

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий

институт

Кафедра «Информатика»

кафедра

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

Лабораторная работа No 1. Управление процессами в ОС GNU Linux

тема

Преподаватель

подпись, дата

А.С. Кузнецов

инициалы, фамилия

Студент КИ18-17/16 031831229

номер группы, зачетной книжки

подпись, дата

В.А. Прекель

инициалы, фамилия

Красноярск 2019

СОДЕРЖАНИЕ

1 Цель и задача работы.....	3
1.1 Цель работы.....	3
1.2 Задача работы.....	3
2 Описание и пояснение к работе.....	3
3 Структура проекта.....	4
3.1 CMakeLists.txt.....	4
3.2 ChildProgram/CMakeLists.txt.....	4
3.3 ChildProgram/Input.h.....	5
3.4 ChildProgram/Input.c.....	6
3.5 ChildProgram/Matrix.h.....	7
3.6 ChildProgram/Matrix.c.....	9
3.7 ChildProgram/MatrixTests.h.....	13
3.8 ChildProgram/MatrixTests.c.....	14
3.9 ChildProgram/main.c.....	19
3.10 ChildProgram/Doxyfile.....	24
3.11 ParentProgram/CMakeLists.txt.....	24
3.12 ParentProgram/main.c.....	24
4 Примеры использования.....	26
4.1 Запуск юнит-тестов.....	26
4.2 Запуск дочернего процесса.....	27
4.3 Запуск родительского процесса.....	28
4.4 Запуск ln -l / с помощью родительского процесса.....	29

1 Цель и задача работы

1.1 Цель работы

Изучение особенностей программной реализации многозадачных приложений в ОС GNU/Linux.

1.2 Задача работы

Требуется: разработать программу в виде Linux-приложения, которая представляет собой родительский процесс. Результат выполнения выводится на терминал/консоль. Программа должна быть устойчива к некорректному пользовательскому вводу. В следующих вариантах заданий оговаривается только функционал программы, представляющей собой дочерний процесс.

Вариант 16. Программа принимает от пользователя две квадратные матрицы одинакового размера, значение которого также вводит пользователь. а затем выводит на экран сумму матриц и значение ее определителя.

2 Описание и пояснение к работе

Используется система сборки CMake. Проект родительского процесса – ParentProgram. Дочернего – ChildProgram. Находятся в соответствующих папках, обязательно содержащих файл main.c с соответствующими функциями main.

В дочернем проекте используется библиотека юнит-тестирования, подключаемая компоновщиком статически. Для запуска юнит-тестов требуется запустить ChildProgram с единственным любым аргументом.

3 Структура проекта

Файловая структура проекта:

- CMakeLists.txt
- ChildProgram
 - + CMakeLists.txt
 - + Input.h
 - + Input.c
 - + Matrix.h
 - + Matrix.c
 - + MatrixTests.h
 - + MatrixTests.c
 - + main.c
 - + Doxyfile
- ParentProgram
 - + CMakeLists.txt
 - + main.c

3.1 CMakeLists.txt

```
cmake_minimum_required(VERSION 3.10)
project(Lab_01 C)
```

```
set(CMAKE_C_STANDARD 11)
```

```
add_subdirectory(ChildProgram)
add_subdirectory(ParentProgram)
```

3.2 ChildProgram/CMakeLists.txt

```

cmake_minimum_required(VERSION 3.10)
project(ChildProgram C)

set(CMAKE_C_STANDARD 11)

add_executable(ChildProgram main.c Matrix.c Matrix.h Input.c Input.h
               MatrixTests.c MatrixTests.h)

target_link_libraries(ChildProgram PRIVATE cunit)

```

3.3 ChildProgram/Input.h

```

/*! \file
* \brief Functions to input with checks
*
* \details Functions to input with checker.
* \bug If input with chars at end, "12fsdf" for example, chars will
* be ignored and 12 will be returned in example.
*/

#ifndef INPUT_H
#define INPUT_H

#include <stdbool.h>

/*! \brief Reads line
*
* \details Reads line.
*
* \param str Pointer to string of line input.
* \return String length.
*/
int InputLine(char* str);

/*! \brief Input int with checks.
*
* \details Write output phrase and reads integer with checker and check for
* non-number chars at begin.
*

```

```

    *   \param output String to output before input.
    *   \param pChecker Pointer to function which check value.
    *   \return Read integer.
    */
int CycleInputInt(char* output, bool(* pChecker)(int));

#endif // INPUT_H

