МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

## Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

**Отчет по лабораторной работе № 7**

по дисциплине:” Системное программирование”

на тему: ”*Автоматическая сборка многофайловых проектов в Linux****”***

Выполнил**:** студент группы 10702221

Будчанин В.А.

Жегздринь А.Р.

Принял**:** Давыденко Н.В.

Минск, 2023

**Цель работы :**

Изучить встроенный инструментарий для разработки приложений под семейство ОС Linux и фундаментальные основы системного программирования с использованием компиляторов gcc/g++, отладчика gdb и других для проектирования, компиляции, отладки и запуска приложений на языке программирования С/C++

**Требования :**

1) Разработать модульное консольное приложение на С/С++ согласно варианту задания. Для компиляции, компоновки и выполнения программы использовать стандартный компилятор Linux gcc/g++, а для отладки - gdb.

2) Размерность массива задаётся пользователем на стадии выполнения программы. Для повышения производительности программы необходимо использовать динамическое выделение памяти.

3) Во время работы программы должны выводить на экран исходные и конечные данные.

4) При выполнении задания запрещается использовать интегрированные средства разработки (Integrated Development Environment, IDE). Рекомендуется задействовать любой текстовый редактор (к примеру, gedit) и набор компиляторов GNU Compiler Collection (GCC), в частности, компиляторы языков программирования С/C++ gcc/g++.

5) При разработке программ необходимо придерживаться соглашения по написанию кода на C/C++ (Code-Convention).

**Упражнения:**

**Упражнение 1.**

**Изучите и выполните требования, представленные в лабораторной работе №2 «Компиляция и отладка простейшего приложения в Linux». Разработайте исходные коды приложений в соответствии с заданиями. Компиляцию приложений проведите с помощью консольных команд**

***Задание лабораторной работы №2****. Компиляция и отладка простейшего приложения в Linux.*

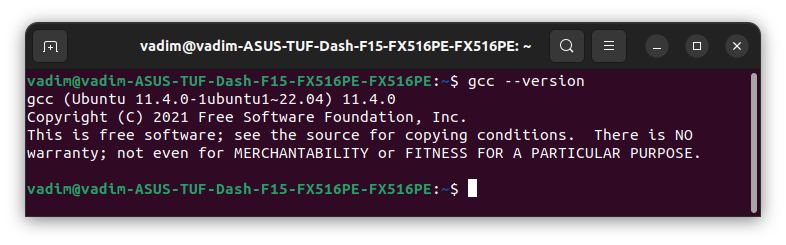
Ввести массив вещественных чисел размером N. Найти его наибольший и наименьший элементы и поменять их местами. Найти сумму и произведение всех элементов массива: в одномерном массиве, состоящем из вещественных элементов, вычислить: количество элементов массива, больших заданного числа и произведение элементов массива, расположенных после максимального по модулю элемент.

**Упражнение 2.**

**Изучите и выполните требования, представленные в лабораторной работе №3 «Многофайловые проекты и шаблон MVC». Разработайте исходные коды приложений в соответствии с заданиями. Компиляцию приложений проведите с помощью консольных команд**

***Задание лабораторной работы №3****. Многофайловые проекты и шаблон MVC.* Разработать интерактивную программу «TrytoGuesstheNumber» («Попробуй угадать число»), которая эмулирует классическую игру на отгадывание числа. Суть программы (игры) сводиться к следующему: компьютер генерирует случайное число из диапазона, к примеру, от 1 до 100, а пользователь (далее игрок) пытается отгадать число за минимальное количество попыток. При каждой очередной попытке компьютер «подсказывает» игроку, как соизмеряется предложенный вариант игрока с действительным загаданным компьютером числом: загаданное число больше или меньше указанного (higher/lower). Как только игрок отгадывает число, компьютер должен «поздравить» его с выводом на экран угаданного числа и количество затраченных игроком попыток. Далее компьютер может «предложить» повторно сыграть игру или выйти из программы. Для универсальности предложенной программы можно добавить возможность выбора диапазона генерирования компьютером случайных чисел, а также задания ограничения на количество попыток. В случае, если игрок не укладывается в заданное количество попыток (т.е. проигрывает), программа должна выводить суровую надпись «GameOver».

