# Задания к работе 3 по алгоритмам и структурам данных.

Все задания реализуются на языке программирования C++ (стандарт C++14 и выше). Реализованные в заданиях приложения не должны завершаться аварийно.

Во всех заданиях запрещено использование глобальных переменных (включая errno).

Во всех заданиях запрещено использование оператора безусловного перехода (goto).

Во всех заданиях запрещено пользоваться функциями, позволяющими завершить выполнение приложения из произвольной точки выполнения, вне контекста исполнения функции main.

Во всех заданиях при реализации необходимо разделять контексты работы с данными (поиск, сортировка, добавление/удаление, модификация и т. п.) и отправка данных в поток вывода / выгрузка данных из потока ввода.

Во всех заданиях все параметры функций и вводимые (с консоли, файла, командной строки) пользователем данные должны подвергаться валидации в соответствии с типом валидируемых данных, если не сказано обратное; валидация должна зависеть от типа данных и логики применения этих данных для выполнения целевой подзадачи. При передаче аргументов приложению в командную строку, их количество также должно валидироваться.

Во всех заданиях необходимо контролировать ситуации с невозможностью [пере]выделения памяти; во всех заданиях необходимо корректно освобождать всю выделенную динамическую память.

Все ошибки, связанные с операциями открытия файла, должны быть обработаны; все открытые файлы должны быть закрыты.

Во всех заданиях запрещено использование глобальных переменных. Во всех заданиях при реализации функций необходимо обеспечить возможность обработки ошибок различных типов на уровне вызывающего кода.

Во всех заданиях сравнение (на предмет эквивалентности или отношения порядка) вещественных чисел на уровне функции должно использовать значение эпсилон, которое является параметром этой функции.

Во всех заданиях при реализации функций необходимо максимально ограничивать возможность модификации (если она не подразумевается) передаваемых в функцию параметров (используйте ключевое слово const), а также вызывающего объекта, в случае вызова его методов.

Для реализованных компонентов должны быть переопределены (либо перекрыты / оставлены реализации по умолчанию - при обосновании) следующие механизмы классов C++: конструктор копирования, деструктор, оператор присваивания.

Во всех заданиях необходимо уменьшать количество копирований нетривиально копируемых объектов.

Во всех заданиях необходимо проектировать компоненты с учетом SOLID принципов. Компонент не должен управлять ресурсом, если это не является его единственной задачей.

Запрещается пользоваться элементами стандартной библиотеки языка C, если существует их аналог в стандартной библиотеке языка C++.

Запрещается использование STL.

1. Разработайте приложение для моделирования автоматизированной системы контроля работы лифтов.
   Основные требования к системе контроля движения лифтов:
   Время в системе дискретное с шагом 1 минута.
   Имеется $n$−этажное здание. В здании имеется $k$ лифтов. Один лифт может находиться в следующих состояниях:
   - стоит с закрытыми дверями;
   - стоит с открытыми дверями;
   - движется вверх;
   - движется вниз.