```

3.4 ChildProgram/Input.c

```

/*! \file
 *
 *   \brief Implements functions of Input.h
 */

#include <stdbool.h>
#include <stdio.h>
#include <string.h>

#include "Input.h"

#define MAX_STRING_LENGTH 50

int InputLine(char* str)
{
    fgets(str, MAX_STRING_LENGTH, stdin);
    int size = strlen(str);
    str[strlen(str) - 1] = '\0';
    return size;
}

int CycleInputInt(char* output, bool(* pChecker)(int))
{
    int n;
    char stringNumber[MAX_STRING_LENGTH];
    while (true)
    {
        printf("%s", output);
        fflush(stdout);
    }
}

```

```

        InputLine(stringNumber);
        int code = sscanf(stringNumber, "%d", &n);
        if (!pChecker(n)) continue;
        if (code > 0) break;
    }
    return n;
}

```

3.5 ChildProgram/Matrix.h

```

/*! \file
* \brief Matrix structure and functions to use it
*
* \details Matrix structure and functions to calculate sum, determinant,
* etc...
* \bug Unclear first and second indices are row or column.
*/

#ifndef MATRIX_H
#define MATRIX_H

/*! \struct Matrix
* \brief Matrix struct
*
* \details Matrix structure. Uses memory allocation, needs frees up memory.
*/
typedef struct
{
    /*!
    * Maximum 1st index.
    */
    int FirstCount;
    /*!
    * SecondCount Maximum 2nd index.
    */
    int SecondCount;
    /*!
    * pData Matrix array. To get element use
    * matrix->pData[firstindex][secondindex].
    */
    int** pData;

```

```

} Matrix;

/*! \brief Create matrix filled with zeros
 *
 * \details Allocate memory for new matrix and initialize with zeros.
 *
 * \param firstCount Maximum 1st index.
 * \param secondCount Maximum 2nd index.
 * \return Pointer to new matrix.
 */
Matrix* CreateBlankMatrix(int firstCount, int secondCount);

/*! \brief Create non-initialized matrix
 *
 * \details Allocate memory for new matrix. Matrix filled with garbage.
 *
 * \param firstCount Maximum 1st index.
 * \param secondCount Maximum 2nd index.
 * \return Pointer to new matrix.
 */
Matrix* CreateEmptyMatrix(int firstCount, int secondCount);

/*! \brief Adds matrices
 *
 * \details Allocate memory for new matrix equals to matrixA + matrixB.
 *
 * \param pMatrixA 1st matrix to add.
 * \param pMatrixB 2nd matrix to add.
 * \return Pointer to sum.
 */
Matrix* SumMatrices(Matrix* pMatrixA, Matrix* pMatrixB);

/*! \brief Calculates minor of matrix
 *
 * \details Allocate memory for new matrix which has't specified row and
 *         column.
 *
 * \param pMatrix Matrix to create minor.
 * \param firstIndex 1st-dimension to exclude (column).
 * \param secondIndex 2nd-dimension to exclude (row).
 * \return Pointer to created minor.
 */
Matrix* GetMinor(Matrix* pMatrix, int firstIndex, int secondIndex);

```



```

/*! \brief Calculates determinant of 2x2 matrix
*
* \details Calculates determinant only for 2x2 matrices.
*
* \param pMatrix 2x2 matrix to calculate determinant.
* \return Determinant integer.
*/
int CalculateDeterminant2x2(Matrix* pMatrix);

/*! \brief Calculates determinant matrix
*
* \details Calculates matrix determinant.
*
* \param pMatrix Matrix to calculate determinant.
* \return Determinant integer.
*/
int CalculateDeterminant(Matrix* pMatrix);

/*! \brief Frees up matrix memory
*
* \details Frees up memory to destroy matrix
*
* \param pMatrix Matrix to free.
*/
void FreeMatrix(Matrix* pMatrix);

#endif // MATRIX_H

```

3.6 ChildProgram/Matrix.c

```

/*! \file
*
* \brief Implements functions of Matrix.h
*/

#include <malloc.h>
#include <assert.h>

#include "Matrix.h"

