CS19001: Programming and Data Structures Laboratory

Soumyajit Dey, Aritra Hazra;
CSE, IIT Kharagpur

http://cse.iitkgp.ac.in/~aritrah/course/lab/PDS/Autumn2018/CS19101_PDS-Lab_Autumn2018.html

27-Aug-2018

Arrays

Arrays are our first example of *structured data*.

Declaration

```
int A[400];
float B[123];
```

Each of the elements A[0],A[1],...,A[399] is a variable of type int

Declaration and Initialization

```
int A[5] = { 51, 29, 0, -34, 67 };
char C[4] = { 'g', 'o', 'd', '\0' };
char C[4] =  "god";

printf("%d\n",A[5]); /*prints a garbage value*/
```

Nested loops

One or more loops can be nested inside another loop.

Sorting

Suppose you have an array A of n elements (say, integers). They are stored in the array locations

```
A[0],A[1],...,A[n-1]
```

We want to rearrange these integers in such a way that after the rearrangement we have

```
A[0] <= A[1] <= A[2] <= ..... <= A[n-1]
```

The resultant array is **sorted**. There are many such sorting methods. One is bubble sort.

Bubble sort

Code

```
for (i=n-2; i>=0; --i)
{
  for (j=0; j<=i; ++j)
  {
    if (A[j] > A[j+1])
    {
      t = A[j];
      A[j] = A[j+1];
      A[j+1] = t;
    }
  }
}
```

A[4]=4,3,2,1

i,j: A $\rightarrow$ A'
2,0: 4,3,2,1 $\rightarrow$ 3,4,2,1
2,1: 3,4,2,1 $\rightarrow$ 3,2,4,1
2,2: 3,2,4,1 $\rightarrow$ 3,2,1,4
bubble till position i=4-2=2.
1,0: 3,2,1,4 $\rightarrow$ 2,3,1,4
1,1: 2,3,1,4 $\rightarrow$ 2,1,3,4
bubble till position i=1
0,0: 2,1,3,4 $\rightarrow$ 1,2,3,4
bubble till position i=0

Programming Assignments

Complete and submit during lab

Assignment 1 [Merry-Go-Round]

You have to read an *n* element array, shift circularly (right or left) the array to *m* positions and print the array. For this, you have to implement the following steps:

1. Display the following to the user:
   ```
   List of Activities:
   0.Exit, 1.Enter Array, 2.Print Array,
   3.Right-Shift Array, 4.Left-Shift Array
   Enter Your Choice:
   ```
2. Take from user a choice (0, 1, 2, 3, 4).
3. If the user enter 0 as choice, then terminate the program.
4. For Choice 1, (a) user inputs the number of array elements, (b) user inputs all the elements of the array, and (c) you store the elements in the array.
   For Choice 2, print all the elements of the array.
   For Choice 3 and similarly Choice 4, (a) user inputs the number of positions to shift right (for 3) or left (for 4), and (b) you shift the elements in the array circularly. However, you may not print the shifted array here (since you already have a print option 2).
5. *Repeat* Step-1 (displaying list of activities).

Assignment 2 [Game-of-Thrones]

Suppose, there are n persons participating in a ‘Survive-or-Eliminate’ game. The persons are labelled using numbers $0, 1, 2, \ldots, n-1$ and they are standing in a circle. The rule of the game goes as follows:

- There will be a fixed number $m > 0$ and there will be a counting which starts from person labelled 0.
- The counting progresses over the persons circularly through, $0, 1, 2, \cdots, n-2, n-1, 0, 1, \cdots$.
- While counting, every m-th person is eliminated and not considered in the game anymore.
- The last remaining person is the Survivor (and hence the Winner).

Your task is to write a C-program which:

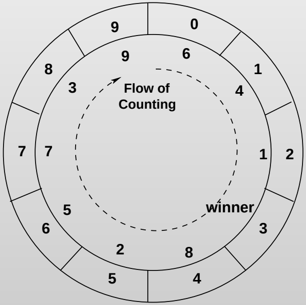
- Takes as input from user the integers, n and $m > 0$.
- Creates an n element array and initialize 0 to everyone to make them unmarked/survivor.
- If person labelled i is the k-th person to be eliminated, writes k in the i-th index of the array.
- Print the array informing which person is eliminated in which step and also reporting the survivor.

Assignment 2 [Game-of-Thrones]

A demo run of the program will create the following output.

```
Enter Number of Persons:  10
Enter Elimination Gap:  3

The Elimination Order:
Person #0 eliminated at Step #6
Person #1 eliminated at Step #4
Person #2 eliminated at Step #1
Person #3 Survives!!
Person #4 eliminated at Step #8
Person #5 eliminated at Step #2
Person #6 eliminated at Step #5
Person #7 eliminated at Step #7
Person #8 eliminated at Step #3
Person #9 eliminated at Step #9
```



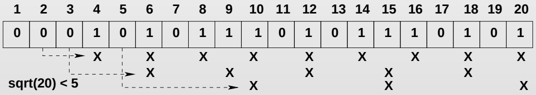
Assignment 3 [Prime-Time]

Write a C-program that reads a positive integer n and lists all primes between 1 and n. Use the sieve of Eratosthenes described below:

- Use an array of n cells indexed 1 through n. Since C starts indexing from 0, one may, for the ease of referencing, use an array of $n + 1$ cells (rather than n). Initially all the array cells are unmarked. During the process one marks the cells with composite indices. An unmarked cell holds the value 0, a marked cell holds 1. Henceforth, let us abbreviate ‘marking the cell at index i’ as ‘marking i’.
- Any positive integral multiple of a positive integer k, other than k itself, is called a proper multiple of k. Starting with $k = 2$, mark all proper multiples of 2 between 1 and n. Then look at the smallest integer > 2 that has not been marked. This is $k = 3$ and must be a prime. Mark all the proper multiples of 3 and then look at the next unmarked integer – this is $k = 5$. Then mark the proper multiples of 5 and so on. The process need continue as long as $k \leq sqrt(n)$, since every composite integer m, $1 < m \leq n$, must have a prime divisor $\leq sqrt(n)$.

Assignment 3 [Prime-Time]

- After the loop described in the last paragraph terminates, report the indices of the unmarked cells in your array. These are precisely all the primes in the range $1, 2, \ldots, n$.



Now adjust the bound n in order to detect the t-th prime, where t is ten-thousand plus your PC number. Report n, your *PC* number and the t-th prime only in the output. Report no other primes.
Remark: *No credit will be given, if any method other than the Prime-Time is used to detect the desired prime.*
Output: *Two runs with different values of n are shown below.*

```
n = 100000
The desired prime is not found.  Increase n.

n = 120000
10128-th prime is 106277
```

Thank You