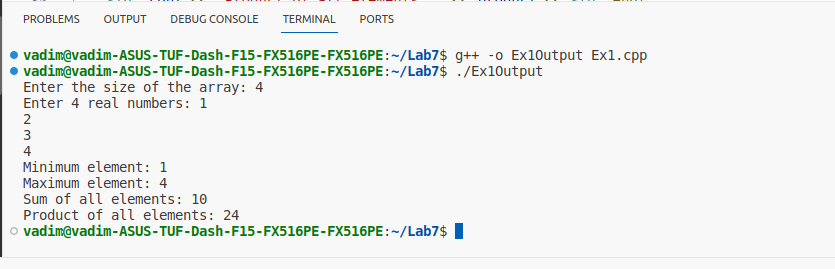
**Выполнение упражнений:**

Версия gcc

**Задание лабораторной №2.**

**Основное**

Ввести массив вещественных чисел размером N. Найти его наибольший и наименьший элементы и поменять их местами. Найти сумму и произведение всех элементов массива.



**Код к заданию:**

#include <iostream>

#include <limits>

#include <vector>

#include <algorithm>

int main() {

int N;

std::cout << "Enter the size of the array: ";

std::cin >> N;

std::vector<double> arr(N);

std::cout << "Enter " << N << " real numbers: ";

for (int i = 0; i < N; ++i) {

std::cin >> arr[i];

}

double minElement = \*std::min\_element(arr.begin(), arr.end());

double maxElement = \*std::max\_element(arr.begin(), arr.end());

int minIndex = std::min\_element(arr.begin(), arr.end()) - arr.begin();

int maxIndex = std::max\_element(arr.begin(), arr.end()) - arr.begin();

std::swap(arr[minIndex], arr[maxIndex]);

double sum = 0;

double product = 1;

for (int i = 0; i < N; ++i) {

sum += arr[i];

product \*= arr[i];

}

std::cout << "Minimum element: " << minElement << std::endl;

std::cout << "Maximum element: " << maxElement << std::endl;

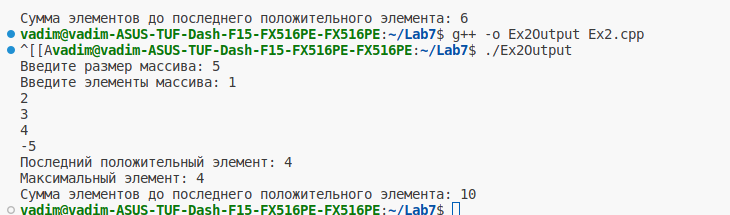
std::cout << "Sum of all elements: " << sum << std::endl;

std::cout << "Product of all elements: " << product << std::endl;

return 0;

}

**3 вариант:** В одномерном массиве, состоящем из n вещественных элементов, вычислить: максимальный элемент массива и сумму элементов массива, расположенных до последнего положительного элемента.



**Код к заданию:**

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

int n;

std::cout << "Введите размер массива: ";

std::cin >> n;

std::vector<double> arr(n);

std::cout << "Введите элементы массива: ";

for (int i = 0; i < n; i++) {

std::cin >> arr[i];

}

double maxElement = \*std::max\_element(arr.begin(), arr.end());

double sum = 0;

int lastPositiveIndex = -1;

for (int i = --n; i >= 0; i--) {

if (arr[i] > 0){

lastPositiveIndex = i;

break;

}

}

for (int i = 0; i <= lastPositiveIndex; i++){

sum += arr[i];

}

std::cout << "Последний положительный элемент: " << lastPositiveIndex + 1 << std::endl;

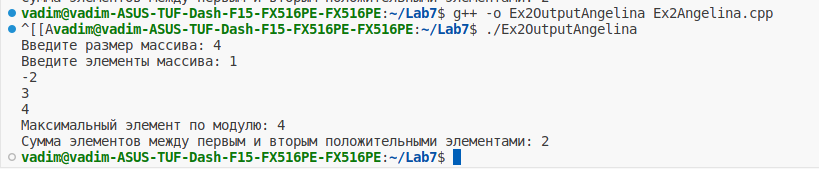
std::cout << "Максимальный элемент: " << maxElement << std::endl;

std::cout << "Сумма элементов до последнего положительного элемента: " << sum << std::endl;

return 0;

}

**5 вариант: В одномерном массиве, состоящем из n вещественных элементов, вычислить: максимальный по модулю элемент массива и сумму элементов массива, расположенных между первым и вторым положительными элементами.**



