# Recitation 2 (Scope + arrays + pointers + pass-by-reference)

**Question 1**: Write a global function that takes an integer array and its size and returns the minimum and maximum values in this array. You may assume that the size of this array is at least 1.

This function should return two values. You may return one of these values using the return value of this function. However, for returning the second value, you need to use the parameter list. (Or alternatively, you can define the return type as void and you can use the parameter list for returning both values.)

There are many alternatives to implement this function. Below shows only a subset of these alternatives.

```
// This function uses pass-by-reference

void MinMax(int arr[], int size, int &min, int &max){
     min = max = arr[0];

     for (int i = 1; i < size; i++){
             if (min > arr[i])
                   min = arr[i];
             if (max < arr[i])
                   max = arr[i];
     }
}
// You can also use int *arr instead of int arr[] in the parameter list,
// they are the same. In other words, you can define the function prototype as
// void MinMax(int *arr, int size, int &min, int &max)
```

```
// This function simulates pass-by-reference via pointers.

void MinMax(int arr[], int size, int *min, int *max){
     *min = *max = arr[0];

     for (int i = 1; i < size; i++){
             if (*min > arr[i])
                   *min = arr[i];
             if (*max < arr[i])
                   *max = arr[i];
     }
}
// Similarly, you can define the function prototype as
// void MinMax(int *arr, int size, int *min, int *max)
```

```
// The main function shows how to call these functions

int main(){
     int a[] = {4,56,7,-1,3,10,12}, min, max;

     MinMax(a,7,min,max);
     MinMax(a,7,&min,&max);

     int *b = new int[5];

     for (int i = 0; i < 5; i++)
             b[i] = i;

     MinMax(b,5,min,max);
     MinMax(b,5,&min,&max);
     delete []b;
     return 0;
}
```

**<u>Question 2</u>:** What is the output of the following program?

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

void func1(int a, int &b){
     static int c = 2;

     a = 3;
     b = 4;
     c *= a;

     cout << "func1: a = " << a << " b = " << b << " c = " << c << endl;
}
void func2(int *a, int b){
     int c = 2;

     *a = 5;
     b = 6;
     c *= *a;

     cout << "func2: a = " << *a << " b = " << b << " c = " << c << endl;
}
int main(){
     int x = 15, y = 25;

     cout << "Initially: x = " << x << " y = " << y << endl;

     cout << "After calling the functions for the first time" << endl;

     func1(x,y);
     cout << "After func1: x = " << x << " y = " << y << endl;

     func2(&x,y);
     cout << "After func2: x = " << x << " y = " << y << endl;

     cout << "After calling the functions for the second time" << endl;

     func1(x,y);
     cout << "After func1: x = " << x << " y = " << y << endl;

     func2(&x,y);
     cout << "After func2: x = " << x << " y = " << y << endl;

     return 0;
}
```

```
Initially: x = 15 y = 25
After calling the functions for the first time
func1: a = 3 b = 4 c = 6
After func1: x = 15 y = 4
func2: a = 5 b = 6 c = 10
After func2: x = 5 y = 4
After calling the functions for the second time
func1: a = 3 b = 4 c = 18
After func1: x = 5 y = 4
func2: a = 5 b = 6 c = 10
After func2: x = 5 y = 4
```

**<u>Question 3</u>:** What is the output of the following program?

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

int a = 5;

void f1(int a){
     cout << a << endl;
}

void f2(){
     cout << a << endl;
}

int main(){
     int a = 10, b = 7;

     cout << a << endl;

     if (a > b){
            int a = 40;
            cout << a << endl;
     }

     cout << a << endl;

     f1(b);
     f2();

     cout << a << endl;

     return 0;
}
```

```
10
40
10
7
5
10
```

**Question 4**: What is the output of the following program?

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

int main(){
     int A, B;
     int *x = &A, *y = &B;
     x = &B;
     *x = 24;
     cout << "B   = " << B   << endl;
     cout << "*y  = " << *y  << endl;

     y = &A;
     *y = 32;
     cout << "&*y = " << &*y << endl;
     cout << "*x  = " << *x  << endl;
     cout << "A  = "  << A   << endl;
     cout << "&A  = " << &A  << endl;

     return 0;
}
```

```
B  = 24
*y  = 24
&*y = 0x7fff5aaf3bc4
*x  = 24
A  = 32
&A  = 0x7fff5aaf3bc4
```

**Question 5**: What is the output of the following program?

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

void displayArray(const int A[], const int no){
     for (int i = 0; i < no; i++)
      cout << A[i] << "\t";
     cout << endl;
}
void modifyArray(int *arr, int index){
     arr[index] = arr[0] + 10;
}
int main(){
     int B[] = {1,2,3,4,5,6,7,8};

     displayArray(B,8);
     displayArray(B+3,4);
     modifyArray(B,2);
     modifyArray(B+5,1);
     modifyArray(&(B[3]),4);
     displayArray(B,8);

     return 0;
}
```

```
1       2       3       4       5       6       7       8
4       5       6       7
1       2       11      4       5       6       16      14
```

