Задания к работе №4 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 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 приложение для организации макрозамен в тексте. На вход приложению через аргументы командной строки подаётся путь к текстовому файлу, содержащему в начале набор директив #define, а далее обычный текст. Синтаксис директивы соответствует стандарту языка C:

   #define <def_name> <value>

   Аргументов у директивы нет, директива не может быть встроена в другую директиву. Ваше приложение должно обработать содержимое текстового файла, выполнив замены последовательностей символов <def_name> на <value>. Количество директив произвольно, некорректных директив нет, объём текста во входном файле произволен. В имени <def_name> допускается использование символов латинского алфавита (прописные и строчные буквы не отождествляются) и символов арабских цифр; значение <value> произвольно и завершается символом переноса строки или символом конца файла. Для хранения имен макросов и макроподстановок используйте хеш-таблицу размера HASHSIZE (начальное значение равно 128). Для вычисления хеш-функции интерпретируйте <def_name> как число, записанное в системе счисления с основанием 62 (алфавит этой системы счисления состоит из символов {0, …,9, A, …, Z, a, …, z}). Хеш-значение для <def_name> в рамках хеш-таблицы вычисляйте как остаток от деления эквивалентного для <def_name> числа в системе счисления с основанием 10 на значение HASHSIZE. Для разрешения коллизий используйте метод цепочек. В ситуациях, когда после модификации таблицы длины самой короткой и самой длинной цепочек в хеш-таблице различаются в 2 раза и более, пересобирайте хеш-таблицу с использованием другого значения HASHSIZE (логику модификации значения HASHSIZE продумайте самостоятельно) до достижения примерно равномерного распределения объектов структур по таблице. Оптимизируйте расчёт хэш-значений при пересборке таблицы при помощи кэширования.

2. Реализуйте приложение – интерпретатор операций над целочисленными массивами. Приложение оперирует целочисленными массивами произвольной длины с именами из множества {A, B, …, Z}. Система команд данного интерпретатора (прописные и строчные буквы отождествляются):
    - Load A, in.txt; - загрузить в массив A целые числа из файла in.txt (во входном файле могут произвольно присутствовать сепарирующие символы - пробелы, табуляции и переносы строк; также могут быть невалидные строковые репрезентации элементов массива);
    - Save A, out.txt; - выгрузить элементы массива A в файл out.txt;
    - Rand A, count, lb, rb; - заполнить массив A псевдослучайными элементами из отрезка $[lb; lr]$ в количестве count штук.
    - Concat A, b; - сконкатенировать два массива A и B результат сохранить в массив A;
    - Free(a); - очистить массив A и сопоставить переменную A с массивом из 0 элементов;
    - Remove a, 2, 7; - удалить из массива a 7 элементов, начиная с элемента с индексом 2;
    - Copy A, 4, 10, b; - скопировать из массива A элементы с 4 по 10 (оба конца включительно) и сохранить их в b;
    - Sort A+; - отсортировать элементы массива A по неубыванию;
    - Sort A-; - отсортировать элементы массива A по невозрастанию;
    - Shuffle A; - переставить элементы массива в псевдослучайном порядке;
    - Stats a; - вывести в стандартный поток вывода статистическую информацию о массиве A: размер массива, максимальный и минимальный элемент (и их индексы), наиболее часто встречающийся элемент (если таковых несколько, вывести максимальный из них по значению), среднее значение элементов, максимальное из значений отклонений элементов от среднего значения;
    - Print a, 3; - вывести в стандартный поток вывода элемент массива A стоящий на позиции с номером 3;
    - Print a, 4, 16; - вывести в стандартный поток вывода элементы массива A, начиная с 4 по 16 включительно оба конца;
    - Print a, all; - вывести в стандартный поток вывод все элементы массива A.

    Индексирование в массивах начинается с 0. Для сортировки массивов и реализации инструкции Shuffle используйте стандартную функцию qsort, свои реализации алгоритмов сортировки не допускаются. Предоставьте текстовый файл с инструкциями для реализованного интерпретатора и продемонстрируйте его работу. Обработайте ошибки времени выполнения инструкций.