```

```

Matrix* CreateBlankMatrix(int firstCount, int secondCount)
{
    Matrix* pRet = (Matrix*) malloc(sizeof(Matrix));
    pRet->SecondCount = secondCount;
    pRet->FirstCount = firstCount;
    pRet->pData = (int**) malloc(pRet->FirstCount * sizeof(int));

    for (int i = 0; i < pRet->FirstCount; i++)
    {
        pRet->pData[i] = (int*) malloc(pRet->SecondCount * sizeof(int));
        for (int j = 0; j < pRet->SecondCount; j++)
        {
            pRet->pData[i][j] = 0;
        }
    }

    return pRet;
}

Matrix* CreateEmptyMatrix(int firstCount, int secondCount)
{
    Matrix* pRet = (Matrix*) malloc(sizeof(Matrix));
    pRet->SecondCount = secondCount;
    pRet->FirstCount = firstCount;
    pRet->pData = (int**) malloc(pRet->FirstCount * sizeof(int));

    for (int i = 0; i < pRet->FirstCount; i++)
    {
        pRet->pData[i] = (int*) malloc(pRet->SecondCount * sizeof(int));
    }
    return pRet;
}

Matrix* SumMatrices(Matrix* pMatrixA, Matrix* pMatrixB)
{
    assert(pMatrixA->SecondCount == pMatrixB->SecondCount);
    assert(pMatrixA->FirstCount == pMatrixB->FirstCount);

    Matrix* pRet =
        CreateEmptyMatrix(pMatrixA->FirstCount, pMatrixA->SecondCount);

```

```

    for (int i = 0; i < pRet->FirstCount; i++)
    {
        for (int j = 0; j < pRet->SecondCount; j++)
        {
            pRet->pData[i][j] = pMatrixA->pData[i][j] + pMatrixB->pData[i][j];
        }
    }

    return pRet;
}

```

```

Matrix* GetMinor(Matrix* pMatrix, int firstIndex, int secondIndex)
{
    Matrix* pRet = CreateEmptyMatrix(
        pMatrix->FirstCount - 1,
        pMatrix->SecondCount - 1);

    for (int i = 0; i < pRet->FirstCount; i++)
    {
        for (int j = 0; j < pRet->SecondCount; j++)
        {
            int oldi = i;
            int oldj = j;
            if (i >= firstIndex)
            {
                oldi++;
            }
            if (j >= secondIndex)
            {
                oldj++;
            }
            pRet->pData[i][j] = pMatrix->pData[oldi][oldj];
        }
    }

    return pRet;
}

```

```

int CalculateDeterminant2x2(Matrix* pMatrix)
{
    assert(pMatrix->FirstCount == 2);
    assert(pMatrix->SecondCount == 2);
}

```

```

    int a = pMatrix->pData[0][0];
    int b = pMatrix->pData[0][1];
    int c = pMatrix->pData[1][0];
    int d = pMatrix->pData[1][1];
    return a * d - c * b;
}

int CalculateDeterminant(Matrix* pMatrix)
{
    assert(pMatrix->FirstCount == pMatrix->SecondCount);

    int n = pMatrix->FirstCount;

    if (n == 1)
    {
        return pMatrix->pData[0][0];
    }
    if (n == 2)
    {
        return CalculateDeterminant2x2(pMatrix);
    }
    if (n >= 3)
    {
        int ret = 0;
        for (int i = 0; i < pMatrix->FirstCount; i++)
        {
            int sign = (i % 2) ? -1 : 1;
            Matrix* minor = GetMinor(pMatrix, 0, i);
            int det = CalculateDeterminant(minor);
            int firstRow = pMatrix->pData[0][i];
            ret += sign * firstRow * det;
            FreeMatrix(minor);
        }
        return ret;
    }
}

void FreeMatrix(Matrix* pMatrix)
{
    for (int i = 0; i < pMatrix->FirstCount; i++)
    {

```

```

        free(pMatrix->pData[i]);
    }
    free(pMatrix->pData);
    free(pMatrix);
}

```

3.7 ChildProgram/MatrixTests.h

```

/*! \file
 * \brief Tests for matrix
 *
 * \details Tests for matrix structure and functions.
 */

#ifndef MATRIXTESTS_H
#define MATRIXTESTS_H

#include <Unit/Basic.h>

#include "Matrix.h"

/*! \brief Test for create an empty 3x2 Matrix
 */
void Blank3x2_MatrixTest(void);

/*! \brief Test for sum 2x2 Matrix
 */
void Sum2x2_MatrixTest(void);

/*! \brief Test for calculate determinant 2x2 Matrix
 */
void Det2x2_MatrixTest(void);

/*! \brief Test for calculate determinant 3x3 Matrix
 */
void Det3x3_MatrixTest(void);

/*! \brief Test for calculate determinant 7x7 Matrix
 */

```

```

void Det7x7_MatrixTest(void);

/*! \brief Test for calculate 2x3 minor for 3x4 Matrix
  */
void Minor3x4_MatrixTest(void);

#endif // MATRIXTESTS_H

