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: ”Компиляция и отладка простейшего приложения в Linux ***”***

Выполнил**:** студент группы 10701321 Нестерков Д.А.

Принял**:** пр.Давыденко Н.

Минск 2023

# **Лабораторная работа № 7**

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++.

**Задания**

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

**Индивидуальное задание**

В одномерном массиве, состоящем из к целых элементов, вычислить: количество положительных элементов массива и сумму элементов массива, расположенных после последнего элемента, равного нулю.

**Выполнение лабораторной работы**

**Код программы**

#include <iostream>

#include <cmath>

int task1(){

// Vvod razmera massiva

int n;

std::cout << "Vvedite razmer massiva: ";

std::cin >> n;

// Dinamicheskoe vydelenie pamyati

double\* arr = new double[n];

// Vvod elementov massiva

std::cout << "Vvedite elementy massiva:\n";

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

std::cin >> arr[i];

}

// Poisk min i max

double min = arr[0];

double max = arr[0];

int min\_i = 0;

int max\_i = 0;

for(int i = 1; i < n; ++i) {

if(arr[i] < min) {

min = arr[i];

min\_i = i;

}

if(arr[i] > max) {

max = arr[i];

max\_i = i;

}

}

// Obmen min i max

double temp = arr[min\_i];

arr[min\_i] = arr[max\_i];

arr[max\_i] = temp;

// Vyvod rezul'tata

std::cout << "\nMassiv s zamenennymi min i max:\n";

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

std::cout << arr[i] << " ";

}

// Vychislenie summy i proizvedeniya

double sum = 0;

double mult = 1;

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

sum += arr[i];

mult \*= arr[i];

}

std::cout << "\nSumma elementov: " << sum;

std::cout << "\nProizvedenie elementov: " << mult;

// Osvobozhdenie pamyati

delete[] arr;

return 0;

}

int task2(){

int k;

std::cout << "Enter number of elements: ";

std::cin >> k;

int arr[k];

int positive\_count = 0;

int sum\_after\_zero = 0;

bool zero\_found = false;

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

std::cin >> arr[i];

if(arr[i] > 0) {

positive\_count++;

}

if(arr[i] == 0) {

zero\_found = true;

}

if(zero\_found && arr[i] != 0) {

sum\_after\_zero += arr[i];

}

}

std::cout << "Positive elements count: " << positive\_count << "\n";

std::cout << "Sum after last zero: " << sum\_after\_zero << "\n";

return 0;

