# Programming Technique

## Lab 5 – Array, string and struct

### 1 Expected outcomes

- Be able to define new data types with typedef and struct.
- Declare struct and array variables.
- Write and read data from members of a struct.
- Use struct and array to solve a part or a whole of a problem.
- Working with string.

### 2 Mandatory exercises

**Exercise 1.**

- Define new data types to store the data of these object types:
  - Printer
  - Subject
  - Corporate
  - Bird

  For each type, you must define at least **four** relevant attributes (name, ID, etc.) and their appropriate data types.

- Declare an array for each of the above type. An array can store at most 100 instances. Use macro to for this number 100.
- For each type, assign data to an instance in the array.
- Print the information you have just assigned.
- Declare a variable of type Printer, initialize the attributes with any values you want. Assign the new Printer instance to one of the 100 Printer array at any position. Do the same for Subject, Corporate and Bird.

Guidelines:

Below is an example written for another data type called computer. It has two attributes: name (string) and resolution (float):

```
typedef struct {
      string name;
      float resolution[2];
  } computer;
```

The new data type was created using struct. There are two way to create a variable with this kind of data type:

**Method 1 (valid in C):**

```
computer com = {
      .name = "ASUS",
      .resolution = { 1366, 768 },
  };
```

If we don't want to initialize the name of the instance, we can write as follow:

```
computer com = {
      .resolution = { 1366, 768 },
  };
```

**Method 2 (valid in C++):**

```
computer com = {
      "ASUS",
      { 1366, 768 },
 };
```

You can compile the above snippets with CodeChef to test them with the following compilers:

- C (GCC-4.9.2)
- C++ 4.9.2
- C++ 14

The assignment operation for new types is the same as built-in data type:

- com2 = com;
- com_array[11] = com;

An attribute of a struct can be assigned like this:

```
com.name = "My new computer";
```

**Note:** the value of the right-hand side must correspond to that of the defined attribute.

We can access the value of an attribute by writing:

```
var_struct.attr
```

Here:

- var_struct is the name of the variable with a struct data type.
- attr: the attribute we want to access.

Define an array with MAX elements (MAX must be constant):

```
computer com_array[MAX];
```

We can access the value of an element of the array as follow:

```
var_array [index].attr
```

Where:

- var_array is the name of the array variable.
- index: the position we want to access.
- attr: the attribute we want to access.

**Exercise 2.** Write a program that allows:

- Read two N-dimensional vectors from a text file called "input.txt". Call the two vectors a and b. N is the dimension of the vectors. You must find N when you read the first line of the text file. N must be <= 50. For example, the file "input.txt" listing 3-dimensional vectors has the following format:

  ---

  ----- Vectors ------

  ---

  12.4 34.2 44.2

  3.5 2.21 21.56

  When reading the above file, if there is an error or when the second line has dimensionality different from the first line, the program should print an error and stop.

- Compute the following:
  - The length of a vector. For example, a = [a1 a2 a3 a4] in 4 dimensions has the length |a| = sqrt(a1*a1 + a2*a2 + a3*a3 + a4*a4)
  - The dot product and cross product (only compute the cross product when there are 3 dimensions) between 2 vectors:
    - Let $\mathbf{a}=[a_1, a_2, a_3]^T$ và $\mathbf{b}=[b_1, b_2, b_3]^T$.
    - Dot product: $\mathbf{a} \bullet \mathbf{b} = a_1b_1 + a_2b_2 + a_3b_3$
    - Cross product: $\mathbf{a} \times \mathbf{b} = \begin{vmatrix} a_2 & a_3 \\ b_2 & b_3 \end{vmatrix}\mathbf{i} - \begin{vmatrix} a_1 & a_3 \\ b_1 & b_3 \end{vmatrix}\mathbf{j} + \begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix}\mathbf{k}$
  - Compute and print the angle between a and b:
    - **alpha** = acos(**a•b**)/(|**a**|*|**b**|).
  - Compute and print the length of the projection of a on b and vice versa:
    - Length of the projection of a on b: **a•b/|a|**

Requirements:

- The student must use macro for the number 50.
- The student must define a struct called Vector and decide what to store in it.

