Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ КОСМИЧЕСКИХ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Кафедра «Вычислительная техника»

УТВЕРЖДАЮ
Заведущий кафедрой
_____ А. И. Легалов
подпись
«_____» _____ 20___г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 Информатика и вычислительная техника

ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ ДЛЯ ФУНКЦИОНАЛЬНО-ПОТОКОВОГО ЯЗЫКА ПРОГРАММИРОВАНИЯ ПИФАГОР

Пояснительная записка

Руководитель _____ профессор, к.т.н. О. В. Непомнящий
подпись, дата

Выпускник _____ М. А. Вайман
подпись, дата

Нормоконтролер _____ доцент, к.т.н. В. И. Иванов
подпись, дата

Красноярск 2016

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Активное развитие параллельных вычислительных архитектур обуславливает проблемы для переноса программ с одной архитектуры на другую. Существующие языки программирования не позволяют создавать переносимые программы, поэтому развиваются архитектурно-независимые методы программирования, в том числе функционально-потоковое программирование, позволяющее писать программы независимо от того на какой архитектуре они будут исполняться. Это позволяет сначала отлаживать программы, находить и исправлять в них ошибки, а потом адаптировать их под конкретную архитектуру вручную либо автоматически.

Создание прикладных параллельных программ, ориентированных на обработку информационных потоков, удобнее осуществлять с применением функциональных языков параллельного программирования, в которых выполнение каждого оператора осуществляется по готовности его данных. Одним из языков поддерживающих архитектурно-независимый стиль программирования, является функционально-потоковый язык параллельного программирования Пифагор, который рассмотрен в ряде работ [3, 4, 7].

Такие языки не требуют явного описания параллелизма задачи, который в этом случае определяется в соответствии с информационными связями. Использование языка Пифагор позволяет:

- создавать переносимые программы с параллелизмом на уровне операторов, ограниченным лишь методом решения задачи;
- обеспечивать перенос программы на конкретную архитектуру, не распараллеливая программу, а сжимая ее максимальный параллелизм;
- проводить оптимизацию программы по множеству параметров с учетом специфики архитектуры ВС, для которой осуществляется трансляция без учета управляющих связей программы.

В языке Пифагор вычисления осуществляются по готовности данных, этот стиль удобен для решения вычислительных задач и задач обработки данных. Однако при работе с данным языком имеется проблема не полной поддержки библиотек функций, что не позволяет успешно разрабатывать сложные программы и проводить тестирование исполнительных систем, трансляторов, которые в данный момент находятся в стадии разработки.

Существует инструментарий в виде IDE Pifagor для разработки программ для языка Пифагор. Требуется интегрированный инструмент в IDE для работы с библиотеками функций.

Таким образом требуется провести анализ интегрированной среды разработки, устранить её ошибки, разработать модуль поддержки репозитория и интегрировать его в среду разработки,

1 Анализ программного обеспечения и постановка задач

1.1 Язык программирования Пифагор

Функциональный язык параллельного программирования Пифагор был разработан как изначально предназначенный для использования в параллельных вычислениях. Используя потоковую модель вычислений при срабатывании операторов по готовности данных, разработчикам удалось добиться возможности описания в программе естественного параллелизма присущего задаче. Это позволяет считать Пифагор архитектурно независимым при условии, что выполнение программы будет производиться путём сжатия параллелизма программы с учётом архитектурных ограничений статически, либо динамически с возможностью перераспределения вычислительной нагрузки между узлами многопроцессорной вычислительной машины или кластера.

Язык поддерживает распараллеливание как на уровне функций, так и на уровне элементарных операций. Как и большинство функциональных языков, Пифагор имеет возможности для ленивых вычислений, которые в нём задаются при помощи задержанных списков. Следствием наличия в языке механизма задержанных списков, вычисления в которых производятся только по явному требованию, могут явиться новые приёмы программирования, которые на основе алгоритмов рекурсивной редукции позволят производить построение вычисляемого выражения на первом уровне рекурсии, как бы достраивая выражение при помощи задержанных списков, что позволит избежать многочисленного копирования данных при рекурсивных вызовах [4].

Пифагор, обладает следующими характеристиками:

- описание распараллеливания на уровне операций;
- архитектурная независимость, достигаемая за счёт описания в программе только информационных связей;

- асинхронный параллелизм, достигаемый за счёт выполнения операций по готовности данных;
- отсутствие переменных, что позволило избежать конфликтов, связанных с совместным использованием памяти параллельными процессами;
- отсутствие операторов цикла, что позволяет избежать конфликтов при использовании различными данными одних и тех же фрагментов параллельной программы.

Хотя язык Пифагор и показал себя эффективным средством для разработки параллельных программ, всё же ему присущи некоторые недостатки. В настоящее время в языке используется принцип неявной типизации, и тип результата операции зависит от типов аргументов и правил семантики. Также в языке отсутствуют средства контроля типов и определения пользовательских типов и сложных структур данных.

1.1.1 Модель функционально-потоковых параллельных вычислений

Язык «Пифагор» представляет собой язык программирования, ориентированный на произведение параллельных вычислений. В основе модели лежит управление вычислениями по готовности данных. Сами вычисления протекают внутри бесконечных ресурсов, что позволяет неявно задавать параллелизм без анализа ресурсных конфликтов. Это позволило реализовать максимальный параллелизм, ограниченный только информационными связями текущей задачи. Перенос задачи в таком случае сводится к распределению ресурсов в соответствии с требованиями данной архитектуры.

Выбор операций и аксиом, составляющих базовый набор, ориентирован на наглядное текстовое представление информационного графа модели при описании его на языке программирования. Результатом этого стал несколько необычный формат языка. Кроме того, в языке отсутствуют вентили,

обеспечивающие условную передачу данных, ввиду того, что их достаточно сложно структурировать в текстовом режиме без дополнительной синхронизации. Концепция цикла также отсутствует; потому граф данной модели является ациклическим.

Модель задается тройкой

$M = (G, P, S_0)$,

где G – ациклически ориентированный граф, задающий информационную структуру;

P – правила функционирования модели, заполнения разметки;

S_0 – начальная разметка.

1.1.2 Типы операторов

В языке определены следующие типы операторов:

Оператор интерпретации (рисунок 1). В качестве входных параметров имеет некую функцию либо оператор и соответствующий набор аргументов. Существует в двух форматах – префиксный (аргументы: функция) и постфиксный (функция^аргументы).

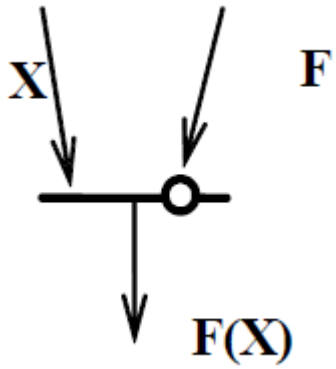


Рисунок 1 – Оператор интерпретации

Константный оператор (рисунок 2). Генерирует константу. Формат записи значение соответствующего типа.

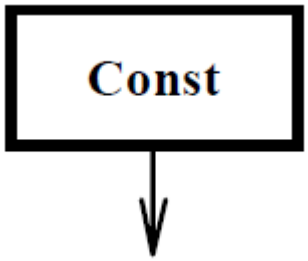


Рисунок 2 – Константный оператор

Оператор копирования (рисунок 3). Вход – некоторая величина. Выход – некоторое количество копий. Записывается в префиксной (величина>>имя) или постфиксной (имя<<величина) формах.