}

int main() {

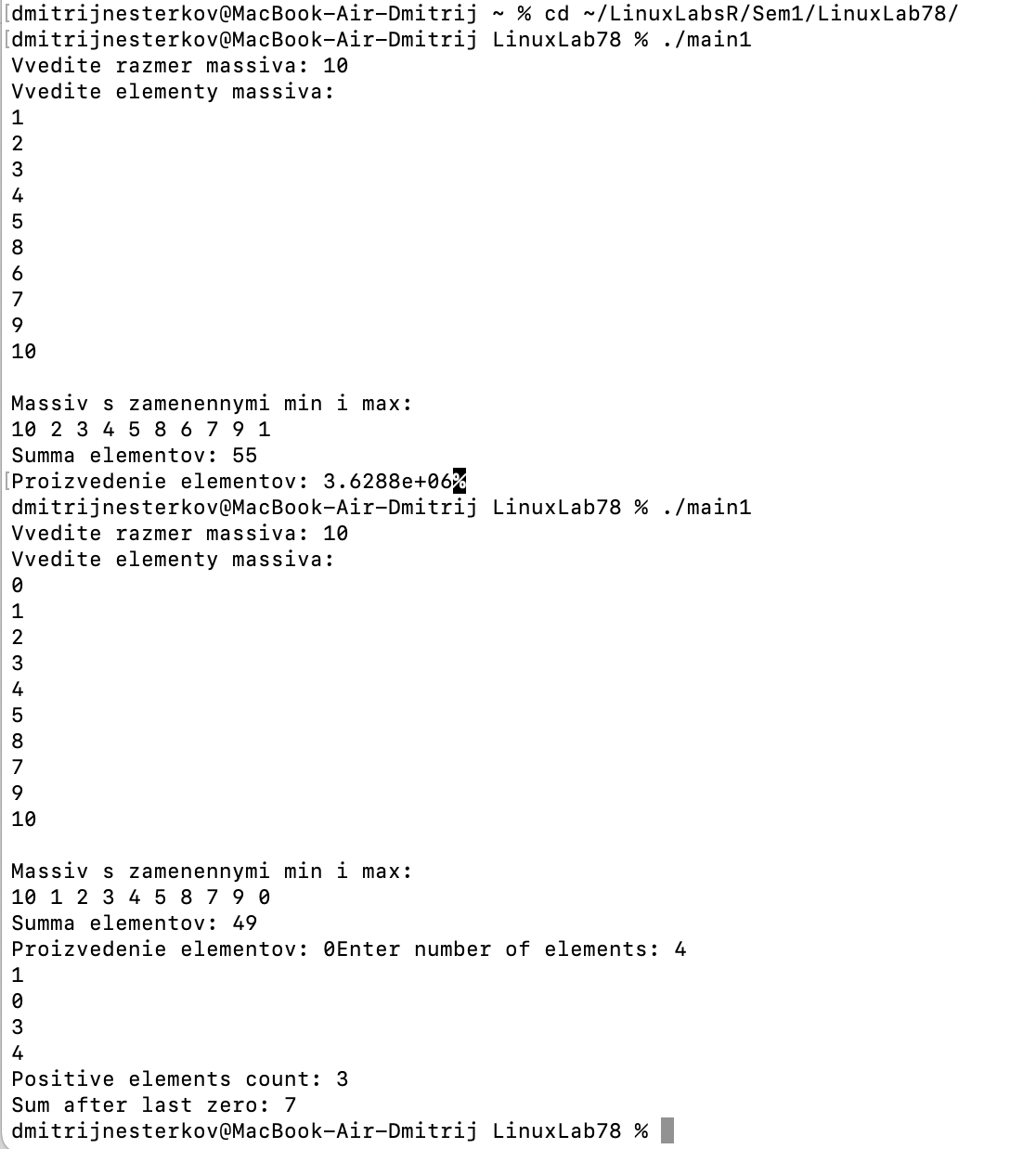
task1();

task2();

return 0;

