

## Домашнє завдання №15

Скласти програму (C/C++), що виконує множення матриць. Для виконання завдання замість простого двовимірного масиву використати власну реалізацію.

### Вибір варіанту

$$(N_{\text{ж}} + N_{\text{г}} + 1) \% 2 + 1$$

де:  $N_{\text{ж}}$  – порядковий номер студента в групі, а  $N_{\text{г}}$  – номер групи (1,2,3,4,5,6,7,8 або 9)

### Варіанти завдань

Варіант	Розмір першої матриця		Розмір другої матриця	
	кількість рядків	кількість стовпців	кількість рядків	кількість стовпців
1	4	5	5	4
2	5	4	4	5

## Приклад коду

Лістинг

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#define MAX_DEEP 32

#define SIZE_1 3
#define SIZE_2 5

#define MATRIX_A_SIZE_I SIZE_1
#define MATRIX_A_SIZE_J SIZE_2

#define MATRIX_B_SIZE_I SIZE_2
#define MATRIX_B_SIZE_J SIZE_1

#define MATRIX_C_SIZE_I SIZE_1
#define MATRIX_C_SIZE_J SIZE_1

template<typename T>
struct MyVLA {
    union {
        T value;
        unsigned int value_;
    };
    T* data;
    unsigned int* levelInfo;
    unsigned int* deep;
    unsigned int offset;
    unsigned int pointerLevel;
    //
    MyVLA(T* data, unsigned int* deep, unsigned int* levelInfo, unsigned int offset,
    unsigned int pointerLevel)
        : value(0), data(data), deep(deep), levelInfo(levelInfo), offset(offset),
```

```

pointerLevel(pointerLevel) {
    (pointerLevel + 1) ? 0 : (value_ = 0, value = *(T*)(data + offset));
}

MyVLA(void*, int number, ...) : offset(0), pointerLevel(number - 1) {
    unsigned int size = 1;
    unsigned int levelInfo[MAX_DEEP];
    va_list argList;
    va_start(argList, number);
    for (int index = 0; index < number; ++index)
    {
        size *= levelInfo[index] = va_arg(argList, unsigned int);
    }
    /*((unsigned int*)&number + 1) + index);
    va_end(argList);
    data = (T*)malloc(size * sizeof(T)+(1 + MAX_DEEP) * sizeof(unsigned int));
    this->deep = (unsigned int*)((char*)data + size * sizeof(T));
    *this->deep = number;
    this->levelInfo = this->deep + 1;
    for (int index = number - 1; index >= 0; --index)
    {
        this->levelInfo[index] = size /= levelInfo[index];
    }
}

MyVLA<T> operator[](int index) {
    return MyVLA<T>(this->data, this->deep, this->levelInfo, offset + index *
this->levelInfo[this->pointerLevel], this->pointerLevel - 1); // level ? --level : 0;
};

MyVLA<T>& operator = (const MyVLA<T>& refMyVLA) {
    *(this->data + this->offset) = refMyVLA.value;
    return *this;
};

MyVLA<T>& operator += (const MyVLA<T>& refMyVLA) {
    *(this->data + this->offset) += refMyVLA.value;
    return *this;
};

MyVLA(const T& data) : data(0), deep(0), levelInfo(0), offset(0), pointerLevel(0)
{
    this->value = data;
};

operator T() {
    return *(data + offset);
};

};

void createVariativeDemisionVLA_scenario(void) {
    unsigned int demisionCount;
    unsigned int demisions[MAX_DEEP] = { 0 };

    printf("Input demision count: ");

    scanf("%d", &demisionCount);

    if (demisionCount > MAX_DEEP) {
        demisionCount = MAX_DEEP;
        printf("demision count set %d by MAX_DEEP\r\n", MAX_DEEP);
    }

    printf("unsigned int data");
    for (unsigned int index = 0; index < demisionCount; ++index) {

```

```

        printf("[D%d]", demisionCount - index - 1);
    }
    printf(";\\r\\n");
    for (unsigned int index = 0; index < demisionCount; ++index) {
        printf("Input D%d: ", index);
        scanf("%d", &demisions[index]);
    }