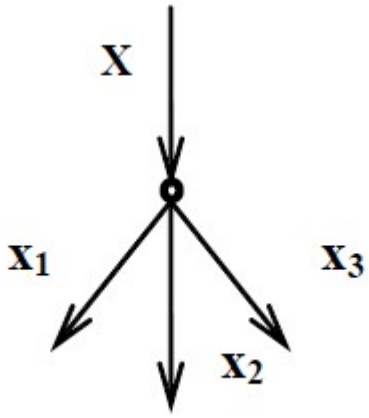


Рисунок 3 – Оператор копирования данных

Оператор группировки в список (рисунок 4). На входе – некое количество значений. На выходе – составленный из этих значений список. Задается в виде ограничения элементов круглыми скобками – (x_1,x_2,x_3,x_4).

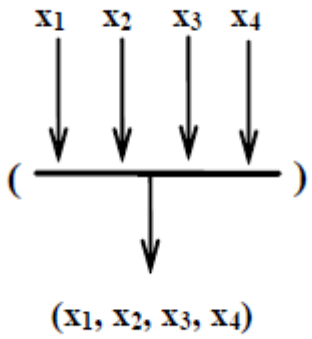


Рисунок 4 – Оператор группировки в список

Оператор группировки в параллельный список (рисунок 5). Аналогичен оператору группировки в список, однако на выходе генерируется параллельный список. Задается в виде ограничения элементов квадратными скобками – $[x_1,x_2,x_3,x_4]$.

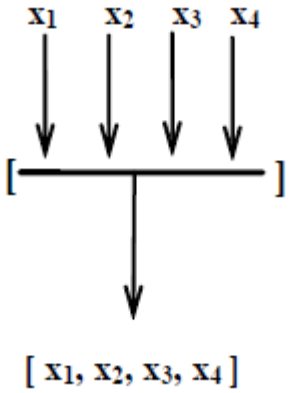


Рисунок 5 – Оператор группировки в параллельный список

Оператор группировки в задержанный список (рисунок 6). Аналогичен оператору группировки в список, однако на выходе генерируется задержанный список. Задается в виде ограничения элементов фигурными скобками – $\{x_1,x_2,x_3,x_4\}$.

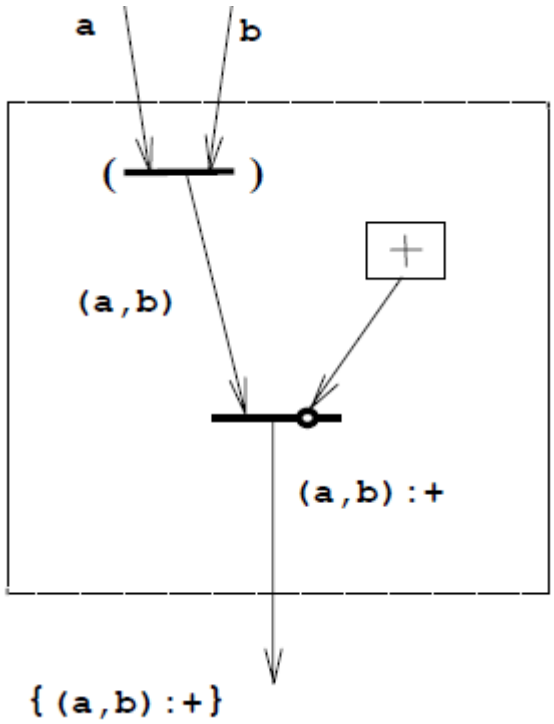


Рисунок 6 – Список задержанных вычислений

Применение обычного списка в качестве аргумента функции имеет смысл лишь в том случае, если это предусмотрено самой функцией. Применение параллельного списка на функцию равнозначно списку применений каждого из элементов списка на функцию:

$[x_1,x_2,x_3,x_4]:F \Rightarrow [x_1:F,x_2:F,x_3;F,x_{4:}:F]$

Задержанный список в качестве элементов содержит фрагменты графа – фактически, фрагменты кода. После произведения операции раскрытия списка на место соответствующих элементов подставляются результаты их выполнения.

1.1.3 Типы данных

В языке определены следующие типы данных:

- *Сигнал (signal)*. Пустое значение, смыслом которого является сам факт его существования. Применение его в качестве аргумента функции равнозначно запуску функции без аргументов. Применение его в качестве функции со списком в виде аргумента реализует операцию раскрытия списка.
- *Целочисленное значение (int)*. Помимо применения как собственно числа, может выполнять функцию выборки или исключения элемента из списка. Положительное целочисленное значение при применении в качестве функции на список возвращает соответствующий по номеру элемент списка. Отрицательное целочисленное значение при применении в качестве функции на список исключает из списка соответствующий по номеру элемент. Если затребованный индекс не существует в списке, возвращается ошибка **BOUNDERROR**.
- *Действительное значение (float).* Соответствует классическому действительному типу. Не имеет нестандартных применений.
- *Символьный тип (char).* Стандартный символьный тип. Не имеет нестандартных применений.
- *Логический тип (bool).* Может выполнять роль клапана: применение значения “истина” на любой атом сохраняет атом. Применение значения “ложь” возвращает пустое значение.
- *Специальные знаки (spec).* Хранят в себе операторы и спецсимволы языка. Применение их равнозначно применению их содержимого.
- *Константы ошибок (error).* Хранят в себе сообщения об ошибках.

Область допустимых значений для констант ошибки задается неупорядоченным множеством, которое в дальнейшем предполагается пополнять. В настоящий момент выделяются следующие ошибки:

- **ERROR** - неидентифицируемая ошибка;
- **REALERROR** - некорректное преобразование действительного числа;

- **INTERROR** - некорректное преобразование целого числа;
- **ZERODIVIDE** - деление на ноль;
- **INTERPRERROR** - ошибка операции интерпретации;
- **BOUNDERROR** - ошибка выхода за границы диапазона;
- **BASEFUNCERROR** – неправильное использование предопределенной функции.

Эти имена запрещается использовать в программе в другом контексте.

Функция - составной объект, конструируемый специальным образом. Она задается определением, начинающимся с ключевого слова **funcdef**. Состоит из заголовка и тела. В заголовке указывается идентификатор аргумента, обеспечивающего передачу в тело функции необходимых данных. В теле описывается алгоритм обработки аргумента. Доступ к исходным данным осуществляется только через аргумент, тип которого и значение в данной версии языка могут быть произвольными. Тело функции состоит из элементов, заключенных в фигурные скобки и разделяемых между собой символом **";"**.

1.1.4 Служебные функционалы

Наряду с описанными ранее операторами, базовый набор языка содержит следующие служебные функционалы.

- Символ ‘.’. Работа его была описана ранее в описании сигналов.
- Символ ‘+’. При воздействии на двухэлементный числовой список производит арифметическое сложение. При воздействии на список логических элементов из двух и более элементов производит логическое сложение (дизъюнкцию).
- Символ ‘-’. При воздействии на двухэлементный числовой список производит арифметическое вычитание. При воздействии на список логических элементов из двух и более элементов производит логическое

сложение по модулю два (исключающее или). При воздействии на одиночное число меняет его знак на противоположный.

