###### -МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

###### ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

###### НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

###### Факультет информационных технологий

**Кафедра параллельных вычислений**

ОТЧЕТ

О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ

«Кэш память процессора. Скорость обращения и размеры.»

студента 2 курса

***extio1***

Новосибирск 2022

**СОДЕРЖАНИЕ**

[ЦЕЛЬ 3](#__RefHeading___Toc18443921)

[ЗАДАНИЕ 3](#__RefHeading___Toc18443922)

[ОПИСАНИЕ РАБОТЫ 4](#__RefHeading___Toc18443923)

[ЗАКЛЮЧЕНИЕ 8](#__RefHeading___Toc18443924)

[Приложения 9](#__RefHeading___Toc18443925)

# ЦЕЛЬ

Исследовать зависимость времени доступа к данным в памяти от их  
объема. Исследовать зависимости времени доступа к данным в памяти от  
порядка их обхода. Сравнить результаты замеров с реальным размером кэш-памяти.

# ЗАДАНИЕ

Написать программу, многократно выполняющую обход массива заданного размера тремя способами (прямой, обратный и случайный). Для каждого размера массива и способа обхода измерить среднее время доступа к одному элементу (в тактах процессора). Построить графики зависимости среднего времени доступа от размера массива. Определить размеры кэш-памяти точным образом. Оценить размеры кэш-памяти различных уровней, обосновать ответ, сопоставить результат с известными реальными значениями. Определить размеры массива, при которых время доступа к элементу массива при случайном обходе больше, чем при прямом или обратном; объяснить причины этой разницы во временах.

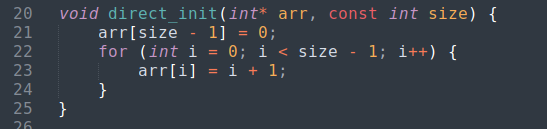
# ОПИСАНИЕ РАБОТЫ

Выполнение практической работы началось с написания программы, заполняющую массивы для прямого, обратного, случайного обхода массива, и замеряющую среднее время обращения к ячейке массива при каждом из обходов. Полный код в приложении 1.

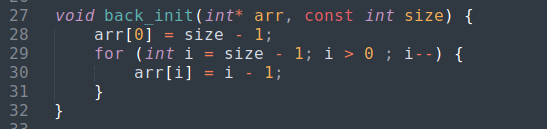


Функция test() выполняет обход и замеряет время обращение к элементу массива. direct\_init, back\_init, random\_init инициализирует массив для прямого, обратного, случайного порядка обхода соответственно. Результаты замеров записываются в .csv (Comma-Separated Values) файл, для того чтобы было удобнее работать с данными для постороения графиков.

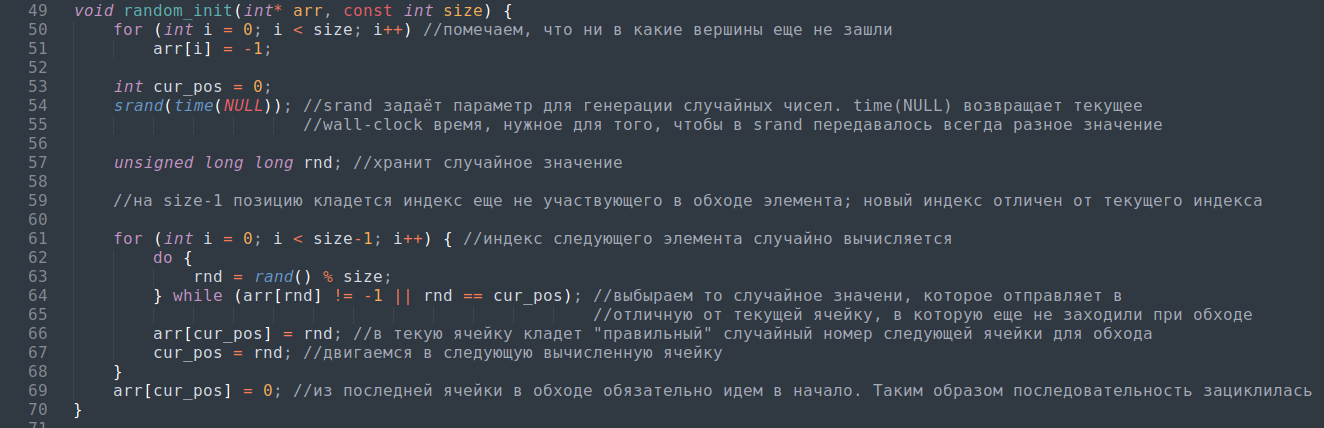
Функция заполнения массива для прямого обхода.