**Код к заданию:**

#include <iostream>

#include <vector>

#include <cmath>

#include <algorithm>

int main() {

int n;

std::cout << "Введите размер массива: ";

std::cin >> n;

std::vector<double> arr(n);

std::cout << "Введите элементы массива: ";

for (int i = 0; i < n; i++) {

std::cin >> arr[i];

}

double maxAbsoluteElement = std::abs(\*std::max\_element(arr.begin(), arr.end()));

double sum = 0;

int positiveCount = 0;

for (int i = 0; i < n; i++) {

if (arr[i] > 0) {

positiveCount++;

if (positiveCount == 2) {

sum += arr[i];

break;

}

}

if (positiveCount == 1) {

sum += arr[i];

}

}

std::cout << "Максимальный элемент по модулю: " << maxAbsoluteElement << std::endl;

std::cout << "Сумма элементов между первым и вторым положительными элементами: " << sum << std::endl;

return 0;

}

**Задание лабораторной №3.**

Разработать интерактивную программу «Try to Guess the Number» («Попробуй

угадать число»), которая эмулирует классическую игру на отгадывание числа.

Суть программы (игры) сводиться к следующему: компьютер генерирует слу-

чайное число из диапазона, к примеру, от 1 до 100, а пользователь (далее иг-

рок) пытается отгадать число за минимальное количество попыток. При каж-

дой очередной попытке компьютер «подсказывает» игроку, как соизмеряется

предложенный вариант игрока с действительным загаданным компьютером

числом: загаданное число больше или меньше указанного (higher/lower). Как

только игрок отгадывает число, компьютер должен «поздравить» его с выво-

дом на экран угаданного числа и количество затраченных игроком попыток.

Далее компьютер может «предложить» повторно сыграть игру или выйти из

программы.

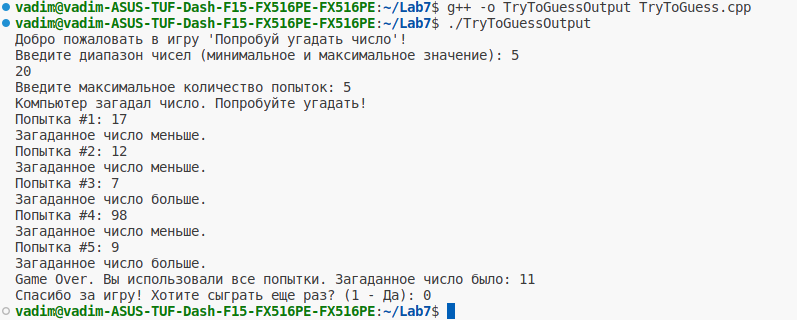
Для универсальности предложенной программы можно добавить возмож-

ность выбора диапазона генерирования компьютером случайных чисел, а

также задания ограничения на количество попыток. В случае, если игрок не

укладывается в заданное количество попыток (т.е. проигрывает), программа

должна выводить суровую надпись «Game Over».



**Код к заданию:**

#include <iostream>

#include <cstdlib>

#include <ctime>

int main() {

int minRange, maxRange, maxAttempts;

std::cout << "Добро пожаловать в игру 'Попробуй угадать число'!" << std::endl;

std::cout << "Введите диапазон чисел (минимальное и максимальное значение): ";

std::cin >> minRange >> maxRange;

std::cout << "Введите максимальное количество попыток: ";

std::cin >> maxAttempts;

srand(time(0));

int secretNumber = rand() % (maxRange - minRange + 1) + minRange;

int guess;

int attempts = 0;

std::cout << "Компьютер загадал число. Попробуйте угадать!" << std::endl;

while (attempts < maxAttempts) {

std::cout << "Попытка #" << attempts + 1 << ": ";

std::cin >> guess;

if (guess == secretNumber) {

std::cout << "Поздравляю! Вы угадали число " << secretNumber << " с " << attempts + 1 << " попытки." << std::endl;

break;

} else if (guess < secretNumber) {

std::cout << "Загаданное число больше." << std::endl;

} else {

std::cout << "Загаданное число меньше." << std::endl;

}

attempts++;

}

if (attempts == maxAttempts) {

std::cout << "Game Over. Вы использовали все попытки. Загаданное число было: " << secretNumber << std::endl;