- Символ ‘*’. При воздействии на двухэлементный числовой список производит арифметическое умножение. При воздействии на список логических элементов из двух и более элементов производит логическое умножение (конъюнкцию).
- Символ ‘/’. При воздействии на двухэлементный числовой список производит арифметическое деление.
- Символ ‘%’. При воздействии на двухэлементный числовой список производит целочисленное деление с формированием остатка.
- Символы ‘=’,’!=’,’<’,’>’,’<=’,’>=’. Производят соответствующие операции сравнения.
- Символ ‘|’. При воздействии на список возвращает количество его элементов (первого уровня вложенности).
- Символ ‘?’. При воздействии на список возвращает список из номеров его истинных элементов.
- Символ ‘#’. При воздействии на список из списков производит транспонирование его. Результат – список из списков; первый список состоит из первых элементов всех изначальных списков, второй – из вторых и так далее.
- Символы ‘()’,‘[]’,‘{}’. Создают соответствующие списки.
- Символ ‘..’. В качестве входных данных воспринимает двухэлементный или трехэлементный список. На выходе генерируется числовой список, сгенерированный по следующим правилам: первый элемент входных данных – первый элемент списка, второй элемент входных данных – последний элемент списка, третий – шаг между элементами.
- Функция ‘dup’. Генерирует список из одинаковых элементов путем дублирования. В качестве входного аргумента требует двухэлементный

список: первый элемент – объект для дублирования, второй – количество дубликатов.

- Функция ‘type’. Возвращает тип входного аргумента.
- Функция ‘int’. Преобразовывает входной аргумент в целочисленное значение.
- Функция ‘float’. Преобразовывает входной аргумент в действительное значение.
- Функция ‘char’. Преобразовывает входной аргумент в символьное значение.
- Функция ‘bool’. Преобразовывает входной аргумент в логическое значение.
- Функция ‘signal’. Преобразовывает входной аргумент в сигнал.
- Функция ‘datalist’. Аналог ‘()’
- Функция ‘parlist’. Аналог ‘[]’
- Функция ‘delaylist’. Аналог ‘{}’

Разумеется, наряду с базовыми функциями могут использоваться определенные пользователем функции. Определение функции производится следующим образом:

```
F1 << funcdef x
{
 …
return << …;
}
```

Каждая функция должна возвращать некое значение, которое следует присваивать к служебному аргументу return. На данный момент не реализована возможность создавать функции, не возвращающие ничего. На практике их альтернативой являются функции, возвращающие пустое значение.

Возможно объявление нескольких функций с одинаковой сигнатурой. В таком случае, вызов сигнатуры будет вызывать одновременное срабатывание всех функций с таким названием, что приведет к генерации в качестве

результата параллельного списка из результатов всех сработавших функций. К имени перегруженной функции можно добавлять приоритет – числовое значение, которое определит, какой по счету будет вызвана данная перегрузка.

Еще одним важным элементом языка являются блоки. Фактически, *блок* – выделенный набор элементов функции. Служит он для объединения группы операторов, образующих законченное действие, либо для локализации обозначений. Блок выделяется с двух сторон фигурными скобками и предваряется служебным словом block. Выход из блока осуществляется либо по присвоению значения служебному идентификатору break (аналогично идентификатору return для функций), либо по выполнении всех команд.

Ранее было упомянуто об отсутствии вентилей в рамках данной модели. Однако, реализовать их своеобразный аналог возможно и здесь. Для этого можно использовать список из условий и задержанный список из соответствующих им вариантов действий. Список условий подвергаются анализу функцией ‘?’; результат этого в дальнейшем производит выборку нужных элементов из списка вариантов действий.

1.1.5 Синтаксис

Стиль и синтаксис программ разработанных на данном языке отличается от C++ и ему подобных языков. Это несколько затрудняет написание программ.

Пример описания функции суммы чисел списка:

```
Summa << funcdef Param
{
    x << Param : 1;
    y << Param : 2;
    (x, y) : + >> return
}
```

Из списка Param выделяется первый и второй элементы суммы. Следующим этапом возвращается из функции сумма элементов списка.

Вызов функции будет выглядеть так:

```
(1,3):Summa
```

Результатом будет:

```
4
```

Так как язык функционально-потоковый в нём нет цикла. Но цикл можно организовать рекурсией. То есть вызывая функцию из тела самой функции.

```
Cycle << funcdef  param
{
   x << param:1;
   n << param:n;
   return << .^[((x,n):[<,=>]):?]^
   (
        {
            block {
                                ...
            x << (x,1):+;
            (x,n):Cycle  >> break
            }
        },
      param
   )
}
```

Выглядит достаточно громоздко по сравнению с языком C++, аналогичный пример выглядит так:

```
for (i=1;i<=n;i++)
      {
        ...
      }
```

Конечно возможно лучше организовать подобный цикл на Пифагоре, но лучше обходится без них, на сколько это возможно.

1.2 Pifagor IDE

Pifagor IDE представляет собой программный комплекс предназначенный для разработки программ на языке программирования Пифагор (рисунок 7). Разработан Матковским И. В. на языке программирования C++ и библиотек Qt. Содержит утилиты для просмотра, изменения, написания программ в графическом интерфейсе.

