Задания к работе №4 по фундаментальным алгоритмам.

Приложения не должны завершаться аварийно.

Во всех заданиях запрещено пользоваться функциями, позволяющими завершить выполнение приложения из произвольной точки выполнения.

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

Во всех заданиях все вводимые (с консоли, файла, командной строки) пользователем данные должны подвергаться валидации в соответствии с типом валидируемых данных, если не сказано обратное.

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.

1. Крестьянину нужно перевезти через реку волка, козу и капусту. Но лодка такова, что в ней может поместиться только крестьянин, а с ним или один волк, или одна коза, или одна капуста. Но если оставить на берегу волка с козой, то волк съест козу, а если оставить на берегу козу с капустой, то коза съест капусту. Как перевезти свой груз крестьянину? Реализуйте клиент-серверный комплекс приложений (Unix/Windows), клиент которого принимает на вход команды от пользователя и делегирует их выполнение серверному приложению посредством очереди сообщений. Продумайте возможность обработки сообщений от многих пользователей, для этого разработайте систему идентификации пользователей.

   В силу сложившихся обстоятельств крестьянин понимает достаточно ограниченный набор команд:

   1) take <object> - взять в лодку заданный объект. Вместо <object> может быть написано wolf, goat или cabbage. При этом команда может быть выполнена только если в лодке есть место;
   2) put; - выложить на берег то, что есть в лодке. При этом команда может быть выполнена только если в лодке помимо крестьянина есть ещё какой-либо объект;
   3) move; - переплыть реку на лодке. При этом команда может быть выполнена в любом случае вне зависимости от того, что именно находится в лодке.

   Клиентские приложения передают серверному приложению инструкции по одной из текстового файла (путь к этому файлу передаётся через аргументы строки), в котором лежит вся последовательность инструкций для крестьянина. Придумайте различные сценарии: которые решают данную задачу, а также с неверной последовательностью инструкций. Реализуйте программу так, чтобы были обработаны всевозможные ошибки.

2. Реализуйте пользовательский тип данных полинома от нескольких переменных. Полином должен представлять собой контейнер мономов, основанный на структуре данных типа двусвязный список. Тип монома от нескольких переменных содержит целочисленное значение коэффициента и ассоциативный контейнер пар значений “имя переменной - степень” (степени переменных являются целыми неотрицательными числами). Например, строковое представление монома “<5*x^2_x1^3_y12^3_y333^4_z_x1^1>” соответствует значению $5x^2x_1^4y_{12}^3y_{333}^4z$ (формат строкового представления придумайте самостоятельно, при этом коэффициент монома равный 1 и степени равные 0 и 1 могут быть опущены в строковом представлении; допускаются повторяющиеся имена переменных в рамках строкового представления монома). Имена переменных при этом могут состоять из символов букв (прописные и строчные буквы отождествляются) и символов 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. Для демонстрации работы с вашим типом разработайте клиент-серверное приложение для обработки регистрозависимых команд следующего вида:

   Add(4*x^2-x_y^2+4*x_y+1, x^2_y^3+2*x_y^2+_y^2-4*x_y); % сложить многочлены
   Mult(4*x^2_y^3+4xy+1, x^2_y^3+2*x_y^2+3*x^2_y^2-4*x_y+5); % умножить многочлены

   Допустимые инструкции:
   - однострочный (начинается с символа ‘%’) комментарий; многострочный (начинается с символа ‘[‘, заканчивается символом ‘]’, вложенность недопустима) комментарий;
   - Add – сложение;
   - Sub - вычитание;
   - Mult – умножение;
   - Eval – вычисление значения полинома в заданной точке;
   - Diff – дифференцирование;
   - Grad – вычисление градиента (результат возвращается или в символьном виде или в виде значения в заданной точке);

- Print – вывод в стандартный поток вывода полинома, находящегося в сумматоре.

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

Mult(x^2+3x-1,2x+x^3); % умножить два полинома друг на друга, результат умножения сохранить в сумматор;
Add(4x1y^2-8z); % в качестве первого аргумента операции сложения будет взято значение из сумматора, результат будет занесен в сумматор;
Eval({1,2,3}); % необходимо взять полином из сумматора и для него вычислить значение в 1.