```

3.8 ChildProgram/MatrixTests.c

```

/*! \file
  *
  * \brief Implements tests of MatrixTests.h
  */

#include "MatrixTests.h"

void Blank3x2_MatrixTest(void)
{
    Matrix* pMatrix = CreateBlankMatrix(3, 2);

    CU_ASSERT_EQUAL(pMatrix->FirstCount, 3);
    CU_ASSERT_EQUAL(pMatrix->SecondCount, 2);

    CU_ASSERT_EQUAL(pMatrix->pData[0][0], 0);
    CU_ASSERT_EQUAL(pMatrix->pData[1][1], 0);
    CU_ASSERT_EQUAL(pMatrix->pData[2][0], 0);
    CU_ASSERT_EQUAL(pMatrix->pData[0][1], 0);
    CU_ASSERT_EQUAL(pMatrix->pData[1][0], 0);
    CU_ASSERT_EQUAL(pMatrix->pData[2][1], 0);

    FreeMatrix(pMatrix);
}

void Sum2x2_MatrixTest(void)
{
    Matrix* pMatrix1 = CreateBlankMatrix(2, 2);
    Matrix* pMatrix2 = CreateBlankMatrix(2, 2);

```

```

    pMatrix1->pData[0][0] = 1;
    pMatrix1->pData[0][1] = -65;
    pMatrix1->pData[1][0] = 33;
    pMatrix1->pData[1][1] = -33;
    pMatrix2->pData[0][0] = 1;
    pMatrix2->pData[0][1] = 12;
    pMatrix2->pData[1][0] = -8;
    pMatrix2->pData[1][1] = 3;

    Matrix* pSum = SumMatrices(pMatrix1, pMatrix2);

    CU_ASSERT_EQUAL(pSum->pData[0][0], 2);
    CU_ASSERT_EQUAL(pSum->pData[0][1], -53);
    CU_ASSERT_EQUAL(pSum->pData[1][0], 25);
    CU_ASSERT_EQUAL(pSum->pData[1][1], -30);

    FreeMatrix(pMatrix1);
    FreeMatrix(pMatrix2);
    FreeMatrix(pSum);
}

void Det2x2_MatrixTest(void)
{
    Matrix* pMatrix = CreateBlankMatrix(2, 2);

    pMatrix->pData[0][0] = 1;
    pMatrix->pData[0][1] = -65;
    pMatrix->pData[1][0] = 33;
    pMatrix->pData[1][1] = -33;

    int det = CalculateDeterminant(pMatrix);
    int det2x2 = CalculateDeterminant2x2(pMatrix);

    CU_ASSERT_EQUAL(det, det2x2);

    CU_ASSERT_EQUAL(det, 2112);

    FreeMatrix(pMatrix);
}

```

```

void Det3x3_MatrixTest(void)
{
    Matrix* pMatrix = CreateBlankMatrix(3, 3);

    pMatrix->pData[0][0] = 1;
    pMatrix->pData[0][1] = -65;
    pMatrix->pData[0][2] = -2;
    pMatrix->pData[1][0] = 33;
    pMatrix->pData[1][1] = -33;
    pMatrix->pData[1][2] = -6;
    pMatrix->pData[2][0] = 33;
    pMatrix->pData[2][1] = 4;
    pMatrix->pData[2][2] = -65;

    int det = CalculateDeterminant(pMatrix);

    CU_ASSERT_EQUAL(det, -126828);

    FreeMatrix(pMatrix);
}

```

```

void Det7x7_MatrixTest(void)
{
    Matrix* pMatrix = CreateBlankMatrix(7, 7);

    pMatrix->pData[0][0] = 1;
    pMatrix->pData[0][1] = 6;
    pMatrix->pData[0][2] = -2;
    pMatrix->pData[0][3] = 2;
    pMatrix->pData[0][4] = 1;
    pMatrix->pData[0][5] = 6;
    pMatrix->pData[0][6] = 2;

    pMatrix->pData[1][0] = 3;
    pMatrix->pData[1][1] = 3;
    pMatrix->pData[1][2] = 6;
    pMatrix->pData[1][3] = 3;
    pMatrix->pData[1][4] = 3;
    pMatrix->pData[1][5] = -6;
    pMatrix->pData[1][6] = 6;

