**21.5 Практическая работа**

Задача

В практической работе есть:

Обязательные задачи помогут проверить, как вы усвоили материал модуля. Их нужно сдать на проверку куратору.

Дополнительные задачи подойдут тем, кто хочет ещё немного потренироваться, решая задачки по программированию. Вы также можете отправлять их на проверку и получать обратную связь куратора, хотя это необязательно.

Цели практической работы

Приобрести практическое умение работы со структурами. Научиться:

* объявлять структуры данных,
* инициализировать структуры данных,
* считывать и записывать поля структур данных,
* заводить массив структур.

Что входит в работу

Обязательные задачи:

1. Реализовать ведомость учёта.
2. Реализовать подробную модель данных посёлка.
3. Реализовать математический тип двумерного вектора.

Дополнительные задачи:

1. Реализовать ролевую пошаговую игру.

**Общие условия**

Разработка выполняется в среде [CLion](https://www.jetbrains.com/clion" \t "_blank). Текстовые документы можно редактировать в стандартном блокноте.

Задание 1. Реализация ведомости учёта

Что нужно сделать

Вы уже создавали программу чтения и записи ведомости. Теперь её надо будет обобщить и дополнить использованием структур.

Формат ведомости прежний: сначала идёт имя и фамилия получателя денежных средств, далее — дата выдачи в формате ДД.ММ.ГГГГ, затем — сумма выплаты в рублях. Данные разделяются пробелами. В конце каждой записи должен быть перевод строки.

Структура данных записи должна соответствовать указанному формату.

При старте программы пользователь указывает команду, которую следует осуществить: list или add. Команда list осуществляет чтение из файла ведомости, как и прежде, только уже в структуры данных, и отображает их на экране.

Команда add добавляет новую запись в конец ведомости.

Советы и рекомендации

Чтобы определить конец файла, используйте, как и прежде, функцию file.eof(). Из-за завершающего переноса строки также стоит проверить и имя. Если оно пустое, то файл закончился.

Используйте вектор структур. Для выбора команды потребуется подобная конструкция:

if (command == "list")   
{   
…   
}

Что оценивается

Корректность работы программы. Программа должна всегда использовать структуры и все операции проводить через них.

Как отправить работу на проверку

Пришлите ссылку на repl.it или файл CPP с решением через форму ниже.

Задание 2. Модель данных для посёлка

Что нужно сделать

С помощью структур данных создайте подробную модель посёлка.

В посёлке множество участков, каждый ~~имеет уникальный номер~~. На каждом участке должен быть расположен ~~как минимум один жилой дом~~, но возможны и ~~другие сооружения~~, такие как гараж, сарай (бытовка) и баня. ~~Для каждого здания предусмотрена информация о площади~~, которую он занимает на участке.

~~В жилом доме может быть 1–3 этажа~~, на ~~каждом из этажей — 2–4 комнаты: спальня, кухня, ванная, детская, гостиная~~. Для каждой комнаты предусмотрены данные о площади. ~~Для каждого этажа также есть данные о высоте потолка~~. В ~~жилом доме и бане может быть печь с трубой~~, информация об этом также должна быть предусмотрена в данных.

Для всех упомянутых сущностей надо объявить соответствующие структуры данных.

Вначале работы программы пользователь заполняет данные о посёлке: вводит общее количество участков и далее начинает детализировать каждый участок.

Для участка сначала заполняется общее количество построек, а затем детализируются постройки, этажи, комнаты на этажах.

По итогам введённых данных вы можете рассчитать суммарный параметр, например процент земли, занимаемой постройками, к общей площади всего посёлка. Главное в этой задаче — не операция подсчёта, а структуры данных.

Советы и рекомендации

Используйте все имеющиеся у вас знания о типах данных в C++. Можно использовать уже изученные перечисления и векторы, создавать структуры внутри структур и так далее.

Вам потребуется перечислять типы комнат и зданий подобным образом:

enum room\_type   
{   
    living,   
    children,   
    …

А затем использовать полученный тип внутри структуры. Алгоритм заполнения каждого объекта будет похож: создать объект нужного типа, ввести с клавиатуры значения, добавить полученный объект в вектор при помощи push\_back.

Что оценивается

Полнота структур, описывающих данные посёлка.

Как отправить работу на проверку

Пришлите ссылку на repl.it или файл CPP с решением через форму ниже.

Задание 3. Реализация математического вектора

Что нужно сделать

Реализуйте структуру двумерного математического вектора и основные операции над ним. Обе координаты вектора (x и y) должны быть вещественными числами.

