**Современные системы программирования**

**Лабораторная работа №1 (6 часов)**

**Функциональное программирование**

**на языке Haskell**

**Лабораторная работа №1**

**«Функциональное программирование. Язык Haskell»**

1. **Цель работы:**

Приобрести навыки работы с интерпретатором языка Haskell. Получить представление об основных типах языка Haskell. Научиться определять простейшие функции.

1. **Порядок выполнения работы:**
2. Изучить основные теоретические положения.
3. Выполнить все задания, встречающиеся в лабораторной работе по ходу изучения материала.
4. Решить задачи на языке Haskell согласно варианту в журнале.
5. Написать отчет по всем выполненным заданиям методички.
6. Защитить отчет у преподавателя.
7. **Основные теоретические положения:**

**3.1. Что такое Haskell?**

Haskell является чисто функциональным языком программирования общего назначения, который включает много последних инноваций в разработке языков программирования. Haskell обеспечивает функции высокого порядка, нестрогую семантику, статическую полиморфную типизацию, определяемые пользователем алгебраические типы данных, сопоставление с образцом, описание списков, модульную систему, монадическую систему ввода - вывода и богатый набор примитивных типов данных, включая списки, массивы, целые числа произвольной и фиксированной точности и числа с плавающей точкой. Haskell является и кульминацией, и кристаллизацией многих лет исследования нестрогих функциональных языков.

Hasckell является одним из самых распространённых нестрогих языков программирования. Назван по имени математика Хаскелла Карри.

Язык появился в 1990 году. Haskell популярен в академических кругах, но малоизвестен среди прикладных программистов. В последнее время расширяется набор прикладных библиотек, язык интегрируется в распространённые программные системы (.Net, COM/ActiveXHaskellScript, Javajaskell), что делает язык всё более и более привлекательным для профессиональных программистов.

**3.2.** **Основы работы с интерпретатором Hugs**

Для выполнения лабораторных работ будет использоваться интерпретатор языка Haskell. Существует несколько реализаций интерпретатора; в настоящем курсе будет использоваться интерпретатор Hugs.

После запуска интерпретатора Hugs (Hugs (Haskell98 mode)) на экране появляется диалоговое окно среды разработчика, автоматически загружается специальный файл предопределений типов и определений стандартных функций на языке Haskell (Prelude.hs), и выводится стандартное приглашение к работе. Это приглашение имеет вид:

**Prelude>**

Вообще, перед символом **>**  выводится имя последнего загруженного модуля. Если приглашение интерпретатора изменено на **Main>**, значит загружен какой-то модуль по умолчанию. Перед началом работы следует выгрузить лишние модули. Для этого зайдите в меню **File -> Module Manager** и удалите файл с именем Hugs.hs. Если приглашение интерпретатора изменилось на **Prelude>** , значит вы все сделали верно.

После вывода приглашения можно вводить выражения языка Haskell, либо команды интерпретатора. Команды интерпретатора отличаются от выражений языка Haskell тем, что начинаются с символа двоеточия (**:**). Примером команды интерпретатора является команда **:quit**, по которой происходит завершение работы интерпретатора. Команды интерпретатора можно сокращать до одной буквы; таким образом, команды **:quit** и **:q** эквивалентны. Команда **:set** используется для того , чтобы установить различные опции интерпретатора. Команда **:?** выводит список доступных команд интерпретатора.

**3.3.Типы**

Программы на языке Haskell представляют собой выражения, вычисление которых приводит к значениям. Каждое значение имеет тип. Интуитивно тип можно понимать просто как множество допустимых значений выражения. Для того, чтобы узнать тип некоторого выражения, можно использовать команду интерпретатора **:type** (или **:t**). Кроме того, можно выполнить команду **:set +t**, для того, чтобы интерпретатор автоматически печатал тип каждого вычисленного результата.   
Основными типами языка Haskell являются:

• Типы Integer и Int используется для представления целых чисел, причем значения типа Integer не ограничены по длине.

• Типы Float и Double используется для представления вещественных чисел.

• Тип Bool содержит два значения: True и False, и предназначен для представления результата логических выражений.

• Тип Char используется для представления символов. Имена типов в языке Haskell всегда начинаются с заглавной буквы.

Язык Haskell является сильно типизированным языком программирования. Тем не менее в большинстве случаев программист не обязан объявлять, каким типам принадлежат вводимые им переменные. Интерпретатор сам способен вывести типы употребляемых пользователем переменных.