3*. Реализуйте пользовательский тип данных полинома от нескольких переменных. Полином должен представлять собой контейнер мономов, основанный на структуре данных вида двусвязный список. Тип монома от нескольких переменных содержит вещественное значение коэффициента и ассоциативный контейнер пар значений “имя переменной - степень” вида бор. Степени переменных являются целыми неотрицательными числами; хранение в ассоциативном контейнере переменных, степень которых равна 0, не допускается в произвольный момент времени. Например, строковое представление монома “<5.114_x^2_x1^3_y12^3_y333^4_-2.1_z_x1^-1_-1>” соответствует значению $10.7394x_1^2y_{12}^3y_{333}^4z$ (формат строкового представления продумайте самостоятельно, при этом коэффициент монома равный 1 и степени переменных, равные 0 и 1, могут быть опущены в строковом представлении; допускаются повторяющиеся имена переменных в рамках строкового представления монома; допускаются повторяющиеся значения коэффициентов). Имена переменных имеют произвольную длину (не менее 1), могут состоять из символов букв (прописные и строчные буквы отождествляются) и символов арабских цифр, при этом могут начинаться только с символа буквы.

Реализуйте функционал для Ваших полиномов: сложение, вычитание, умножение полиномов; вычисление значения полинома в заданной точке конечномерного пространства (значения координат точки могут быть вещественными числами); нахождение частной производной многочлена по соответствующей переменной (имя переменной должно являться параметром), нахождения значения градиента для полинома, нахождения частной первообразной полинома (с точностью до константы). Для демонстрации работы с вашим типом разработайте приложение для обработки регистрозависимых команд следующего вида:

Add(4*x^2-x_y^2+4*x_y+1,[где лабы] x^2_y^3+2*x_y^2+_y^2-4*x_y); % сложить многочлены, сохранить результат в сумматор
Mult(4*x^2_y^3+4xy+1, [x^2_[я комментарий]]y^3+2*x_y^2+3*x^2_y^2-4*x_y+5); % умножить многочлены, сохранить результат в сумматор

Допустимые инструкции: однострочный (начинается с символа ‘%’) комментарий, многострочный (начинается с символа ‘[‘, заканчивается символом ‘]’, вложенность произвольна) комментарий; Add – сложение многочленов, Sub - вычитание многочлена из многочлена, Mult – умножение многочленов, Eval – вычисление многочлена в заданной точке конечномерного пространства (на вход подаётся строковое представление набора пар вида “имя переменной - значение переменной”), PartDeriv – частная производная многочлена, Cmps – композиция двух многочленов. Разделителем параметров в инструкциях (кроме комментариев) является символ ‘,’.

Если в инструкции отсутствует первый параметр, то это означает, что вместо него используется текущее значение сумматора. Например:

Mult(x^2+3_x-1,2x+x^3); % умножить два полинома друг на друга, результат умножения сохранить в сумматор;
Add(4_x_1.2_y^2-8_z); % в качестве первого аргумента операции сложения будет взято

значение из сумматора, результат будет занесен в сумматор;
Eval({x:-1.11,y:2.1,z:0.81}); % необходимо взять полином из сумматора и для него вычислить значение в заданной точке конечномерного пространства.

Результаты всех операций над полиномами должны сохраняться в сумматор. В начале обработки входного потока данных значение сумматора равно нулю.
Входным потоком данных является файл, путь к которому является аргументом командной строки. В результате работы приложения необходимо вывести на консоль промежуточные значения сумматора после выполнения каждой инструкции из входного файла. Предоставьте текстовый файл с инструкциями для реализованного интерпретатора и продемонстрируйте его работу. Обработайте ошибки времени выполнения инструкций.