Guideline:

a) Data:
- The struct Vector must have the attributes to store its value and dimension. You should find the appropriate data type for them.
- Define MAX_DIMENSION to help with the input and PI, DEG to convert the angle from radian to degree.
- A variable to read the input file:
  - string str;
  - isstringstream iss;
- Declare variables related to vector operations:
  - double sum_square, scalar_p, angle, etc.

b) Algorithm:

How to read the input:

- Use `getline(cin, str)` to read a line from file and assign that line to str.
- Skip the first three lines.
- Read the first vector (the fourth line in the input file).

- Use `istringstream::str()` to reassign the contain to iss. This allows us to reuse iss to read other lines in the file.
- Use a while loop to read the values one-by-one and assign them to the vector. The while loop stops when there is an error or when the reading is done.
- Clear the error flag of istringstream.
- Do the same for the second vector.

**Exercise 3.** Write a program to:

a) Prompt the user to enter a list of non-negative real numbers and store them in a 1-dimensional array. The prompting only stops when the program reads a negative number or the amount of numbers exceeds MAX_SIZE. MAX_SIZE is created using #define.
b) Compute and print: the average value and the standard deviation of the list. If the array has N elements, the standard deviation can be computed as follow:

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x-\bar{x})^2}$$

c) Invert the list and print it **without using an additional array**.

Guidelines:

The program should print guidelines on how the user should input.

One way to solve problem a) is:

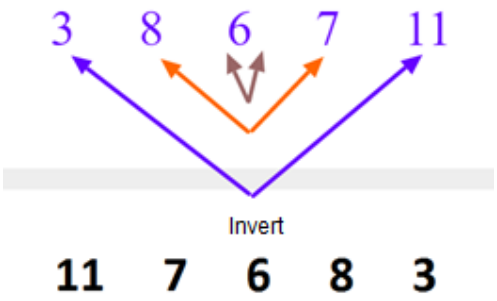
- Use a loop to loop MAX_SIZE times, the looping will stop early if user enters a negative number.
- We can add another while loop inside the for loop to catch invalid input (character instead of number), this is optional.

For problem b), you need to write a loop to:

- Calculate the sum then the average by dividing the sum to the number of valid elements in the array (<= MAX_SIZE).
- Calculate the sum of squares just as the standard deviation equation requires then compute the standard deviation.

To invert the array without using an additional array:

- Looping N/2 times using for (1/2 the number of elements in the array).
- Each time we loop, we can swap the current element with the respective one from bottom to top:




The same idea can work with a while loop:

- Initialize **left** and **right** with the first and the last positions of the array (0 and N-1).
- Swap the **left** element with the **right** element.
- Increase **left** by one and **decrease** right by 1 after swapping. Continue the loop.
- The stopping condition is left > right.

**Exercise 4.** Write a program to:

a) Define a struct to store an 2D vertex.
b) Declare an array of the above struct to store MAX_VERT vertices. MAX_VERT needs to be created by #define.
c) Write a program that let user enter N (N is the number of vertices and $2 < N \leq MAX\_VERT$). If the input was incorrect, prompt the user to reenter until it is correct.
d) Randomly generate N vertices. The coordinates must be in the range [-100, 100].
e) Assume that all the points form a polygon, compute and print its perimeter.
f) Redirect the output streaming result to "output.txt" (instead of printing on the console screen).

Guidelines:

a) Data:

The program should have the following variables:

- N: float (To check if it is a non-negative integer).

- Define a struct called Vertice to represent a vertex in 2D:
  - x: float.
  - y: float.

You should know the difference between struct + typedef and struct without typedef.