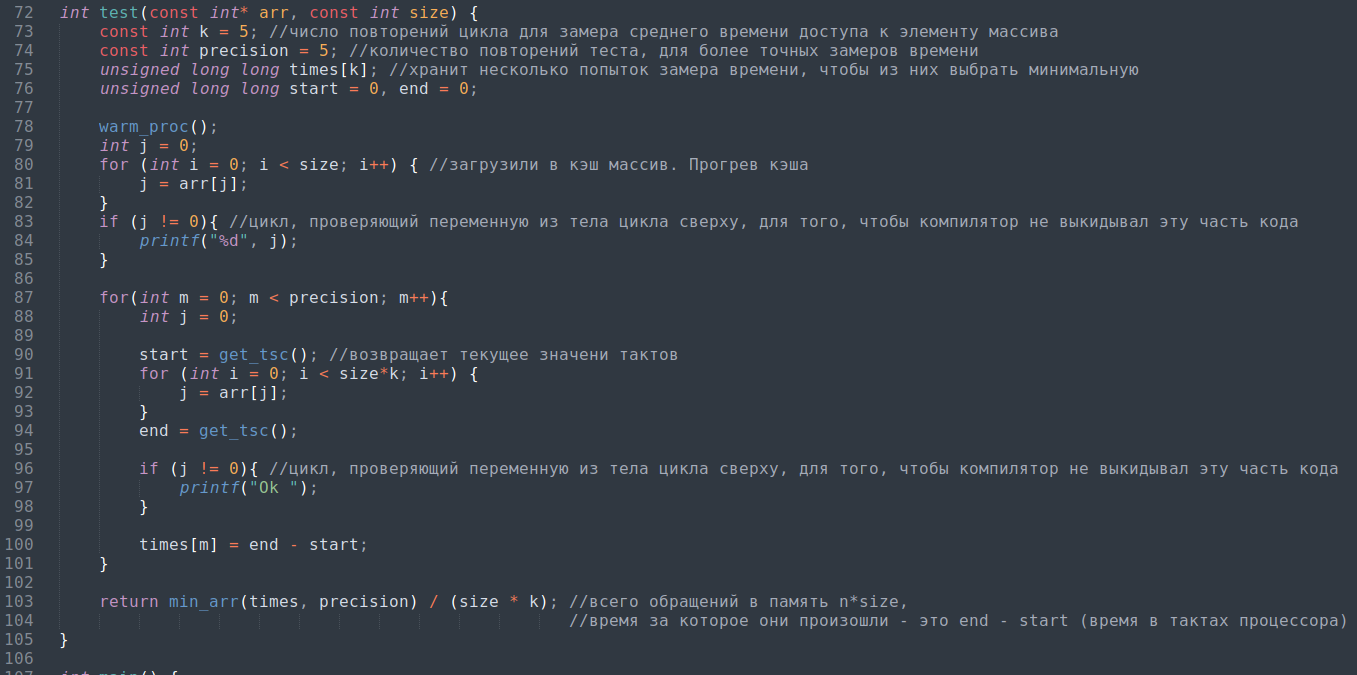


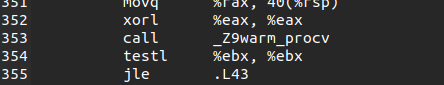
Функция заполнения массива для обратного обхода.

