Первая лабораторная по курсу "Практика программирования с использованием C++". Можно считать это даже не как лабораторную, а как макродомашку. Какие-то задачи будут с защитой, какие-то с автоматическим тестированием и ручной проверкой кода (будет описано ниже). Система оценивания приведена сразу после описания заданий. Если внутри текста встречается жирный шрифт, то там установлена гиперссылка.

1 Представление типов данных в памяти

Для успешного выполнения этой части лабы вам может потребоваться тип **union**. Этот типа данных позволяет трактовать одну и туже область памяти (последовательность бит) как переменную разных типов, что удобно для предстоящих задач.

(Если вы знаете reinterpret_cast, то можете, конечно, и его использовать, но к вам будет больше вопросов на сдаче)

1.1 Тип *int*

В этой небольшой части посмотрим на битовое представление *int* в памяти и то, как *int* и *unsigned int* смотрят на одну и ту же последовательность битов по-разному.

Итак список задач для анализа представления чисел типа *int* в памяти:

1. Вывести побитовое представление переменной типа *int* в памяти. Сделать это для положительных и отрицательных значений переменной. Про дополнительный код можно прочитать **здесь**.

2. Вывести побитовое представление переменной типа *unsigned int* в памяти, сравнить представление положительных чисел типа *int* и *unsigned int* в памяти. Действительно ли *unsigned int* "сдвинут"на половину диапазона?

3. После того, как появилось осознание, как представляются положительные и отрицательные числа типа int в памяти, проведите эксперимент с прибалением 1 к INT_MAX. Какой получился ответ? Как произошло прибавление с точки зрения двоичного кода? Почему комьютер трактует результат именно так? Аналогично проведите экперимент с вычитанием 1 из 0 типа *unsigned int*. Поругайте создателей компьютера за контринтуитивное поведение целочисленных типов.

1.2 Тип *float*

Теперь более сложный монстр. Посмотрим на представление числа типа float в памяти и на интересные особенности операции с ними. Напоминаю, что числа типа float реализуют действительные числа в машинной арифметике, а поэтому операции с ними должны быть примерно походи на операции с действительными числами в математике. Предлагается посмотреть, так ли это на самом деле, однако начнем по порядку. Теорию по числам с плавающей точкой можно посмотреть в книге **"Методы численного анализа"** (глава 6).

Оговорю сразу, что для чисел типа double рассматриваемые в этой части лабы эффекты тоже существуют и подчиняются тем же принципам, просто их более тяжело ловить, чем на float'ах (для этого

двойная точность и была придумана). Поэтому все, что предлагается сделать в этой части, можно сделать и на типе *double*.

1. Вывести двоичное представление числа типа *float*, показать где мантисса и где экспонента. На примере чисел, которые абсолютно точно могут быть представлены в памяти показать как переводится двоичный код в значение действительного числа.

2. Мы понимаем, что диапазон *float* конечен и дискретен из-за конечности битовой последовательности. Так же мы понимаем, что из-за способа хранения чисел типа *float* расстояние между соседними числами a и b, которые точно можно закодировать увеличивается с увеличением модуля этих чисел. Значит, что в какой-то момент расстояние между соседними числами станет равным 1. Задача: найти первые такие два числа a и b. Показать, что $\forall x > a \longrightarrow x == x + 1$, то есть значение суммы округляется обратно.

3. Самый занимательный пункт: показать, что для арифметических операций с *float* нет свойства ассоциативности, то есть $\exists\, a, b, c : (a + b) + c \neq a + (b + c)$

4. Показать, что в компьютерной арифметике с *float* ряд $\sum_{k=1}^{n}(1/k)$ сходится и найти первый элемент (пусть его номер k), прибавление которого не увеличивает сумму (происходит округление, что рассмотрено в пункте 2).

5. Найти значение частичной суммы гармонического радя из п.3 до значения k + 100. Показать, что если начать суммирование с конца, то результат получится другой. Объяснить эффект на основе полученных в п.1–4 знаний.