```



```
pMatrix->pData[2][0] = 3;
pMatrix->pData[2][1] = -4;
pMatrix->pData[2][2] = 6;
pMatrix->pData[2][3] = 3;
pMatrix->pData[2][4] = 4;
pMatrix->pData[2][5] = 6;
pMatrix->pData[2][6] = 3;
```

```
pMatrix->pData[3][0] = -3;
pMatrix->pData[3][1] = 4;
pMatrix->pData[3][2] = -6;
pMatrix->pData[3][3] = 3;
pMatrix->pData[3][4] = 4;
pMatrix->pData[3][5] = -6;
pMatrix->pData[3][6] = 3;
```

```
pMatrix->pData[4][0] = 3;
pMatrix->pData[4][1] = -3;
pMatrix->pData[4][2] = 6;
pMatrix->pData[4][3] = 3;
pMatrix->pData[4][4] = -3;
pMatrix->pData[4][5] = 6;
pMatrix->pData[4][6] = 6;
```

```
pMatrix->pData[5][0] = 1;
pMatrix->pData[5][1] = 6;
pMatrix->pData[5][2] = 2;
pMatrix->pData[5][3] = -2;
pMatrix->pData[5][4] = 1;
pMatrix->pData[5][5] = 6;
pMatrix->pData[5][6] = 2;
```

```
pMatrix->pData[6][0] = 1;
pMatrix->pData[6][1] = 6;
pMatrix->pData[6][2] = 2;
pMatrix->pData[6][3] = 2;
pMatrix->pData[6][4] = 1;
pMatrix->pData[6][5] = 6;
pMatrix->pData[6][6] = 2;
```

```
int det = CalculateDeterminant(pMatrix);
```

```

    CU_ASSERT_EQUAL(det, 355968);

    FreeMatrix(pMatrix);
}

void Minor3x4_MatrixTest(void)
{
    Matrix* pMatrix = CreateBlankMatrix(3, 4);

    pMatrix->pData[0][0] = 1;
    pMatrix->pData[0][1] = 2;
    pMatrix->pData[0][2] = 3;
    pMatrix->pData[0][3] = 4;

    pMatrix->pData[1][0] = 5;
    pMatrix->pData[1][1] = 6;
    pMatrix->pData[1][2] = 7;
    pMatrix->pData[1][3] = 8;

    pMatrix->pData[2][0] = 9;
    pMatrix->pData[2][1] = 10;
    pMatrix->pData[2][2] = 11;
    pMatrix->pData[2][3] = 12;

    Matrix* pMinor1 = GetMinor(pMatrix, 0, 0);

    CU_ASSERT_EQUAL(pMinor1->FirstCount, 2);
    CU_ASSERT_EQUAL(pMinor1->SecondCount, 3);
    CU_ASSERT_EQUAL(pMinor1->pData[0][0], 6);
    CU_ASSERT_EQUAL(pMinor1->pData[0][1], 7);
    CU_ASSERT_EQUAL(pMinor1->pData[0][2], 8);
    CU_ASSERT_EQUAL(pMinor1->pData[1][0], 10);
    CU_ASSERT_EQUAL(pMinor1->pData[1][1], 11);
    CU_ASSERT_EQUAL(pMinor1->pData[1][2], 12);

    Matrix* pMinor2 = GetMinor(pMatrix, 1, 1);

    CU_ASSERT_EQUAL(pMinor2->FirstCount, 2);
    CU_ASSERT_EQUAL(pMinor2->SecondCount, 3);

```

```

CU_ASSERT_EQUAL(pMinor2->pData[0][0], 1);
CU_ASSERT_EQUAL(pMinor2->pData[0][1], 3);
CU_ASSERT_EQUAL(pMinor2->pData[0][2], 4);
CU_ASSERT_EQUAL(pMinor2->pData[1][0], 9);
CU_ASSERT_EQUAL(pMinor2->pData[1][1], 11);
CU_ASSERT_EQUAL(pMinor2->pData[1][2], 12);

Matrix* pMinor3 = GetMinor(pMatrix, 2, 3);

CU_ASSERT_EQUAL(pMinor3->FirstCount, 2);
CU_ASSERT_EQUAL(pMinor3->SecondCount, 3);
CU_ASSERT_EQUAL(pMinor3->pData[0][0], 1);
CU_ASSERT_EQUAL(pMinor3->pData[0][1], 2);
CU_ASSERT_EQUAL(pMinor3->pData[0][2], 3);
CU_ASSERT_EQUAL(pMinor3->pData[1][0], 5);
CU_ASSERT_EQUAL(pMinor3->pData[1][1], 6);
CU_ASSERT_EQUAL(pMinor3->pData[1][2], 7);

Matrix* pMinor4 = GetMinor(pMatrix, 2, 1);

CU_ASSERT_EQUAL(pMinor4->FirstCount, 2);
CU_ASSERT_EQUAL(pMinor4->SecondCount, 3);
CU_ASSERT_EQUAL(pMinor4->pData[0][0], 1);
CU_ASSERT_EQUAL(pMinor4->pData[0][1], 3);
CU_ASSERT_EQUAL(pMinor4->pData[0][2], 4);
CU_ASSERT_EQUAL(pMinor4->pData[1][0], 5);
CU_ASSERT_EQUAL(pMinor4->pData[1][1], 7);
CU_ASSERT_EQUAL(pMinor4->pData[1][2], 8);

FreeMatrix(pMatrix);
FreeMatrix(pMinor1);
FreeMatrix(pMinor2);
FreeMatrix(pMinor3);
FreeMatrix(pMinor4);
}