   Для всех состояний важным представляется значение номера этажа, на котором лифт в данный момент находится. Двигаться лифт может только закрытым.
   В лифте имеется $n$ кнопок с номерами этажей. Кнопки с номерами этажей могут находиться в двух положениях: “Нажата” и “Отжата”. Нажатая кнопка с номером этажа означает, что имеется задание на движение лифта в направлении к данному этажу. Направление движения должно определяться положением лифта в текущий момент относительно целевого этажа. В каждый момент времени в состоянии “Нажата” может находиться несколько кнопок с номерами этажей. При движении в каком-либо направлении по достижении следующего нужного этажа лифт останавливается, и соответствующая кнопка принимает положение “Отжата”.
   На каждом этаже имеется кнопка вызова лифта. После нажатия кнопки вызова лифта кнопка остается в нажатом состоянии до остановки лифта на данном этаже. Если кнопка вызова нажимается на этаже по пути следования лифта, лифт должен остановиться на данном этаже, после чего кнопка вызова на данном этаже должна быть автоматически отжата. Для обслуживания этажа выбирается любой из лифтов, наименее удаленных от этажа, которым нет необходимости менять направление движения. Иными словами, если в системе два лифта: первый лифт на первом этаже, а второй с пассажирами на 7 этаже поднимается вверх и нажата кнопка вызова лифта на 6 этаже, то на 6 этаж поедет первый лифт, а не второй.
   Расстояние между соседними этажами лифт преодолевает за $3 + 5 \times m_i$ секунд, где $m_i$ − текущая загруженность лифта (отношение массы перевозимого груза к максимальной нагрузке $M_i$), с округлением в бóльшую сторону. В лифте одновременно может находиться несколько человек (также лифт может двигаться пустым). Лифт не должен начинать движение, если суммарный перемещаемый вес превышает $M_i$ кг ( $i = 1, 2, \ldots, k$). Если суммарный вес людей, находящихся в лифте, превышает $M_i$ кг, в лифте возникает ситуация перегрузки, при этом последний зашедший должен покинуть кабину лифта (при этом в лифт могут зайти другие пассажиры, ожидающие лифт на этом же этаже) и, после того, как лифт уедет, повторно вызвать лифт.
   Для каждого пассажира должны быть определены следующие атрибуты: уникальный идентификатор, время появления (с точностью до минут), исходный этаж, целевой этаж, вес (в кг) - вещественное число двойной точности.
   Для моделирования работы системы лифтов разработайте интерфейс, обеспечивающий обработку набора заданий. Задания описываются текстовыми файлами, пути к которым передаются приложению в виде аргументов командной строки; в файлах указаны идентификаторы пассажиров, время появления, их вес, исходный этаж (этаж, на котором пассажир вызывает лифт), целевой этаж. Входные файлы считаются корректными. Пример файла:

   ```
   1 80 3 00:01 8
   11 65 2 00:15 6
   25 65 2 00:15 6
   ```

   Ещё один текстовый файл (путь к нему передаётся первым аргументом командной строки) содержит значения параметров $n$, $k$, $M_1 \ldots M_k$.
   Результатом работы моделирующего приложения должны являться два текстовых файла, в которых предоставлены отчеты о пассажирах и работе лифтов соответственно:
   - для каждого пассажира: время его появления, исходный этаж, целевой этаж, время погрузки в лифт, общее время движения, список пассажиров, с которыми он пересекался во время своего пути, покидал ли кабину лифта в связи с перегрузкой;
   - для каждого лифта: время простоя, время в пути, количество пройденных пролётов между этажами, суммарный перевезенный груз (значение в кг), максимальная выполненная нагрузка, количество перегрузок.

   Содержимое полученных отчётов должно быть сгруппированно по пассажирам и лифтам соответственно (порядок групп может быть произволен).

2. Некоторое инфекционное заболевание от человека к человеку передается контактным путем. Вероятность заразиться им при контакте равна $p_1 \in (0; 1]$; инкубационный период заболевания равен $I$ дней; заразившийся может излечиться с вероятностью $p_2 \in [0; 1]$; при этом излечение начинается сразу после инкубационного периода и требует $[T \ldots 3T]$ дней ($T > 3$); во время излечения заразившийся "самоизолирован" и не контактирует со своим окружением (с теми, с кем он непосредственно знаком), в остальное время контактирование с каждым из своего окружения постоянно; в случае неуспешного излечения заболевание становится хроническим, при этом носитель хронического заболевания не является его распространителем. На иммунитет к повторному заболеванию после успешного излечения влияет параметр $r$ (логическое значение: $true$ или $false$). Вспышкой болезни будем называть случайное возникновение заболевания у какого-либо псевдослучайного человека. Реализуйте приложение, моделирующее распространение заболевания в течение $d$ дней. Время в системе дискретно (шаг - 1 день).
В первый день моделирования происходит вспышка болезни.
Во входном файле построчно содержатся данные о человеке и его знакомствах. Формат одной записи:

<id>,<name>,<surname>,[<familiar people id's>,...]

На последней строке входного файла указываются значения параметров $p_1$, $p_2$, $d$, $r$, $T$, $I$.

