UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

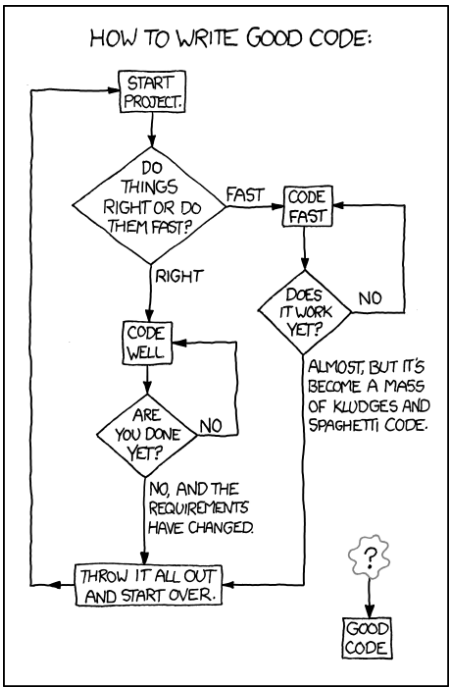
Computer Programming 143

Practical 8

2019

Aim of Practical 8:

- use 1-D array as 2-D matrix
- Develop Matrix library



Instructions

1. Attendance is **compulsory** for all the practical sessions of your assigned group. See the study guide for more details.
2. The last section (usually the last 30 minutes) of the practical will be used for a test.
3. If more than two tests have been missed for what ever reason, you will receive an **incomplete** for the subject. See the study guide for more details.
4. You must do all assignments **on your own**. Students are encouraged to help each other **understand** the problems and solutions, but each should write his/her own code. By simply copying someone else's code or solutions, you will not build an understanding of the work.
5. You are responsible for your own progress. Ensure that you understand the practical work. Check your work against the memorandum that will be posted on Wednesday afternoons on learn.sun.ac.za.
6. Use H:\CP143 as your Code::Blocks workspace folder for all projects. But it is highly suggested that you also use a **flash drive to backup** all your work.
7. Create a new project **for each assignment**. See *Creating a Code::Blocks Project* in Practical 0 for instructions on how to do this.
8. Include a comment block at the top of each source file according to the format given. It must include the correct filename and date, your name and student number, the copying declaration, and the title of the source file.
9. **Indent your code correctly.** Making your code readable is not beautification, it is a time and life saving habit. Adhere to the standards (refer to the documents on SUNLearn).
10. Comment your code sufficiently well. It is required for you and others to understand what you have done.

Question 8A

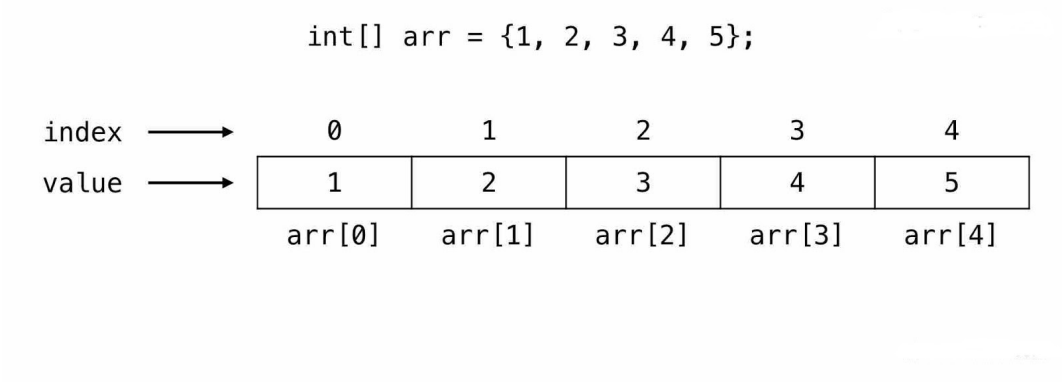
Goal: *Develop a matrix manipulation library.*