```

3.9 ChildProgram/main.c

```

/*! \file
* \brief Main file of child program

```

```

*
*  \details Main file which contains the main function.
*/

#include <stdio.h>
#include <stdbool.h>
#include <malloc.h>

#include <Unit/Basic.h>

#include "Matrix.h"
#include "Input.h"
#include "MatrixTests.h"

/*! \brief Check number for matrix size
*
*  \details Check number for matrix size which must be great than 0.
*
*  \param n Number to check.
*  \return Bool.
*/
bool MatrixSizeChecker(int n)
{
    return n >= 1;
}

/*! \brief Check number for matrix element
*
*  \details Check number for matrix element which may be any integer.
*
*  \param n Number to check.
*  \return Always true bool.
*/
bool MatrixElementChecker(int n)
{
    return true;
}

/*! \brief Main function
*
*  \details Main function. If one argument given, will be run unit tests.

```

```

* Else will be program which reads two matrices, adds them and calculates
* determinant of sum.
*
* \param argc Count program arguments.
* \param argv Array string which contains args.
* \return Integer 0 upon exit success.
*/
int main(int argc, char** argv)
{
    if (argc == 2)
    {
        CU_pSuite suite;
        CU_initialize_registry();
        suite = CU_add_suite("main_suite", NULL, NULL);
        CU_ADD_TEST(suite, Blank3x2_MatrixTest);
        CU_ADD_TEST(suite, Sum2x2_MatrixTest);
        CU_ADD_TEST(suite, Det2x2_MatrixTest);
        CU_ADD_TEST(suite, Det3x3_MatrixTest);
        CU_ADD_TEST(suite, Det7x7_MatrixTest);
        CU_ADD_TEST(suite, Minor3x4_MatrixTest);
        CU_basic_run_tests();

        CU_cleanup_registry();
        return CU_get_error();
    }

    int firstCount = 0;
    int secondCount = 0;

    firstCount = secondCount =
        CycleInputInt("Введите порядок матриц: ", MatrixSizeChecker);

    Matrix* pMatrix1 = CreateBlankMatrix(firstCount, secondCount);
    Matrix* pMatrix2 = CreateBlankMatrix(firstCount, secondCount);

    printf("Введите матрицу pMatrix1:\n");
    fflush(stdout);
    for (int i = 0; i < firstCount; i++)
    {
        for (int j = 0; j < secondCount; j++)
        {
            char* format = "pMatrix1[%d][%d] = ";

```

```

        ssize_t len = snprintf(NULL, 0, format, i, j);
        char* s = (char*) malloc(len + 1 * sizeof(char));
        snprintf(s, len + 1, format, i, j);

        pMatrix1->pData[i][j] = CycleInputInt(s, MatrixElementChecker);

        free(s);
    }
}
printf("\n");
fflush(stdout);

printf("Введите матрицу pMatrix2:\n");
fflush(stdout);
for (int i = 0; i < firstCount; i++)
{
    for (int j = 0; j < secondCount; j++)
    {
        char* format = "pMatrix2[%d][%d] = ";
        ssize_t len = snprintf(NULL, 0, format, i, j);
        char* s = (char*) malloc(len + 1 * sizeof(char));
        snprintf(s, len + 1, format, i, j);

        pMatrix2->pData[i][j] = CycleInputInt(s, MatrixElementChecker);

        free(s);
    }
}
printf("\n");
fflush(stdout);

Matrix* pSum = SumMatrices(pMatrix1, pMatrix2);

printf("Матрица pMatrix1:\n");
for (int i = 0; i < firstCount; i++)
{
    for (int j = 0; j < secondCount; j++)
    {
        printf("%d ", pMatrix1->pData[i][j]);
        fflush(stdout);
    }
}