Пример файла:

1,Иван,Иванов,2,3
2,Юрий,Андреев,1,4
3,Александр,Васильев,1,4
4,Учи,Матчасть,2,3
0.15 0.825 50 true 7 4

В выходной файл (используя класс логгера из задания 2) необходимо вывести информацию, сгруппированную следующим образом:

- фамилии и имена всех не заразившихся в течение моделирования людей;
- фамилии и имена всех исцелившихся в течении моделирования людей;
- фамилии и имена исцелившихся людей, окружение которых не исцелилось;
- фамилии и имена всех не заразившихся людей, у которых всё их окружение заразилось более одного раза;
- фамилии и имена всех людей, переболевших более одного раза (только в случае, если $r$ имеет значение $true$);
- фамилии и имена людей, у которых заболевание перешло в хроническую форму;

Для демонстрации работы реализуйте приложение-генератор входного файла, который будет подан на вход основному приложению. Выходной файл приложения должен содержать записи о 10'000-1'000'000 людях с псевдослучайно сгенерированными фамилиями и именами, а также знакомствами (граф знакомств должен быть связным). Информация о связях между людьми в выходном файле должна быть согласована (если человек с id = 52 знает человека с id = 1089, то и человек с id = 1089 знает человека с id = 52).

3. Инфекционное отделение поликлиники имеет смотровую на *N* человек и *M* докторов, которые готовы оказать помощь заболевшим или проконсультировать здоровых людей. По правилам инфекционного отделения в смотровую может зайти здоровый человек, если в ней есть свободное место и в ней находятся только здоровые люди, и наоборот: при наличии свободных мест заболевший человек может войти в смотровую, если там только заболевшие. Как только человек вошел в смотровую, он занимает свое место и ожидает доктора. Незанятый доктор выбирает пациента, пришедшего раньше других и проводит приём, который длится от 1 до *T* временных единиц. В особых случаях, доктор может попросить у другого не занятого доктора помощи, которая также может длиться от 1 до *T* временных единиц. Если все места в смотровой заняты, то пришедшие пациенты встают в очередь. При этом в очереди спустя некоторое время, при наличии заболевшего человека, заражается вся очередь. Количество пациентов и интервал их появления произволен и случаен.
   Реализуйте приложение, моделирующее работу инфекционного отделения поликлиники. Ваша программа должна вести лог-файл с подробной историей работы инфекционного отделения. Необходимо сохранять информацию о:
   - пришедших пациентах и их состоянии;
   - времени работы докторов и времени оказания помощи каждому отдельному пациенту;
   - количестве временных единиц, проведённых пациентом с момента поступления в отделение до момента окончания его приёма;
   - заражении всей очереди с указанием количества новых заболевших.

   Продемонстрируйте работу вашей программы при различных значениях параметров *N*, *M*, *T*. Подберите параметры так, чтоб показать особые случаи, которые могут возникнуть в инфекционном отделении.

4. Реализовать систему усиления игровых персонажей с помощью случайных элементов экипировки для некоторой MMORPG.

   Каждый персонаж может иметь некоторое количество ячеек для установки усиления (очень часто под ячейками имеют в виду слоты для элементов одежды, например, шлем, наплечники, перчатки, плащ и др.). Для каждого персонажа определены главные его характеристики: здоровье, броня и, в зависимости от класса персонажа: сила, ум, ловкость, и вторичные характеристики: меткость, везение, мастерство. Персонажи классифицируются следующим образом: защитники, целители, бойцы ближнего боя и бойцы дальнего боя. В зависимости от класса персонажа для него имеет значение та или иная главная характеристика. Для защитников – это здоровье, для целителей – ум, для бойцов ближнего боя – ловкость или сила, для бойцов дальнего боя – ум.

   Вторичные характеристики действуют одинаково для всех классов: меткость определяет возможность промаха при атаке, везение определяет возможность нанесения двойного урона, мастерство увеличивает наносимый урон.

   Продумайте и реализуйте закон для расчета урона (или лечения), наносимого персонажем в зависимости от его главных и вторичных характеристик.

   Продумайте и реализуйте для каждого класса персонажа его способности, а также систему генерации усилений. При создании персонажа никаких усилений (экипировки) у него нет. Продумайте возможность генерации необходимых элементов. Обратите внимание что на каждом предмете экипировки могут быть следующие характеристики: здоровье, броня, ум или сила, или ловкость, и одна или две вторичные характеристики.