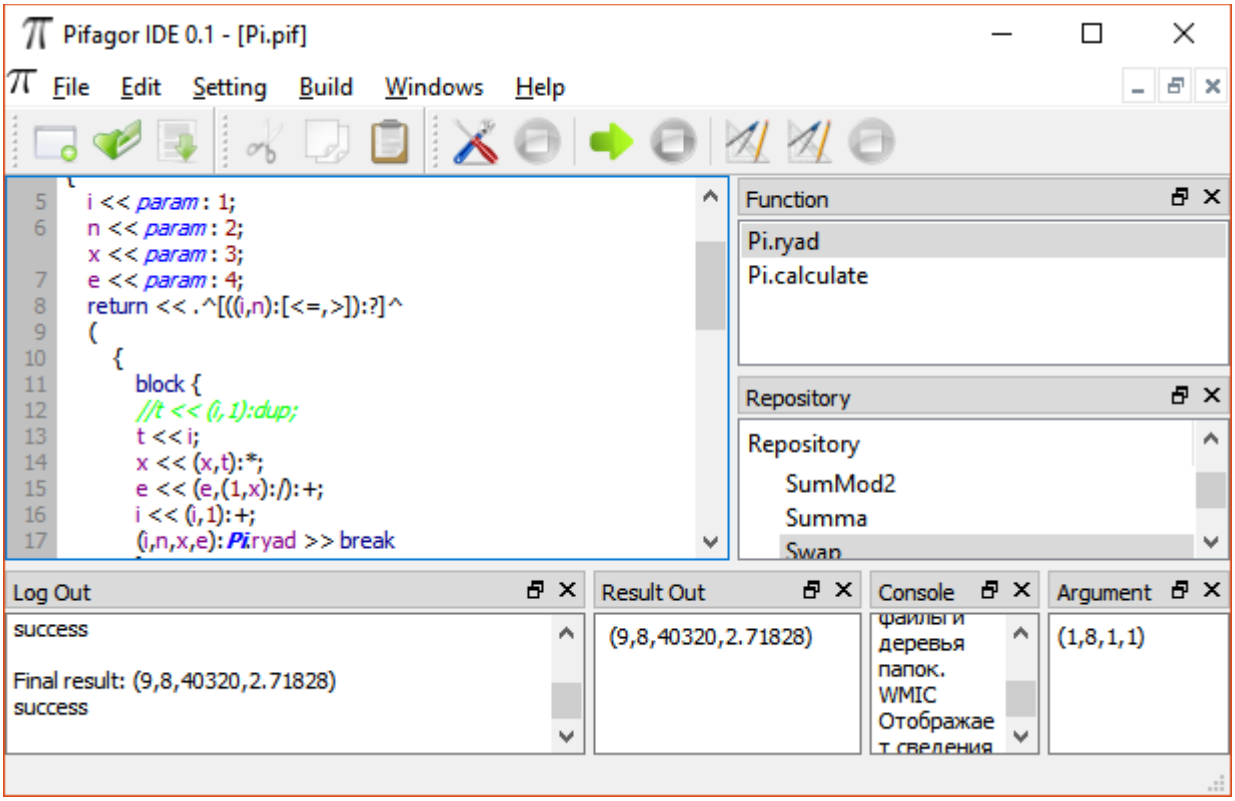


Рисунок 7 – Окно программы IDE Пифагор

1.2.1 Обзор

Просмотр и изменение кода функций в среде обеспечивает окно просмотра содержимого. Окно содержит панель вкладок, обеспечивающую быстрый доступ к содержимому ранее открытых документов, посредством вкладок. Рабочая область окна может быть разделена на несколько окон для просмотра содержимого нескольких ранее открытых документов. Вынесение вкладки за пределы рабочей области окна приводит к открытию содержимого соответствующего документа в отдельном окне.

Просмотр результата работы программ в среде обеспечивает окно просмотра результатов работы утилит. В него выводится информация выходных данных и список ошибок и предупреждений. Окно выходных данных содержит текущий результат работы программы, написанной на языке программирования

Пифагор. Список ошибок и предупреждений хранит соответствующую информацию в виде списка.

Просмотр доступных к выполнению функций в среде обеспечивает окно функций содержащихся в открытом проекте. Функции отображены в виде списка. Из этого списка можно выбрать функцию в которую будут подаваться аргументы.

Окно репозитория содержит дерево репозитория, которые обеспечивают просмотр кода функций находящихся в распределенном репозитории. В окне можно открыть функцию и использовать её код в своём проекте.

Недостаток функционала IDE состоит в нехватке интегрированного инструмента для сортировки функций выше обозначенных функций. Хотя функционал для выделения функций из проекта для внесения в репозиторий есть. Работает автоматически.

1.2.2 Исправление ошибок

Интегрированная среда разработки была предоставлена в исходном коде проекта разработанном в Qt Creator, для дальнейшей компиляции программы. Но при попытке провести операцию компилятор MinGW вывел ошибки о не хватке заголовочных файлов. Проанализировав файлы проекта были исправлены пути их местоположения для правильного определения их компилятором.

С помощью системы управления версиями Subversion был сделана ревизия для автора программы с исправленным файлами проекта.

1.3 Выводы

Язык Пифагор был предназначен для разработки архитектурно-независимых параллельных программ. Оптимизация программ под определенную архитектуру ЭВМ происходит на этапе трансляции. Это позволяет описывать алгоритмы работы программ без особенностей архитектуры под которую разрабатывается программа.

Были обнаружены ошибки работы в IDE, которые были в следствии исправлены для дальнейшей нормальной работы в интегрированной среде разработки.

Требуется разработать инструмент для отсортировки функций и разработать новые функции, для эффективной разработки программ. И интегрировать их в IDE, в составе репозитория.

2 Разработка библиотеки функций

Библиотека функций разделена на категории по назначению функций:

- Операции над векторами (Vector);
  - Поиск минимального элемента вектора (MinEl);
  - Сумма элементов вектора (SumEl);
  - Поэлементное произведение двух векторов (ScalMult);
  - Поиск максимального элемента вектора (MaxEl);
- Операции над матрицами (Matrix);
  - Определитель квадратной матрицы (Detr);
  - Проверка квадратности матрицы (IsSquare);
  - Проверка матрицы на симметричность (IsSymmetry);
  - Умножение матрицы на матрицу (MatrMult);
- Сортировка списка (Sort);
  - Выделение элементов больше первого (FilterSortMs);
  - Выделение элементов меньше первого (FilterSortLs);
  - Выделение элементов равного первому (FilterSortEq);
  - Быстрая сортировка по возрастанию (QuickMs);
- Математические функции (Math);
  - Модуль числа (Abs);
  - Возведения числа в квадрат (PowSqr);
  - Квадратный корень числа (RootSqr);
  - Факториал (Fact)
- и т. д.

В приложении Б указан список функций для языка Пифагор с описанием их применения. Анализируя эти функции было принято разработать ещё ряд функций для более продуктивной разработки.

2.1 Функция вычисления числа Эйлера

e – основание натурального логарифма, математическая константа, иррациональное и трансцендентное число. Приблизительно равно 2,71828. Иногда называют числом Эйлера или числом Непера. Обозначается строчной латинской буквой «e».

Тождество Эйлера:

$$e^{i\pi} + 1 = 0, \tag{1}$$

Поиск числа Эйлера для этой функции на C++:

```
int n, i;
double x = 1;
long double e = 1;

for (i=1;i<=n;i++) // ряд Тейлора, за точность берем количество знаков
после запятой
{
	x = x*i;
	e = e + 1/x;
}
```

На языке Пифагор поиск числа Эйлера выглядит так:

```
Math.CalcNumE << funcdef param{
i << param : 1; n << param : 2; // i – итерация, n - точность
x << param : 3; e << param : 4; // x – основание, e – число Эйлера
return << .^[((i,n):[<=,>]):?]^
({
	block {
	t << i; x << (x,t):*;
	e << (e,(1,x):/):+;
	i << (i,1):+;
	(i,n,x,e):Math.CalcNumE >> break
	}},param)}
```

2.2 Функция вычисления числа Пи

Число Пи - математическая константа, равная отношению длины окружности к длине её диаметра. То, что отношение длины окружности к диаметру одинаково для любой окружности, и то, что это отношение немногим более 3, было известно ещё древнеегипетским, вавилонским, древнеиндийским и древнегреческим геометрам.

Ряд Лейбница описывает число Пи такой закономерностью:

$$\frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots = \frac{\pi}{4} \quad (2)$$

Из него делаем вывод формулы для нахождения числа Пи:

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} - \frac{1}{15} + \frac{1}{17} - \frac{1}{19} + \frac{1}{21} - \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}. \quad (3)$$

2.3 Вывод

В результате разработаны функции для репозитория языка программирования Пифагор. Были предложены категории, по которым группируются функции по назначению. Так же были разработаны математические функции вычисления числа Пи, функция вычисления числа Эйлера и другие.

3 Модуль работы с репозиторием

Изначально необходимо точно сформулировать основные идеи, которые определяют функционал модуля работы с репозиторием. Грамотно составленные требования к работе являются гарантией того, что она будет выполнена максимально качественно и будет иметь весь необходимый функционал.