4*. На вход приложению через аргументы командной строки подаётся путь к текстовому файлу, содержащему последовательность строк, в каждой из которых записаны один или несколько операторов над булевскими векторами с односимвольными именами из множества {A, B, …, Z}, а также однострочные и многострочные комментарии. Символом однострочного комментария является символ ‘%’, а символами многострочного – символы ‘{‘ и ‘}’ (вложенность не допускается). Возможный вид операторов (операция “:=” означает присваивание переменной слева результата вычисления выражения справа):

1) A := B <op> C; <op> - бинарная поразрядная операция из списка:
   - + (дизъюнкция);
   - & (конъюнкция);
   - -> (импликация);
   - <- (обратная импликация);
   - ~ (эквиваленция);
   - <> (сложение по модулю 2);
   - +> (коимпликация);
   - ? (штрих Шеффера);
   - ! (стрелка Пирса);
2) X := \W; - поразрядное отрицание;
3) read(D, base); - ввод значения в переменную D, входная строка репрезентирует число в системе счисления с основанием base (в диапазоне [2..36]);
4) write(Q, base); - вывод значения из переменной Q в виде числа в системе счисления с основанием base (в диапазоне [2..36]).

Разделителем между операторами является символ ‘;’. Сепарирующие символы (пробелы, символы табуляций и символы переносов строк) могут присутствовать произвольно, различий между прописными и строчными буквами нет, вложенные комментарии допускаются, уровень вложенности произвольный.

При наличии в командной строке флага “/trace” , следующим после него аргументом указывается путь к файлу трассировки, необходимо выводить в файл трассировки подробную информацию о выполнении каждой инструкции из файла: значения переменных до и после выполнения инструкции, информацию о произошедших ошибках времени выполнения, о синтаксических и семантических ошибках в инструкциях; при отсутствии флага аналогичный вывод выполняется в стандартный поток вывода. Предоставьте текстовый файл с инструкциями для реализованного интерпретатора и продемонстрируйте его работу. Обработайте ошибки времени выполнения инструкций.

5*. На вход приложению через аргументы командной строки подаются пути к текстовым файлам, содержащим выражения (в каждой строке находится одно выражение), а также флаг (--table или --calculate).

1. Флаг --calculate, программа вычисления выражений. Выражения в файлах могут быть произвольной структуры: содержать произвольное количество арифметических операций (сложение, вычитание, умножение, целочисленное деление, взятие остатка от деления, возведение в целую неотрицательную степень), круглых скобок (задают приоритет вычисления подвыражений).

   В вашем приложении необходимо для каждого выражения из каждого входного файла:
   - проверить баланс скобок для каждого выражения;
   - построить дерево арифметического для каждого выражения;
   - построить обратную польскую запись для каждого выражения;
   - вычислить значение выражения с использованием алгоритма вычисления выражения, записанного в обратной польской записи.

   В результате работы приложения для каждого входного файла в стандартный поток вывода необходимо вывести путь к файлу и список выражений из него, для каждого выражения вывести: исходное выражение, значение выражения при заданных (через стандартный поток ввода) значениях переменных и файлы сгенерированных инструкций.

   В случае обнаружения ошибки в расстановке скобок либо невозможности вычислить значение выражения, для каждого файла, где обнаружены вышеописанные ситуации, необходимо создать текстовый файл, в который выписать для каждого ошибочного выражения из исходного файла: само выражение, его порядковый номер в файле (индексация с 0) и причину невозможности вычисления.

   Для решения задачи используйте собственную реализацию структуры данных вида стек на базе структуры данных вида односвязный список.
   Предоставьте текстовый файл с выражениями для реализованного приложения и продемонстрируйте его работу. Обработайте ошибки времени выполнения инструкций.

2. Флаг --table — приложение, которое по заданной булевой формуле строит таблицу истинности.

   Каждый файл содержит произвольное количество строк, в которых записаны булевы формулы.

В этой формуле могут присутствовать:

- односимвольные имена логических переменных;
- константы 0 (ложь) и 1 (истина);
- & - оператор логической конъюнкции;
- | - оператор логической дизъюнкции;
- ~ - оператор логической инверсии;
- -> - оператор логической импликации;
- +> - оператор логической коимпликации;
- <> - оператор логического сложения по модулю 2;
- = - оператор логической эквиваленции;
- ! - оператор логического штриха Шеффера;
- ? - оператор логической функции Вебба;
- операторы круглых скобок, задающие приоритет вычисления подвыражений.

