EE6411 Lab Sheet #4
References, Pointers and Arrays

Reiner Dojen[1]

[1]reiner.dojen@ul.ie

1 Objectives

- Understand difference between pass-by-value and pass-by-reference.
- Use reference type parameter.
- Use pointer types to achieve pass-gy-reference.
- Understand operations on pointers.
- Utilize pointer arithmetic to access array elements.
- Use the `vector` class as a dynamic array.

2 Preparation

You should be familiar on how to create/compile/execute applications in Visual Studio IDE. I also recommend to have worked through E-Activities 1-4 Automatic Code Format and 1-5 Managing multiple Projects in a Solution.

3 Exercises

Ex4.1 Using Pass-by-Reference

Create a function `void recip(...)` that has one parameter of type reference to double. Have the function change the value of the object that the parameter points to into its reciprocal (e.g. if the user passes the value 5.0 to the function, it should be change to $\frac{1}{5.0} = 0.2$). Add a main function to your program to demonstrate that it works correctly (it should test it at least for the values 1, 5, 10, 0.5, 0, -0.25, -1, -5).

Ex4.2 Returning a Value and a Status

Modify exercise Ex4.1: In addition to modifying the parameter, let your function `recip(...)` return a success/failure notification (boolean) - i.e. return true on success and false on failure (such as attempted division by 0).

Ex4.3 Using Pass-by-Reference for Out Parameters

Write a single function that counts the number of positive, negative and zero values in an integer array (Yes, I know, this is not good style as a function should only perform a single action ☺). Return the number of positive values, the number of negative values and the number of zero values. Use pass-by-reference to use parameters as “out parameter”: these are variables passed by reference into a function with the purpose of holding the result of the function after it terminates. Your function should have the following declaration:

```
void countPosNegZero(int array[], int numElements,
         int &positive, int &negative, int &zero);
```

where array is the array with the values, numElements is the number of elements in the array, positive will hold the number of positive values in the array, negative will hold the number of negative values in the array and zero will hold the number of zero values in the array.

Add a test-main function that tests your program with several different arrays (use the `rand()` method to create arrays with random values ranging over negative and positive values).

Ex4.4 Using Pass-by-Reference by Pointer

Create a function `void recip(...)` that has one parameter of type pointer to double. Have the function change the value of the object that the parameter points to into its reciprocal (e.g. if the user passes the value 5.0 to the function, it should be change to $\frac{1}{5.0} = 0.2$). Add a main function to your program to demonstrate that it works correctly (it should test it at least for the values 1, 5, 10, 0.5, 0, -0.25, -1, -5).

Ex4.5 Returning a Value and a Status

Modify exercise Ex4.4: In addition to modifying the parameter, let your function `recip(...)` return a success/failure notification (boolean) - i.e. return true on success and false on failure (such as attempted division by 0).

Ex4.6 Using Pass-by-Reference by Pointer for Out Parameters

Write a single function that counts the number of positive, negative and zero values in an integer array (This is still not a good idea - but if the lecturer insists ... ☺). Return the number of positive values, the number of negative values and the number of zero values. Use pass-by-reference to use parameters as "out parameter": these are variables passed by reference into a function with the purpose of holding the result of the function after it terminates. Your function should have the following declaration:

```
void countPosNegZero(int array[], int numElements,
         int *positive, int *negative, int *zero);
```

where array is the array with the values, numElements is the number of elements in the array, positive will hold the number of positive values in the array, negative will hold the number of negative values in the array and zero will hold the number of zero values in the array.

Add a test-main function that tests your program with several different arrays (use the `rand()` method to create arrays with random values ranging over negative and positive values).

Ex4.7 Accessing Array Elements via Pointer

Create an array with 10 random numbers in range 0-100 (both inclusive). Print out these 10 random numbers - but use pointer arithmetic (`ptr + i`) instead of array subscription (`array[i]`) to access the elements of the array.

Ex4.8 Passing Arrays as Pointers

Create a function `printArray(int *data, int size)` that accepts an array as first parameter (here passed as a pointer together with the size of the array) and prints out the elements of th array (use pointer arithmetic instead of array subscription to access the elements).

Add a main function that:

- Creates an array numbers1 of 10 random elements.
- Calls function `printArray(...)` to print numbers1 to the screen.
- Creates an array numbers2 of 20 elements.
- Calls function `printArray(...)` to print the screen.

Ex4.9 Using Array Subscription on Pointers

Re-do exercise Ex4.8. However, this time use array subscription to access the array passed via pointer `int *data`.

Ex4.10 Treating Pointer as Arrays

Re-do exercise Ex4.8. However, this time use an array as first parameter (`printArray(int data[], int size)`) and use pointer arithmetic to access elements of `int data[]`.

Ex4.11 Subtracting Pointers/Addresses

Write a program that creates an array of 5 random integer numbers. Use a for-loop to iterate through all elements of the array and print out the difference of the addresses of each element and the first element of the array (`&array[i]-&array[0]` or `&array[i]-array` or `(array+i)-array` - all three achieve the same outcome).

Then repeat the same with an array of 5 random double numbers - do you expect a different outcome or the same? Compile and run your program to confirm your expectation.

Ex4.12 Vector as Array

Use the `std::vector` class to implement the following: Develop a program that manages the grades for a class. Implement the following functions:

`void enter_percent(vector<double> &student_percentages):`
Prompts the user to enter the percentages each student has achieved and stores the entered values in the passed vector. This will continue until the user enters a negative value. If the vector already contains elements, these will be removed before the user is prompted.

```
void get_grades(const vector<double> &student_percentages,
                vector<char> &student_grades):
```
Converts the percentages from the vector `student_percentages` into grades A to F and stores the result in the vector `student_grades`. Make sure that percentages and grades are stored in the same order. Percentages are converted as follows to grades: $A \geq 80.0, B \geq 65.0, C \geq 50.0, D \geq 35.0, F < 35.0$.

`void print_grade_summary(const vector<char> &student_grades):`
Counts the occurences of each grade (A to F) in the vector and prints the result to `cout`.

`int main():`
Creates two (static) arrays: one for percentage vectors (`vector<double>`) and one for grade vectors (`vector<char>`). In each of these arrays store three empty vectors of the appropriate type. For each of the elements in the percentage vectors perform the following:

- Use the function `enter_percent(...)` to fill it with student results.
- Pass the percent vector and the (still empty) matching grade vector to the function `get_grades(...)`.
- Pass the grade vector to the `print_grade_summary(...)` function to get the result overview.

Hint: If you want to simplify testing of your program (without the need to always enter grades) utilize a text file as outlined in exercise "Duplicate Elimination" on lab sheet #3.