Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Факультет комп’ютерних наук

# ЛАБОРАТОРНА РОБОТА № 1

з дисципліни «Операційні системи»

Тема: «Основи мови програмування С. Компілятор GCC»

Виконав:

студент 3 курсу

групи КС-32

Безрук Юрій Русланович

Перевірив: Споров О.Є.

Харків – 2020

Целью данной работы является ознакомление с работой компилятора GCC и сборочной системы make и написание простых программ на языке С для проверки работы основных их функций и разных способов подключения многофайловой структуры проектов (прямая сборка, динамические и статические библиотеки).

# ХОД РАБОТЫ

**Задание 0.**

В задании 0 необходимо было создать модель системы точек в пространстве, имеющих массу и координаты. Согласно условию задачи, с некоторого момента времени точка с наименьшей массой исчезает, передавая свой вес ближайшей. Этот процесс должен продолжаться до тех пор, пока точка не останется одна.

Для решения данной задачи было решено создать модель «мир», вмещающую в себя массив структур «точка». Проект разбит на 3 файла. В заголовочном файле world.h располагаются подключения основных библиотек, прототипы функций и обьявления структур «точка» (состоит из трех координат и веса) и «мир» (содержащей массив точек, поле с их количеством, а также служебные параметры пространства, а именно максимальные и минимальные значение генерируемого веса и координат для данного «мира»). Так же для удобства в этом файле были объявлены служебные константы TRUE и FALSE.

В файле world.c содержится программная реализация системы: определены все функции, которые не взаимодействуют с пользователем, а именно: функция генерации массива точек в заданном простанстве, функция нахождения минимальной точки, функция, описывающая исчезновение и передачу массы ближайшей точке (расстояние вычисляется по теореме пифагора на осонове координат), печать информации о всех точках, а так же функция создания самого «мира» по заданным параметрам пространства.

В файле main.c находятся реализации функций, взаимодействующих с пользователем: функции, инициализирующей параметры выбранного пространства и непосредственно функции main.

Для удобства проведений экспериментов программа сразу даёт возможность создать несколько «миров» с разными параметрами. После указания количества миров, пользователь поочередно инициализирует их стартовыми значениями, после чего можно «запускать» любой из миров по его номеру. При выборе номера мира начинается процедура исчезновения точек, пока не останется последняя. После этого мир «умирает» и пользователь может приступить к обработке других заданных пространств.

Для реализации массивов миров и точек было применено динамическое управление памятью через функции malloc() (при создании), realloc() (при обновлении) и free() (при освобождении).

В скрипте сборки makefile указано выполнение сначала компиляции системы в объектные файлы main.o и world.o, а после сборка в исполняемый файл task0, очистка от объектных файлов и запуск.

**Задание 4**

Согласно условию задачи, необходимо реализовать на языке С подобие структуры данных ArrayList из Java, позволяющее хранить в неограниченном объёме элементы различных типов.

Т.к. язык С не является объектно-ориентированным, вместо создания класса с полями и методами пришлось использовать структуру list с полями массива указателей на данные типа void\* и целочисленным полем размера. Проект тоже является многофайловым: в подпапке include находится заголовочный файл list.h, содержащий объявление структуры list и методов, обслуживающих его. Файл list.c содержит реализацию функций работы со списком. Этот файл запакован в библиотеку liblist, хранящуюся в папке lib. В файле main.c содержится точка входа в программу для проверки работы реализованной структуры.

Для того, чтобы реализовать возможность хранения значений любого типа, в структуре обьявлен массив универсальных указателей void\*. Таким образом, в списке хранятся не сами значения, а только их адреса, что соответствует Java ArrayList. Сам список не объявлялся глобально, поэтому в большинство функций передается он сам либо указатель на него. Коротко о реализованных функциях:

create\_list(void\* item) – создает и возвращает структуру типа list с переданным элементом item.

add(struct list\* ls, void\* item) – добавляет новый элемент в конец списка, довыделяя память и увеличивая размер на 1.

insert(struct list\* ls, int index, void\* item) – добавляет новый элемент в указанное место списка, довыделяя память, производя сдвиг элементов и увеличивая размер на 1, при условии, что индекс валидный.

size(struct list ls) – возвращает текущий размер списка.

delete(struct list\* ls, int index) – аналог метода remove ArrayList. Удаляет элемент по заданному индексу из списка, производя обратный сдвиг и урезание памяти с уменьшением размера на 1 при условии, что индекс валидный.

set(struct list\* ls, int index, void\* item) – устанавливает новый элемент на указанную позицию, перезатирая старое значение при условии, что индекс валидный.

get(struct list\* ls, int index) – возвращает элемент из списка по заданной позиции при условии, что индекс валидный.

Таким образом, данные внутри списка хранятся в виде адресов и при их получении их нужно конвентировать в желаемый формат. Согласно условию задания, проект нужно было реализовать при помощи подключения статической и динамической библиотек (т.е. двумя способами). К отчету прикрепляются соответственно, две папки с проектами. Для создания статической библиотеки генерируется объектный файл list.o и из него создается библиотека liblist.a. и перемещается в указанную директорию. И далее запускается сборка проекта и указанием местонахождения заголовочных и библиотечных файлов через атрибуты и подключением библиотеки.

Для создания динамической библиотеки в системную переменную LD\_LIBRARY\_PATH задается местонахождение папки с библиотекой проекта, после чего переменная экспортируется. Сама же библиотека liblist.so генерируется при помощи специальных атрибутов компилятора GCC и перемещается в нужную директорию, после чего запускается финальная стадия сборки.

К проектам прикреплены make-файлы для обоих вариантов.

# ВЫВОДЫ

Таким образом, в ходе выполнения данной работы были разработаны, скомпилированы и собраны две системы (одна из них также двумя вариантами) согласно условию задачи с применением компилятора GCC и системы сборки make. Папки проектов заархивированы.