}

std::cout << "Спасибо за игру! Хотите сыграть еще раз? (1 - Да): ";

int playAgain;

std::cin >> playAgain;

if (playAgain == 1) {

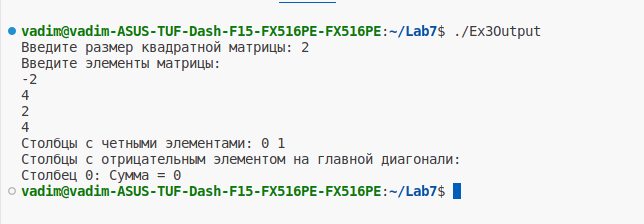
main();

}

return 0;

}

**3 вариант:** Выведите номера столбцов, все элементы, которых четны. Для каждого столбца с отрицательным элементом на главной диагонали вывести его номер и сумму всех элементов этого столбца.

****

**Скрипт к заданию:**

#include <iostream>

#include <vector>

int main() {

int n;

std::cout << "Введите размер квадратной матрицы: ";

std::cin >> n;

std::vector<std::vector<int>> matrix(n, std::vector<int>(n));

std::cout << "Введите элементы матрицы: " << std::endl;

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

std::cin >> matrix[i][j];

}

}

std::cout << "Столбцы с четными элементами: ";

for (int j = 0; j < n; j++) {

bool allEven = true;

for (int i = 0; i < n; i++) {

if (matrix[i][j] % 2 != 0) {

allEven = false;

break;

}

}

if (allEven) {

std::cout << j << " ";

}

}

std::cout << std::endl;

std::cout << "Столбцы с отрицательным элементом на главной диагонали: " << std::endl;

for (int j = 0; j < n; j++) {

if (matrix[j][j] < 0) {

int sum = 0;

for (int i = 0; i < n; i++) {

sum += matrix[i][j];

}

std::cout << "Столбец " << j << ": Сумма = " << sum << std::endl;

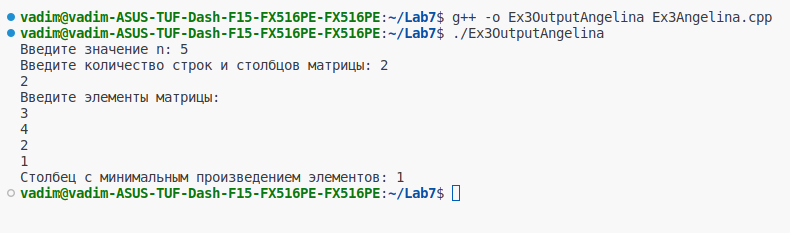
}

}

return 0;

}

**5 вариант:** Среди столбцов заданной матрицы, содержащих только такие элементы, которые по модулю не больше n, найти столбец с минимальным произведением элементов.

****

**Скрипт к заданию:**

#include <iostream>

#include <vector>

#include <limits>

int main() {

int n;

std::cout << "Введите значение n: ";

std::cin >> n;

int rows, columns;

std::cout << "Введите количество строк и столбцов матрицы: ";

std::cin >> rows >> columns;

std::vector<std::vector<int>> matrix(rows, std::vector<int>(columns));

std::cout << "Введите элементы матрицы: " << std::endl;

for (int i = 0; i < rows; i++) {

for (int j = 0; j < columns; j++) {

std::cin >> matrix[i][j];

}

}

int minProductColumn = -1;

int minProduct = std::numeric\_limits<int>::max();

for (int j = 0; j < columns; j++) {

int product = 1;

bool validColumn = true;

for (int i = 0; i < rows; i++) {

if (std::abs(matrix[i][j]) > n) {

validColumn = false;

break;

}

product \*= matrix[i][j];

}

if (validColumn && product < minProduct) {

minProduct = product;

minProductColumn = j;

}

}

if (minProductColumn != -1) {

std::cout << "Столбец с минимальным произведением элементов: " << minProductColumn << std::endl;

} else {

std::cout << "Не найден допустимый столбец." << std::endl;

}

return 0;