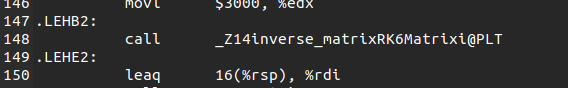


Функция заполенения массива для случайного обхода.

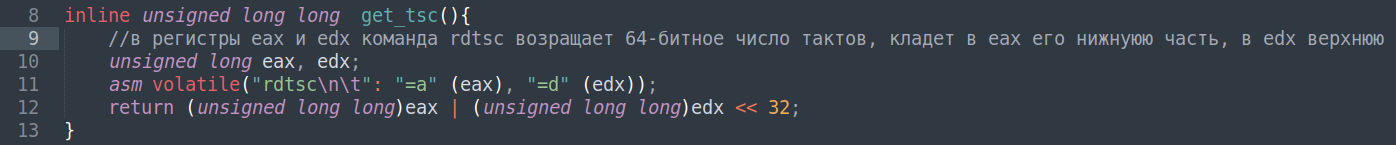
Основная функция, выполняющая обход и замеряющая время обращения к ячейке.

 Для того, чтобы проверить, что компилятор не выкинул фунцию warm\_proc, был скомпилирован cpp файл с ключем -S и изучен asm листинг. Main содержит обращение к warm\_proc, а warm\_proc вызыавает функцию нахождения обратной матрицы.