    MyVLA<int> variativeDemisionVLA(NULL, demisionCount
        , demisions[0], demisions[1], demisions[2], demisions[3]
        , demisions[4], demisions[5], demisions[6], demisions[7]
        , demisions[8], demisions[9], demisions[10], demisions[11]
        , demisions[12], demisions[13], demisions[14], demisions[15]
        , demisions[16], demisions[17], demisions[18], demisions[19]
        , demisions[20], demisions[21], demisions[22], demisions[23]
        , demisions[24], demisions[25], demisions[26], demisions[27]
        , demisions[28], demisions[29], demisions[30], demisions[31]);

    //    add code to using VLA here ...
    if (demisionCount == 3) { // for demisionCount=3 example
        // set zero element
        variativeDemisionVLA[0][0][0] = 123;
        // read zero element
        printf("value: %d: ", (int)variativeDemisionVLA[0][0][0]);
    }
}

void matrixMul_scenario(void) {
    MyVLA<int> matrixA(NULL, 2, MATRIX_A_SIZE_J, MATRIX_A_SIZE_I);
    //
    for (unsigned int iIndex = 0; iIndex < MATRIX_A_SIZE_I; ++iIndex) {
        for (unsigned int jIndex = 0; jIndex < MATRIX_A_SIZE_J; ++jIndex) {
            matrixA[iIndex][jIndex] = (int)((long long int)iIndex *
MATRIX_A_SIZE_J + jIndex);
        }
    }
    //
    MyVLA<int> matrixB(NULL, 2, MATRIX_B_SIZE_J, MATRIX_B_SIZE_I);
    //
    for (unsigned int iIndex = 0; iIndex < MATRIX_B_SIZE_I; ++iIndex) {
        for (unsigned int jIndex = 0; jIndex < MATRIX_B_SIZE_J; ++jIndex) {
            if (iIndex == jIndex) {
                matrixB[iIndex][jIndex] = (int)(2);
            }
            else {
                matrixB[iIndex][jIndex] = (int)(0);
            }
        }
    }
    //
    printf("\\r\\n matrixA:\\r\\n");
    for (unsigned int iIndex = 0; iIndex < MATRIX_A_SIZE_I; ++iIndex) {
        for (unsigned int jIndex = 0; jIndex < MATRIX_A_SIZE_J; ++jIndex) {
            printf(" %5d ", (int)matrixA[iIndex][jIndex]);
        }
        printf("\\r\\n");
    }

    //
    MyVLA<int> matrixC(NULL, 2, MATRIX_C_SIZE_J, MATRIX_C_SIZE_I);
    // matrixC = matrixA * matrixB;
    for (int iIndex = 0; iIndex < MATRIX_C_SIZE_I; ++iIndex) {
        for (int jIndex = 0; jIndex < MATRIX_C_SIZE_J; ++jIndex) {
            matrixC[iIndex][jIndex] = 0;
            for (int kIndex = 0; kIndex < SIZE_2; ++kIndex) {

```

```

matrixC[iIndex][jIndex] += matrixA[iIndex][kIndex] *
matrixB[kIndex][jIndex];
    }
}

//
printf("\r\n matrixB:\r\n");
for (unsigned int iIndex = 0; iIndex < MATRIX_B_SIZE_I; ++iIndex) {
    for (unsigned int jIndex = 0; jIndex < MATRIX_B_SIZE_J; ++jIndex) {
        printf(" %5d ", (int)matrixB[iIndex][jIndex]);
    }
    printf("\r\n");
}

//
printf("\r\n matrixC = matrixA * matrixB:\r\n");
for (unsigned int iIndex = 0; iIndex < MATRIX_C_SIZE_I; ++iIndex) {
    for (unsigned int jIndex = 0; jIndex < MATRIX_C_SIZE_J; ++jIndex) {
        printf(" %5d ", (int)matrixC[iIndex][jIndex]);
    }
    printf("\r\n");
}

}

int main() {
    matrixMul_scenario();

#ifdef __linux__
    (void)getchar();
#elif defined(_WIN32)
    system("pause");
#else
#endif

    return 0;
}

```