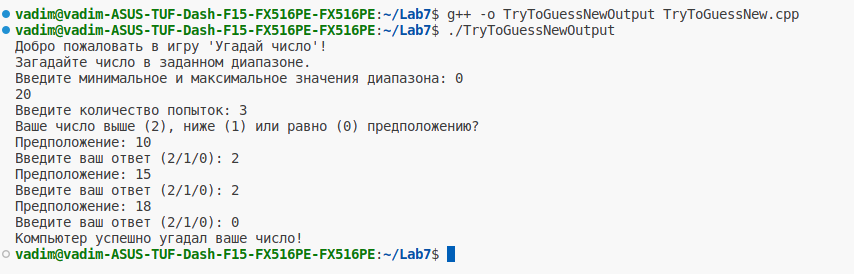
}

**Задание лабораторной №4.**

Необходимо переработать основное задание из предыдущей лабораторной

работы таким образом, чтобы пользователь (игрок) загадывал число, а компь-

ютер, используя оптимальный и эффективный алгоритм, его отгадывал.



**Скрипт к заданию:**

#include <iostream>

int main() {

int minRange, maxRange;

std::cout << "Добро пожаловать в игру 'Угадай число'!" << std::endl;

std::cout << "Загадайте число в заданном диапазоне." << std::endl;

std::cout << "Введите минимальное и максимальное значения диапазона: ";

std::cin >> minRange >> maxRange;

int guess;

int response;

int attempts = 0;

std::cout << "Введите количество попыток: ";

int maxAttempts;

std::cin >> maxAttempts;

std::cout << "Ваше число выше (2), ниже (1) или равно (0) предположению?" << std::endl;

while (attempts < maxAttempts) {

guess = (minRange + maxRange) / 2;

std::cout << "Предположение: " << guess << std::endl;

std::cout << "Введите ваш ответ (2/1/0): ";

std::cin >> response;

if (response == 2) {

minRange = guess + 1;

} else if (response == 1) {

maxRange = guess - 1;

} else if (response == 0) {

std::cout << "Компьютер успешно угадал ваше число!" << std::endl;

break;

} else {

std::cout << "Некорректный ответ. Пожалуйста, введите 2, 1 или 0." << std::endl;

}

attempts++;

}

if (attempts == maxAttempts) {

std::cout << "Компьютер не смог угадать ваше число за заданное количество попыток." << std::endl;

}

return 0;

}

**Задания:**

**Задание 1**

Используя материалы (исходные файлы) упражнения №1 проведите компиляцию однофайлового проекта с прохождением всех стадий комппиляции.

Для ускорения работы примените скрипты bash.

Исходные файлы программ обязательно должны содержать коментарии.

Сборка проекта должна содержать файлы с результатами препроцессинга. Исследуйте фалы препроцессора, найдите в них код своей программы.

Определите размеры исходных, препроцессорных, ассемблерных, объектных

и исполняемых файлов. С помощью соответствующий консольных команд

определите форматы этих файлов. Результаты подтвердите скриншотами.

**Задание 2**

Используя материалы (исходные файлы) упражнения №1 и №2 создайте один

многофайловый проект, руководствуясь принципом Single Responsibility

Principle. Каждое задание должно быть представлено отдельным исходным

файлом. Для связывания файлов проекта обязательно используйте заголовочные файлы. Все файлы могут располагаться в одной директории.

Все задания должны вызываться консольным меню. Для выхода из приложения предусмотрите отдельный параметр.

Проведите автосборку проект с прохождением всех стадий компиляции с использованием утилиты make.

**Задание 3**

Используя материалы (исходные файлы) задания №2 создайте один многофайловый проект, руководствуясь принципом Single Responsibility Principle.

Каждое задание должно быть представлено отдельным исходным файлом. Используйте архитектурный шаблон проектирования MVC. Здесь под шаблоном

MVC понимается файлы или группа файлов в трех отдельных директориях

(папках), а не просто три отдельных файла.