```

```

        printf("\n");
    }
    printf("\n");
    fflush(stdout);

    printf("Матрица pMatrix2:\n");
    for (int i = 0; i < firstCount; i++)
    {
        for (int j = 0; j < secondCount; j++)
        {
            printf("%d ", pMatrix2->pData[i][j]);
            fflush(stdout);
        }
        printf("\n");
    }
    printf("\n");
    fflush(stdout);

    printf("Сумма матриц pMatrix1 + pMatrix2:\n");
    for (int i = 0; i < firstCount; i++)
    {
        for (int j = 0; j < secondCount; j++)
        {
            printf("%d ", pSum->pData[i][j]);
            fflush(stdout);
        }
        printf("\n");
    }
    printf("\n");
    fflush(stdout);

    int det = CalculateDeterminant(pSum);
    printf("Определитель суммы матриц det = %d\n", det);
    fflush(stdout);

    FreeMatrix(pMatrix1);
    FreeMatrix(pMatrix2);
    FreeMatrix(pSum);

    return 0;

```

```
}
```

3.10 ChildProgram/Doxyfile

Файл слишком большой.

3.11 ParentProgram/CMakeLists.txt

```
cmake_minimum_required(VERSION 3.10)
project(ParentProgram C)

set(CMAKE_C_STANDARD 11)

add_executable(ParentProgram main.c)
```

3.12 ParentProgram/main.c

```
/*! \file
 * \brief Main file of parent program
 *
 * \details Main file which contains the main function.
 */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <wait.h>

/*! \brief Spawn a child process
 *
 * \details Spawn a child process running a new program, as given by PROGRAM.
 *
 * \param program The name of the program to runs;
 *                the path will be searched for this program.
 * \param argList A NULL-terminated list of character strings to be
 *                passed as the program's argument list.
```



```

    * \return          Identifier of the spawned process.
    */
pid_t Spawn(char* program, char** argList)
{
    pid_t childPid;

    childPid = fork();

    if (childPid == 0)
    {
        execvp(program, argList);

        fprintf(stderr, "Ошибка при выполнении execvp\n");
        fflush(stderr);
        abort();
    }

    return childPid;
}

/*! \brief Main function
 *
 * \details Main function. If no argument given, will be exec Child
 * program via relative path "../ChildProgram/ChildProgram". If arguments
 * given, will be exec program in argument. Then wait child for exit.
 *
 * \param argc Count program arguments.
 * \param argv Array string which contains args.
 * \return Integer 0 upon exit success.
 */
int main(int argc, char** argv)
{
    char* child = NULL;
    char** argList = NULL;

    if (argc > 1)
    {
        child = argv[1];
        argList = argv + 1;
    } else
    {

```

```

        child = "../ChildProgram/ChildProgram";
        char* argListTmp[] = {child, NULL};
        argList = argListTmp;
    }

    int childPid = Spawn(child, argList);
    waitpid(childPid, NULL, 0);

    return 0;
}

```

4 Примеры использования

4.1 Запуск юнит-тестов

/home/vladislav/Projects/SystemProgramming/cmake-build-debug/0x02/Lab_01/ChildProgram/
ChildProgram 123

CUnit - A unit testing framework for C - Version 3.1.0-cunity
<http://cunit.sourceforge.net/>

Run Summary	-	Run	Failed	Inactive	Skipped
Suites	:	1	0	0	0
Asserts	:	48	0	n/a	n/a
Tests	:	6	0	0	0

Elapsed Time: 0.002(s)

Process finished with exit code 0

4.2 Запуск дочернего процесса

```
/home/vladislav/Projects/SystemProgramming/cmake-build-debug/0x02/Lab_01/ChildProgram/  
ChildProgram
```

Введите порядок матриц: 2

Введите матрицу pMatrix1:

```
pMatrix1[0][0] = 1
```

```
pMatrix1[0][1] = 2
```

```
pMatrix1[1][0] = qwe
```

```
pMatrix1[1][0] = dsaf
```

```
pMatrix1[1][0] = 3
```

```
pMatrix1[1][1] = 4
```

Введите матрицу pMatrix2:

```
pMatrix2[0][0] = qweq
```

```
pMatrix2[0][0] = wqe
```

```
pMatrix2[0][0] = 123
```

```
pMatrix2[0][1] = -312
```

```
pMatrix2[1][0] = 2
```

```
pMatrix2[1][1] = 1
```