Однако, если все же для каких-либо целей необходимо объявить, что некоторое значение принадлежит некоторому типу, используется конструкция вида: переменная :: Тип. Если включена опция интерпретатора +t, он печатает значения в таком же формате.

Ниже приведен пример протокола сессии работы с интерпретатором. Предполагается, что текст, следующий за приглашением Prelude>, вводит пользователь, а следующий за этим текст представляет ответ системы.

***Текст программы:***

**Prelude>:set +t**

**Prelude>1**

**1 :: Integer**

**Prelude>1.2**

**1.2 :: Double**

**Prelude>’a’**

**’a’ :: Char**

**Prelude>True**

**True :: Bool**

*Задание 1. Выполните команду :set +t и по приведенному примеру проверьте, каких типов будут следующие данные:*

* 50000
* 5.555
* ‘5’
* “5555”
* True

Из данного протокола можно сделать вывод, что значения типа Integer, Double и Char задаются по тем же правилам, что и в языке Си.   
Развитая система типов и строгая типизация делают программы на языке Haskell безопасными по типам. Гарантируется, что в правильной программе на языке Haskell все типы используются правильно. С практической точки зрения это означает, что программа на языке Haskell при выполнении не может вызвать ошибок доступа к памяти (Accessviolation). Также гарантируется, что в программе не может произойти использование неинициализированных переменных. Таким образом, многие ошибки в программе отслеживаются на этапе ее компиляции, а не выполнения.

**3.4. Арифметика**

Интерпретатор Hugs можно использовать для вычисления арифметическихвыражений. При этом можно использовать операторы +, -,\*, / (сложение, вычитание, умножение и деление) с обычными правилами приоритета. Кроме того, можно использоватьоператор ^ (возведение в степень). Таким образом, сеанс работы может выглядетьследующим образом:

***Текст программы:***

**Prelude>2\*2**

**4 :: Integer**

**Prelude>4\*5 + 1**

**21 :: Integer**

**Prelude>2^3**

**8 :: Integer**

Кроме того, можно использовать стандартные математические функции sqrt (квадратный корень), sin, cos, exp и т.д. В отличие от многих других языков программирования, в Haskell при вызове функции не обязательно помещать аргумент в скобки. Таким образом, можно просто писать sqrt 2, а не sqrt(2). Пример:

***Текст программы:***

**Prelude>sqrt 2**

**1.4142135623731 :: Double**

**Prelude>sqrt 2 + 1**

**2.4142135623731 :: Double**

**Prelude>sqrt (2 + 1)**

**1.73205080756888 :: Double**

Из данного примера можно сделать вывод, что вызов функции имеет более высокий приоритет, чем арифметические операции, так что выражение sqrt 2 + 1 интерпретируется как (sqrt 2) + 1, а не sqrt (2 + 1). Для задания точного порядка вычисления следуетиспользовать скобки, как в последнем примере. Также следует заметить, что в отличие от большинства других языков программирования, целочисленные выражения в языке Haskell вычисляются с неограниченным числом разрядов. В отличие от языка Си, где максимально возможное значение типа int ограничено разрядностью машины, тип Integer в языке Haskell может хранить целые числа произвольной длины.