}

Снимок экрана 2023-10-18 в 08.23.44

****

**Рисунок 1 результаты выполнения**

**Главный файл**

#!/bin/bash

set -x

# Компилируем исходный файл

g++ -o my\_program main1.cpp

# Выполняем препроцессинг и сохраняем результат

g++ -E main1.cpp -o main\_preprocessed.cpp

# Создаем ассемблерный файл

g++ -S main1.cpp

# Создаем объектный файл

g++ -c main1.cpp

set +x

# Выводим размеры файлов

ls -lh main1.cpp main\_preprocessed.cpp main1.s main1.o my\_program

# Определяем форматы файлов

file main1.cpp main\_preprocessed.cpp main1.s main1.o my\_program

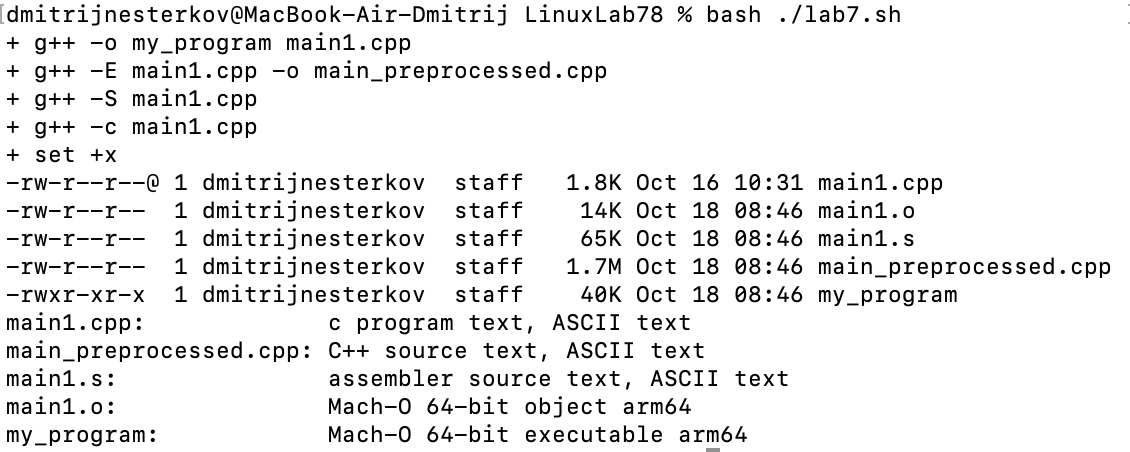


Рисунок 1.1

**Контрольные вопросы**

Что такое GCC? Опишите наиболее востребования опции компилятора gcc/g++. Чем отличается gcc и g++?

**GCC (GNU Compiler Collection) - это набор компиляторов, включающий в себя:**

**- gcc - компилятор языка Си**

**- g++ - компилятор языка C++**

**- и другие компиляторы для Fortran, Java, Ada и др. языков**

**Отличия gcc и g++:**

**- gcc используется для компиляции кода на Си, g++ - для C++**

**Задание**

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 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 «Game Over».

**Индивидуальное задание**

Найти номер строки заданной матрицы, в которой находится самая длинная серия (последовательность одинаковых элементов).

Код программы

#include <iostream>

#include <cstdlib>

#include <ctime>

void task2(){

int rows, cols;

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

std::cin >> rows;

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

std::cin >> cols;

int mat[rows][cols];

// Ввод матрицы

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

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

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

}

}

// Поиск строки с максимальной серией

int max\_row = 0;

int max\_streak = 0;

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

int curr\_streak = 1;

for(int j = 1; j < cols; j++) {

if(mat[i][j] == mat[i][j-1]) {

curr\_streak++;

} else {

curr\_streak = 1;

}

if(curr\_streak > max\_streak) {

max\_streak = curr\_streak;

max\_row = i;

}

}

}

std::cout << "Номер строки с максимальной серией: " << max\_row << "\n";

}

int main() {

srand(time(0)); // инициализация генератора случайных чисел

int lower\_bound = 1;

int upper\_bound = 100;

std::cout << "Введите нижнюю границу диапазона: ";

std::cin >> lower\_bound;

std::cout << "Введите верхнюю границу диапазона: ";

std::cin >> upper\_bound;

int max\_attempts = 10;

std::cout << "Введите максимальное кол-во попыток: ";

std::cin >> max\_attempts;

int number = rand() % (upper\_bound - lower\_bound + 1) + lower\_bound;

int attempts = 0;

while(attempts < max\_attempts) {

int guess;

std::cout << "Ваша догадка: ";

std::cin >> guess;

attempts++;

if(guess == number) {

std::cout << "Поздравляю, вы угадали число за " << attempts << " попыток!";

break;

}

else if(guess < number) {

std::cout << "Загаданное число больше!\n";

}

else {

std::cout << "Загаданное число меньше!\n";

