Задания к работе №3 по Фундаментальным алгоритмам.

Все задания реализуются на языке программирования *C* (стандарт *C99* 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 при помощи возврата целевых результатов функции через параметры функции и возврата из функции значения (либо типа *int*, либо перечислимого типа (*enum*)), репрезентирующего статус-код функции, в целях обработки последнего в вызывающем коде через оператор *switch/case* либо через управляющие конструкции языка *if / else if / else*. Возвращаемые статус-коды функций необходимо продумать самостоятельно так, чтобы были покрыты всевозможные ошибки времени выполнения функций.

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*).

1. Реализуйте функцию перевода числа из десятичной системы счисления в систему счисления с основанием 2^r, $r = 1, ..., 5$. При реализации функции разрешается использовать битовые операции и операции обращения к памяти, запрещается использовать стандартные арифметические операции. Продемонстрируйте работу реализованной функции.

2. Напишите функцию с переменным числом аргументов, на вход которой передаются: размерность пространства n; экземпляры структур, содержащие в себе координаты векторов из n-мерного пространства; указатели на функции, вычисляющие нормы векторов. Ваша функция должна для каждой переданной нормы вернуть самый длинный переданный вектор (если таковых векторов несколько, необходимо вернуть их все).
   Замечание. Реализуйте возможность вычислять следующие нормы:
   $$\|x\|_\infty = max_j|x_j|,$$
   $$\|x\|_p = \left(\sum_{j=1}^{n}|x_j|^p\right)^{\frac{1}{p}},\ p \geq 1,$$
   $$\|x\|_A = \sqrt{(Ax,\ x)},$$
   где A - положительно определенная матрица размерности $n \times n$. Передачу функций реализуйте с помощью универсального типа указателя на функцию. Продемонстрируйте работу реализованной функции.

3. На вход программе через аргументы командной строки подается путь ко входному файлу, флаг (флаг начинается с символа ‘-’ или ‘/’, второй символ - ‘a’ или ‘d’) и путь к выходному файлу. В файле в каждой строчке содержится информация о сотруднике (для этой информации определите тип структуры *Employee*): id (целое неотрицательное число), имя (непустая строка только из букв латинского алфавита), фамилия (непустая строка только из букв латинского алфавита), заработная плата (неотрицательное вещественное число). Программа должна считать записи из файла в динамический массив структур и в выходной файл вывести данные, отсортированные (с флагом ‘-a’/‘/a’ - по возрастанию, с флагом ‘-d’/‘/d’ - по убыванию) первично - по зарплате, далее (если зарплаты равны) - по фамилии, далее (если зарплаты и фамилии равны) - по именам, наконец, по id. Для сортировки коллекции экземпляров структур используйте стандартную функцию *qsort*, своя реализация каких-либо алгоритмов сортировки не допускается.

4. **1.** Опишите тип структуры *String*, содержащую в себе поля для указателя на динамический массив символов типа *char* и количества символов (длины строки) типа *int*. Для описанного типа структуры реализуйте функции:
    - создания экземпляра типа *String* на основе значения типа *char **
    - удаления внутреннего содержимого экземпляра типа *String*
    - отношения порядка между двумя экземплярами типа *String* (первично по длине строки, вторично по лексографическому компаратору)
    - отношения эквивалентности между двумя экземплярами типа *String* (лексографический компаратор)
    - копирования содержимого экземпляра типа *String* в существующий экземпляр типа *String*
    - копирования содержимого экземпляра типа *String* в новый экземпляр типа *String*, размещённый в динамической памяти
    - конкатенации к содержимому первого экземпляра типа *String* содержимого второго экземпляра типа *String*.

    Продемонстрируйте работу реализованного функционала.

    2. Экземпляр структуры *Mail* содержит в себе экземпляр структуры *Address* получателя (город (непустая строка), улица (непустая строка), номер дома (натуральное число), корпус (строка), номер квартиры (натуральное число), индекс получателя (строка из шести символов цифр)), вес посылки (неотрицательное вещественное число), почтовый идентификатор (строка из 14 символов цифр), время создания (строка в формате “dd:MM:yyyy hh:mm:ss”), время вручения (строка в формате “dd:MM:yyyy hh:mm:ss”). Экземпляр структуры *Post* содержит указатель на экземпляр структуры *Address* текущего почтового отделения и динамический массив экземпляров структур типа *Mail*. Реализуйте интерактивный диалог с пользователем, предоставляющий функционал для добавления и удаления объектов структур типа *Mail* в объект структуры типа *Post*, информативный вывод данных об отправлении при поиске объекта типа *Mail* по идентификатору. Объекты структуры *Mail* должны быть отсортированы по индексу получателя (первично) и идентификатору посылки (вторично) в произвольный момент времени. Также в интерактивном диалоге реализуйте опции поиска всех доставленных отправлений, а также всех отправлений, срок доставки которых на текущий момент времени (системное время) истёк. Информацию о доставленных/недоставленных отправлениях выводите в порядке времени создания по возрастанию (от старых к новым). Для хранения строковых данных используйте структуру *String* из п. 1.