*Задание 2. Выполните арифметические действия:*

* *2\*Cos*

Операторы, определённые в стандартном модуле Prelude. Значения ассоциативности и приоритета для этих операторов также указаны в таблице.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Символ | Значение | Тип | Ас. | Пр. |
| !! | Индекс | [a] -> Int -> a | Л | 9 |
| . | Композиция | (a -> b) -> (c -> a) -> c -> b | П | 9 |
| ˆ | Экспонента | (Integral b, Num a) => a -> b-> a | П | 8 |
| ˆˆ | Экспонента | (Fractional a, Integral b) => a -> b -> a | П | 8 |
| \*\* | Экспонента | Floating a => a -> a -> a | П | 8 |
| \* | Умножение | Num a => a -> a -> a | Л | 7 |
| / | Деление | Rational a => a -> a -> a | Л | 7 |
| quot | Целое деление | Num a => a -> a -> a | Л | 7 |
| rem | Остаток | Num a => a -> a -> a | Л | 7 |
| div | Целое деление | Num a => a -> a -> a | Л | 7 |
| mod | Остаток | Num a => a -> a -> a | Л | 7 |
| :% | Дробь | Integral a => a -> a -> Ratio a | Л | 7 |
| % | Сокращение | Integral a => a -> a -> Ratio a | Л | 7 |
| + | Сложение | Num a => a -> a -> a | Л | 6 |
| - | Вычитание | Num a => a -> a -> a | Л | 6 |
| : | Создание списка | a -> [a] -> [a] | П | 5 |
| ++ | Конкатенация | [a] -> [a] -> [a] | П | 5 |
| /= | Неравенство | Eq a => a -> a -> Bool |  | 4 |
| == | Равенство | Eq a => a -> a -> Bool |  | 4 |
| < | Меньше | Ord a => a -> a -> Bool |  | 4 |
| <= | Меньше или равно | Ord a => a -> a -> Bool |  | 4 |
| > | Больше | Ord a => a -> a -> Bool |  | 4 |
| >= | Больше или равно | Ord a => a -> a -> Bool |  | 4 |
| elem | Существование | Eq a => a -> [a] -> Bool |  | 4 |
| notElem | Несуществование | Eq a => a -> [a] -> Bool |  | 4 |
| && | Логическое И | Bool -> Bool -> Bool | П | 3 |
| || | Логическое ИЛИ | Bool -> Bool -> Bool | П | 2 |
| >> | Связывание | m a -> m b -> m b | Л | 1 |
| >>= | Связывание | m a -> (a -> m b) -> m b | Л | 1 |
| =<< | Связывание | Monad m => (a -> m b) -> m a -> m b | П | 1 |
| $ | Стр. композиция | (a -> b) -> a -> b | П | 0 |

**3.5. Кортежи**

В языке Haskell пару можно задать, перечислив компоненты через запятую и взяв их в скобки: (5,3). Компоненты пары не обязательно должны принадлежать одному типу: можно составить пару, первым элементом которой будет строка, а вторым - число и т.д. Например, пара (5,3) имеет тип (Integer, Integer); пара (1, 'a') принадлежит типу (Integer, Char). Можно привести и более сложный пример: пара ((1,'a'),1.2) принадлежит типу ((Integer,Char),Double). Проверьте это с помощью интерпретатора.

Кроме пар, аналогичным образом можно определять тройки, четверки и т.д. Их типы записываются аналогичным образом.

Для работы с парами в языке Haskell существуют стандартные функции fst и snd, возвращающие, соответственно, первый и второй элементы пары. Эти функции определены только для пары и не работают для других кортежей.

***Текст программы:***

**Prelude>fst (5, True)**

**5 :: Integer**

**Prelude>snd (5, True)**

**True :: Bool**

**Prelude>(1,2,3,4)**

**(1,2,3,4) :: (Integer,Integer,Integer,Integer)**

Элементом кортежа может быть значение любого типа, в том числе и другой кортеж. Для доступа к элементам кортежей, составленных из пар, может использоваться комбинация функций fst и snd. Следующий пример демонстрирует извлечение элемента 'a' из кортежа(1, ('a', 23.12)):

***Текст программы:***

**Prelude>**

**fst (snd (1, ('a', 23.12)))**

**'a' :: Char**

*Задание 3.Зададим кортеж вида: (5, ((“hello’’, “world”),’a’))*

*Извлеките из него:*

* *число 5;*
* *слово «hello»;*
* *символ ‘а’.*

**3.6. Списки**

В отличие от кортежей, список может хранить произвольное количество элементов. Чтобы задать список в Haskell, необходимо в квадратных скобках перечислить его элементы через запятую. Все эти элементы должны принадлежать одному и тому же типу. Тип списка с элементами, принадлежащими типу a, обозначается как [a]. В списке может не быть ни одного элемента. Пустой список обозначается как [].

Элементами списка могут быть любые значения- числа, символы, кортежи, другие списки и т.д.

Оператор : (двоеточие) используется для добавления элемента в начало списка. Его левым аргументом должен быть элемент, а правым – список. С помощью оператора (:) и пустого списка можно построить любой список.

***Текст программы:***

**Prelude>1:[2,3]**

**[1,2,3] :: [Integer]**

**Prelude>1:(2:(3:[]))**

**[1,2,3] :: Integer**

Оператор (:) ассоциативен вправо, поэтому в приведенном выше выражении можноопустить скобки:

***Текст программы:***

**Prelude>1:2:3:[]**

**[1,2,3] :: Integer**

Для работы со списками в языке Haskell существует большое количество функций. В данной лабораторной работе рассмотрим только некоторые из них:

Функция head возвращает первый элемент списка.