}

}

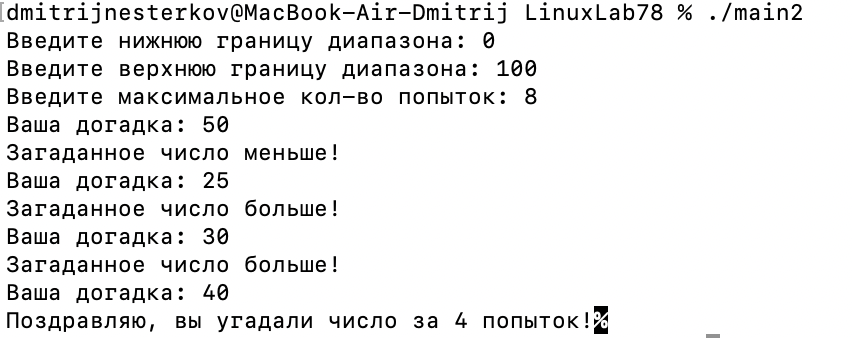
if(attempts == max\_attempts) {

std::cout << "Игра окончена. Вы проиграли.";

}

return 0;

}



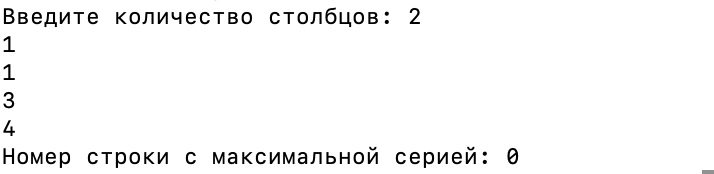


Рисунок 1-2 результаты

Главный файл

CC=g++

CFLAGS=-c -Wall

LDFLAGS=

SOURCES=main.cpp task1.cpp task2.cpp

OBJECTS=$(SOURCES:.cpp=.o)

EXECUTABLE=app3

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)

$(CC) $(LDFLAGS) $(OBJECTS) -o $@

.cpp.o:

$(CC) $(CFLAGS) $< -o $@

clean:

rm -f $(OBJECTS) $(EXECUTABLE)

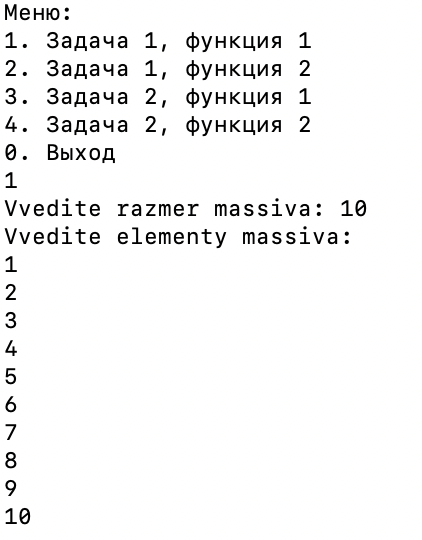


Рисунок 1.2

**Контрольные вопросы**

Объясните принцип "Единственной ответственности" (Single Responsibility Principle, SRP), который используется бля проектирования и реализации независимого или слабосвязанного кода.

Принцип единственной ответственности (Single Responsibility Principle) - это один из основных принципов проектирования кода, согласно которому каждый модуль, класс или функция должны нести ответственность только за одну вещь и решать только одну задачу.

Это позволяет сделать код:

- Понятным

- Гибким

- Поддерживаемым

**Задание**

Необходимо переработать основное задание из предыдущей лабораторной работы таким образом, чтобы пользователь (игрок) загадывал число, а компьютер, используя оптимальный и эффективный алгоритм, его отгадывал.

**Индивидуальное задание**

Для программ, которые были разработаны в двух предыдущих лабораторных работах №2 и №3 необходимо добавить автоматизацию сборки многофайлового проекта с использованием сценарных оболочек и автосборщиков.