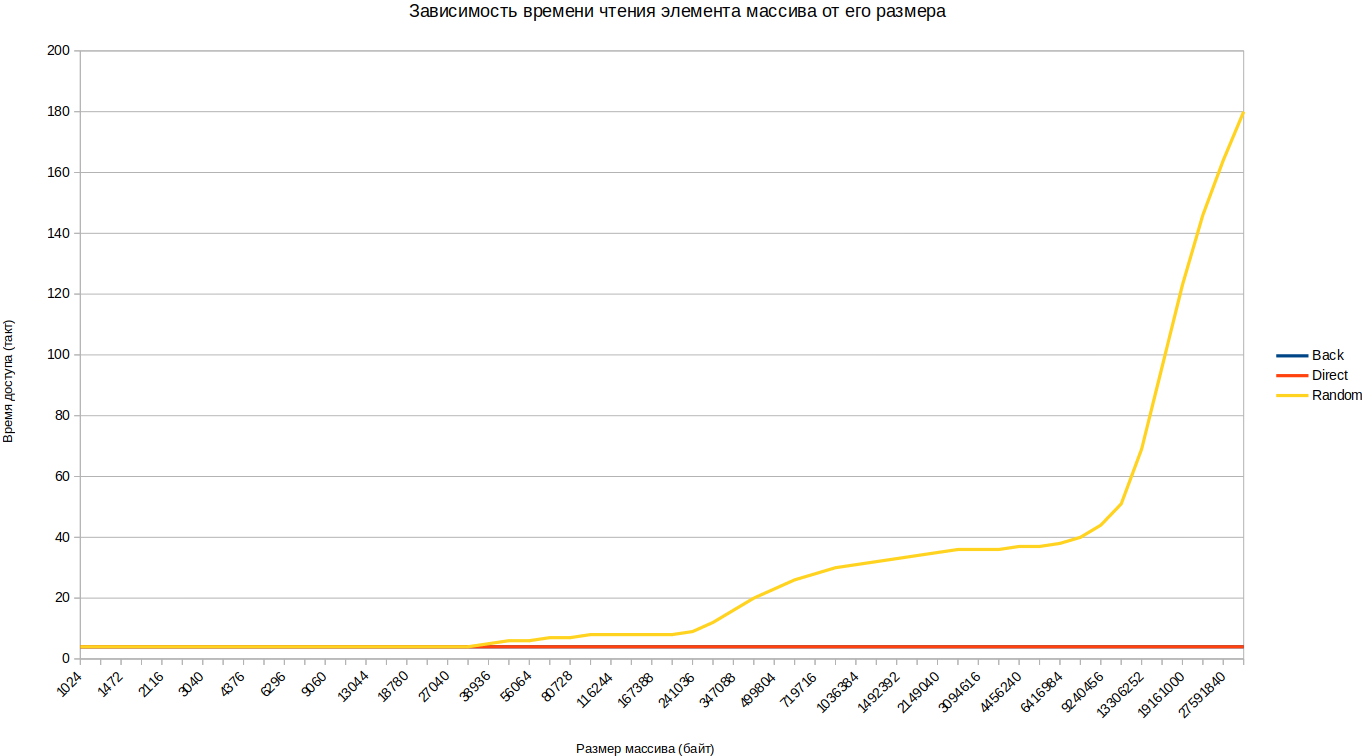




Функция, возвращающая значение счетчика тактов.



По полученным данным из файла cacheout.csv были построенный графики зависимости обращения к ячейке массива от его размера при разных обходах массива. (все графики в Приложении 2)

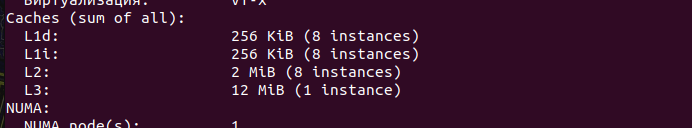
 **Анализ полученных данных**

На графиках видно, что скорость обращения к элементу массива не растет с увеличением размера массива при прямом и обратном обходе в отличие от случайного. Это объясняется тем, что процессор умеет распознавать прямой и обратный обход, поэтому он заразанее подгружает в кэш первого уровня следующую кеш-строку. Поэтому время доступа к элементу массива при прямом и обратном обходе равен времени чтения из кэша первого уровня.

При случайном обходе наблюдается рост времени обращения. Процессор не может рассмотреть закономерность в общениях в память, поэтому каждый раз данные он читает из кеша 1,2 или 3 уровня, если массив поместился в кеш, а при больших размерах и из оперативной памяти.

По времени обращении к памяти случайного обхода можно определить размеры кэшей. Обратим внимание на точки перегиба. Когда массив перестает помещаться в кеш одного из уровней, то время чтения увеличивается, что сказываться на среднем времени доступа к элементу массива, так как увеличивается число элементов, доступ к которым требует большего времени. Тогда видно, что у L1 размер ~32Kb, L2: ~250Kb, L3: ~8Mb

Обратимся к теримналу и введем команду lscpu, чтобы узнать размер кэша:

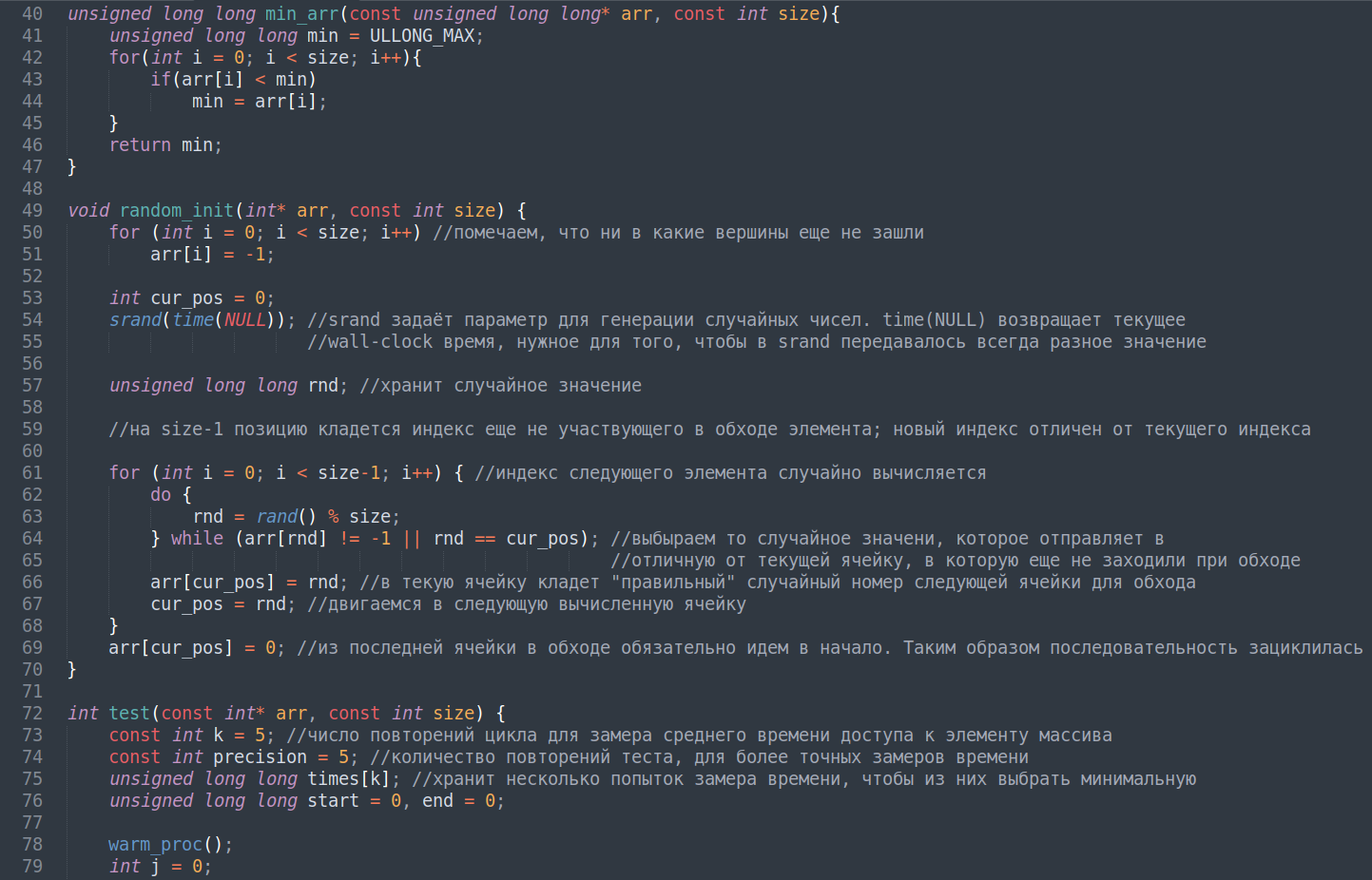