5. Экземпляр структуры типа *Student* содержит поля: id студента (целое неотрицательное число), имя (непустая строка только из букв латинского алфавита), фамилия (непустая строка только из букв латинского алфавита), группа (непустая строка) и оценки за 5 экзаменов (динамический массив элементов типа *unsigned char*). Через аргументы командной строки программе на вход подаётся путь к файлу, содержащему записи о студентах. При старте программа считывает поданный файл в динамический массив структур типа *Student*. В программе должен быть реализован поиск всех студентов по:
   - id;
   - фамилии;
   - имени;
   - группе,

   сортировка (для сортировки необходимо передавать компаратор для объектов структур) студента(-ов) по:
   - id;
   - фамилии;
   - имени;
   - группе.

   Добавьте возможность вывода в трассировочный файл (путь к файлу передаётся как аргумент командной строки) данные найденного по id студента: ФИО, группу и среднюю оценку за экзамены. Также добавьте возможность вывести в трассировочный файл фамилии и имена студентов, чей средний балл за все экзамены выше среднего балла за все экзамены по всем считанным из файла студентам. Все вышеописанные опции должны быть выполнимы из контекста интерактивного диалога с пользователем. Для сортировки коллекции экземпляров структур используйте стандартную функцию *qsort*, своя реализация каких-либо алгоритмов сортировки не допускается.

6. В некотором уездном городе ввели систему учета городского транспорта. Каждый остановочный пункт регистрирует каждую единицу останавливающегося общественного транспорта, то есть в текстовый файл записывается номер (непустая строка) транспортного средства, время остановки (строка в формате “dd.MM.yyyy hh:mm:ss”), время отправления (строка в формате “dd.MM.yyyy hh:mm:ss”) и маркер, указывающий является ли данная остановка промежуточной, начальной или конечной для данного транспортного средства. Каждый файл содержит координаты остановочного пункта, на котором этот файл был сгенерирован. Необходимо разработать приложение для обработки данных с остановочных пунктов. Реализуйте на базе односвязного списка односвязных списков хранилище информации. Элементами основного списка являются списки пройденных маршрутов от начальной до конечной точки со всеми промежуточными точками для каждого транспортного средства. Элементы в списке пройденных остановок необходимо упорядочить по возрастанию временных меток. Входными аргументами вашего приложения является массив путей к файлам (с каждого остановочного пункта), при этом файлы подаются в произвольном порядке, записи в файле также не упорядочены не по какому-либо критерию, но гарантия целостности записей поддерживается. Для вашего хранилища маршрутов посредством интерактивного диалога реализуйте поиск транспортного средства,

которое проехало больше/меньше всех маршрутов, поиск транспортного средства, которое проехало самый длинный/короткий путь, поиск транспортного средства с самым длинным/коротким маршрутом и транспортное средство с самой долгой/короткой остановкой на остановочном пункте и транспортное средство с самым большим временем простоя (стоянки на своих остановках).

7. В текстовом файле находится информация о жителях (тип структуры *Liver*) некоторого поселения: фамилия (непустая строка только из букв латинского алфавита), имя(непустая строка только из букв латинского алфавита), отчество (строка только из букв латинского алфавита; допускается пустая строка), дата рождения (в формате число, месяц, год), пол (символ 'M' - мужской символ 'W' - женский), средний доход за месяц (неотрицательное вещественное число). Напишите программу, которая считывает эту информацию из файла в односвязный упорядоченный список (в порядке увеличения возраста). Информация о каждом жителе должна храниться в объекте структуры *Liver*. Реализуйте возможности поиска жителя с заданными параметрами, изменение существующего жителя списка, удаления/добавления информации о жителях и возможность выгрузки данных из списка в файл (путь к файлу запрашивайте у пользователя с консоли). Добавьте возможность отменить последние $N/2$ введённых модификаций, то есть аналог команды Undo; N - общее количество модификаций на текущий момент времени с момента чтения файла/последней отмены введённых модификаций.