Код программы:

#include <iostream>

int main() {

int number;

std::cout << "Загадайте число от 1 до 100: ";

std::cin >> number;

int lower\_bound = 1;

int upper\_bound = 100;

while (lower\_bound <= upper\_bound) {

int guess = (lower\_bound + upper\_bound) / 2;

if (guess == number) {

std::cout << "Я угадал ваше число! Это " << guess << std::endl;

break;

}

else if (guess < number) {

std::cout << "(<)Мое число:" << guess << std::endl;

lower\_bound = guess + 1;

}

else {

std::cout << "(>)Мое число: " << guess << std::endl;

upper\_bound = guess - 1;

}

}

return 0;

}

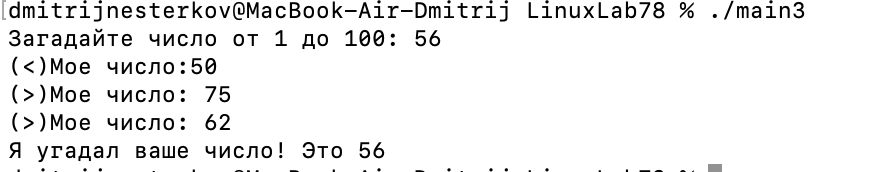


Рисунок 1 результат

# Определение цели по умолчанию

all: app

# Сборка приложения

app: main2.o

g++ main2.o -o app

# Компиляция main2.cpp

main2.o: main2.cpp

g++ -c main2.cpp

# Цель для очистки временных файлов

clean:

rm -f \*.o app

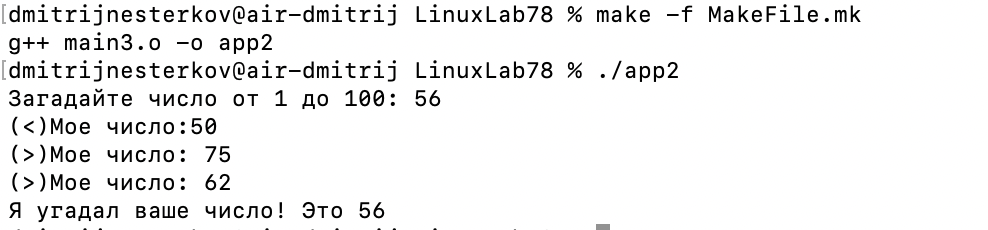


Рисунок 2 makefile

Главный файл

Контрольные вопросы

Makefile

# Compiler

CXX = g++

# Compiler flags

CXXFLAGS = -std=c++11 -Wall

# Directories

MODEL\_DIR = Model

VIEW\_DIR = View

CONTROLLER\_DIR = Controller

# Source files

MODEL\_SOURCES = $(wildcard $(MODEL\_DIR)/\*.cpp)

VIEW\_SOURCES = $(wildcard $(VIEW\_DIR)/\*.cpp)

CONTROLLER\_SOURCES = $(wildcard $(CONTROLLER\_DIR)/\*.cpp)

MAIN\_SOURCE = main.cpp

# Object files

MODEL\_OBJECTS = $(MODEL\_SOURCES:.cpp=.o)

VIEW\_OBJECTS = $(VIEW\_SOURCES:.cpp=.o)

CONTROLLER\_OBJECTS = $(CONTROLLER\_SOURCES:.cpp=.o)

MAIN\_OBJECT = $(MAIN\_SOURCE:.cpp=.o)

# Executable name

EXECUTABLE = app

all: $(EXECUTABLE)

$(EXECUTABLE): $(MODEL\_OBJECTS) $(VIEW\_OBJECTS) $(CONTROLLER\_OBJECTS) $(MAIN\_OBJECT)

  $(CXX) $(CXXFLAGS) $(MODEL\_OBJECTS) $(VIEW\_OBJECTS) $(CONTROLLER\_OBJECTS) $(MAIN\_OBJECT) -o $(EXECUTABLE)

# Rule for creating object files

$(MODEL\_DIR)/%.o: $(MODEL\_DIR)/%.cpp

  $(CXX) $(CXXFLAGS) -c $< -o $@

$(VIEW\_DIR)/%.o: $(VIEW\_DIR)/%.cpp

  $(CXX) $(CXXFLAGS) -c $< -o $@

$(CONTROLLER\_DIR)/%.o: $(CONTROLLER\_DIR)/%.cpp

  $(CXX) $(CXXFLAGS) -c $< -o $@

%.o: %.cpp

  $(CXX) $(CXXFLAGS) -c $< -o $@

clean:

  rm -f $(MODEL\_DIR)/\*.o $(VIEW\_DIR)/\*.o $(CONTROLLER\_DIR)/\*.o \*.o $(EXECUTABLE)

