
    for (int row = 0; row < NUMBER_OF_ROWS; row++)
    {
        for (int digit = 0; digit < numOfDigits; digit++)
        {
            // iteratively extracting each bit from the bit pattern and printing it
            for (int bit = 0; bit < NUMBER_OF_BITS; bit++)
                printf("%c",
                ((BIG_DIGITS[row][decimals[digit]] >> bit) & 1) == 1 ? '@' : ' ');
        }
        puts("");
    }
}
unsigned int getNumberOfDigits(unsigned int n) {
    return (unsigned int) log10(n) + 1;
}
```

3. Array processing (elimination of three largest values) (one of many array reduction problems)

The array a(1..n) contains arbitrary integers. Write a function reduce(a, n) that reduces the array a(1..n) by eliminating from it all values that are equal to three largest different integers. For example, if a=(9, 1, 1, 6, 7, 1, 2, 3, 3, 5, 6, 6, 6, 7, 9) then three largest different integers are 6, 7, 9, and after reduction the reduced array would be a=(1, 1, 1, 2, 3, 3, 5), n=7. The time complexity of the solution should be in O(n).

The answer is listed on the pages TBD through TBD.

4. Iteration versus recursion (an opportunity for performance measurement)

Make a sorted integer array a[i]=i, i=0,...,n-1. Let bs(a, n, x) be a binary search program that returns the index i of the array a[0..n-1] where a[i]=x. Obviously, the result is bs(a, n, x)=x, and the binary search function can be tested using the loop

```
for (j=0; j < K; j++)
    for (i=0; i < n; i++)
        if (bs(a, n, i) != i)
            cout << "\nERROR";
```

Select the largest n your software can support and then K so that this loop with an iterative version of bs runs 3 seconds or more. Then measure and compare this run time and the run time of the loop that uses a recursive version of bs. Compare these run times using maximum compiler optimization (release version) and the slowest version (minimum optimization or the debug version). If you use a laptop, make measurements using AC power, and then the same measurements using only the battery. What conclusions can you derive from these experiments? Who is faster? Why?

The answer is listed on the pages TBD through TBD.

5. Iteration versus recursion (another opportunity for performance measurement)

Write a recursive function Frec(n) that computes Fibonacci numbers. Then write an iterative version of Fibonacci number function Fit(n). Functions Frec(n) and Fit(n) return the same value but with different performance.

Write the main program that discovers the value N10 so that Frec(N10) runs on your machine exactly 10 seconds. Then measure the run time of Fit(N10) and compute how many times is Fit(N10) faster than Frec(N10). Show what is N10 on your machine.

Notes:

1. When you measure the speed, your machine should be disconnected from the Internet, it should use the AC power supply, and it should run only one program (your performance measurement program).

2. In C++ you can measure current time in seconds using the following function:

```
double sec(void)
{
    return double(clock()) / double(CLOCKS_PER_SEC);
}
```

To measure the run time of fast programs you must repeat them many times inside a loop. Take care to eliminate the overhead generated by the loop.

The answer is listed on the pages TBD through TBD.