6. Подводим итоги. Наверное, вы уже слышали рецепт: *float* не сравнивают через ==. Объяснить, почему такое сравнение опасно. Опасно ли сравнить числа типа *float* через ==, если с ними не было произведено арифметических операций?

2 Работа с памятью и указателями

Сортировки Как вы уже поняли, в этом пункте нужно написать сортировки, которые сортируют массив, переданный по указателю. Правильность написания сортировки нужно показать замерами асимптотики. Объявления (интерфейсы) функций, которые нужно реализовать, будут лежать в файлике *sort_ interface.cpp*.

1. Написать сортировку бинарной кучей

2. Написать сортировку слиянием в рекурсивной форме (тому, кто решит использовать std::merge, задание не засчитывается)

2.1 Дополнительные задачи

- Реализовать алгоритм Гаусса решения СЛАУ, используя двумерные динамические массивы. Напишите пару тестовых примеров с не очень тривиальными матрицами, показывающих, что алгоритм работает как нужно.

Внимание: матрица 3x3, заполненная последовательностью от 1 до 9, вырождена, не попадите в ловушку.

- Реализуйте множественную двустороннюю связь, используя структуры и указатели. Для этого создайте два массива, в которых будут храниться экземпляры некоторых структур. Например, пусть в первом массиве хранятся экземпляры структуры Student, а во втором массиве экземпляры структуры Lesson, при этом в каждом экземпляре структуры Student должен существовать массив указателей на экземпляры структуры Lesson и наоборот, т.е. каждый студент связан с несколькими занятиями и каждое занятие связано с несколькими студентами. Реализуйте функцию, которая по экземпляру Student выводит все соответствующие ему Lesson, и наоборот, по Lesson выводит всех соответствующих Student.

3 Структуры данных

В этом разделе лабы нужно будет реализовать две структуры данных, которые мы с вами обсуждали на паре: vector и list. Описание структур и интерфейсы функций приведенны в файлах *interface_ vector.cpp* и *itrefrace_ list.cpp*.

Ещё раз напомню, что vector – это структура данных с произвольным доступом за O(1) и вставкой за O(n), гарантирующая, что элементы в ней лежат последовательно в памяти. list – структура данных с произвольным доступом за O(n) и вставкой за O(1), не дающая никаких гарантий по памяти, элементы могут лежать где угодно. Ещё раз простым языком можно посмотреть **тут**.

Тестовый код представлен в *vector_ profiler.cpp* и *list_ profiler.cpp* соотвественно названию.

4 Оцнивание

- Первая часть неделима и оценивается полностью в 2.5 балла. Для успешной сдачи необходимо показать код с комментариями-ответами на вопросы, поставленные в пунктах задач. Предполагается сдача с обсуждением того, как трактовать ваши результаты. Не бойтесь: текста и смысла много, а кода по факту будет мало.

- Вторая часть делится. За сортировку кучей можно набрать 0.5 балл, за слияние – 1 балла. При этом необходимо доказать, что сортировка работает верно, то есть предоставить замеры асимптотики. Без этого не совсем очевидно, что именно вы написали. Эта часть подразумевает код-ревью с возможной отправкой на доработку.

  И да, я знаю примерно 10 сайтов, откуда можно взять реализацию сортировок на C++, однако копипаста без понимания бессмысленна, также она выскрывается одним вопросом по коду. Вот и думайте.

- Во второй части есть две дополнительные задачи. Каждая из них оценивается в 1 балл

- Корректная реализация структуры данных vector оценивается в 2.5 балла

- Корректная реализация структуры list оценивается в 3.5 балла

За все лабы результат будет суммироваться, и уже финальная сумма будет округляться.

Итого: $2.5 + 0.5 + 1 + 2.5 + 3.5 + (1 + 1) = 12$ баллов за лабу. Я вижу baseline следующий: vector + сортировки + часть 1, за что получается 6.5 баллов. Каждая из частей, сделанная полностью, позволит вам существенно лучше разобраться в том, что мы будем дальше проходить, а свойства float — важнейшая вещь в численном решении физических задач.

Успехов!