**Question 6:** Write a global function called **splitArray** that takes an array, its size, and two indices as input, creates two new arrays from the input array, and returns these two arrays together with their sizes. The first output array will contain the input array entries in between the given two indices and the second output array will contain the rest of the entries. Note that the input array should remain unchanged.

For example, if the input array is [ 45 2 6 88 12 10 3 2 10 21 5 ] and the two input indices are 3 and 7, you should return [ 88 12 10 3 2 ] as the first output array and 5 as its size, and [ 45 2 6 10 21 5 ] as the second output array and 6 as its size.

In this question, you may assume that both of the indices are valid (e.g., they are not out of the range, the first index is always smaller than the second one, the difference of index values is always greater than 0 and smaller than the size of the original array).

This function returns four outputs (two arrays and two sizes). Thus, you cannot simply use the return value of that function. You need to pass these outputs to the function using pass-by-reference or you need to simulate pass-by-reference using pointers. The following two solutions show these alternatives.

```
// This function uses pass-by-reference

void splitArray(int *arr, int size, int ind1, int ind2,
                 int *&first, int &firstSize, int *&second, int &secondSize){

     firstSize = ind2 - ind1 + 1;
     secondSize = size - firstSize;

     first = new int[firstSize];
     second = new int[secondSize];

     for (int i = 0; i < ind1; i++)
            second[i] = arr[i];

     for (int i = ind1; i <= ind2; i++)
            first[i - ind1] = arr[i];

     for (int i = ind2 + 1; i < size; i++)
            second[i - firstSize] = arr[i];
}
```

```
// This function simulates pass-by-reference via pointers.

void splitArray(int *arr, int size, int ind1, int ind2,
                 int **first, int *firstSize, int **second, int *secondSize){

     *firstSize = ind2 - ind1 + 1;
     *secondSize = size - *firstSize;

     *first = new int[*firstSize];
     *second = new int[*secondSize];

     for (int i = 0; i < ind1; i++)
            (*second)[i] = arr[i];

     for (int i = ind1; i <= ind2; i++)
            (*first)[i - ind1] = arr[i];

     for (int i = ind2 + 1; i < size; i++)
            (*second)[i - *firstSize] = arr[i];
}
```

```
// The following main function shows how to call these functions
int main(){
     int arr[] = {45, 2, 6, 88, 12, 10, 3, 2, 10, 21, 5};
     int *a1, a1Size, *a2, a2Size;

     splitArray(arr,11,3,7,a1,a1Size,a2,a2Size);

     // ...

     delete []a1;
     delete []a2;
     // you need to call delete for a1 and a2 before calling splitArray2
     // in order to prevent memory leak

     splitArray(arr,11,3,7,&a1,&a1Size,&a2,&a2Size);
     // ...

     delete []a1;
     delete []a2;

     // you should not call delete for arr since it was not dynamically created

     return 0;
}
```

**<u>Question 7</u>:** Write a <u>global function</u> called **selectNegatives** that takes an array and its size, creates an output array containing only the negative entries of the input array, and returns this output array together with its size.

For example, if the input array is [ -45 11 6 38 -12 0 ], you should return [ -45 -12 ] as the output array and 2 as its size. If the input array does not contain any negative items, you should return NULL for the output array and 0 as its size.