Вложенность скобок произвольна. Для вычисления булевой формулы постройте бинарное дерево выражения и вычисление значения булевой формулы на конкретном наборе переменных выполняйте с помощью этого дерева. Приоритеты операторов (от высшего к низшему): 3(~), 2(?,!,+>,&), 1(|, ->, <>, =).

В результате работы приложения необходимо получить выходной файл (имя файла псевдослучайно составляется из букв латинского алфавита и символов арабских цифр, файл должен быть размещён в одном каталоге с исходным файлом), в котором для каждого выражения должна быть построена своя таблица истинности, заданной входной булевой формулой.

6*. Реализуйте приложение, выполняющее моделирование почтового сервиса.

Программа моделирует работу почтового сервиса, который включает взаимодействие между почтовыми отделениями и письмами. В каждом отделении может храниться ограниченное количество писем любого состояния.

На вход программе вторым аргументом командной строки подаётся путь к файлу маппингов (связей) между почтовыми отделениями в виде пар: id первого отделения, id второго отделения.

Система должна поддерживать следующие команды:

- Добавление почтового отделения: Команда добавляет новое почтовое отделение с уникальным ID, максимальной вместимостью писем n, и списком id отделений, имеющих связь с новым отделением. В отделении может храниться не более n писем одновременно.
- Удаление почтового отделения: Команда удаляет почтовое отделение по его ID. Письма, не связанные с этим отделением, отправляются в другие отделения. Письма, связанные с удаляемым отделением, помечаются как "Не доставлено" и удаляются из системы.
- Добавление письма: Команда добавляет письмо с полями тип, приоритет, id отделения отправителя, id отделения получателя, а также строкой технических данных. Каждому письму присваивается уникальный ID, который генерируется с помощью static переменной, чтобы обеспечить последовательное увеличение значения ID.
- Пометить письмо как недоставленное
- Попытка взять письмо: Команда позволяет попытаться забрать письмо по его ID в месте назначения. Если письмо успешно доставлено, оно удаляется из системы.
- Получение списка всех писем: Команда выводит список всех писем в том числе доставленных и недоставленных в указанный выходной файл.
- Выход из программы.

Работа с письмами: Каждые 0.2 секунды самое приоритетное письмо в каждом отделении передаётся в одно из связанных с ним отделений, в направлении к месту назначения. Для хранения писем внутри отделений необходимо использовать структуру данных **косую приоритетную очередь**. Письма с более высоким приоритетом обрабатываются раньше. Письмо передаётся от одного отделения к другому, пока не достигнет пункта назначения или не будет помечено как недоставленное.

Свойства писем: Каждое письмо имеет следующие свойства: ID (уникальный идентификатор, задаётся автоматически с помощью статической переменной), Тип (например, обычное, срочное), Состояние (доставлено, не доставлено, в процессе

отправки) Приоритет (целое число, чем выше, тем важнее письмо), ID отделения отправителя, ID отделения получателя, Технические данные (произвольная строка).

Если письмо не может быть доставлено в конечное отделение по причине недоступности или полной загруженности получателя, письмо отправляется в ближайшее доступное отделение, и система сохраняет такую "петлю" перенаправления до тех пор, пока письмо не станет возможным доставить в исходное место назначения.

Выходные данные: реализуйте пользовательский интерфейс в командной строке «для домохозяек», по взаимодействию с вашей программой, а также обеспечьте логирование в файл передвижение всех писем и изменения состояний писем и отделений. (название генерируется либо псевдослучайно, либо передаётся третьим аргументом командной строки).

7*. Опишите тип структуры *MemoryCell*, содержащей имя переменной и её целочисленное значение.
Через аргументы командной строки в программу подается файл с инструкциями вида

```
myvar=15;
bg=25;
ccc=bg+11;
print ccc;
myvar=ccc;
bg=ccc*myvar;
print;
```

