**Exercise#1: find the max element in an array**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **23** | **65** | **12** | **87** | **12** | **343** | **098** | **12** | **76** | **312** |

* **preconditions:** The array arr is not null.

The length of the array arr is greater than or equal to 1.

* **Postconditions:** The function returns the maximum element in the array arr.
* **Loop invariant:** At the beginning of each iteration of the loop, the variable max contains the maximum element of the first i elements of the array arr.
* **Loop variant:** The loop variable i is incremented by 1 on each iteration of the loop.

**Pseudo code:**

public class Main

{

int arr []

static int max ()

{

int i;

int max = arr [0];

for (i = 1; i < arr. length; i++)

if (arr[i] > max)

max = arr[i];

return max;

}

**Exercise#2: Convert roman numbers to integers**

**Precondition**: The input string s is a valid Roman numeral containing only the characters 'I', 'V', 'X', 'L', 'C', 'D', and 'M'.

* **Postcondition**: The method romanToInt returns an integer value equal to the decimal representation of the input Roman numeral s.
* **Loop variant**: The loop variant is i, which is initialized to s. length () - 1 and decremented by 1 on each iteration until it reaches 0.
* **Loop invariant**: At the beginning of each iteration of the loop, result contains the sum of the decimal values of the Roman numerals processed so far, and prevValue contains the decimal value of the Roman numeral immediately following the numeral being processed on the previous iteration (or 0 if it is the first iteration). Additionally, the Roman numeral characters from index i+1 to the end of the string s have already been processed and their decimal values have been added to result.

**Pseudo code:**

int result = 0;

int prevValue = 0;

for (int i = s. length () - 1; i >= 0; i--) {

int currValue = getValue(s. charAt(i));

if (currValue < prevValue) {

result -= currValue;

} else {

result += currValue;

}

prevValue = currValue;

}

return result;

}

private static int getValue (char c) {

switch(c) {

case 'I': return 1;

case 'V': return 5;

case 'X': return 10;

case 'L': return 50;

case 'C': return 100;

case 'D': return 500;

case 'M': return 1000;

default: return 0;

}

**Exercise#3: Move zeroes to the end of Array**

**Takeaway:**

* **Learn problem-solving using single loop and two pointers approach.**

Some mistakes in loop programming

**Precondition:** The input integer array arr is not null.

**Postcondition:** The elements of the input array arr are rearranged such that all non-zero elements are moved to the beginning of the array, in their original order, followed by all zero elements.

**Loop variant**: The loop variant is i, which is initialized to 0 and incremented by 1 on each iteration until it reaches n-1.

**Loop invariant:** At the beginning of each iteration of the loop, the elements of the input array arr with indices from 0 to count-1 have already been rearranged such that they contain all non-zero elements in their original order, and the elements with indices from count to i-1 contain only zeros. Additionally, the element at index i has not yet been processed.

**Loop correct programming**

int n = arr. length;

int count = 0;

for (int i = 0; i < n; i++) {

if (arr[i]! = 0) {

int temp = arr[i];

arr[i] = arr[count];

arr[count] = temp;

count++;

}