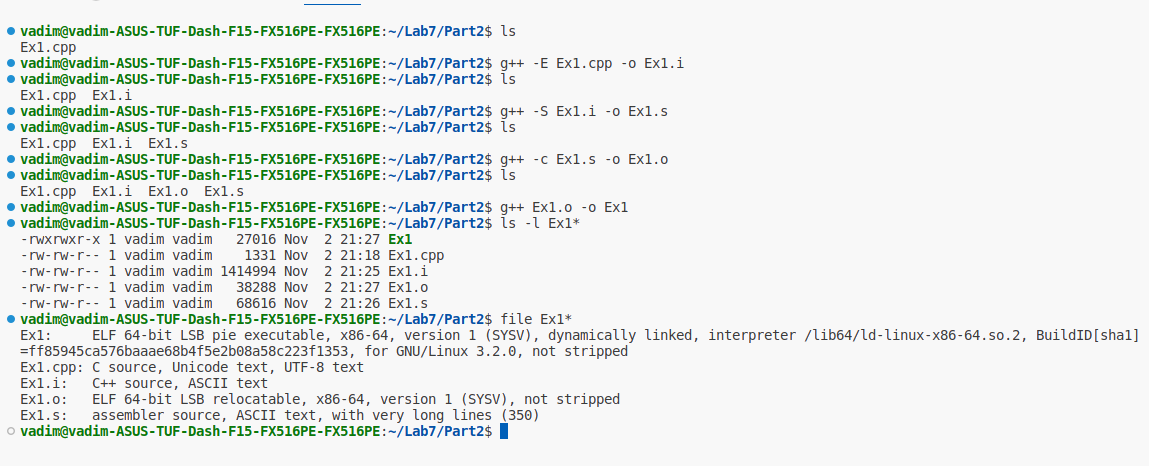
Все задания должны вызываться консольным меню. Для выхода из приложения предусмотрите отдельный параметр. Для связывания файлов проекта обязательно используйте заголовочные файлы.

Проведите рекурсивную автосборку проект с прохождением всех стадий компиляции с использованием утилиты make

**Выполнение заданий:**

**Задание 1.**

Компиляция однофайлового проекта по стадиям:

**Текстовая версия:**

vadim@vadim-ASUS-TUF-Dash-F15-FX516PE-FX516PE:~/Lab7/Part2$ ls

Ex1.cpp

vadim@vadim-ASUS-TUF-Dash-F15-FX516PE-FX516PE:~/Lab7/Part2$ g++ -E Ex1.cpp -o Ex1.i

vadim@vadim-ASUS-TUF-Dash-F15-FX516PE-FX516PE:~/Lab7/Part2$ ls

Ex1.cpp Ex1.i

vadim@vadim-ASUS-TUF-Dash-F15-FX516PE-FX516PE:~/Lab7/Part2$ g++ -S Ex1.i -o Ex1.s

vadim@vadim-ASUS-TUF-Dash-F15-FX516PE-FX516PE:~/Lab7/Part2$ ls

Ex1.cpp Ex1.i Ex1.s

vadim@vadim-ASUS-TUF-Dash-F15-FX516PE-FX516PE:~/Lab7/Part2$ g++ -c Ex1.s -o Ex1.o

vadim@vadim-ASUS-TUF-Dash-F15-FX516PE-FX516PE:~/Lab7/Part2$ ls

Ex1.cpp Ex1.i Ex1.o Ex1.s

vadim@vadim-ASUS-TUF-Dash-F15-FX516PE-FX516PE:~/Lab7/Part2$ g++ Ex1.o -o Ex1

vadim@vadim-ASUS-TUF-Dash-F15-FX516PE-FX516PE:~/Lab7/Part2$ ls -l Ex1\*

-rwxrwxr-x 1 vadim vadim 27016 Nov 2 21:27 Ex1

-rw-rw-r-- 1 vadim vadim 1331 Nov 2 21:18 Ex1.cpp

-rw-rw-r-- 1 vadim vadim 1414994 Nov 2 21:25 Ex1.i

-rw-rw-r-- 1 vadim vadim 38288 Nov 2 21:27 Ex1.o

-rw-rw-r-- 1 vadim vadim 68616 Nov 2 21:26 Ex1.s

vadim@vadim-ASUS-TUF-Dash-F15-FX516PE-FX516PE:~/Lab7/Part2$ file Ex1\*

Ex1: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=ff85945ca576baaae68b4f5e2b08a58c223f1353, for GNU/Linux 3.2.0, not stripped

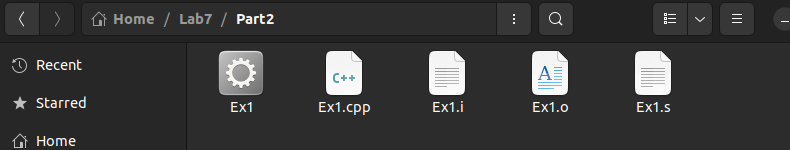
Ex1.cpp: C source, Unicode text, UTF-8 text

Ex1.i: C++ source, ASCII text

Ex1.o: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), not stripped

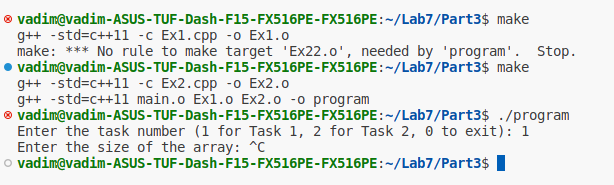
Ex1.s: assembler source, ASCII text, with very long lines (350)

**Демонстрация того, что файлы были созданы:**



**Задание 2.**

Создание многофайлового проекта.



