# Лабораторная работа №3

## Цель работы

Изучить процесс создания динамических библиотек при помощи набора компиляторов иутилит GCC и особенности их применения.

## Задание

1. Написать программу в соответствие с вариантом.
2. Массив и матрицу заполнять случайными числами от -50 до 50.
3. Функции для работы с массивами и матрицами поместить в две отдельные динамические библиотеки.
4. При запуске программы пользователю должно быть представлено меню, в котором можно выбрать с чем будет происходить работа: с матрицей или с массивом.
5. В зависимости от выбора пользователя, загружается одна или другая динамическая библиотека.
6. Библиотеки должны быть скомпилированы с учётом возможного использования в ОС семейств Linux или Windows/
7. Основная программа должна при помощи директив препроцессора поддерживать мультиплатформенность в рамках этих двух семейств ОС.

### Вариативная часть (Вариант 14)

В массиве K (73) и матрице R (7х10) найти число элементов, которые делятся на 7 без остатка.

## Результат выполненной работы

Исходный код (**файл main.c**):

#include <stdio.h>  
#include <stdlib.h>  
  
#ifdef \_\_linux  
  
#include <dlfcn.h>  
  
void LoadRun(const char\* const s, const char\* const func\_name) {  
 void\* lib;  
 void (\*fun)(void);  
 lib = dlopen(s, RTLD\_LAZY);  
 if (!lib) {  
 printf("cannot open library '%s'\n", s);  
 return;  
 }  
 fun = (void (\*)(void))dlsym(lib, func\_name);  
 if (fun == NULL) {  
 printf("cannot load function func\n");  
 }  
 else {  
 fun();  
 }  
 dlclose(lib);  
}  
  
#elif defined \_WIN32  
  
#include <windows.h>  
  
void LoadRun(const char\* const s, const char\* const func\_name) {  
 void\* lib;  
 void (\*fun)(void);  
 lib = LoadLibrary(s);  
 if (!lib) {  
 printf("cannot open library '%s'\n", s);  
 return;  
 }  
 fun = (void (\*)(void))GetProcAddress((HINSTANCE)lib, func\_name);  
 if (fun == NULL) {  
 int error = GetLastError();  
 printf("cannot load function %s, error is %d\n", func\_name, error);  
 }  
 else {  
 fun();  
 }  
 FreeLibrary((HINSTANCE)lib);  
}  
  
#endif  
  
#include "array\_proc.h"  
#include "matrix\_proc.h"  
#include "iostream"  
using namespace std;  
  
  
int main(int argc, char\*\* argv)  
{  
 setlocale(LC\_ALL, "Rus");  
  
   
 int selected = 1;  
   
  
 while (selected)  
 {  
  
 cout << "Что вы хотите использовать?" << endl;  
 cout << "1. Массив" << endl;  
 cout << "2. Матрица" << endl;  
 cout << "0. Выход" << endl;  
  
  
 cin >> selected;  
 switch(selected)  
 {  
  
 case 1:  
 {  
 LoadRun("array\_proc.dll", "array\_proc\0");  
  
 int\* mas = new int[73];  
  
 for (int i = 0; i < 73; i++)  
 {   
 mas[i] = rand() % 100 - 50;  
 }  
 int count\_7 = array\_proc(mas, 72);  
 printf("Результат %d\n\n", count\_7);  
  
 delete[] mas;  
 break;  
 }  
  
 case 2:  
 {  
 LoadRun("matrix\_proc.dll", "matrix\_proc\0");  
  
  
 int\*\* matrix = new int\*[7];  
 for (int i = 0; i < 7; i++)  
 {  
 matrix[i] = new int[10];  
 }  
  
 for (int i = 0; i < 7; i++)  
 {  
 for (int j = 0; j < 10; j++)  
 {  
 matrix[i][j] = rand() % 100 - 50;  
 }  
 }  
 int count\_matrix\_7 = matrix\_proc(matrix, 7, 10);  
 printf("Результат %d\n\n", count\_matrix\_7);  
   
 break;  
 }  
  
 }  
  
 }  
  
 return 0;  
}

Исходный код (файл array\_proc.h):