   Общая характеристика персонажа определяется как суммарная совокупность значений характеристик на всех его элементах экипировки.

   При демонстрации работы приложения Вам необходимо показать генерацию элементов экипировки созданного персонажа, наносимый урон способностями в зависимости от текущих характеристик. Также необходимо иметь возможность создать произвольное количество случайных персонажей (со случайной экипировкой). Использование различных контейнеров (массивы, списки, хеш-таблицы, приоритетные очереди, деревья поиска, стеки, очереди, деки и т. д.) для обеспечения хранения данных в системе должно быть обосновано.

5. Реализовать класс монома от нескольких переменных, содержащий в качестве полей значение коэффициента (объект типа дроби из работы 2 по АиСД), а также ассоциативный контейнер вида АВЛ-дерево, ключом которого является имя переменной (непустая строка, состоящая исключительно из символов букв латинского алфавита), а значением - степень этой переменной. В классе должны быть определены и реализованы:
   - конструкторы (один из которых конструктор от char const *: например, “<5/8*x^2*yy^3*zzz^0>”);
   - методы доступа к членам класса;
   - перегруженные операторы для выполнения стандартных арифметических операций: сложение (+=, +), вычитание (-=, -), умножение (*=, *);
   - перегруженные операторы для выполнения операций отношения эквивалентности на множестве мономов от нескольких переменных (==, !=);
   - перегруженные операторы выгрузки из потока / вставки в поток для корректного ввода / вывода монома.

   На основе реализованного класса монома необходимо реализовать класс полинома от многих переменных. Класс полинома представляет собой контейнер мономов, основанный на структуре данных вида двусвязный список. В классе полинома необходимо реализовать:
   - конструкторы (один из которых конструктор от char const *: например, “<5*xy^2*yz^3*zz^0-1/2*xy^3>”);
   - методы доступа к членам класса;
   - перегруженные операторы для выполнения стандартных арифметических операций: сложение (+=, +), вычитание (-=, -), умножение (*=, *);
   - перегруженные операторы для выполнения операций отношения эквивалентности на множестве мономов от нескольких переменных (==, !=);
   - перегруженные операторы выгрузки из потока / вставки в поток для корректного ввода / вывода монома.

   Для демонстрации работы вашей программы реализуйте возможность обработки текстового файла следующего вида:

   <4x^2y^3-xy^2+4xy+1>+<x^2y^3+2xy^2+3x^2y^2-4xy+5>;
   <4x^2y^3 +4xy+1>*< x^2y^3+2xy^2+3x^2y^2-4xy+5>;
   < x^2y^3+2xy^2+3x^2y^2-4xy+5>-<4x^2y^3-xy^2+4xy+1>;
   < x^2y^3+2xy^2>==< x^2y^3+2xy^2>;
   < x^2y^3+2xy^2>!=< x^2y^3+2xy^2>;

   *Замечание.* Арифметические операции, возвращающие новый объект, необходимо реализовать с помощью соответствующих им операций, модифицирующих вызывающий объект: например, операция + должна быть реализована с помощью операции +=. Продемонстрировать передачу аргументов в функции по значению и по ссылке. Также необходимо продемонстрировать возврат объекта из функции. Реализуйте возможность сохранения и восстановления ваших объектов в/из потоков. Формат строкового представления для монома и полинома продумайте самостоятельно.

6. На вход программе подается набор путей к текстовым файлам, содержащим инструкции вида:

```
{
        ShowVar;
        var1=new(<size>);
        {
                var2=new(<size1>);
                ShowVar;
                {
                        {
                                var3=new(<size2>);
                        }
                        var5=new(<size3>);
                        ShowVar;
                        var21=new(<size12>);
                }
                var22=new(<size21>);

        }
        var2=new(<size1>);
        ShowVar;
}
```