Функция last возвращает последний элемент списка.

Функция tail возвращает список без первого элемента

Функция init возвращает список без последнего элемента

Функция null проверяет список на пустоту. Если в качестве аргумента этой операции будет задан пустой список, то функция выдаст значение True, в противном случае –False.

Функция length возвращает длину списка.

Функция elem проверяет наличие элемента в списке.

Функция take возвращает список, состоящий из n первых элементов исходного списка.

Функция zip возвращает список, состоящий из пар объединенных исходных списков.

Функция !! возвращает элемент, номер которого задан (начиная с 0).

Функции head и tail определены для непустых списков. При попытке применить их к пустому списку интерпретатор сообщает об ошибке.

Примеры работы с указанными функциями:

***Текст программы:***

**Prelude>head [1,2,3]**

**1 :: IntegerPrelude>tail [1,2,3]**

**[2,3] :: [Integer]**

**Prelude>tail [1]**

**[] :: Integer**

**Prelude>length [1,2,3]**

**3 :: Int**

**Prelude>elem 2 [1,2,3]**

**True :: Bool**

**Prelude> take 2 [1,2,3]**

**[1,2] :: [Integer]**

**Prelude> zip ["20","30"] [1,2,3]**

**[("20",1),("30",2)] :: [([Char],Integer)]**

**Prelude> [1,2,3,4,5] !! 3**

**4 :: Integer**

Заметьте, что результат функции length принадлежит типу Int, а не типу Integer.

Для соединения (конкатенации) списков в Haskell определен оператор ++.

*Задание 4. Приведите пример нетривиальных выражений, принадлежащих следующему типу:*

* *((Char,Integer), String, [Double])*
* *[(Double,Bool,(String,Integer))]*
* *([Integer],[Double],[(Bool,Char)])*
* *[[[(Integer,Bool)]]]*
* *(([Double],[Bool]),[Integer])*

*Требование нетривиальности в данном случае означает, что встречающиеся в выражениях списки должны содержать больше одного элемента.*

*Задание 5. Решите задание согласно своего варианта:*

*1. Извлеките второй элемент списка.*

*2. Извлеките предпоследний элемент списка.*

*3. Найдите длину списка.*

4. *Дан список целых чисел. Добавьте элемент в конец списка.*

*5. Даны два списка целых чисел. Объедините их попарно в один список.*

*6. Извлеките третий с конца элемент списка.*

*7. Дан список целых чисел. Проверьте наличие числа 5 в списке.*

*8. Извлеките последний элемент списка.*

*9. Извлеките список, состоящий из 2-го, 3-го и 4-го элементов списка.*

*10. Даны два списка целых чисел. Извлеките первые элементы списков.*

**3.7. Строки**

Строковые значения в языке Haskell, как и в Си, задаются в двойных кавычках. Они принадлежат типу String.

***Текст программы:***

**Prelude>"hello"**

**"hello" :: String**

В действительности строки являются списками символов; таким образом, выражения"hello", ['h','e','l','l','o'] и 'h':'e':'l':'l':'o':[] означают одно и то же, а тип String является синонимом для [Char]. Все функции для работы со списками можно использовать при работе со строками:

Для преобразования числовых значений в строки и наоборот существуют функции read и show:

***Текст программы:***

**Prelude>head "hello"**

**'h' :: Char**

**Prelude>show 1**

**"1" :: [Char]**

**Prelude>"Formula " ++ show 1**

**"Formula 1" :: [Char]**

**Prelude>1 + read "12"**

**13 :: Integer**

Если функция read не сможет преобразовать строку в число, она сообщит об ошибке.

**3.8. Функции**

До сих пор мы использовали встроенные функции языка Haskell. Теперь пришла пора научиться определять собственные функции. Для этого нам необходимо изучить еще несколько команд интерпретатора (напомним, что эти команды могут быть сокращены до одной буквы):

Команда :load позволяет загрузить в интерпретатор программу на языке Haskell, содержащуюся в указанном файле.

Команда :edit запускает процесс редактирования последнего загруженного файла.

Команда :reload перечитывает последний загруженный файл.