Содержание Makefile’а:

CXX = g++

CXXFLAGS = -std=c++11

SRCS = main.cpp Ex1.cpp Ex2.cpp

OBJS = $(SRCS:.cpp=.o)

EXEC = program

.PHONY: all clean

all: $(EXEC)

$(EXEC): $(OBJS)

$(CXX) $(CXXFLAGS) $^ -o $@

%.o: %.cpp

$(CXX) $(CXXFLAGS) -c $< -o $@

clean:

rm -f $(OBJS) $(EXEC)

**Задание 3.**

Содержание основного Makefile’а:

all: main program

.PHONY: all clean

program:

echo "./main" > program

chmod +x program

main: controller/main.o model/Ex1.o model/Ex2.o

g++ $^ -o $@

controller/main.o: controller/main.cpp view/Ex1.h view/Ex2.h

g++ -c $< -o $@

model/Ex1.o: model/Ex1.cpp view/Ex1.h

g++ -c $< -o $@

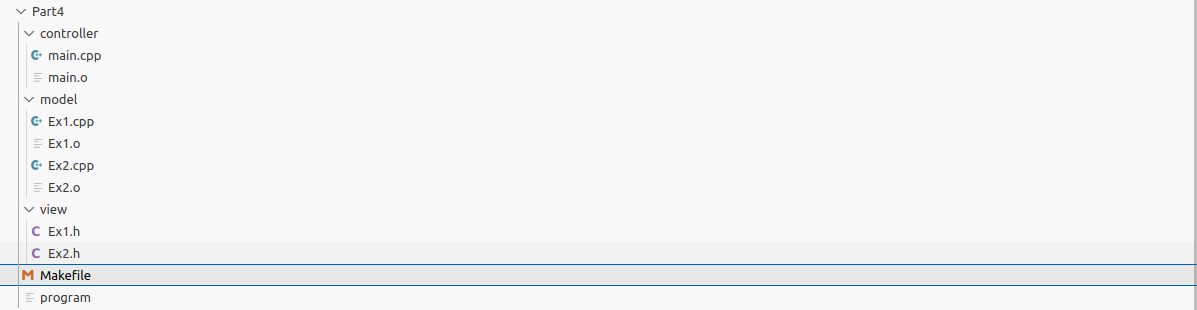
model/Ex2.o: model/Ex2.cpp view/Ex2.h

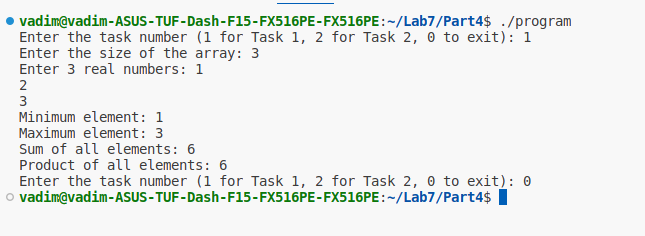
g++ -c $< -o $@

clean:

rm -f controller/\*.o model/\*.o view/\*.o main

Созданная для задания директория:

Результат работы скомпилированного файла для исполнения программы:



**Контрольные вопросы:**

**1) Что такое компоновка (линковка, сборка)?**

Компоновка (линковка, сборка) - это процесс объединения нескольких объектных файлов (object files) вместе для создания исполняемого файла или библиотеки.

**2) Что такое рефакторинг кода?**

Рефакторинг кода - это процесс изменения структуры и внутреннего устройства программного кода с целью улучшить его качество, читаемость, поддерживаемость и эффективность, не меняя при этом его внешнего поведения.