Файл не содержит ошибок и все инструкции корректны. В рамках приложения реализуйте чтение данных из файла и выполнение всех простых арифметических операций (сложение, вычитание, умножение, целочисленное деление, взятие остатка от деления), инициализации переменной и присваивания значения переменной (=) и операции print (вывод в стандартный поток вывода либо значения переменной, имя которой является параметром операции print, либо значений всех объявленных на текущий момент выполнения переменных с указанием их имён). В каждой инструкции может присутствовать только одна из вышеописанных операций; аргументами инструкций могут выступать значение переменной, подаваемое в виде имени этой переменной, а также целочисленные константы, записанные в системе счисления с основанием 10. При инициализации переменной по необходимости требуется довыделить память в динамическом массиве структур типа *MemoryCell*; инициализация переменной (по отношению к операции перевыделения памяти) должна выполняться за амортизированную константу. Для поиска переменной в массиве используйте алгоритм дихотомического поиска, для этого ваш массив в произвольный момент времени должен находиться в отсортированном состоянии по ключу имени переменной; для сортировки массива используйте стандартную функцию qsort, реализовывать непосредственно какие-либо алгоритмы сортировки запрещается. Имя переменной может иметь произвольную длину и содержать только символы латинских букв, прописные и строчные буквы при этом не отождествляются. В случае использования в вычислениях не объявленной переменной необходимо остановить работу интерпретатора и вывести сообщение об ошибке в стандартный поток вывода. Предоставьте текстовый файл с инструкциями для реализованного интерпретатора и продемонстрируйте его работу. Обработайте ошибки времени выполнения инструкций.

8*. Необходимо смоделировать систему управления процессами в реальном времени с учетом приоритетов и времени ожидания. Каждый процесс обладает <pid> - уникальным идентификатором (целое число), обладает <name> - именем процесса (строка), <time> – временем необходимым на обработку (целое число) и <priority> приоритетом на исполнение (целое число). Также у процесса могут быть дочерние процессы, в таком случае родительский процесс не может быть завершён пока не завершены все дочерние. Ваша система управления должна выполнять максимально приоритетные процессы в текущий момент времени, и каждый процесс в сумме должен отработать <time> времени. Процессы могут быть приостановлены в случае появления нового процесса с более высоким приоритетом или изменения приоритета существующего процесса. В случае равенства приоритетов, отдать предпочтение первому добавленному процессу.

Для организации хранения процессов используйте структуру данных **биномиальную кучу**. Для организации хранения строк используйте структуру данных **бор**. За единицу времени исполнения для моделирования системы возьмите 0.2 секунды.

Реализуйте обработку следующих команд (из стандартного потока, либо из файла, переданного аргументом командной строки с флагом --input <path>):

- !np <name> <pid> <priority> <time> — добавляет новый процесс в очередь на исполнение.
- !pp <parent pid> <name> <pid> <priority> <time> — добавляет новый дочерний процесс в очередь на исполнение.
- !wait <pid> <time> — запрашивает приостановку текущего исполняемого процесса на указанное количество времени time. Процесс идентифицируется по pid. В это время процесс находится в состоянии ожидания.
- !list — Вывод информации о активных процессах.
- !all — Вывод информации о всех процессах.
- !exec <count> — Изменение числа активных процессов
- !edit <pid> <new priority> — изменяет приоритет процесса с идентификатором pid. Если измененный процесс имеет более высокий приоритет по сравнению с текущими исполняемыми процессами, система должна приостановить текущие процессы и начать исполнять процесс с более высоким приоритетом.

Также система должна выводить информацию о следующих событиях (в стандартный поток либо в файл, переданный аргументом командной строки с флагом –output <path>):

- Добавление процесса в очередь
- Начало исполнения процесса
- Остановка процесса
- Завершение процесса

9**. 1. Разработайте приложение, моделирующее работу центра клиентской поддержки. На вход приложению через аргументы командной строки вторым аргументом подаётся натуральное число *max_priority* и пути к файлам. В файле, расположенном по пути, передаваемом в качестве второго аргумента командной строки, содержится информация о параметрах модели; построчно:

- вид структуры данных, предназначенной для хранения заявок конкретным отделением; допускаются следующие варианты (структуры данных соответствовать интерфейсу приоритетной очереди и поддерживать операции добавления данных по ключу, удаления данных с максимальным приоритетом по ключу, поиска данных с максимальным приоритетом по ключу, слиянием двух приоритетных очередей без разрушения исходных, слиянием двух приоритетных очередей с разрушением исходных):
  - BinaryHeap - двоичная приоритетная очередь;
  - BinomialHeap - биномиальная приоритетная очередь;
  - FibonacciHeap - фибоначчиева приоритетная очередь;
  - SkewHeap - косая приоритетная очередь;
  - LeftistHeap - левосторонняя приоритетная очередь;
  - Treap - декартово дерево;
- вид структуры данных, предназначенной для хранения по ключу идентификатора отделения значений типа указатель на объект приоритетной очереди, хранящей заявки конкретного отделения; допускаются следующие варианты (для структур данных предусмотрите единый интерфейс взаимодействия):
  - HashSet - хеш-таблица с методом цепочек разрешения коллизий;
  - DynamicArray - динамический массив (отсортированный по ключу) с дихотомическим поиском;
  - BinarySearchTree - двоичное дерево поиска;
  - Trie - бор;
- дата и время начала моделирования (в формате ISO 8601);
- дата и время конца моделирования (в формате ISO 8601);
- минимальное время обработки заявки оператором (в минутах, натуральное число);
- максимальное время обработки заявки оператором (в минутах, натуральное число);
- количество отделений (натуральное число в диапазоне [1..100]);
- для каждого отделения: количество операторов, обрабатывающих заявки (натуральное число в диапазоне [10..50]); числа разделены символом пробела;
- вещественное число *overload_coeff* (1.0 и выше), являющееся коэффициентом перегрузки отделения (отделение становится перегруженным, если число сообщений в очереди превышает число операторов отделения в *overload_coeff* раз и более).

В файлах, расположенных по остальным переданным в качестве аргументов командной строки путям, содержится информация о заявках пользователей. Формат строки с информацией о заявке следующий:

<время поступления заявки> <приоритет> <идентификатор отделения> “<текст заявки>”

Приоритет может иметь значение в диапазоне [0..*max_priority*], от низшего к высшему.
Поступающие отделению заявки обрабатываются в порядке согласно приоритету заявки (первичный приоритет) и времени приходи заявки в систему (вторичный приоритет). Любой свободный оператор отделения извлекает очередную заявку и работает с ней, после обработки заявки он работает уже с другой заявкой и так далее.
В случае если отделение становится перегруженным, по возможности необходимо передать все стоящие в очереди заявки этого отделения наименее загруженному обработкой заявок отделению (используйте операцию *meld* из реализации приоритетной очереди).
Время в системе дискретно с шагом в 1 минуту.
В результате моделирования необходимо получить лог-файл с информацией о процессе и результатах моделирования, а также о произошедших ошибках. Формат лога:

[*<время возникновения ситуации в рамках модели в формате ISO 8601>*] [*<код ситуации>*]: *<описание>*

Логгируйте следующие ситуации:

Код ситуации	Что необходимо описать
NEW_REQUEST	Поступление заявки в систему; Идентификатор заявки и идентификатор отделения, куда поступила заявка на обработку
REQUEST_HANDLING_STARTED	Начало обработки заявки; идентификатор заявки и имя оператора, который начал заниматься её обработкой
REQUEST_HANDLING_FINISHED	Окончание обработки заявки; идентификатор заявки, время её обработки (в минутах) и имя оператора, который занимается её обработал
DEPARTMENT_OVERLOADED	Перегрузка отделения; идентификатор заявки, после поступления которой произошла перегрузка, идентификатор отделения, которому были переданы все заявки из очереди (если таковое имеется)

Имена операторов должны быть уникальны в рамках каждого отдела и должны быть сгенерированы псевдослучайно (алфавит состоит из прописных и строчных букв латинского алфавита).
Предоставьте текстовый файл с выражениями для реализованного приложения и продемонстрируйте его работу. Обработайте ошибки времени выполнения инструкций.

2. Реализуйте вспомогательное приложение для задачи 9.1 для генерации файла с информацией о параметрах модели, с настройкой параметров посредством интерактивного диалога с пользователем.
3. Реализуйте вспомогательное приложение для задачи 9.1, генерирующее файлы со входными данными, без интерактивного диалога с пользователем (количество данных, объёмы данных и сами данные псевдослучайны).

