Задания к работе 3 по прикладному программированию.

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. 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.

2. 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.

3. 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 задания 2.

4. 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.

5. В текстовом файле находится информация о жителях (тип структуры *Citizen*) 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 *Citizen*. 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.

6. 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.

7. **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.

8. 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.

9. 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.

10. 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.

11. 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.