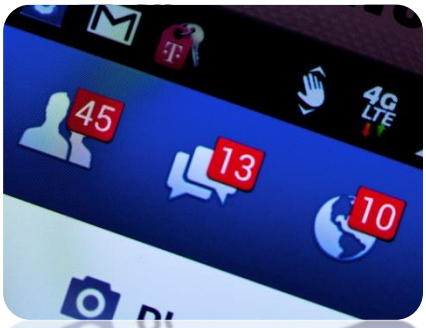
Computational Problem Solving (CS100)

Shafay Shamail, Arif Zaman and

Safee Ullah Chaudhary



Lab 10 Objectives:

- Using and manipulating Arrays
- Writing recursive functions.

Lab Guidelines

1. Make sure you get your work graded before the lab time ends.
2. You put all your work onto the LMS folder designated for the lab (i.e. “Lab10”) before the time the lab ends.
3. Talking to each other is NOT permitted. If you have a question, ask the lab assistants.
4. If you are a hot-shot C++ expert, you are still not allowed to use any feature of C++ that has not been covered in class or in the lab.
5. The object is not simply to get the job done, but to get it done in the way that is asked for in the lab.

Arrays

An array is a list of elements of the same data type that are placed in contiguous memory locations.

For instance, five values of type `int` can be declared as an array without having to declare 5 different variables (each with its own identifier). Instead, using an array, the five `int` values are stored in contiguous memory locations, and all five can be accessed using the same identifier, with the proper index.

An array containing 5 integer values of type `int` called my_array could be represented as:

```
int my_array[5] = {2,33,4,6,9};
```

my_array	2	33	4	6	9

General syntax for declaring and initializing an array:

Type name[size_of_array] = {elements of array};

Note: If you do not initialize an array, it will contain garbage values at each index. This may lead to unexpected results.

Accessing the values of an array

Following the previous examples in which `my_array` had 5 elements and each of those elements was of type `int`, the name which can be used to refer to each element is the following:

	my_array[0]	my_array[1]	my_array[2]	my_array[3]	my_array[4]
my_array	2	33	4	6	9

For instance:

```
cout << my_array[2] << endl;
```

output:

4

Similarly, if you want to update or modify specific values of an array, this can be done as follows:

```
my_array[2] = 16;

cout << my_array[2] << endl;
```

output:

16

Task 0: Arrays 101

Initialize an array of size 10 and then input 1 4 9 16 25 36 49 64 81 100 in the array using a loop.

Array as parameters

At some point, we may need to pass an array to a function as a parameter. But it is important to note that arrays are passed by *Reference* in C++. This essentially means that the function can directly modify the content of the array you passed to it as an argument (because memory address of the array is passed to the function. Any updates made to the array in the function will be made to the memory address of the array and hence the actual value of the of array indicies will be changed).

To accept an array as parameter for a function, the parameters can be declared as the array type, but with empty brackets, omitting the actual size of the array. For example:

For instance,

```
void func(int my_array[])
{
      my_array[2] = 0;
}
int main()
{
      int my_array[3] = {1,2,3};
      func(my_array);
      cout << my_array[2]  <<  endl;
}
```

Output of the above code will be:

```
0
```

Note: The function does not know the size of the array and we will need to pass a separate argument specifing the size if we need to traverse the array in the function. The method to do it is as follows:

```
void func(int my_array[], int size){}
```

Task 1:

a) Write a program that calculates the sum of all the numbers in the array you created in task 1. It should also output the minimum and maximum values of the array. The program should not be limited to the array above but should work for any array of any size.

b) Now convert theses three operations(sum, min, max) into individual functions. The functions should take the array as input and return appropriate values.

Task 2:

Write a program that initializes an array of size defined by the user. The program should then ask the user for successive values and add them to the array respectively. If the user enters 0, your program should exit and display the elements of the array.

Task 3:

Write a program that merges two arrays of same size into one big array in such a manner that it takes alternate values from each array.

For instance,

Arr1[3] = {1,3,5};

Arr2[3] = {2,4,6};

Merged_array = {1,2,3,4,5,6};

Recursion

When we talk about recursion, we are really talking about a function that calls itself.

Let's start by looking at a basic loop.

```
for(int i=0;  i<10; i++) {
     cout << "The number is: " << i << endl;
}
```

This cycle will continue to repeat for as long as the value of 'i' is less than 10. The last sentence displayed would read, "The number is: 9". As you can see the basic 'for loop' has three parts to its declaration, a starting value, what must remain true in order to continue repeating, and a modifying expression

Now with recursion, we won't need to use a 'for loop' because we will set it up so that our function calls itself. Let's recreate this same program one more time, only this time we will do it without a 'for loop'. We will use a recursion instead, like this.

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

void my_function(int i) {
  cout << "The number is: " << i << endl;
  i++;
  if(i<10) {
    my_function(i);
  }
}

int main()
{

int i = 0;
my_function(i);

return 0;
}
```

Try and understand how the recursive and iterative codes are similar.

Lets look at another example.

```
string ordinal(int n)
{
       if (n==1) return "st";
      if (n==2) return "nd";
      if (n==3) return "rd";
      if (n<20) return "th";
      if (n<100) return ordinal( n % 10 );
      else return ordinal(n % 100);
}
int main()
{
      int n=0;
      do
       {
            cin >> n;
            cout << n << ordinal(n) << endl;
      } while (n);
}
```

The base case is essentially the stopping condition. For instance, in the above example, our recursive call will return when any of the base condtions are met. i.e.

```
      if (n==1) return "st";
      if (n==2) return "nd";
      if (n==3) return "rd";
      if (n<20) return "th";
```

The base case is important in recursion since without it, the function will never stop and eventually overflow the stack (you never want that to happen!).

Task 4:

a) Write a recursive function that takes a number as input and return the product of all the even number from 2 to that number.
b) Initialize an array of size 10 and insert values of your own choice. Now apply the function you made in part 4(a) to each of the values and store the answer at the same postition in the array.