Большое домашнее задание 2

2 декабря 2024 г.

1 Общее описание задачи

Вам дан код для симуляции поведения нескольких несжимаемых жидкостей в замкнутом пространстве.

В целях упрощения (или усложнения?) задания физика жидкостей в симуляторе весьма далека от реальной.

Посредственное качество кода – тоже часть задачи.

Код использует для всех вычислений один тип данных – `FIXED(32, 16)` (о типах данных – в секции 2).

Как вы знаете, типы данных могут сильно влиять на скорость исполнения кода и потребление памяти.

И так, ваша задача – обобщить заданный код на произвольные типы данных (понимать логику работы при этом необязательно).

Помимо этого, надо научить код принимать входные данные (начальное расположение жидкостей, ускорение свободного падения g и плотности rho) из файла.

2 Типы данных

Все вычисления в симуляторе ведутся в вещественных числах. Необходимо реализовать поддержку как минимум следующих типов для их хранения:

- `FLOAT` – он же обычный платформо-зависимый `float` из C++.
- `DOUBLE` – неожидано, платформо-зависимый `double` из C++.
- `FIXED(N, K)` – вещественное число с фиксированной точкой. Представляет из себя N-битное знаковое целое число, деленное на 2^K. В отличии от предыдущих, платформо-независимый. Всегда занимает ровно N бит. Если такой размер чисел не поддерживается целевой архитектурой – ошибка компиляции. В примере уже написан `FIXED(32, 16)`, остальные напишите сами.

- `FAST_FIXED(N, K)` – отличается от предыдущего тем, что может занимать больше N бит. Импользует в качестве носителя быстрейший тип, имеющий хотя-бы N бит (то есть стандартные `int_fast8_t`, `int_fast16_t`, ...). Если целевая архитектура не поддерживает целочисленные типы размера $\geqslant N$ то это приводит к ошибке компиляции.

3 Более подробная постановка задачи

Ожидаемый результат – файл (или набор файлов) с кодом, который можно скомпилировать с опцией `-DTYPES=...`, где ... – это перечисленные через запятую типы. Например `-DTYPES=FLOAT,FAST_FIXED(13, 7),FIXED(32,5),DOUBLE`. Опционально можно передать еще опцию `-DSIZES=...`, где перечислены доступные размеры поля через запятую в формате `S(N,M)` (например `-DSIZES=S(1920,1080),S(10,10),S(42,1337)`).

Далее скомпилированный результат можно запустить, передав ему параметры `--p-type=..., --v-type=..., --v-flow-type=...`, которые задают типы, в которых должны быть соответствующие величины из примера (массивы p, velocity и velocity_flow соответственно). Если один из представленных типов не был передан при компиляции то программа сообщает об ошибке.

Далее программа считывет входные данные их файла (его тоже можно передать, как аргумент командной строки) и начинает симуляцию. При этом, если размер поля, считанный из файла соответствует одному из размеров, переданных при компиляции, то программа должна использовать поле статического размера, иначе можно использовать динамический размер.

Формат файла вы выбираете сами, можно использовать текст, json, какой-то свой бинарный формат или что-то еще.

Учтите, что выбор типов и размеров при компиляции сделан для повышения производительности, то есть прятать типы за какими-нибудь абстрактными классами и вызывать на каждую арифметическую операцию по виртуальной функции не стоит – теряется весь смысл.

4 Предлагаемый план решения

1. Вынести код из глобального неймспейса в класс.
2. Сделать класс шаблоном, получающим в параметрах типы и размеры поля.
3. Написать шаблон `Fixed<N, K>`.

4. Преобразовать опции компиляции в более понятные шаблонные типы (тут пригодятся макросы).

5. Написать на шаблонах нечто, которое будет преобразовывать параметры из рантайма в статические наборы параметров для вашего класса. Т.е. во время исполнения выбирать одну из предкомпилированных реализаций. Это основня часть задачи.

6. Сделать считывание из файла.

5 Система оценки

- Ввод из фала – 1 балл
- Написаный шаблон Fixed – 1 балл
- Шаблонный клас симулятора с выбором типов, размеров и возможностю использовать динамический размер – 2 балла
- Выбор типов и размеров в рантайме (без потерии производительности по сравнению с выбором во время компиляции) – 4 балла
- То же самое, но с потерей производительности (стирание типа числа) – 2 балла, если не сделан предыдущий пункт.
- Возможность сохранить параметры симуляции на каждом шаге, сохранить результат и продолжить его с произвольной точки – до 2 баллов, требует подробнее разобраться в коде.