8. На основе односвязного списка реализуйте тип многочлена от одной переменной (коэффициенты многочлена являются целыми числами). Реализуйте функции, репрезентирующие операции сложения многочленов, вычитания многочлена из многочлена, умножения многочленов, целочисленного деления многочлена на многочлен, поиска остатка от деления многочлен на многочлен, вычисления многочлена в заданной точке, нахождения производной многочлена, композиции многочленов.
   Для демонстрации работы вашей программы реализуйте возможность обработки текстового файла (с чувствительностью к регистру) следующего вида:

   Add(2x^2-x+2,-x^2+3x-1); % сложить заданные многочлены;

   Div(x^5,x^2-1); % разделить нацело заданные многочлены.

   Возможные инструкции в файле: однострочный (начинается с символа '%') комментарий, многострочный (начинается с символа '[', заканчивается символом ']', вложенность запрещена) комментарий; Add – сложение многочленов, Sub - вычитание многочлена из многочлена, Mult – умножение многочленов, Div – целочисленное деление многочлена на многочлен, Mod – остаток от деления многочлена на многочлен, Eval – вычисление многочлена в заданной точке, Diff – дифференцирование многочлена, Cmps – композиция двух многочленов.

   Если в инструкции файла один параметр, то это означает, что вместо первого параметра используется текущее значение сумматора. Например:

Mult(x^2+3x-1,2x+x^3); % умножить два многочлена друг на друга результат сохранить в сумматор и вывести на экран;

Add(4x-8); % в качестве первого аргумента будет взято значение из сумматора, результат будет занесен в сумматор;

Eval(1); % необходимо будет взять многочлен из сумматора и для него вычислить значение в 1.

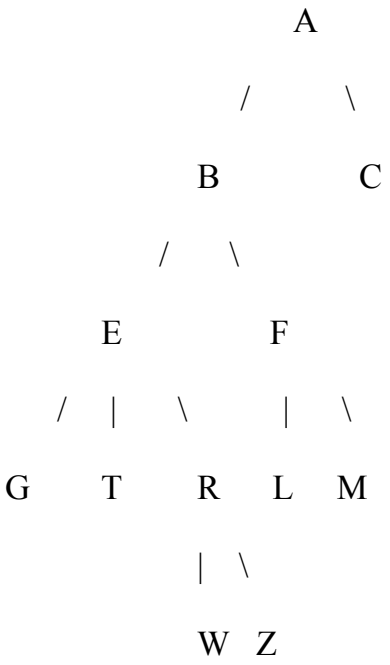
Значение сумматора в момент начала выполнения программы равно 0.

9. **A)** Реализуйте приложение для сбора статистических данных по заданному тексту. Результатом работы программы является информация о том, сколько раз каждое слово из файла встречается в данном файле (путь к файлу - первый аргумент командной строки). Слова в файле разделяются сепараторами (каждый сепаратор - символ), которые подаются как второй и последующие аргументы командной строки. В интерактивном диалоге с пользователем реализуйте выполнение дополнительных опций: вывод информации о том сколько раз заданное слово встречалось в файле (ввод слова реализуйте с консоли); вывод первых *n* наиболее часто встречающихся слов в файле (значение *n* вводится с консоли); поиск и вывод в контексте вызывающего кода в консоль самого длинного и самого короткого слова (если таковых несколько, необходимо вывести в консоль любое из них). Размещение информации о считанных словах реализуйте посредством двоичного дерева поиска.

   B) Для построенного дерева в пункте A реализуйте и продемонстрируйте работу функции поиска глубины данного дерева.

   C) Для построенного дерева в пункте A реализуйте функции сохранения построенного дерева в файл и восстановления бинарного дерева из файла. При этом восстановленное дерево должно иметь точно такую же структуру и вид, как и до сохранения. Продемонстрируйте работу реализованных функций.

10. Реализуйте приложение для построения дерева скобочного выражения. На вход программе через аргументы командной строки подается путь к файлу, в котором содержится произвольное число строк. Каждая строка является корректным скобочным выражением. Ваша программа должна обработать каждое скобочное выражение из файла и вывести в текстовый файл (путь к файлу - аргумент командной строки) деревья скобочных записей в наглядной форме (форму вывода определите самостоятельно). Возникающие при работе программы деревья не обязаны быть бинарными. Например, запись A (B (E (G, T, R (W, Z)), F (L, M)), C) соответствует дереву



Формат вывода не фиксирован и определяется удобством восприятия.