- Define an 1D array called vertices[MAX_VERT] with type Vertice to store the points.

b) Algorithm:

- Verify N entered by user. It must be in the [3, MAX_VERT] range. If N is invalid, the user must re-enter.
- Learn how to generate random values each time we re-execute the program. This can be done using `seed` and `srand()`. If we do not use `srand()` and only use `rand()`, why are the numbers the same each time we re-execute the program? You should be able to answer this.
- Use `rand()` to generate real-valued numbers in the range [-100, 100].
- To compute the perimeter, we need to sum all lengths of the sides of the polygon.
- To calculate the lengths of the polygon, we can define a macro to compute the length of every two adjacent points. Note that when you use macro, all of the parameters of the macro must be within a pair of parentheses (). This is to avoid priority problem.
- To redirect the output stream to write to "output.txt", we can use command line (cmd). Assume that the program is named Vertex.exe, to write outputs to "output.txt", use the follow command:

```
Vertex.exe > output.txt
```

**Exercise 5.** Given an input file storing array-like data, the format is given below:

- It has N lines. N ranges from 3 to 100.
- Each line has M numbers. M ranges from 2 to 100.

a) Write a program to enter vector X and matrix W based on the input matrix from the file. Here, X is the first column of the matrix. W is a matrix consisting N lines and the other M – 1 columns.

b) Calculate X * W and print the results.

- Note: use macro for the above constants (M and N). You should be able to write the file yourself and test the corresponding results.

Guidelines:

a) Data: we need appropriate type to store X and W. Note that the type should be able to consist the matrix in its largest possible size (M and N).
b) Algorithm:

- To read the file we can use redirection operator to redirect the input to std:stdin and read them normally. Initially, we need to read the first line to see how many columns it has. For the other lines, we can use getline combined with a loop to read the input file line-by-line. In each loop, we can use stringstream and a loop to read each number in a line and assign it to the previously-defined structure. The looping will end once there is no more line (check this with stringstream::eof). While reading, find the number of columns of each line to see if it is the same as the first line. If there is a difference, print an error and end the program.
- After the data has been read and stored in the variables, we clearly know the size of the vector and the matrix. With this, we can calculate the vector-matrix multiplication X * W. The multiplication between X and W will result in a vector K. The $i^{th}$ element of this vector will be calculated as follow:
  - $K[i] = \sum_{j=1}^{N} X[j] * W[i][j]$. You can use a loop to compute the sum of all the N terms. Here is an example with N = 3:

$$(1 \quad 2 \quad 3) \times \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix} = \begin{pmatrix} 1*1+2*2+3*3 \\ 1*4+2*5+3*6 \\ 1*7+2*8+3*9 \end{pmatrix}^T = \begin{pmatrix} 14 \\ 32 \\ 50 \end{pmatrix}^T$$

- You should find an appropriate format to print the values of K. Remember to specify the correct alignment and field width.

**Exercise 6.** Write a program that receives a string, calculate its length. This means you have to re-implement the strlen function in `<string.h>`.

Guideline:

a) Data:

- A variable to store the input string: char array.
- A variable to store the length: data type is size_t.

b) Algorithm:

- Use macro #define to explicitly specify the size of the buffer storing the string inputted from keyboard.
- Use `cin.getline` to get the string from keyboard (do not use cin because cin ignores white space).

- Iterate over the inputted string until the terminating null character '\0' is found. (This character tells the program that the string has ended, **the length of a string does not take this character into account)**.
- The number of characters found (except for the null) is the length of the string.

**Exercise 7.** Write a program that receives a string, remove all white spaces in front of and at the end of the string. Additionally, if there are more than one white space between two non-white space characters, remove them so they only have one.

Guidelines:

a) Data: a variable to store the input string, data type is char array.
b) Algorithm:

- Use macro #define to define the maximum buffer size.
- Use `cin.getline` to get the string from keyboard (do not use cin because cin ignores white space).
- Iterate over white spaces and count the number of white spaces.
- Remove the white spaces between two consecutive non-white space characters (but keep one!).
- Repeat the previous two step until we reach the end of the string.

Example:

" Programming in C++ " → "Programming in C++ " → "Programming in C++ " → "Programming in C++ " -> "Programming in C++".

**Exercise 7.** Write a program that receives a string and invert the **words** (not characters) of the string. Print the original string and the inverted string.

Guidelines:

a) Data: A variable to store the string inputted from keyboard, char array.
b) Algorithm:

- Use macro #define to define the maximum buffer size.
- Use `cin.getline` to get the string from keyboard (do not use cin because cin ignores white space).
- Invert all of the characters in a string:
  "Programming in C++" → "++C ni gnimmargorP".
- Invert the characters of each word in the string:
  "++C ni gnimmargorP" → "C++ in Programming".