10**. Реализуйте приложение-интерпретатор с настраиваемым синтаксисом и возможностями отладки программ, написанных для реализованного интерпретатора.

Для настройки интерпретатора приложению через аргументы командной строки подается файл с описанием имён инструкций и их синтаксиса. Файл настроек содержит сопоставления операций, которые может выполнить интерпретатор, и их псевдонимов, которые будут использованы в программах, которые будут поданы на вход. Файл настроек может содержать однострочные комментарии, которые начинаются с символа #.

Интерпретатор оперирует 32-х разрядными целочисленными беззнаковыми переменными, имена которых могут содержать один и более символов (в качестве символов, входящих в имена переменных, допускаются символы букв латинского алфавита, арабских цифр и подчерка (‘_’); имя переменной не может начинаться с символа цифры; длина имени переменной произвольна; прописные и строчные буквы не отождествляются).

Команды, которые могут быть выполнены интерпретатором:

- Унарные:
  - not - поразрядная инверсия;
  - input -ввод значения из стандартного потока ввода в системе счисления с основанием base_input;
  - output -вывод значения переменной в стандартный поток вывода в системе счисления с основанием base_output;
- Бинарные:
  - add - сложение;
  - mult - умножение;
  - sub - вычитание;
  - pow - возведение в целую неотрицательную степень по модулю 2^{32} алгоритмом быстрого возведения в степень по модулю;
  - div - целочисленное деление;
  - rem - взятие остатка от деления;
  - xor - поразрядное сложение по модулю 2;
  - and - поразрядная конъюнкция;
  - or - поразрядная дизъюнкция;
  - = - присваивание значения переменной или её инициализация значением выражения или константой в системе счисления с основанием base_assign.

В рамках инструкции, обрабатываемой интерпретатором, допускается выполнение нескольких команд. Пример файла (при base_assign = 16):

```
var_1 = 1F4;
var_2 = mult(var_2, 4);
Var3 = add(div(var_2, 5) , rem(var_1, 2));
print(Var3);
```

Для каждой из вышеописанных команд можно задать синоним в файле настроек интерпретатора. Для этого на отдельной строке файла необходимо сначала указать

оригинальное название команды, далее через пробел - синоним. Если одна и та же команда в файле настроек заменяется синонимом несколько раз, в результате должен быть применён только последний встреченный синоним; при задании нескольких синонимов для одной и той же команды в одном файле настроек оригинальное написание команды не сохраняется на уровне файла настроек. Если для команды не задаётся синоним, она сохраняет своё оригинальное написание.

Помимо синонимов для команд, выполняемых интерпретатором, необходимо реализовать возможность конструирования синтаксиса инструкций относительно файла настроек:

- Сохранение результатов выполнения операций:
  - left= - при наличии этой инструкции в файле настроек, в обрабатываемом файле переменные, в которые будет сохранён результат выполнения операции, должны находиться слева от операции. Пример:
    Var=add(Smth,OtheR);
  - right= - при наличии этой инструкции в файле настроек, в обрабатываемом файле переменные, в которуе будет сохранён результат выполнения операции, должны находиться справа от операции. Пример:
    add(Smth,OtheR)=Var;
- Взаимное расположение операндов и операции, выполняемой над операндами:
  - Для унарных операций:
    - op() - при наличии этой инструкции в файле настроек, аргумент операции находится после операции и обрамляется скобками. Пример:
      result=operation(argument);
    - ()op - при наличии этой инструкции в файле настроек, аргумент операции находится перед операцией и обрамляется скобками. Пример:
      result=(argument)operation;
  - Для бинарных операций:
    - op() - при наличии этой инструкции в файле настроек, аргументы операции находятся после операции и обрамляются скобками. Пример:
      result=operation(argument1,argument2);
    - (op) - при наличии этой инструкции в файле настроек, первый аргумент операции находится перед операцией, второй аргумент операции находится после операции. Пример:
      result=argument1 operation argument2;
    - ()op - при наличии этой инструкции в файле настроек, аргументы операции находятся после операции и обрамляется скобками. Пример:
      result=(argument1,argument2)operation;