This function returns two outputs (one array and one size). Thus, you may use the return value to return one of these outputs but you cannot use the return value for both of the outputs. For this reason, you need to use pass-by-reference or simulate pass-by-reference using pointers. Below are two alternatives. ***You may want to code the other alternatives as an exercise. To ensure that you understood the pointers and arrays, you may also want to write a caller function that calls each of these alternatives.***

```
// This function uses the return value to return the output array
// and pass-by-reference to return the size of this output array

int *selectNegatives(int *arr, int size, int &outputSize){

     // first calculate the number of negative entries in the input array
     // otherwise you cannot make allocation for the output array
     outputSize = 0;

     for (int i = 0; i < size; i++)
            if (arr[i] < 0)
                  outputSize++;

     // if the outputSize is zero, return the NULL value since it means that
     // the input array does not contain any negative items
     if (outputSize == 0)
            return NULL;

     // now make the allocation for the output array and copy the negative items
     int *output = new int[outputSize];
     int cnt = 0;

     for (int i = 0; i < size; i++)
      if (arr[i] < 0)
            output[cnt++] = arr[i];

     return output;
}
```

```
// This function uses the return value to return the size
// and simulate pass-by-reference to return the output array

int selectNegatives(int *arr, int size, int **output){

     int outputSize = 0;

     for (int i = 0; i < size; i++)
            if (arr[i] < 0)
                  outputSize++;

     if (outputSize == 0){
            *output = NULL;
            return outputSize;
     }

     *output = new int[outputSize];
     int cnt = 0;

     for (int i = 0; i < size; i++)
            if (arr[i] < 0)
                  (*output)[cnt++] = arr[i];

     return outputSize;
}
```

Both of these functions do not change the input array. Now, modify the code such that it eliminates the negative entries from the input array and changes its size accordingly. Below is the modified code for the first function. As an exercise, you may modify the second function.

```
// Here you can define the function prototype in different ways. For example,
// instead of using pass-by-reference, you can simulate pass-by-reference.
// In that case, you need to modify your statements as well.

int *selectNegativesModified(int *&arr, int &size, int &outputSize){

     // calculate the number of negative entries in the output array
     outputSize = 0;

     for (int i = 0; i < size; i++)
            if (arr[i] < 0)
                  outputSize++;

     // if this number is 0, it means that there are no negative entries in the
     // input array, and thus, the output array is NULL and the input array
     // and its size remain unchanged
     if (outputSize == 0)
            return NULL;

     // if there are negative entries in the input array, allocate the output
     // array and another temporary array that will contain the non-negative
     // entries. Here we have to allocate a temporary array since we will still
     // use the input array to read its contents. Once we are done with it, we
     // will deallocate the input array and assign the temporary array to it
     int *output = new int[outputSize];
     int tmpSize = size - outputSize;
     int *tmpArr, cnt = 0, tmpCnt = 0;

     // tmpSize = 0 means that all entries in the input array are negatives
     // so that the input array should become empty after the function call
     if (tmpSize == 0)
            tmpArr = NULL;
     else
            tmpArr = new int[tmpSize];

     for (int i = 0; i < size; i++)
            if (arr[i] < 0)
                  output[cnt++] = arr[i];
            else
                  tmpArr[tmpCnt++] = arr[i];

     // as aforementioned, we will deallocate the input array and assign the
     // temporary array, which contains non-negative values, to it. We are also
     // changing the size of the input array
     delete []arr;
     arr = tmpArr;
     size = tmpSize;

     return output;
}
```

**Question 8**: Write a global function called **rowSum** that takes a two-dimensional array of integers and the dimensions of this array as inputs, computes the sum of the values in individual rows, and returns these sums in a separate one-dimensional array. Each element in the one-dimensional array should contain the sum of the corresponding row in the two-dimensional array.

For example, the output array will be [ 8 2 3 10 11] for the following two-dimensional input array:

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 2 |
| 3 | -1 | 1 | -1 |
| 2 | 1 | 2 | -2 |
| 1 | 2 | 5 | 2 |
| 4 | 1 | 3 | 3 |

```
int *rowSum(int **twoD, int row, int column){

     int *oneD = new int[row];

     for (int i = 0; i < row; i++){
            oneD[i] = 0;
            for (int j = 0; j < column; j++)
                  oneD[i] += twoD[i][j];
     }

     return oneD;
}
```

**Question 9**: Consider the following Time class.

```
class Time{

public:
     Time(){
            hour = minute = second = 0;
     }

private:
     int hour, minute, second;
};
```

First, dynamically allocate a two-dimensional array of Time objects, where the first and second dimensions are 4 and 6, respectively. Then, deallocate the memory space used by this array.

```
// Allocation
Time **tm = new Time*[4];
for (int i = 0; i < 4; i++)
     // To make this allocation, you should have a default constructor
     tm[i] = new Time[6];


// Deallocation
for (int i = 0; i < 4; i++)
      delete [] tm[i];
delete [] tm;
```