Таким образом, к разрабатываемую модулю работы с репозиторием предъявляются следующие требования:

- Модульность – инструмент должен быть модулем для лёгкого встраивания его в уже имеющуюся интегрированную среду разработки;
- Функция «Поиск» – необходима поисковая строка, для организации поиска необходимой функции по репозиторию.

3.1 Выбор технологий и инструментов

Для разработки модуля работы с репозиторием используется Qt библиотеки и язык C++, по причине разработанного на них IDE Пифагор, а так же имеющегося опыта работы с ними. В качестве среды разработки используется Qt Creator 3.6.1.

3.1.1 Qt

Qt – кроссплатформенный инструментарий разработки ПО на языке программирования C++. Qt позволяет запускать написанное с его помощью ПО в большинстве современных операционных систем путём простой компиляции программы для каждой ОС без изменения исходного кода. Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и

заканчивая классами для работы с сетью, базами данных и XML. Qt является полностью объектно-ориентированным, легко расширяемым и поддерживающим технику компонентного программирования.

Отличительная особенность Qt от других библиотек – использование Meta Object Compiler (MOC) – предварительной системы обработки исходного кода. MOC позволяет во много раз увеличить мощь библиотек, вводя такие понятия, как слоты и сигналы. Кроме того, это позволяет сделать код более лаконичным[10].

3.2 Разработка

Модуль интегрирован в IDE как Dock-виджет из класса QDockWidget. Это позволяет его перемещать в области окна или скрыть его. Dock-виджет содержит в себе слой, вертикально построенных два виджета: поиска функций findFuncRepos и дерева репозитория repList.

```
    repDock = new QDockWidget(tr("Repository"));
    repDock->setObjectName("repDock");
    QVBoxLayout *headLayout = new QVBoxLayout();
    headLayout->addWidget(findFuncRepos);
    headLayout->addWidget(repList);
    headLayout->setMargin(0);
    headLayout->setSpacing(0);
    QWidget *wgt = new QWidget();
    wgt->setLayout(headLayout);
    repDock->setWidget(wgt);
    addDockWidget(Qt::RightDockWidgetArea, repDock);
```

Виждет поиска функций состоит из текстовой метки lbl класса QLabel и текстового поля findLineEdit класса QLineEdit, построенных горизонтально. В текстовой метке записано «Search» для указания назначения текстового поля.

```
FindFuncRepos::FindFuncRepos(QWidget *parent)
    : QWidget(parent)
{
    QLabel *lbl = new QLabel("Search");
    findLineEdit = new QLineEdit();

    QHBoxLayout *mainLayout = new QHBoxLayout();
    mainLayout->addWidget(lbl);
    mainLayout->addWidget(findLineEdit);
    mainLayout->setMargin(2);
```

```
    //headLayout->addWidget(f_RepList);

    this->setLayout(mainLayout);

}

QString FindFuncRepos::Find()
{
    QString str = findLineEdit->text();
    return str;
}
```

Системой Qt сигнал-слот было организовано соединение события измениния текста в поле findLineEdit и функции поиска findFunction.

```
connect(findFuncRepos->findLineEdit, SIGNAL(textChanged(QString)), this,
SLOT(findFunction()));
```

При любом изменении текста вызывается функция findFunction – поиск функций по дереву репозитория.

```
void Pif_IDE::findFunction()
{
    QString str = findFuncRepos->Find();
    repList->FindFunc(str);
}
```

Из текстового поля получаем текстовую строку и производим по её содержанию поиск. Результат записывается в список объектов дерева. Если объектов в списке больше нуля, то первый в списке объект выделяется setCurrentItem в дереве репозитория.

```
void Rep_List::FindFunc(QString str)
{
    QList<QTreeWidgetItem *> found = findItems(str, Qt::MatchRecursive);
    if (found.size() != 0)
    {
        setCurrentItem(found[0]);
    }
}
```

Само дерево Rep_List репозитория строится функцией WorkDir из файлов функций языка Пифагор находящихся в папке репозитория. Структурная схема репозитория приведена в приложении А.

```
Rep_List::Rep_List(QWidget *parent) : QTreeWidget(parent)
{
      setColumnCount(1);
    setHeaderLabel("Repository");

      showAction = new QAction(QIcon("./images/file.png"), tr("Open File"),
this);
```

```
      showAction->setStatusTip(tr("Open original file"));

    newFileAction = new QAction(QIcon("./images/new.png"), tr("Open as new
file"), this);
    newFileAction->setStatusTip(tr("Open as new file"));

      showRIGjpg = new QAction(QIcon("./images/graf.png"), tr("Open RIG"),
this);
      showRIGjpg->setStatusTip(tr("Open RIG.jpg"));

      showCGjpg = new QAction(QIcon("./images/graf.png"), tr("Open CG"), this);
      showCGjpg->setStatusTip(tr("Open CG.jpg"));
}
void Rep_List::createList()
{
      clear();
    workDir(stdValue.pathRepository,"",NULL);
}
void Rep_List::workDir(QString pathPrefix, QString subPath, QTreeWidgetItem
*parent)
{
    QString fullPath=pathPrefix+"\\"+subPath;
    QDir dir;

    dir.setFilter(QDir::AllDirs | QDir::NoSymLinks);
    dir.setSorting(QDir::Name);
    dir.setPath(fullPath);
    QFileInfoList list = dir.entryInfoList();
    QString currentPath;

    QTreeWidgetItem *item;

    for(int i = 0; i < list.count(); i++)
    {
        currentPath=list.at(i).fileName();
        if(currentPath != "." && currentPath != "..")
        {
            if(parent == NULL)
                addTopLevelItem(item = new QTreeWidgetItem());
            else
                item = new QTreeWidgetItem(parent);
            item->setText(0, currentPath);
            workDir(fullPath,currentPath,item);
        }
    }
}
```

3.3 Выводы

Разработан модуль работы с репозиторием на языке программирования C++ с использованием библиотек Qt.

По мере разработки модуля работы с репозиторием появилась идея организации дополнительного функционала «Живого» поиска. Он

подразумевает собой подсветку искомой функции в дереве репозитория по мере набора части названия функции.

4 Примеры применения

4.1 Функции языка Пифагор

После проведения теста на максимальную точность было выявлено, что функция вычисления числа Эйлера на данный момент имеет максимальную точность в пять знаков (рисунок 8).