Пример файла настроек:

right= #это комментарий

(op)
add sum
#mult prod и это тоже комментарий
[sub minus
pow ^ и это...]
div /
rem %
xor <>
xor ><
#xor <>
input in
output print
= ->

Значения base_assign, base_input, base_output задаются при помощи передачи значений в аргументы командной строки, могут находиться в диапазоне [2..36] и имеют значение по умолчанию равное 10 (при отсутствии в командной строке).
Разделителем между инструкциями в обрабатываемом интерпретатором файле является символ “;”. Сепарирующие символы (пробелы, табуляции, переносы строк) между лексемами могут присутствовать произвольно, различий между прописными и строчными буквами нет. Также в тексте программ могут присутствовать однострочные комментарии, начинающиеся с символа # и заканчивающиеся символом конца строки или символом конца файла; многострочные комментарии, обрамляющиеся символами ‘[‘ и ‘]’, вложенность многострочных комментариев произвольна.
Входной файл для интерпретатора подаётся через аргументы командной строки. Реализуйте обработку интерпретатором инструкций из входного файла. При завершении работы интерпретатор должен “запомнить” последний файл настроек, с которым работал, и при следующем запуске работать с теми же настройками, если это возможно.
В качестве аргумента командной строки приложению можно подать флаг “--debug”, “-d”, “/debug” (перечисленные флаги эквивалентны). При наличии одного из таких флагов в аргументах командной строки, однострочный комментарий с текстом “BREAKPOINT” (без кавычек) интерпретируется приложением как точка останова: выполнение интерпретатором кода из входного файла на момент встречи этого комментария приостанавливается, и пользователь начинает взаимодействовать с программой в интерактивном режиме посредством диалога. Интерактивный диалог должен предложить пользователю выбор из нескольких возможных действий:

- вывести в стандартный поток вывода значение переменной с именем, считываемым из стандартного потока ввода, в системе счисления с основанием 16, а также снять дамп памяти, в которой хранится это значение (байты значения, записанные в системе счисления с основанием 2, в порядке нахождения в памяти “слева направо”; строковые представления байтов должны сепарироваться одним символом пробела); после выполнения действия интерактивный диалог с пользователем продолжается;

- вывести в стандартный поток вывода имена и значения всех переменных, “известных” (с которыми велась работа) на данный момент исполнения кода; после выполнения действия интерактивный диалог с пользователем продолжается;
- изменить значение переменной, имя которой считывается из стандартного потока ввода, на новое значение, считываемое из стандартного потока ввода как число в системе счисления с основанием 16 (переменная должна быть “известна” на момент выполнения операции); после выполнения действия интерактивный диалог с пользователем продолжается;
- “объявить” переменную, имя которой считывается из стандартного потока ввода и проинициализировать её значением, которое вводится, в зависимости от выбора пользователя в интерактивном диалоге, запускаемом из основного интерактивного диалога, как число в цекендорфовом представлении либо как число записанное римскими цифрами (переменная не должна быть “известна” на момент выполнения операции); после выполнения действия интерактивный диалог с пользователем продолжается;
- “отменить” объявление переменной, имя которой считывается из стандартного потока ввода; дальнейшее использование переменной в коде до её повторного “объявления” (через выполнение инструкции либо через отладчик) должно привести к ошибке интерпретатора (переменная должна быть “известна” на момент выполнения операции); после выполнения действия интерактивный диалог с пользователем продолжается;
- завершить интерактивный диалог с пользователем и продолжить выполнение кода;
- завершить работу интерпретатора.

При возникновении ошибок в рамках интерактивного диалога, программа должна вывести в стандартный поток вывода информацию об ошибке отладчика; состояние переменных при этом не изменяется, интерактивный диалог не завершается, выполнение приложения не завершается.

Взаимодействие с переменными уровня интерпретатора (объявление, присваивание, “отмена” объявления) обеспечьте на основе структуры данных вида “бор”.

Предоставьте текстовый файл с выражениями для реализованного интерпретатора и продемонстрируйте его работу с разными файлами настроек, с разными входными текстовыми файлами с наборами инструкций, а также работу в режиме отладки. Обработайте ошибки времени выполнения инструкций.