В начале обработки входного потока данных значение сумматора равно нулю. Результаты всех операций над полиномами должны сохраняться в сумматор.
Разработайте клиент-серверное приложение (Unix/Windows), в котором математический функционал реализован на стороне сервера, а клиентские приложения передают серверному приложению запросы на выполнение операций. Взаимодействие клиентских приложений с серверным приложением с ним реализуйте на основе проецируемых в память файлов. Клиентские приложения получают входные инструкции из текстовых файлов, пути к которым передаются в качестве аргументов командной строки.

3. Через блокируемую посредством семафоров разделяемую память подаётся текстовый файл, содержащий последовательность строк, в каждой из которых записаны один или несколько операторов над булевскими векторами из множества [A, B, …, Z] и, возможно, однострочные или многострочные комментарии. Символом однострочного комментария является символ ‘%’, а символами многострочного – символы ‘{‘ и ‘}’. Возможный вид операторов:
    1) A := B <op> C; <op> - логическая операция из списка:
        - + (дизъюнкция);
        - & (конъюнкция);
        - -> (импликация);
        - <- (обратная импликация);
        - ~ (эквиваленция);
        - <> (сложение по модулю 2);
        - +> (коимпликация);
        - ? (штрих Шеффера);
        - ! (стрелка Пирса);

        := - оператор присваивания.

    2) X := \W; - логическое отрицание.

    3) read(D, base); - ввод значения в переменную D в системе счисления с основанием base (в диапазоне 2..36 включительно оба конца).

    4) write(Q, base); - вывод значения из переменной Q в системе счисления с основанием base (в диапазоне 2..36 включительно оба конца).

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 подробную информацию о выполнении каждой инструкции из файла). На стороне сервера необходимо реализовать интерпретатор инструкций, подаваемых из клиентских приложений, которые, в свою очередь, получают инструкции из текстового файла. Необходимо предусмотреть обработку ошибок.

4. Реализуйте приложение – интерпретатор операций над целочисленными массивами. Приложение оперирует целочисленными массивами произвольной длины с именами A, B, …, Z. Система команд данного интерпретатора (прописные и строчные буквы отождествляются):
    1) 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);
    2) Save A, out.txt; - выгрузить элементы массива A в файл out.txt;
    3) Rand A, count, lb, rb; - заполнить массив A псевдослучайными элементами из отрезка $[lb; lr]$ в количестве count штук.
    4) Concat A, b; - сконкатенировать два массива A и B результат сохранить в массив A;
    5) Free(a); - очистить массив A и сопоставить переменную A с массивом из 0 элементов;
    6) Remove a, 2, 7; - удалить из массива a 7 элементов, начиная с элемента с индексом 2;
    7) Copy A, 4, 10, b; - скопировать из массива A элементы с 4 по 10 (оба конца включительно) и сохранить их в b;
    8) Sort A+; - отсортировать элементы массива A по неубыванию;
    9) Sort A-; - отсортировать элементы массива A по невозрастанию;
    10) Shuffle A; - переставить элементы массива в псевдослучайном порядке;
    11) 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 отклонений элементов от среднего значения;
    12) Print a, 4, 16; - вывести в стандартный поток вывода элементы массива начиная с 4 и по 16;
    13) Print a, all; - вывести на экран все элементы массива.

    Индексирование в массивах начинается с 0. Для сортировки массивов используйте стандартную функцию qsort, свои реализации алгоритмов сортировки не допускаются. Предоставьте текстовый файл с инструкциями для данного интерпретатора. Считается, что синтаксических ошибок в инструкциях из входного файла нет, однако должны быть обработаны все ошибки исполнения инструкций (выход за границы массива, обращение к необъявленной переменной, невалидный диапазон элементов и т. д.).

5. На вход программе через аргументы командной строки подаются пути к текстовым файлам, содержащим арифметические выражения (в каждой строке находится одно выражение). 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. В вашей программе необходимо для каждого выражения:
   - проверить баланс скобок;
   - построить обратную польскую запись выражения;
   - вычислить значение каждого выражения с использованием алгоритма вычисления обратной польской записи.

   В результате работы программы для каждого файла необходимо вывести в стандартный поток вывода имя файла и для каждого выражения из этого файла необходимо вывести:
   - исходное выражение;
   - обратную польскую запись для исходного выражения;
   - значение выражения.

   В случае обнаружения ошибки в расстановке скобок либо невозможности вычислить для каждого файла, где обнаружена ошибка, 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 типа стек на базе односвязного списка.