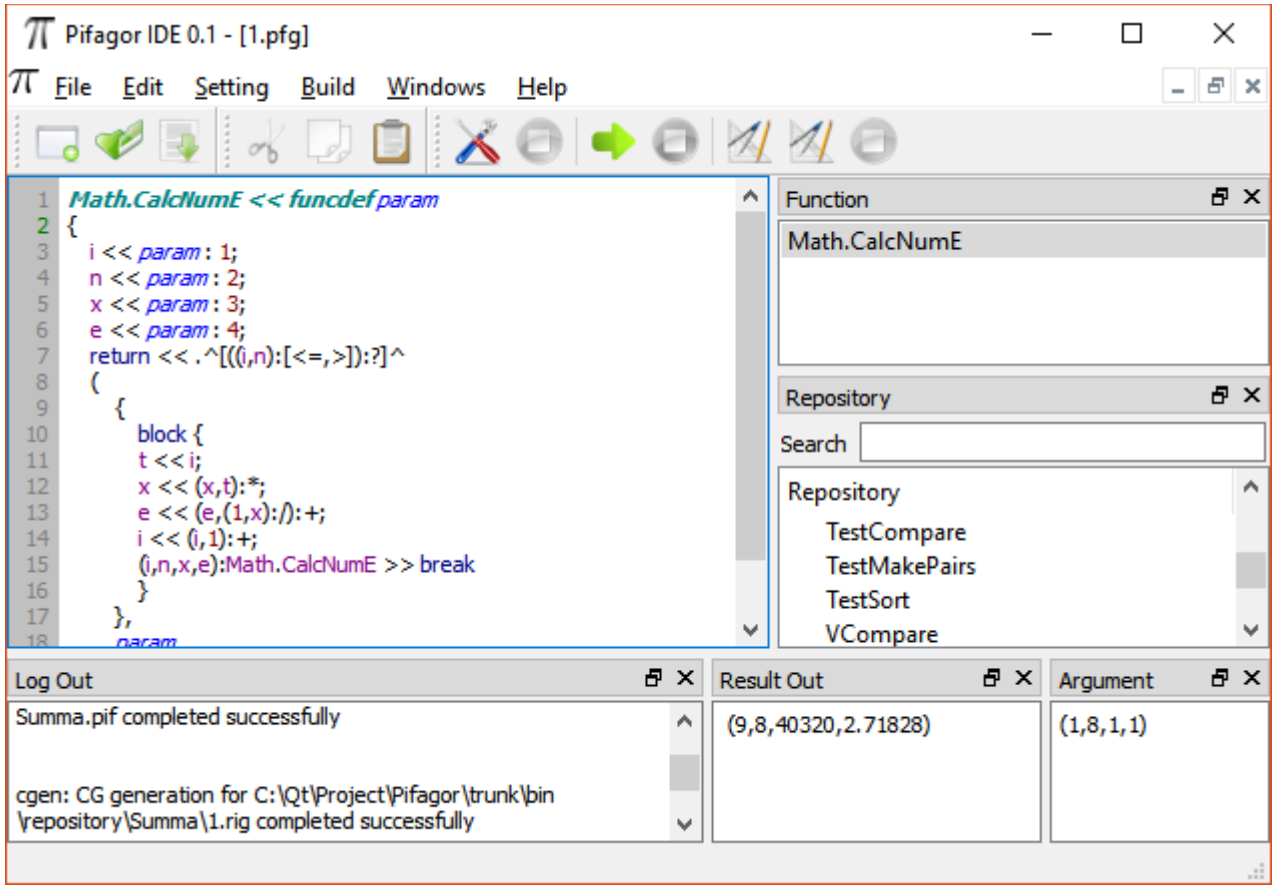


Рисунок 8 - Результат работы функции вычисления числа Эйлера

4.2 Модуль работы с репозиторием

Отдельное окно модуля показано на рисунке 9. Репозиторий функций отображается в виде дерева. Функции одной категории вложены в группу

одного назначения, пример abs – функция возвращает модуль числа, находится в категории Math.

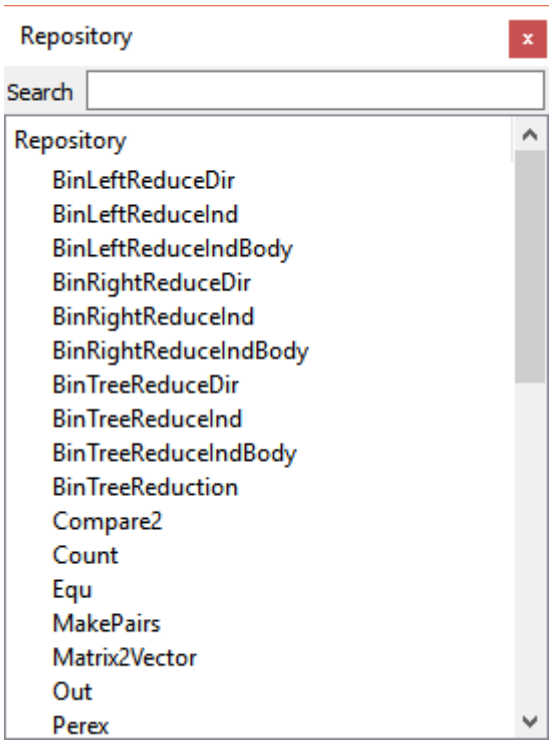


Рисунок 9 - Скриншот модуля работы с репозиторием

Функция поиска работает при наборе текста в соответствующей области. При наборе полного названия функции, сразу в дереве репозитория выделяется искомая функция языка Пифагор (рисунок 10).

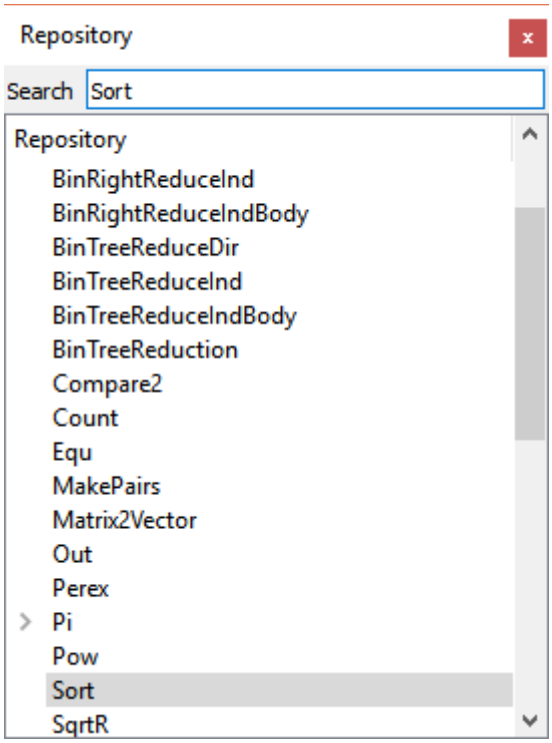


Рисунок 10 - Результат поиска

4.3 Вывод

Протестированы разработанные функции и модуль работы с репозиторием. В ходе тестирования модуля работы с репозиторием ошибок не обнаружено. Функции библиотеки производят вычисления в рамках имеющегося транслятора. Они справляются с поставленными задачами, исполняя назначенный им функционал.

ЗАКЛЮЧЕНИЕ

Был рассмотрен и изучен функционально-потоковый язык программирования Пифагор, цель которого разработка архитектурно независимых параллельных программ. Проанализирована и дополнена интегрированная среда разработки IDE Pifagor, разработанным модулем работы с репозиторием. Разработан комплект функций репозитория на языке Пифагор для пользования ими в дальнейшем разработчиками. Проведено тестирование разработанного программного обеспечения.

Таким образом выполнены все задачи поставленные выпускной квалификационной работой.