Определения пользовательских функций должны находиться в файле, который нужно загрузить в интерпретатор Hugs с помощью команды :load. Для редактирования загруженной программы можно использовать команду :edit. Она запускает внешний редактор (по умолчанию это Notepad) для редактирования файла. После завершения сеанса редактирования редактор необходимо закрыть; при этом интерпретатор Hugs перечитает содержимое изменившегося файла. Однако файл можно редактировать и непосредственно из оболочки Windows. В этом случае, для того чтобы интерпретатор смог перечитать файл, необходимо явно вызывать команду :reload.

Рассмотрим пример. Создайте в каком-либо каталоге файл lab1.hs.

Пусть полный путь к этому файлу с:\labs\lab1.hs. В интерпретаторе Hugs выполните следующие команды:

Prelude>:load "c:\\labs\\lab1.hs"

Если загрузка проведена успешно, приглашение интерпретатора меняется на Main>.

Обратите внимание, что при записи имени файла в аргументе команды :load символы \ дублируются. Также, как и в языке Си, в Haskell символ \ служит индикатором начало служебного символа ('\n' и т.п.) Для того, чтобы ввести непосредственно символ \, необходимо, как и в Си, экранировать его еще одним символом \.

Если подключить файл не удалось, попробуйте его добавить через **File -> Module Manager.**

Запишите в созданный вами файл следующий текст:

***Текст программы:***

**square :: Integer -> Integer**

**square x = x \* x**

Первая строка (square :: Integer ->Integer) объявляет, что мы определяем функцию square, принимающую параметр типа Integer и возвращающую результат типа Integer.

Вторая строка (square x = x \* x) является непосредственно определением функции. Функция square принимает один аргумент и возвращает его квадрат.

Функции в языке Haskell являются значениями “первого класса”. Это означает, чтоони “равноправны” с такими значениями, как целые и вещественные числа, символы, строки, списки и т.д. Функции можно передавать в качестве аргументов в другие функции, возвращать их из функций и т.п. Как и все значения в языке Haskell, функции имеют тип. Тип функции, принимающей значения типа a и возвращающей значения типа b, обозначается как a->b.

Загрузите созданный файл в интерпретатор и выполните следующие команды:

***Текст программы:***

**Main>square 2**

**4 :: Integer**

Заметим, что в принципе объявление типа функции square не являлось необходимым: интерпретатор сам мог вывести необходимую информацию о типе функции из ее определения. Однако, во-первых, выведенный тип был бы более общим, чем Integer ->Integer, а во-вторых, явное указание типа функции является “хорошим тоном” при программировании на языке Haskell, поскольку объявление типа служит своего рода документацией к функции и помогает выявлять ошибки программирования.

Имена определяемых пользователем функций и переменных должны начинаться с латинской буквы в нижнем регистре. Остальные символы в имени могут быть прописными или строчными латинскими буквами, цифрами или символами \_ и ' (подчеркивание и апостроф).

*Задание 6. Напишите функцию, вычисляющую сумму двух чисел и вызовите ее с числами, например 5 и 2.*

**3.8. Условные выражения**

В определении функции в языке Haskell можно использовать условные выражения.

Запишем функцию signum, вычисляющую знак переданного ей аргумента:

***Текст программы:***

**signum :: Integer -> Integer**

**signum x = if x > 0 then 1**

**else if x < 0 then -1**

**else 0**

Условное выражение записывается в виде: if условие then выражение else выражение. Выражения в then-части и в else-части условного оператора должны принадлежать одному типу. Часть else является обязательной.

Выражения типа Bool можно комбинировать с помощью общепринятых логических операторов && и || (И и ИЛИ), и функции отрицания not.

Разумеется, можно определять свои функции, возвращающие значения типа Bool, и использовать их в качестве условий. Например, можно определить функцию isPositive, возвращающую True, если ее аргумент неотрицателен и False в противном случае:

**isPositive :: Integer ->Bool**

**isPositive x = if x > 0 then True else False**

Теперь функцию signum можно определить следующим образом:

**signum :: Integer -> Integer**

**signum x = if isPositive x then 1**

**else if x < 0 then -1 else 0**

Отметим, что функцию isPositive можно определить и проще: Россия

**isPositive x = x > 0**

Следует сделать еще одно замечание, касающееся порядка определения функций. В предыдущем разделе мы определили две функции, signum и isPositive, одна из которых использовала для своего определения другую. Возникает вопрос: какая из этих функций должна быть определена раньше? Напрашивается ответ, что определение isPositive должно предшествовать определению функции signum; однако в действительности в языке Haskell порядок определения функций не имеет значения! Таким образом, функция isPositive может быть определена как до, так и после функции signum.