#pragma once  
  
extern "C" \_\_declspec(dllexport) int array\_proc(int\* mas, int count);

Исходный код (файл array\_proc.c):

#include "array\_proc.h"  
  
extern "C" \_\_declspec(dllexport) int array\_proc(int\* mas, int count)  
{  
 int count\_7 = 0;  
 int i = 0;  
 for (; i < count; i++)  
 {  
 if (mas[i] % 7 == 0)  
 {  
 count\_7++;  
 }  
 }  
 return count\_7;  
}

Исходный код (файл matrix\_proc.h):

#pragma once  
  
extern "C" \_\_declspec(dllexport) int matrix\_proc(int\*\* matrix, int i, int j);

Исходный код (файл matrix\_proc.c):

#include "matrix\_proc.h"  
  
extern "C" \_\_declspec(dllexport) int matrix\_proc(int\* matrix[], int m, int n)  
{  
 int count\_7 = 0;  
 for (int i = 0; i < m; i++)  
 {  
 for (int j = 0; j < n; j++)  
 {  
 if (matrix[i][j] % 7 == 0)  
 {  
 count\_7++;  
 }  
 }  
 }  
 return count\_7;  
}

На основе файлов с исходным кодом matrix\_proc.c и array\_proc.c были созданы объектные файлы с позиционно-независимым кодом с помощью команды g++ -c -fPIC -o array\_proc.o array\_proc.cpp и команды g++ -c -fPIC -o matrix\_proc.o matrix\_proc.cpp.

На основе этих объектных файлов были созданы динамические библиотеки array\_proc.dll и matrix\_proc.dll с помощью команд g++ -shared -o array\_proc.dll array\_proc.o и g++ -shared -o matrix\_proc.dll matrix\_proc.o соотвественно.

Для кроссплатформенности в файле main.cpp были добавлены дерективы препроцессора.

#ifdef \_\_linux  
  
#include <dlfcn.h>  
  
void LoadRun(const char\* const s, const char\* const func\_name) {  
 void\* lib;  
 void (\*fun)(void);  
 lib = dlopen(s, RTLD\_LAZY);  
 if (!lib) {  
 printf("cannot open library '%s'\n", s);  
 return;  
 }  
 fun = (void (\*)(void))dlsym(lib, func\_name);  
 if (fun == NULL) {  
 printf("cannot load function func\n");  
 }  
 else {  
 fun();  
 }  
 dlclose(lib);  
}  
  
#elif defined \_WIN32  
  
#include <windows.h>  
  
void LoadRun(const char\* const s, const char\* const func\_name) {  
 void\* lib;  
 void (\*fun)(void);  
 lib = LoadLibrary(s);  
 if (!lib) {  
 printf("cannot open library '%s'\n", s);  
 return;  
 }  
 fun = (void (\*)(void))GetProcAddress((HINSTANCE)lib, func\_name);  
 if (fun == NULL) {  
 int error = GetLastError();  
 printf("cannot load function %s, error is %d\n", func\_name, error);  
 }  
 else {  
 fun();  
 }  
 FreeLibrary((HINSTANCE)lib);  
}  
  
#endif

Так же в файле main.cpp было добавлено меню для выбора нужной библиотеки.

По итогу выполнения лабораторной работы получилась работоспособная программа, выполняющая все требования задания.

Программа была протестирована, далее представлены снимок экрана с работающей программой.

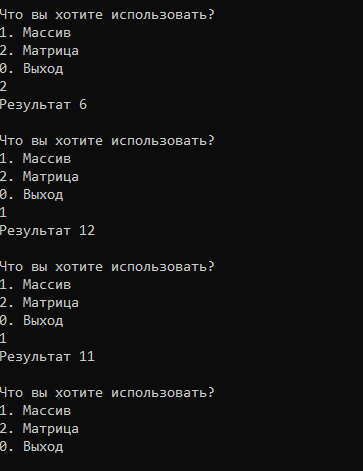


Рисунок 3.1 — Демонстрация работы программы