// main.cpp

#include "Model/Person.h"

#include "View/PersonView.h"

#include "Controller/PersonController.h"

int main() {

    // Create model, view, and controller

    Person person;

    PersonView personView;

    PersonController personController(&person, &personView);

    // Set person details

    personController.setPersonDetails("John Doe", 30);

    // Update and print details

    personController.updateView();

    // Update person details and print again

    personController.setPersonDetails("Jane Smith", 35);

    personController.updateView();

    return 0;

}

// Controller/PersonController.cpp

#include "PersonController.h"

PersonController::PersonController(Person\* model, PersonView\* view) : m\_model(model), m\_view(view) {}

void PersonController::setPersonDetails(const std::string& name, int age) {

    m\_model->setName(name);

    m\_model->setAge(age);

}

void PersonController::updateView() const {

    m\_view->printPersonDetails(m\_model->getName(), m\_model->getAge());

}

// Controller/PersonController.h

#ifndef PERSONCONTROLLER\_H

#define PERSONCONTROLLER\_H

#include "Person.h"

#include "PersonView.h"

class PersonController {

public:

    PersonController(Person\* model, PersonView\* view);

    void setPersonDetails(const std::string& name, int age);

    void updateView() const;

private:

    Person\* m\_model;

    PersonView\* m\_view;

};

#endif

// View/PersonView.cpp

#include "../Model/Person.h"

void PersonView::printPersonDetails(const std::string& name, int age) const {

    std::cout << "Person Details:\n";

    std::cout << "Name: " << name << "\n";

    std::cout << "Age: " << age << "\n";

}

// View/PersonView.h

#ifndef PERSONVIEW\_H

#define PERSONVIEW\_H

#include <iostream>

#include "Person.h"

class PersonView {

public:

    void printPersonDetails(const std::string& name, int age) const;

};

#endif

// Model/Person.cpp

#include "Person.h"

void Person::setName(const std::string& name) {

    m\_name = name;

}

void Person::setAge(int age) {

    m\_age = age;

}

std::string Person::getName() const {

    return m\_name;

}

int Person::getAge() const {

    return m\_age;

}

// Model/Person.h

#ifndef PERSON\_H

#define PERSON\_H

#include <string>

class Person {

public:

    void setName(const std::string& name);

    void setAge(int age);

    std::string getName() const;

    int getAge() const;

private:

    std::string m\_name;

    int m\_age;

};

#endif

- ProjectDirectory

    - Makefile

    - main.cpp

    - Model

        - Person.h

        - Person.cpp

    - View

        - PersonView.h

        - PersonView.cpp

    - Controller

        - PersonController.h

        - PersonController.cpp

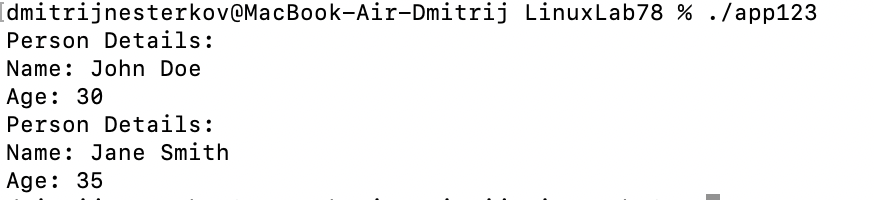


Рисунок 1.3

Параметризация процесса сборки и переменные make-файла?

В Make-файлах вы можете определять переменные, которые позволяют параметризовать процесс сборки. Это полезно, например, для задания параметров компиляции, путей к файлам и других параметров.