6. Напишите приложение, которое по заданной булевой формуле строит её таблицу истинности. На вход программы подается файл, который содержит одну строку, в которой записана булева формула. В этой формуле могут присутствовать:
    - односимвольные имена переменных;
    - константы 0 и 1;
    - & - 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;
    - круглые скобки.

    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 операций: 3(~), 2(?,!,+>,&), 1(|, ->, <>, =).

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. Реализовать чтение данных из файла и выполнение всех простых арифметических операций (+, -, *, /, %), инициализации переменной и присваивания значения переменной (=) и операции print (вывод на экран 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. При инициализации переменной необходимо довыделить память в динамическом массиве структур типа *MemoryCell*. 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 букв, прописные и строчные буквы не отождествляются. В случае использования в вычислениях не объявленной переменной необходимо остановить работу интерпретатора и вывести сообщение об ошибке в стандартный поток вывода.

8. Реализуйте интерпретатор с настраиваемым синтаксисом. Для настройки интерпретатора через аргументы командной строки подается файл с описанием 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 более одного символа (в качестве символов, входящих в имена переменных, допускаются символы букв, цифр и подчерка (‘_’); имя переменной не может начинаться с символа цифры; длина имени переменной произвольна; прописные и строчные буквы не отождествляются).

   Основные команды, которые могут быть выполнены интерпретатором:

   - add (сложение);
   - mult (умножение);
   - sub (вычитание);
   - pow (возведение в степень);
   - div (целочисленное деление);
   - rem (взятие остатка от деления);
   - xor (побитовое сложение по модулю 2);
   - 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)
   - = (присваивание значения переменной или её инициализация)

   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. 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:

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

Значения base_input и base_output задаются при помощи аргументов командной строки, могут находиться в диапазоне [2..36] и имеют значение по умолчанию равное 10.
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 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. Продемонстрируйте работу интерпретатора с разными файлами настроек и с разными входными текстовыми файлами с наборами инструкций.

9. Разработайте приложение для организации макрозамен в тексте. На вход программы подается текстовый файл, который содержит в начале файла набор директив #define и далее обычный текст. Синтаксис директивы соответствует стандарту языка C:

   #define <def_name> <value>

   Аргументов у директивы нет. Ваша программа должна обработать текстовый файл, выполнив замены во всем тексте последовательности символов <def_name> на <value>. Количество директив произвольно, некорректных директив нет, размер текста произволен. В имени <def_name> допускается использование символов латинского алфавита (прописные и строчные буквы не отождествляются) и символов 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.

10. Разработайте приложение, моделирующее работу центра клиентской поддержки. На вход приложению подаются пути к файлам. В файле, расположенном по пути, передаваемом в качестве первого аргумента командной строки, содержится информация о параметрах модели:
    - вид структуры данных, предназначенной для обработки заявок конкретным отделением (может быть “BinomialHeap” для биномиальной пирамиды или “SkewHeap” для косой пирамиды);
    - вид структуры данных, предназначенной для хранения объектов пирамид (может быть “DynamicArray” для динамического массива пирамид, отсортированного по ключу идентификатора отделения или “BinarySearchTree” для бинарного дерева поиска, хранящего в элементах в качестве ключа идентификатора отделения и в качестве значения пирамиду с заявками для этого отделения);
    - дата и время начала и конца моделирования (с точностью до минут);
    - минимальное и максимальное время обработки заявки оператором (в минутах, неотрицательное целое число);
    - количество отделений (в диапазоне [1..100]);
    - для каждого отделения: количество операторов, обрабатывающих заявки (в диапазоне [10..50]);
    - 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 прихода заявки> <приоритет> <идентификатор отделения>
    “<текст заявки>”

    Приоритет может иметь значение в диапазоне [0..*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 заявки этого отделения наименее загруженному обработкой заявок отделению (используйте операцию meld).

    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:

<*системное время возникновения ситуации*> [<*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 занимается её обработкой
REQUEST_HANDLING_FINISHED	Окончание обработки заявки; идентификатор заявки, время её обработки (в минутах) и имя оператора, который занимается её обработкой
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).

Также реализуйте вспомогательное приложение для генерации файла с информацией о параметрах модели, с настройкой параметров посредством интерактивного диалога с пользователем, а также для генерации файлов со входными данными.