Getting started

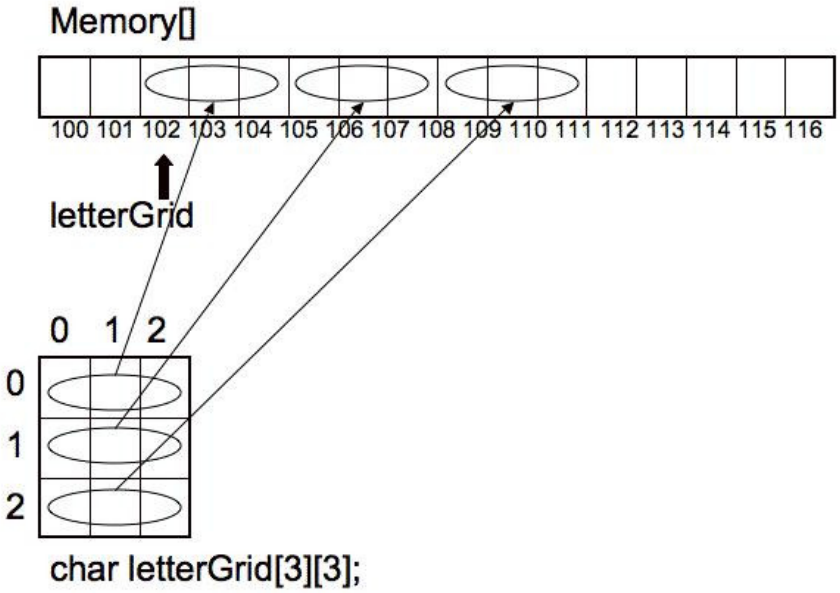
1. Create a project named `Assignment8A`. Make sure that this is the active project in the workspace before compiling the program.
2. Include the **standard comment block** above your main function. Also, comment your whole program appropriately.
3. Create a header and source file called **matrixOperations.h** and **matrixOperations.c**.

Regarding 2D arrays

As you know, 1D arrays are stored as a list of values in memory, this can be seen in the below figure:



But how are 2D arrays stored in memory? If you had a 3 by 3 matrix called letterGrid, how will the values be stored in memory? C makes use of row major memory allocation for 2D arrays. Row major ordering assigns successive elements, moving across the rows and then down the columns, to successive memory locations. You start with the first row (row number zero) and then concatenate the second row to its end. You then concatenate the third row to the end of the list, then the fourth row, etc. This can be seen in the figure below.



The actual mapping is not important as long as two things occur:

- each element maps to a unique memory location
- the mapping is consistent

So why would you want to represent a 2D array as a 1D array? When creating functions that operate on 2D arrays, the size of the second element needs to be specified. Therefore these functions are not generic and will only work for specific size arrays. By representing the 2D array as a 1D array, you can write generic functions that work on any size arrays.

The formula used to access the correct elements in the array is:

- array1D[x*W+y]=array2D[x][y];
- array2D[i/W][i%W]=array1D[i];

where the arrays are defined as array1D[W*H] and array2D[W][H].

Program description

1. In **matrixOperations.c** code the following functions (include the function prototypes in **matrixOperations.h**):

   - **`void`** **DisplayMatrix**(**`int`** matrix[], **`int`** sizeRow, **`int`** sizeCol);
     This function receives an array with its size and displays it to the screen as a 2D array. Use a suitable display format to display the matrix.

- **void FillMatrix**(**int** matrix[], **int** sizeRow, **int** sizeCol);
  This function receives an array with its size and fills it with random numbers in the range from 0 to 9.
- **void MultiplyMatrices**(**int** matrixAns[], **int** matrixA[], **int** matrixB[], **int** sizeRowA, **int** sizeColA, **int** sizeRowB, **int** sizeColB);
  This function receives three matrices. **matrixAns** will store the answer to **matrixA** multiplied by **matrixB**. The size of **matrixA** and **matrixB** are given, what is the size of **matrixAns**?
- **void AddMatrices**(**int** matrixAns[], **int** matrixA[], **int** matrixB[], **int** sizeRow, **int** sizeCol);
  This function receives three matrices. **matrixAns** will store the answer to **matrixA** added to **matrixB**. All three matrices have the same size.
- **void TransposeMatrix**(**int** matrixAns[][], **int** matrixA[][], **int** sizeSquare);
  This function receives two matrices, **matrixAns** will store the transpose of **matrixA**. The size of **matrixA** is given and is a square matrix.

2. In the main function display the following user menu:

```
PLEASE SELECT AN OPTION:
1) Matrix multiplication
2) Matrix addition
3) Matrix transpose
```

3. If the user selected "**Matrix multiplication**" then:
   - In the main function, ask the user to input the number of rows and columns of each of the two matrices
   - Check that the sizes are compatible for matrix multiplication, keep asking until they are
   - Fill the matrices with random numbers
   - Display the matrices to the screen
   - Multiply them and display the result to the screen
4. If the user selected "**Matrix addition**" then:
   - In the main function, ask the user to input the number of rows and columns of the first matrix
   - Fill the matrices with random numbers
   - Display the matrices to the screen
   - Add them and display the result to the screen (how can you modify this to handle matrix subtraction?)
5. If the user selected "**Matrix transpose**" then:
   - In the main function, ask the user to input the number of rows of the matrix (a square matrix)
   - Fill the matrix with random numbers

- Display the matrix to the screen
- Transpose it and display the result to the screen

6. Modify the above code to ask the user to input the individual matrix values. Create a function to do this.

7. Ensure that your code is indented correctly and that the {} braces are on the correct lines. Use the prescribed textbook as guideline.

8. Ensure that you copy the **Assignment8A** project folder to a flash drive as a backup.