СПИСОК СОКРАЩЕНИЙ

ВС – Вычислительная система

ЭВМ – Электронная вычислительная машина

IDE – Интегрированная среда разработки (англ. Integrated Development Environment)

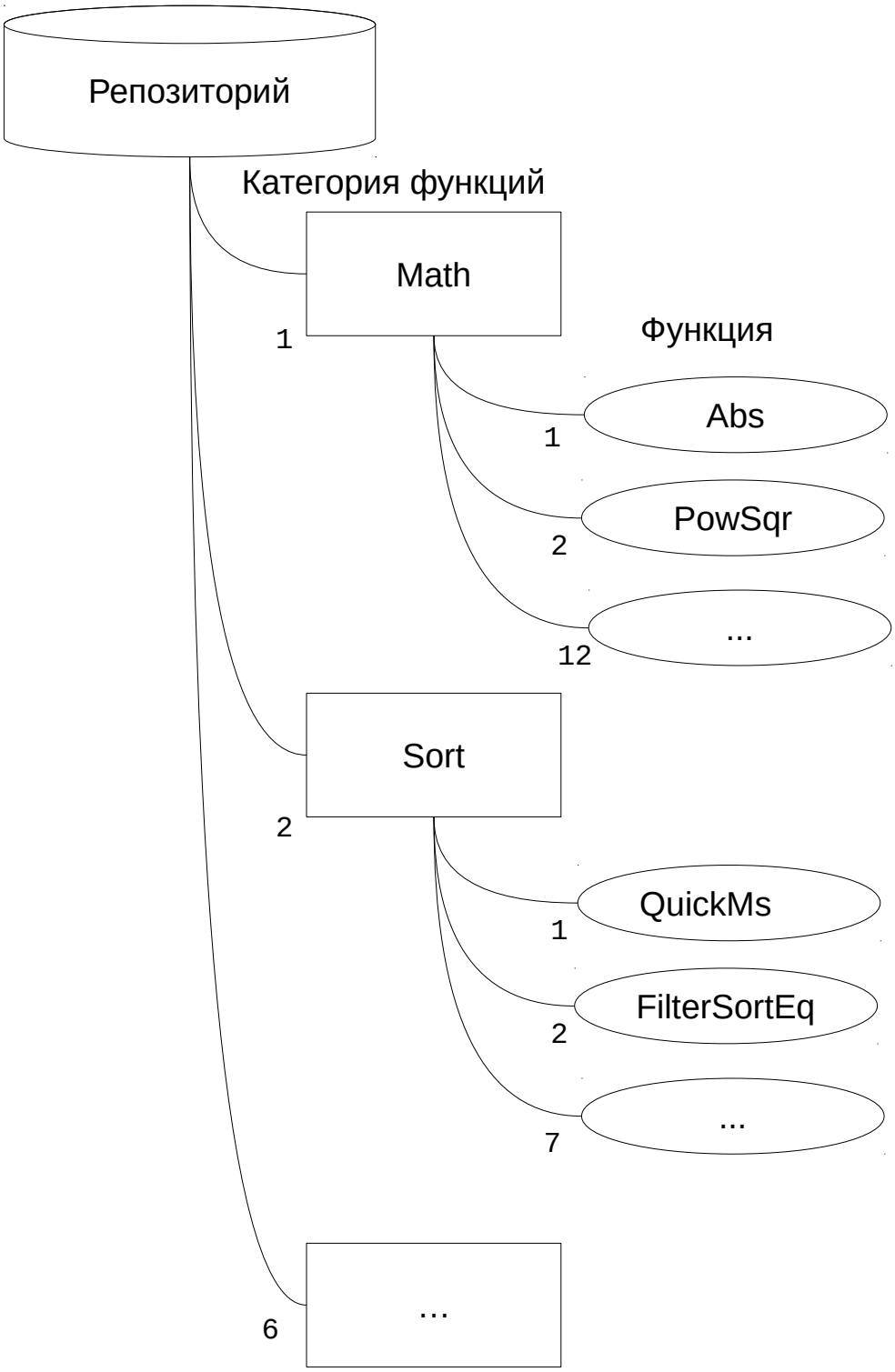
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19.701–90 Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. – Введ. 01.01.1992. – Москва : Стандартинформ, 2010. – 23 с.
2. ГОСТ Р 7.0.5–2008 Система стандартов по информации, библиотечному и издательскому делу. Библиографическая ссылка. Общие требования и правила составления. – Введ. 01.01.2009. – Москва : Стандартинформ, 2008. – 23 с.
3. Легалов, А. И. Функциональная модель параллельных вычислений и язык программирования "Пифагор". [Электронный ресурс] / А. И. Легалов, Ф. А. Казаков, Д. А. Кузьмин, Д. В. Привалихин. // Режим доступа: http://www.softcraft.ru/parallel/fpp/fppcontent.shtml
4. Легалов, А. И. Функциональный язык для создания архитектурно-независимых параллельных программ. / А. И. Легалов // Вычислительные технологии, № 1 (10) – 2005. – С. 71-89.
5. Преддипломная практика и итоговая государственная аттестация: учебно-методическое пособие [Электронный ресурс] / сост.: О. А. Русанова, В. И. Иванов, А. И. Постников. – Электрон. дан. – Красноярск: Сиб. федер. Ун-т, 2012. – 52 с.
6. СТО 4.2–07–2014 Система менеджмента качества. Общие требования к построению, изложению и оформлению документов учебной деятельности. – Введ. 30.12.2013. – Красноярск: ИПК СФУ, 2014. – 60 с.
7. Удалова, Ю. В. Отладка и верификация функционально-потоковых параллельных программ : дис. … канд. тех. наук : 05.13.11 / Удалова Юлия Васильевна. – Красноярск, 2014. – 170 с.

8. Хахаев, И. А. Технологии обработки текстовой информации в LibreOffice / И. А. Хахаев, В. Ф. Кучинский. – Санкт-Петербург: Университет ИТМО, 2016. – 143 с.
9. Шлее, М. Qt 4.8. Профессиональное программирование на C++ / Макс Шлее. – Санкт-Петербург: БХВ-Петербург, 2012. – 912 с.
10. Qt [Электронный ресурс] : Материал из Википедии — свободной энциклопедии, Авторы Википедии // Википедия, свободная энциклопедия. — Электрон. дан. — Сан-Франциско: Фонд Викимедиа, 2016. — Режим доступа: https://ru.wikipedia.org/wiki/Qt;