Матрица pMatrix1:

```
1 2
```

```
3 4
```

Матрица pMatrix2:

```
123 -312
```

```
2 1
```

Сумма матриц pMatrix1 + pMatrix2:

```
124 -310
```

```
5 5
```

Определитель суммы матриц det = 2170

Process finished with exit code 0

4.3 Запуск родительского процесса

```
/home/vladislav/Projects/SystemProgramming/cmake-build-debug/0x02/Lab_01/ParentProgram/
```

```
ParentProgram
```

```
Введите порядок матриц: 3
```

```
Введите матрицу pMatrix1:
```

```
pMatrix1[0][0] = -21
```

```
pMatrix1[0][1] = 1
```

```
pMatrix1[0][2] = 3
```

```
pMatrix1[1][0] = 4
```

```
pMatrix1[1][1] = 5
```

```
pMatrix1[1][2] = 6
```

```
pMatrix1[2][0] = 7
```

```
pMatrix1[2][1] = 8
```

```
pMatrix1[2][2] = 9
```

```
Введите матрицу pMatrix2:
```

```
pMatrix2[0][0] = 1
```

```
pMatrix2[0][1] = 1
```

```
pMatrix2[0][2] = 1
```

```
pMatrix2[1][0] = 1
```

```
pMatrix2[1][1] = 1
```

```
pMatrix2[1][2] = -1
```

```
pMatrix2[2][0] = -1
```

```
pMatrix2[2][1] = 0
```

```
pMatrix2[2][2] = 0
```

```
Матрица pMatrix1:
```

```
-21 1 3
```

```
4 5 6
```

```
7 8 9
```

```
Матрица pMatrix2:
```

```
1 1 1
```

```
1 1 -1
```

```
-1 0 0
```

```
Сумма матриц pMatrix1 + pMatrix2:
```

```
-20 2 4
```

```
5 6 5
```

```
6 8 9
```

Определитель суммы матриц $\det = -294$

Process finished with exit code 0

4.4 Запуск `ln -l` / с помощью родительского процесса

```
/home/vladislav/Projects/SystemProgramming/cmake-build-debug/0x02/Lab_01/ParentProgram/
ParentProgram ls -l /
total 2151796
drwxr-xr-x  2 root root      4096 сен  4 10:46 bin
drwxr-xr-x  3 root root      4096 сен  7 10:48 boot
drwxrwxr-x  2 root root      4096 фев  4  2019 cdrom
-rw-----  1 root root 55828480 фев  4  2019 core
drwxr-xr-x 19 root root      4500 сен  7 11:19 dev
drwxr-xr-x 146 root root    12288 сен  7 12:44 etc
drwxr-xr-x  3 root root      4096 фев  4  2019 home
lrwxrwxrwx  1 root root       32 сен  4 10:30 initrd.img -> boot/initrd.img-5.0.0-27-
generic
lrwxrwxrwx  1 root root       32 сен  4 10:30 initrd.img.old -> boot/initrd.img-
5.0.0-25-generic
drwxr-xr-x 21 root root      4096 апр 20 11:53 lib
drwxr-xr-x  2 root root      4096 апр 20 11:40 lib32
drwxr-xr-x  2 root root      4096 апр 20 11:39 lib64
drwx-----  2 root root    16384 фев  4  2019 lost+found
drwxr-xr-x  3 root root      4096 фев  4  2019 media
drwxr-xr-x  2 root root      4096 окт 18  2018 mnt
drwxr-xr-x  4 root root      4096 мар 15 13:15 opt
dr-xr-xr-x 224 root root       0 сен  7 09:40 proc
drwx-----  6 root root      4096 фев 22  2019 root
drwxr-xr-x 33 root root     1000 сен  7 19:34 run
drwxr-xr-x  2 root root    12288 сен  4 10:46 sbin
drwxr-xr-x 10 root root      4096 июл 19 22:10 snap
drwxr-xr-x  2 root root      4096 окт 18  2018 srv
-rw-----  1 root root 2147483648 июл 20 12:29 swapfile
dr-xr-xr-x 13 root root       0 сен  7 12:53 sys
drwxrwxrwt 19 root root    20480 сен  7 19:53 tmp
drwxr-xr-x 12 root root      4096 апр 20 11:49 usr
drwxr-xr-x 14 root root      4096 окт 18  2018 var
lrwxrwxrwx  1 root root       29 сен  4 10:30 vmlinuz -> boot/vmlinuz-5.0.0-27-
generic
lrwxrwxrwx  1 root root       29 сен  4 10:30 vmlinuz.old -> boot/vmlinuz-5.0.0-25-
```

generic

Process finished with exit code 0