Hamburg University of Applied Sciences
Dept. of Electrical Engineering and Computer Science
Laboratory "Software Construction in C"

6	Dynamic MEM allocation	
Semester / Group		Record editor
Date		Participant
Professor		

Approvals							

After this Lab, you will be able to ...

- work with the MSVC++ Compiler, especially with the debugger
- Develop test strategies for your programs
- Develop complex programs: work with pointers, dynamic memory allocation for 1D- and 2D-arrays on the heap

Note :

You have to prepare Assignment 1 below before the lab and upload your solution before or at the beginning of your lab appointment Jan. 2014.

Note :

- **For the two Tuesday groups :**
  - **The complete solution (Assignment 1&2) for this lab has to be turned in latest next Monday after your lab.**
- **For the Wednesday group :**
  - **The complete solution (Assignment 1&2) for this lab has to be turned in latest next Tuesday after your lab.**

Assignment 1:

Write a program which allocates memory on the heap for a three-dimensional array of type double. Test your program with a matrix matr[4][2][3] (i.e. 4 planes, 2 rows, 3 columns), where each element contains the sum as follows:

`sum_of_element = plane-index * 1000 + row-index*10 + columns index * 100`, as shown below.

```
plane 0
     0    100    200
    10    110    210
plane 1
  1000   1100   1200
  1010   1110   1210
plane 2
  2000   2100   2200
  2010   2110   2210
plane 3
  3000   3100   3200
  3010   3110   3210
```

Assignment 2:

A vector row vector v^T with 3 elements is shown below

$$v^T = \begin{pmatrix} a_0 & a_1 & a_2 \end{pmatrix}.$$

We will restrict ourselves to vectors with real elements, thus
a_i, i=1,..,N are real numbers. N can be of **arbitrary size**. Below, N=3 is assumed.

The following operations shall be implemented in ANSI C using functions.

Vector addition : $\underline{c} = \begin{pmatrix} c_0 \\ c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} + \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}$

Row vector multiplied by column vector (Dot product):

$$dp = \begin{pmatrix} a_0 & a_1 & a_2 \end{pmatrix} \cdot \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}$$

Column vector multiplied by row vector: $\underset{\sim}{m} = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} \cdot \begin{pmatrix} b_0 & b_1 & b_2 \end{pmatrix}$

Note that "*dp*" is a scalar value and $\underset{\sim}{m}$ is in this case a 3x3 matrix.

On the next page, the prototypes of all functions and main( ) is shown. You **must not** change the prototypes.
The code is provided on the homepage (ZIP6), you do NOT have to type it in!

In main( ), the following steps are taken:

- Vectors (stored on the stack) are declared and initialized, pointers are declared
- the number of elements stored in vectors A and B stored on stack are calculated
- space for copies of vectors A and B is allocated on heap
- vectors A and B are copied from stack to heap
- C = A + B is computed. Note : A, B and C are stored on heap, result C is printed
- dot_prod(A, B) is computed, A and B on heap, result is a SCALAR value on stack
- 1D column vector is multiplied with 1D row vector → result is a matrix, matrix is printed
- All memory allocated via malloc is freed

The screen output is shown below:

```
number of elements in vectors : 4

vector addition:
       12.00        24.00        36.00        48.00

dot_prod_val = 330.000000

1D column vector times 1D row vector  :
       11.00        22.00        33.00        44.00
       22.00        44.00        66.00        88.00
       33.00        66.00        99.00       132.00
       44.00        88.00       132.00       176.00
Press any key to continue . . .
```

Assignment:
Write all functions used in main() in order to get the screen output shown above.

```c
//------------------- prototypes -----------------------------
double *        allocate_mem_on_heap_for_vector_1D(int N);
double **       allocate_mem_on_heap_for_matrix_2D(int N);
void            free_mem_on_heap_for_vector_1D(double *);
void            free_mem_on_heap_for_matrix_2D(double **, int N);

void            copy_vector_1D_from_stack_to_heap(
                      double p2vector_1D_on_stack[],
                      double p2vector_1D_on_heap[],
                                           int N);
void            copy_vector_1D_from_heap_to_stack(
                      double vector_1D_on_heap[],
                      double vector_1D_on_stack[],
                                           int N);
double *        vector_1D_add(double *p2A, double *p2B, int N);
double          dot_product(double *p2A, double *p2B, int N);
double **       column_vector_1D_times_row_vector_1D(double *p2A, double *p2B, int N);

void            print_vector_1D(double *p2_vec, int N);
void            print_matrix_2D(double **p2p2_matrix, int N);

//------------------------------------------------------------

//------------------- main program, variables ---------------------
int main(){
        double A_on_stack[] = {1, 2, 3, 4};
        double B_on_stack[] = {11, 22, 33, 44};

        double *p2A_on_heap = NULL;
        double *p2B_on_heap = NULL;
        double *p2C_on_heap = NULL;
        double *p2D_on_heap = NULL;
        double dot_prod_val;
        int elements;
        double **p2_matrix2D_on_heap = NULL;

//--------------------------------------------------------------
// code starts:
// calculate the number of elements
        elements = sizeof(A_on_stack)/sizeof(double);
        printf("number of elements in vectors : %d\n\n", elements);

// allocate space for vectors A and B on heap
        p2A_on_heap = allocate_mem_on_heap_for_vector_1D(elements);
        p2B_on_heap = allocate_mem_on_heap_for_vector_1D(elements);

// copy vectors A and B from stack to heap
        copy_vector_1D_from_stack_to_heap(A_on_stack, p2A_on_heap, elements);
        copy_vector_1D_from_stack_to_heap(B_on_stack, p2B_on_heap, elements);

// computes C = A + B. Note : A, B and C are on heap
        p2C_on_heap = vector_1D_add(p2A_on_heap, p2B_on_heap, elements);
        printf("vector addition:\n");
        print_vector_1D(p2C_on_heap, elements);

// computes dot_prod(A, B), A and B on heap, result is a SCALAR value on stack
        dot_prod_val= dot_product(p2A_on_heap, p2B_on_heap, elements);
        printf("dot_prod_val = %lf\n\n", dot_prod_val);

// calculate 1D column vector times 1D row vector  --> matrix
        p2_matrix2D_on_heap = column_vector_1D_times_row_vector_1D(p2A_on_heap, p2B_on_heap,
elements);
        printf("1D column vector times 1D row vector  :\n");
        print_matrix_2D(p2_matrix2D_on_heap, elements);

// free memory
        free_mem_on_heap_for_vector_1D(p2A_on_heap);
        free_mem_on_heap_for_vector_1D(p2B_on_heap);
        free_mem_on_heap_for_matrix_2D(p2_matrix2D_on_heap, elements);
        return 0;
} // end of main
```