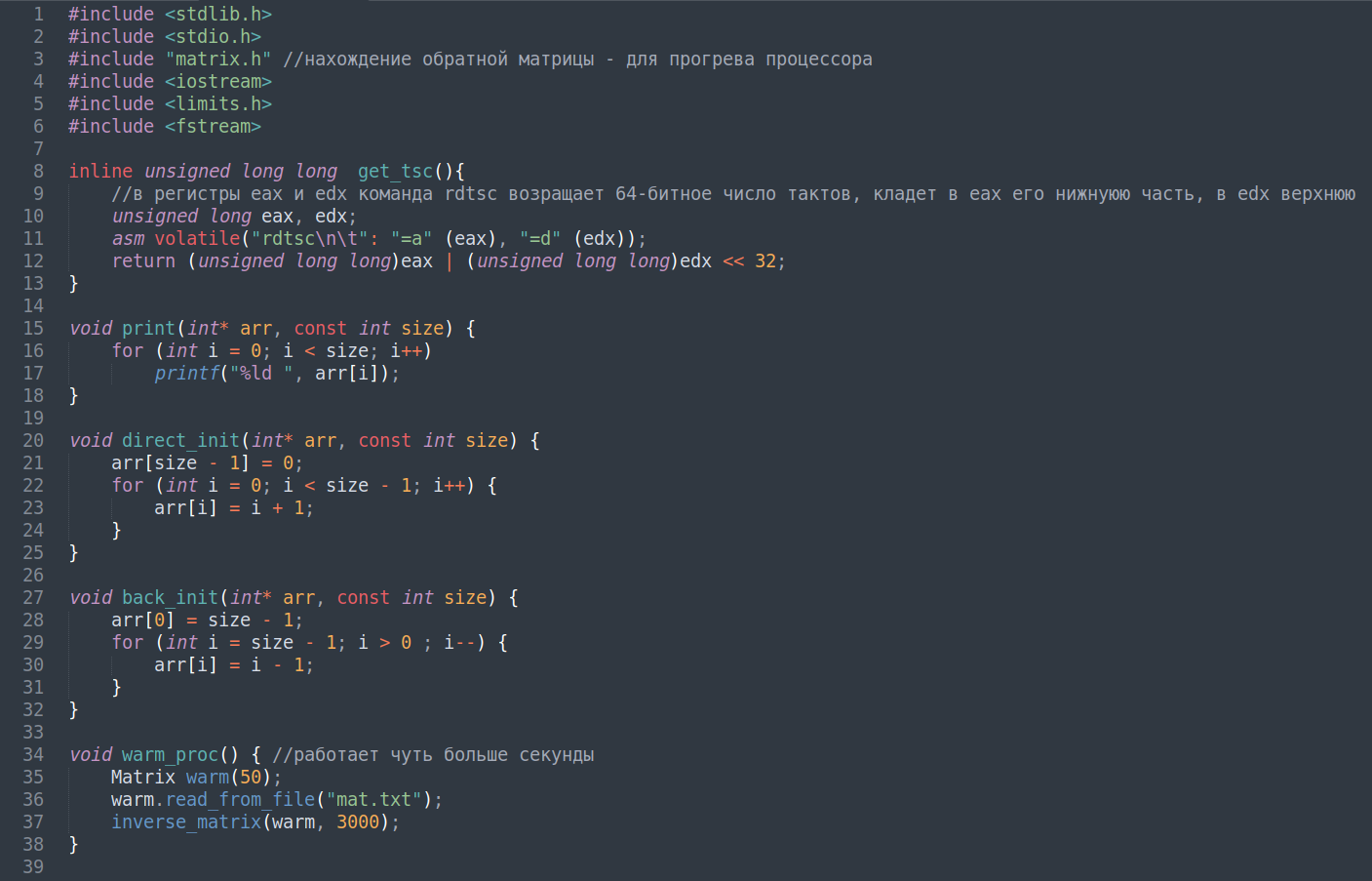