В начале программы пользователь вводит команду, которая соответствует требуемой операции. Далее, в зависимости от операции, пользователь вводит её аргументы. Это могут быть как векторы, так и обычные, скалярные значения. В результате в стандартный вывод помещается результат. Это тоже может быть или вектор, или скаляр.

Реализуйте следующие операции в виде отдельных функций:

* сложение двух векторов — команда add;
* вычитание двух векторов — команда subtract;
* умножение вектора на скаляр — команда scale;
* нахождение длины вектора — команда length;
* нормализация вектора — команда normalize.

Что оценивается

Корректность работы программы и всех операций над вектором или векторами.

Как отправить работу на проверку

Пришлите ссылку на repl.it или файл CPP с решением через форму ниже.

Советы и рекомендации

Снова используйте конструкцию вида if (operation == "add")   для выбора команды. Тут так и напрашиваются отдельные функции для каждой операции.

Задание 4. Реализация ролевой пошаговой игры (дополнительное задание)

Что нужно сделать

Реализуйте сильно упрощённую версию ролевой пошаговой игры.

Игра происходит на карте размером 20 на 20 клеток. По клеткам перемещаются враги и персонаж игрока.

После каждого хода игрока карта показывается вновь со всеми врагами на ней. Игрок помечается буквой P, враги — буквой E, пустые места — точкой.

Каждый персонаж игры представлен в виде структуры с полями: имя, жизни, броня, урон.

Вначале игры создаются 5 случайных врагов в случайных клетках карты. Имена врагам задаются в формате Enemy #N, где N — это порядковый номер врага. Уровень жизней врагам задаётся случайно — от 50 до 150. Уровень брони варьируется от 0 до 50. Урон тоже выбирается случайно — от 15 до 30 единиц.

Игрок конструирует своего персонажа самостоятельно. Задаёт все его параметры, включая имя.

Все персонажи появляются в случайных местах карты.

Игрок осуществляет ход с помощью следующих команд: w, a, s, d. В зависимости от команды выбирается направление перемещения персонажа: влево, вправо, вверх, вниз.

Враги перемещаются в случайном направлении.

Если персонаж (враг или игрок) перемещается в сторону, где уже находится какой-то персонаж, то он бьёт этого персонажа с помощью своего урона. Враги при этом никогда не бьют врагов, а просто пропускают ход и остаются на своём месте. За пределы карты (20 на 20 клеток) ходить нельзя никому. Если кто-то выбрал направление за гранью дозволенного, ход пропускается.

Формула для расчёта урона совпадает с той, что была в занятии. Жизни уменьшаются на оставшийся после брони урон. При этом броня тоже сокращается на приведённый урон.

Игра заканчивается тогда, когда либо умирают все враги, либо персонаж игрока. В первом случае на экран выводится сообщение о поражении, во втором — победа.

Если в начале хода игрок вводит команду save или load вместо направления перемещения, то игра либо делает сохранение своего состояния в файл, либо загружает это состояние из файла соответственно.

Советы и рекомендации

Для определения команды персонажа можно завести специальный флаг внутри структуры данных персонажа.

Для отображения координат персонажей можете использовать структуру вектора из другой задачи, но заменить типы координат.

Для сохранения параметров игры и их загрузки можно работать с текстовыми данными. Однако записать в двоичном формате куда проще, главное соблюдать последовательность полей:

std::ofstream file("save.bin", std::ios::binary);   
    for (int i = 0; i < 6; ++i) {   
        file.write((char\*)&(characters[i].position),   
                   sizeof(characters[i].position));   
…

При загрузке используйте ifstream и ту же последовательность полей.

Что оценивается

* Корректно работает сохранение и загрузка в игре.
* После загрузки все атрибуты игрока, включая имя, восстанавливаются из файла.
* После загрузки количество противников на игровом поле соответствует количеству противников при сохранении.
* Все игроки после загрузки появляются на координатах, в которых они находились при сохранении.
* При ходе игрока не происходит появления новых противников.
* Перед загрузкой файла сохранения проверяется его наличие в операционной системе.

Как отправить работу на проверку

Пришлите ссылку на repl.it или файл CPP с решением через форму ниже.

Критерии оценки

* Программы запускаются и корректно работают, нет синтаксических ошибок.
* Имена переменных корректные, отражают суть хранимых данных.
* Создан текстовый интерфейс для всех программ.
* Во всех задачах корректно использованы структуры.
* Перед загрузкой файла в задачах проверяется его существование на диске.
* Выполнено три задания и более.