Количество инструкций и их порядок произволен. Приложение должно поочерёдно обработать инструкции из каждого переданного файла. Оператор *new* запрашивает нужный размер оперативной памяти (указывается в круглых скобках в виде целого неотрицательного числа, записанного в системе счисления с основанием 16). Оператор *ShowVar* выводит в стандартный поток вывода все видимые из данного места кода переменные, с указанием размеров памяти, который ими занят. Максимально доступный объем оперативной памяти определяется параметром *N*, являющимся аргументом командной строки. Фигурные скобки определяют блоки кода, которые определяют области видимости переменных. Если переменная покидает область видимости, она становится недоступной. При покидании области видимости, занятая память не возвращается автоматически. При этом учтите, что, если очередной вызов оператора new не может быть выполнен, то необходимо запустить операцию освобождения памяти под покинувшие свою область определения переменные, а затем вновь попытаться выделить требуемый блок. При невозможности выделения блока, необходимо завершить выполнение приложения. Для распределения динамической памяти используйте алгоритмы системы двойников.

7. Разработайте приложение для проведения лотереи (аналог Русского лото). Ваше приложение должно обеспечивать генерацию билетов для очередного тиража лотереи. Количество генерируемых билетов произвольно и может быть велико (> 20’000’000 шт.). Учтите ситуацию, что не все сгенерированные билеты могут участвовать в тираже (это типичная ситуация, которая возникает при неполной реализации билетов к тиражу). Смоделируйте проведение розыгрыша: на каждом ходе проверяйте, появился ли победитель; предусмотрите систему выигрышей; предоставьте возможность поиска билетов по заданным критериям: номеру билета, величине выигрыша, и т. д. Сохраняйте информацию о проведенных тиражах для обеспечения поиска данных в будущем. Реализуйте функционал обработки данных таким образом, чтобы тип коллекции/адаптера, в котором размещаются Ваши данные, являлся параметром шаблона. Продемонстрируйте обработку данных с подстановкой в качестве параметра шаблона собственных реализаций косого дерева и двусвязного списка.

8. Разработайте приложение для обработки данных логистической компании. Информация о доставке груза должна содержать информацию о грузе (название, вес, отправитель, получатель, стоимость, стоимость доставки, содержимое и т. д.), а также информацию об участках маршрута доставки (для участка необходимо хранить: пункт отправления, время отправления, пункт назначения, время доставки в пункт назначения, вид транспорта (автомобильный, железнодорожный, морской, авиационный)). Ваше приложение должно обеспечить возможность хранения и обработки данных о доставках (поиск по заданным критериям, добавление/удаление доставок). Для демонстрации работы реализуйте генератор (на базе паттерна “фабричный метод”), выдающий случайным образом сформированную информацию о доставке (необходимо проследить за тем, чтобы пункт назначения $n$ −ного участка и пункт отправления $(n + 1)$ −го участка совпадали). Реализуйте функционал обработки данных таким образом, чтобы тип коллекции/адаптера, в котором размещаются Ваши данные, являлся параметром шаблона. Продемонстрируйте обработку данных с подстановкой в качестве параметра шаблона собственных реализаций красно-чёрного дерева и хеш-таблицы, разрешающей коллизии методом цепочек.

9. Разработайте приложение для моделирования проведения лотереи (аналог Спортлото (6 из 49 или 5 из 36)). Ваше приложение должно обеспечивать генерацию билетов для очередного тиража лотереи. Количество генерируемых билетов произвольно и может быть велико (> 20’000’000 шт.). Учтите ситуацию, что не все сгенерированные билеты могут участвовать в тираже (это типичная ситуация, которая возникает при неполной реализации билетов к тиражу). Смоделируйте проведение розыгрыша: на каждом ходе проверяйте, появился ли победитель; предусмотрите систему выигрышей; предоставьте возможность поиска в интерактивном диалоге всех билетов по заданным критериям: номеру билета, величине выигрыша, и т. д. Сохраняйте информацию о проведенных тиражах для обеспечения поиска данных в будущем. Реализуйте функционал обработки данных таким образом, чтобы тип коллекции/адаптера, в котором размещаются Ваши данные, являлся параметром шаблона. Продемонстрируйте обработку данных с подстановкой в качестве параметра шаблона собственных реализаций стека и очереди на базе односвязного списка.