*Задание 7. Решите задачу согласно своего варианта:*

*1. По введенному номеру месяца вывести его наименование. Предусмотреть сообщение в случае некорректного ввода данных.*

*2. Для данного x вычислить значение функции:*

*3. Дано 3 числа. Упорядочьте их по возрастанию.*

*4. Даны два целых числа. Определить, является ли хотя бы одно из них делителем другого.*

*5. Дано 3 числа. Упорядочьте их по убыванию.*

*6. Дано 3 числа. Определить, имеется ли среди них хотя бы одна пара равных между собой чисел.*

*7. По введенному номеру недели вывести наименование этого дня. Предусмотреть сообщение в случае некорректного ввода данных.*

*8. Для данного x вычислить значение функции:*

*9. Дано 2 числа. Проверить заданные числа на четность/нечетность.*

*10. Если целое число m делится нацело на целое число n, то вывести на экран частное от деления, в противном случае вывести сообщение «m на n нацело не делится».*

**3.9. Функции многих переменных и порядок определения функций**

До сих пор мы определяли функции, принимающие один аргумент. Разумеется, в языке Haskell можно определять функции, принимающие произвольное количество аргументов. Определение функции add, принимающей два целых числа и возвращающей их сумму, выглядит следующим образом:

***Текст программы:***

**add :: Integer ->Integer ->Integer**

**add x y = x + y**

Тип функции add может выглядеть несколько загадочно. В языке Haskell считается, что операция -> ассоциативна вправо. Таким образом, тип функции add может быть прочитан как Integer -> (Integer ->Integer), т.е. в соответствие с правилом каррирования, результатом применения функции add к одному аргументу будет функция, принимающая один параметр типа Integer. Вообще, тип функции, принимающей n аргументов, принадлежащих типам t1, t2,…, tn, и возвращающей результат типа a, записывается в виде t1->t2->...->tn->a.

*Задание 8. Напишите функцию, вычисляющую сумму квадратов двух чисел и вызовите ее с числами, например 2 и 3:*

***Текст программы:***

**f :: Int -> Int -> Int**

**f x y = x\*x + y\*y**

**Main> f 2 3**

*А теперь попробуйте такой вызов:*

**Main> f 2.3 4.2**

Вы получите ошибку. Проблема заключается в том, что 4.2 это не Int.

Одним из вариантов решения будет не указывать явно тип функции f.  
Тогда Haskell выведет для нас наиболее общий подходящий тип:

***Текст программы:***

**f x y = x\*x + y\*y**

**Main> f 2.3 4.2**

В таком случае нам не надо создавать новую функцию для каждого типа данных.

*Задание 9. Напишите функции, решающие следующие задачи:*

*(для решения задач обязательно использовать рекурсию)*

1. *Дан список чисел. Найти произведение всех положительных чисел в списке.*
2. *Реализуйте число Фибоначчи.*
3. *Дан список чисел. Найдите количество чисел, которые больше 10.*
4. *Дан список чисел. Найдите в нем сумму положительных чисел.*
5. *Переведите число n >= 0 в список его десятичных цифр.*
6. *Дан список чисел. Найти их произведение.*
7. *Дан список чисел. Найти в нем количество нулей.*
8. *Дан список чисел. Найдите в нем количество отрицательных чисел.*
9. *Дан список чисел. Найти в нем сумму четных чисел.*
10. *Найдите факториал заданного числа.*
11. *Дан список чисел. Найти в нем произведение нечетных чисел.*

*\* Дано 2 числа. Найти их НОД.*

*\* Дано 2 числа. Найти их НОК.*

*\* Переведите число n >= 0 в список его неотрицательных делителей.*

Лабораторная работа №2

Используя генераторы списков согласно своему варианту решите следующие задачи:

1. **Конструирование конечных списков (количество элементов в списке - 20).**
   1. Список натуральных чисел, составляющих геометрическую прогрессию.
   2. Список нечётных натуральных чисел в арифметической прогрессии с шагом 3.
   3. Список чётных натуральных чисел в арифметической прогрессии с шагом 5.
   4. Список квадратов натуральных чисел.
   5. Список чисел Фибоначи.
   6. Список факториалов.
   7. Список кубов натуральных чисел.
   8. Список степеней двойки.
   9. Список степеней тройки.
   10. Список степеней пятерки.
   11. Список пятых степеней натуральных чисел.
   12. Список степеней семерки.
2. **Конструирование бесконечных списков.**
   1. Список степеней двойки.
   2. Список степеней тройки.
   3. Список кубов натуральных чисел.
   4. Список натуральных чисел, составляющих геометрическую прогрессию.
   5. Список квадратов натуральных чисел.
   6. Список пятых степеней натуральных чисел.
   7. Список чисел Фибоначи.
   8. Список четных чисел в арифметической прогрессии с шагом 3.
   9. Список факториалов.
   10. Список нечетных чисел в арифметической прогрессии с шагом 5.
   11. Список степеней пятёрки.
   12. Список степеней семёрки.
3. **Реализуйте сами следующие функции работы со списками.**
   1. *OddEven*(*L*) — функция перестановки местами соседних элементов в заданном списке.
   2. *Delete* (*A*,*L*) — удаляет первое вхождение заданного элемента из списка.
   3. *Compress(S)* — функция, удаляющая дубликаты из списка.
   4. *ListSumm*(*L*1, *L*2) — функция сложения элементов двух списков. Возвращает список, составленный из сумм элементов списков-параметров. Учесть, что переданные списки могут быть разной длины.
   5. *Reverse*(*L*) — функция, обращающая список (первый элемент списка становится последним, второй — предпоследним, и так далее до последнего элемента).
   6. *Position*(*L*, *A*) — функция, возвращающая номер первого вхождения заданного атома в список.
   7. *Insert*(*L*, *A*, *n*) — функция включения в список заданного атома на определённую позицию.
   8. *Set*(*L*) — функция, возвращающая список, содержащий единичные вхождения атомов заданного списка.
   9. *СhkDups (L)* – функция, возвращающая True, если в списке, являющемся ее аргументом, дважды содержится хотя бы один элемент.
   10. *IsSorted(L)* – функция, возвращающая True, если ее аргумент — отсортированный список/
   11. *Idxs(L*) – функция, которая находит в списке все вхождения элементов, равных заданному, и выдает позиции, в которых он встретился/
   12. *DelGt(L,a,b)* – функция, которая удаляет из списка все элементы, находящиеся в интервале [a,b], если a < b и в интервале [b,a], если b < a.
   13. *Map(F, L)* — функция применения другой переданной в качестве параметра функции ко всем элементам заданного списка.
   14. *Frequency(L)* — функция, возвращающая список пар (символ, частота). Каждая пара определяет атом из заданного списка и частоту его вхождения в этот список.
   15. *DelGt (A,L)* – функция, которая удаляет из списка все элементы, больше заданного.
4. **Напишите функции, решающие следующие задачи:**
5. Три точки A, B, C лежат на одной прямой. Заданы длины AB, BC, AC. Может ли точка A лежать между точками B и C.
6. Даны координаты центров и радиусы двух окружностей на плоскости. Может ли вторая окружность целиком содержаться внутри первой? Если нет, то сколько точек пересечения имеют окружности?
7. Можно ли из круглого листа железа диаметром D метров вырезать квадрат со стороной A метров?
8. Стороны треугольника A, B, C. Найти высоту, опушенную на строну C.
9. Высота равнобедренного треугольника H метров, основание L. Найти углы треугольника и длину боковой стороны.
10. Найти точку, равноудаленную от осей координат и от точки с заданными координатами (x,y).
11. Даны четыре точки A, B, C, D на плоскости. Является ли четырехугольник ABCD параллелограммом?
12. Даны четыре точки A, B, C, D на плоскости. Является ли четырехугольник ABCD квадратом?
13. Определите, является ли список палиндромом.
14. Найти углы треугольника, еcли они пропорциональны заданным числам A, B, C.
15. Периметр равнобедренного треугольника равен P метрам, а боковая сторона L. Найти высоту треугольника.
16. В равнобедренном треугольнике боковая сторона A, а основание B. Найти высоту, опущенную на основание.
17. Можно ли из отрезков с заданными длинами A, B, C построить прямоугольный треугольник?