Заметим, что точки перегиба на графиках и фактические размеры кэшей совпадают для L1, L2. Делим цифру на 8 (у каждого ядра своя кэш память 1 и 2 уровня) и получаем 32 и 256 кбайт. Память 3 уровня в процессоре I7-9700 распределяется динамически среди 8 ядер.

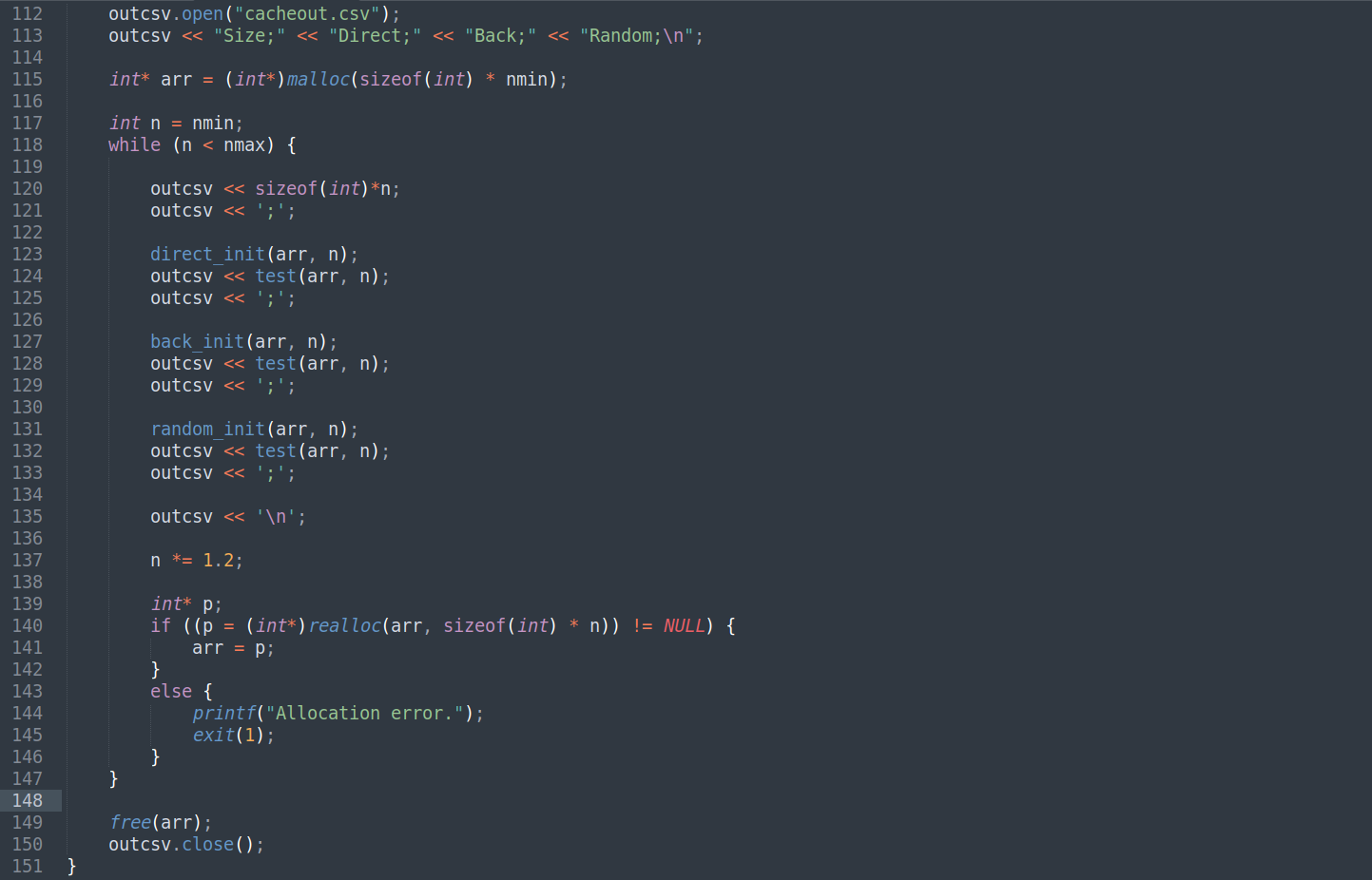
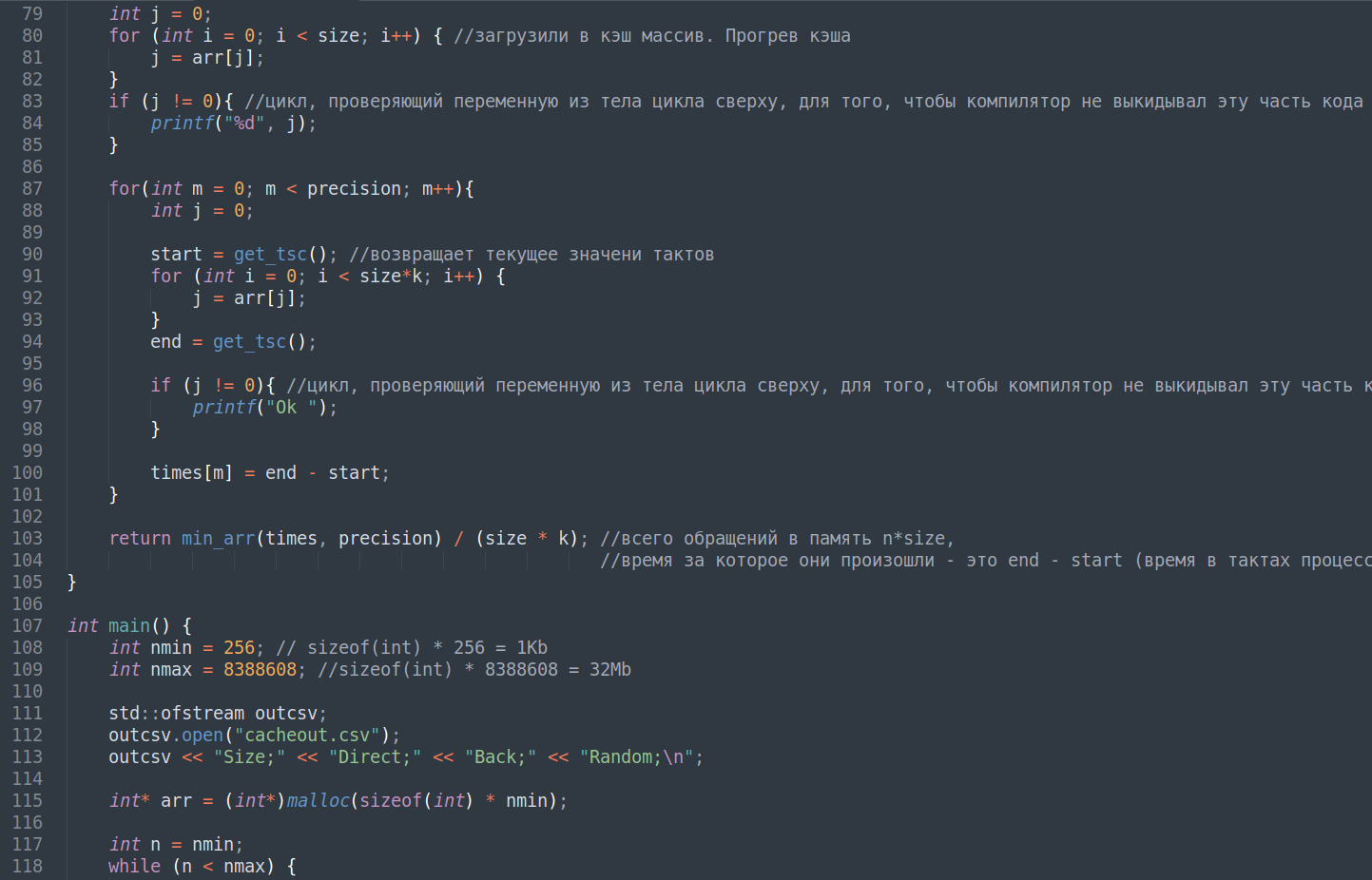
# ЗАКЛЮЧЕНИЕ

При написании данной работы была исследована работа программы с кэшем при чтении массива данных из памяти. Было выяснено, что прямой и обратный обход массива не требует большого количество тактов процессора для прочтения каждого элемента. Процессор распознает обходы таких видов и заранее грузит данные в кэш 1 уровня. Случайный обход массива является самым затратным по времени доступа к элементу способом способом чтения данных. Потому что в этом случае процессор не может оптимизировать расположение данных в памяти, и элемент читается ровно оттуда, где он лежит по факту (в кэше 1, 2, или 3 уровня, а может даже и из памяти).

# Приложение 1. Исходный код и команда для его компиляции



# Приложение 1. продолжение



# Приложение 2. Графики

# Приложение 2. продолжение