**3) Опишите базовый синтаксис make-файла и общие правила определения целевых связок в make-файле: целей (targets), зависимостей (dependencies) и инструкций (instructions)? Какая цель будет обрабатываться самой первой при запуске утилиты make? Что такое «основная цель сборки» и что в неё входит?**

Make-файл - это текстовый файл, который содержит инструкции для утилиты make, которая автоматизирует процесс компиляции и сборки программного проекта. В make-файле определяются цели (targets), зависимости (dependencies) и инструкции (instructions).

Цель (target) - это файл или действие, которое нужно выполнить. Он указывает утилите make, что нужно создать или обновить определенный файл. Цель может быть обычным файлом, объектным файлом, исполняемым файлом или действием, которое нужно выполнить.

Зависимость (dependency) - это файл или другая цель, от которой зависит выполнение текущей цели. Зависимости указываются после двоеточия и могут быть как файлами, так и другими целями. Если зависимость изменяется, то цель будет пересобрана.

Инструкция (instruction) - это команда, которую утилита make выполняет для создания или обновления цели. Инструкции могут быть любыми командами, которые можно выполнить в командной строке.

При запуске утилиты make без указания цели, будет обрабатываться первая цель, которая указана в make-файле. Эта цель называется "основной целью сборки" (default target). Она обычно является целью, от которой зависят все остальные цели в проекте. Основная цель сборки определяется в make-файле с помощью специального синтаксиса, например:

make

all: target1 target2 target3

В этом примере, "all" - это основная цель сборки, а "target1", "target2" и "target3" - ее зависимости. При запуске утилиты make без указания цели, будет выполнена основная цель сборки, и утилита make будет выполнять все зависимости, указанные для этой цели.

Основная цель сборки может включать все необходимые цели для полной сборки проекта, такие как компиляция и линковка исходных файлов, создание исполняемого файла и т.д.

Важно отметить, что синтаксис make-файла может различаться в зависимости от используемой версии утилиты make и операционной системы.

**5) Назовите основные стадии компиляции программы?**

Процесс компиляции состоит из следующих этапов:

1. Лексический анализ.

2. Синтаксический анализ.

3. Семантический анализ.

4. Оптимизация.

5. Ассемблирование.

6. Генерация кода.

7. Связывание.

**6) В чем различие компиляции и интерпритации программы?**

Интерпретатор берет одну инструкцию, транслирует и выполняет ее, а затем берет следующую инструкцию. Компилятор же транслирует всю программу сразу, а потом выполняет ее

**7) Приведите известные вам примеры компилируемых и интерпритируемых языков программирования**

Примерами компилируемых языков являются Pascal, C, C++

Примерами интерпретируемых языков являются PHP, Python, JavaScript

**8) Что такое препроцессор и зачем он нужен?**

Препроцессор — это компьютерная программа, принимающая данные на входе и выдающая данные, предназначенные для входа другой программы

**9) В чем разница между понятиями «аргумент функции» и «параметр функции» ?**

Параметром (формальным параметром) функции называется переменная в функции, которая будет содержать передаваемое снаружи входное значение. Аргументом называют фактическую переменную или выражение, значение которого используется как входное значение при вызове функции

**10) В чем разница использования двух форм директивы: #include <filename> или #include “filename”?**

Для #include <filename> препроцессор выполняет поиск в зависимости от реализации. Для #include "filename" препроцессор сначала выполняет поиск в том же каталоге, что и файл, содержащий директиву, а затем следует по пути поиска

**11) Что такое автосборка? Какие приемущества она дает?**

Сборкой называется процесс подготовки программы к непосредственному использованию. Собирать программы вручную неудобно, поэтому программисты, как правило, прибегают к различным приемам, позволяющим автоматизировать этот процесс. Самый простой способ – написать сценарий оболочки (shell-скрипт), который будет автоматически выполнять все то, что вы обычно вводите вручную

**12) Какой командой можно определить определить формат файла?**

**13) Что такое утилита make?**

Утилита make — наиболее популярное и проверенное временем средство автоматической сборки программ в Linux

**14) В чем разница автосборки скриптами bash и утилитой make?**

make — утилита предназначенная для автоматизации преобразования файлов из одной формы в другую. Bash позволяет создавать сценарии для автоматизации различных операций

**Вывод:** в данной лабораторной работе были изучены встроенный инструментарий для разработки приложений под семейство ОС Linux и этапы такие, как проектирование, компиляция, отладка и запуск.