Dr. B. Güsmann VGU

Electrical Engineering and Information Technology, B.Eng.
Introduction to the C Programming Language: Exercises

Exercise Sheet 11

---

1. A vector

$$A = \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix}$$

describes the position of a point A in 3-dim space for a given coordinate system.

A matrix

$$R(z,\varphi) = \begin{pmatrix} \cos(\varphi) & -\sin(\varphi) & 0 \\ \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Describes the rotation around the z-axis.

The multiplication R x A is defined by the scalar product

$b_i = r_{i1}*a_1 + r_{i2}*a_2 + r_{i3}*a_3$ for each row i = 1,2,3

The vecor B is the result of rotating vector A with angle φ around the z-axis. This technique is used to describe position and orientation of objects moved by robots.

Write a program, which uses 1-dim arrays for A and B and a 2-dim array for R. Write a function with parameters B, R, A, which computes the rotation of A around the z-axis with angle PHI(φ).

Read A and φ in main() from the keyboard, setup R and call your function and write the result B to the screen.

2. A matrix

$$R(x,\varphi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi) & -\sin(\varphi) \\ 0 & \sin(\varphi) & \cos(\varphi) \end{pmatrix}$$

describes the rotation around the x-axis.

A multiplication of matrix R1 x R2 is defined by multiplying each column $R2_j$ of R2 with R1 as defined in exercise 1. In other words

R = R1 x R2 is defined by R_j = R1 x $R2_j$ for each column j = 1, 2, 3 of R2.

Computing R = R(x, φ 1) x R(z, φ 2) means to compute the resulting rotation of first applying R(z, φ 2) and then applying R(x, φ 1).
Write a function CombinedRotation(R, R1, R2) computing the above described matrix product R = R1 x R2. Write a main() setting up R1 and R2, then calling CombinedRotation() and printing R row by row to the screen.

3. Combine exercise 1 and exercise 2. Read in a vector A, an angle φ 1 and an angle φ 2.
Rotate A by φ 2 around the z-axis and by φ 1 around the x-axis. Print the resulting vector B.

4. a) 2-dimensional arrays are stored in memory as one row after the other.
The following example shows how we can use flexible index bounds as function paramters for a 2-dimensional matrix. Try your own examples with other index bounds.

```
#include <stdio.h>

void print_matrix(int *p_matrix, int row, int column);

int main()
/*****************************************************/
/*                                                   */
/* Matrix with variable index bounds as parameter.  */
/*                                                   */
/*****************************************************/
{
int  i, j;
int  matrix[4][4];
int  *p_to_my_matrix;

    for(i = 0; i < 4; i++) {
        for (j = 0; j < 4; j++) {
            matrix[i][j] = (i+1)*(j+2);
        }
    }
    printf("Control print:\n");
    for(i = 0; i < 4; i++) {
        for (j = 0; j < 4; j++) {
            printf("%4d", matrix[i][j]);
        }
        printf("\n");
    }
    printf("\n");

    p_to_my_matrix = (int *)matrix;
    print_matrix(p_to_my_matrix, 4, 4);

    getchar();
}

void print_matrix(int *p_matrix, int row, int column)
/************************************************/
/*                                              */
/* Prints values of an int matrix row by row.  */
/*                                              */
/************************************************/
{
int i, j;

    for(i = 0; i < row; i++) {
        for (j = 0; j < column; j++) {
            printf("%4d", *(p_matrix+i*column+j) );
        }
```

```
        printf("\n");
    }
} /* END_print_matrix(); */
```

b) Where the compiler throws an error message:

```
void define_matrix(int row, int column)
/***********************************************/
/*                                             */
/* Defines values of an int matrix.            */
/*                                             */
/***********************************************/
{
int i, j;
int matrix[row][column];

     for(i = 0; i < row; i++) {
         for (j = 0; j < column; j++) {
             matrix[i][j] = i + j; ;
         }
     }
} /* END_print_matrix(); */
```