ПРИЛОЖЕНИЕ А

Схема структуры репозитория



ПРИЛОЖЕНИЕ Б

Таблица функций

Имя файла	Имя функции	Описание функции
_sort.pfg	BinTreeReduction	Свертка вектора заданной функцией
	change	Превращение аргумента в параллельный список
	Compare2	Проверка, во всех ли заданных парах (X,Y) X<Y и переворот "неправильных" пар
	Equ	Сравнение двух чисел (проверка на равенство)
	MakePairs	Разбиение вектора на пары с возможным переворотом пар
	Matrix2Vector	Преобразование матрицы в вектор
	Sort	Исходный вектор бьется на пары, все они переворачиваются, после чего проверяется, все ли они после этого встали "правильно"
	SumMod2	Сравнение двух чисел (проверка на неравенство)
	Swap	Сортировка пары целых чисел
	TestCompare	Проверка функции Compare
	TestMakePairs	Проверка функции MakePairs
	TestSort	Проверка функции Sort
	VCompare	Сравнение двух векторов
	VSum	Сумма элементов вектора
_tmp.pfg	div	Самодельное целочисленное деление (div)
	div_modTest	Проверка функций mod и div
	divmod	Самодельный div+mod
	divmodTest	Проверка функции divmod
	LongDivMod	Расчет результата деления X на Y с точностью Z-1 знаков после запятой
	LongDivModTest	Тест функции LongDivMod
	Loop	Функция, возвращающая задержанный вызов самой себя
	LoopTest	Бесконечный самовызов функции Loop
	mod	Самодельный остаток (mod)
	notdivmodTest	Проверка операции %
	xxx	Проверка генераторов последовательностей
alpha_reduction.pfg	BinLeftReduceDir	Левая бинарная свертка
	BinLeftReduceInd	Левая бинарная свертка в заданном диапазоне
	BinLeftReduceIndBody	Интерфейс для BinLeftReduceIndBody
	BinRightReduceDir	Правая бинарная свертка
	BinRightReduceInd	Правая бинарная свертка в заданном диапазоне
	BinRightReduceIndBody	Интерфейс для BinRightReduceIndBody
	BinTreeReduceDir	Древовидная бинарная свертка
	BinTreeReduceInd	Интерфейс для BinTreeReduceIndBody
	BinTreeReduceIndBody	Древовидная бинарная свертка в заданном диапазоне
binary.pfg	is_reflective	Проверка матрицы на рефлексивность
	is_symmetry	Проверка матрицы на симметричность
	is_transitive	Проверка матрицы на транзитивность
	select	Выборка элемента из списка
	transitive	Вспомогательная функция для is_transitive
	VAnd	Логическое И от списка логических элементов
eqobj.pfg	EqObjAll	Поиск объекта в списке
	EqObjSort	Поиск объекта в списке
	TestEqObjAll	Проверка EqObjAll
	TestEqObjSort	Проверка EqObjSort
fact01.pfg	fact01	Факториал с проверкой типа аргумента
	SeqFactRec	Факториал
	Test2Fact01	Проверка fact01
	TestFact01	Проверка fact01

продолжение Приложения Б

fact02.pfg	fact02	Факториал с проверкой типа аргумента
	fact02_0	Факториал 0
	fact02_1	Факториал 1
	fact02_all	Вызов вычисления факториалов для всех возможных типов входных данных
	fact02_neg	Факториал отрицательного
	fact02_pos	Факториал положительного
	Test2Fact02	Проверка fact02
	TestFact02	Проверка fact02
fact03.pfg	fact03	Факториал
	overfact	Перегруженная функция расчета факториала для разных аргументов
	Test2Fact03	Проверка fact03
	TestFact03	Проверка fact03
hanoy.pfg	recHanoy	Задача о X ханойских башнях
	rHanoy	Вспомогательная функция для RecHanoy
john.pfg	abs	Модуль
	NOD	НОД двух чисел
	NOD_r	НОД двух чисел
	NOD_V	НОД нескольких чисел
math.pfg	math.abs	Модуль
	math.fact	Факториал
mathlit.pfg	Pi	Число Пи
	PowMinOne	Возведение -1 в степень X
matrix01.pfg	Add_VV	Сложение двух векторов
	Delete_Mik	Удаление i-ой строки и k-го столбца матрицы
	Delete_Vi	Удаление i-ой строки матрицы
	Det01r_M	Определитель квадратной матрицы
	Det02r_M	Определитель квадратной матрицы
	DiagM_V	Построение диагональной матрицы из вектора
	ElemMult_VV	Поэлементное произведение двух векторов
	Insert_Mijx	Вставляет в матрицу заданный элемент на заданную позицию
	Insert_V_ix	Вставляет в вектор заданный элемент на заданную позицию
	IscorRect_M	Проверка корректности (прямоугольности) матрицы
	IsSoglas_MM	Проверка согласованности матриц
	IsSquare_M	Проверка квадратности матрицы
	Mult_MS	Умножение матрицы на скаляр
	Mult_MV	Умножение матрицы на вектор
	Mult_VS	Умножение вектора на скаляр
	ScalMult_VV	Скалярное произведение векторов
	Sub_VV	Разность двух векторов
	Sum_V	Сумма элементов вектора
	znak	Возведение -1 в степень X

maxvec.pfg	VMax	Поиск максимального элемента вектора
mergesort.pfg	merge	Вспомогательная функция (слияние)
	mergesortN	Сортировка слиянием
	mergesortTest	Тест сортировки слиянием
mult.pfg	math.mult.cinco	Возврат 5X
	math.mult.cuatro	Возврат 4X
	math.mult.dos	Возврат 2X
	math.mult.list	Перемножение элементов двухэлементного списка
	math.mult.lister	Вызов всех функций файла
	math.mult.ocho	Возврат 8X
	math.mult.seis	Вовзрат 6X
	math.mult.siete	Возврат 7X
	math.mult.tres	Возврат 3X
	math.mult.uno	Возврат X
osj.pfg	ScalMultVec	Поэлементное произведение двух векторов
	SeqVecMult	Правосворачивающее скалярное произведение векторов
	SeqVecSum	Правосворачивающее сложение вектора
	VecMult	Скалярное произведение векторов
	VecSum	Сумма элементов вектора
parfact.pfg	parfact.fact	Параллельное получение факториала
	parfact.parfact	Вспомогательная функция для parfact.fact
parmin.pfg	ParMin_01	Поиск минимального элемента вектора
part3_exampl.pfg	Abs	Модуль числа
	Abs2	Модуль числа
	AbsAdd	Сумма модулей двух чисел
	BinTreeReduction	Свертка вектора заданной функцией
	BinTreeReductionTest	Тест BinTreeReduction
	Min	Возврат минимального из двух чисел
	ParAddSubMultDiv	Сложение, вычитание, умножение и деление
	VecScalMult	Умножение вектора на скаляр
	VecScalMultBrief	Умножение вектора на скаляр (минимизированная версия)
	VecScalMultNoComment	Умножение вектора на скаляр (без комментариев в коде)
	VecSum	Сумма элементов вектора
	VecSum2	Сумма элементов вектора
productions.pfg	BinTreeReduceDir	Древовидная свертка списка заданной функцией
	EachOpd1WithOpd2	Построение пар из всех комбинаций элементов списка и дополнительного элемента
	MatrMatrProd	Умножение матрицы на матрицу
	Opd1WithEachOpd2	Построение пар из всех комбинаций дополнительного элемента и элементов списка
	VecMatr1Prod	Умножение вектора-строки на вектор-столбец
	VecMatrProd	Умножение вектора на матрицу
	VecVecProd	Перемножение двух векторов одной длины
quicksort1.pfg	hifilter	Выделение элементов больше или равных первому
	hisort	Выделение и сортировка элементов больше или равных первому
	lofilter	Выделение элементов меньше первого
	losort	Выделение и сортировка элементов меньше
	quicksortN	Быстрая сортировка
	quicksortTest	Проверка quickSortN

окончание Приложения Б

rec.pfg	ack	Функция Аккермана
	C	Для (X,Y) количество сочетания из Y по X
	fac.1	Факториал
	fac.2	Факториал
	fib	Расчет числа Фиббоначи с заданным номером
	gcd	НОД двух чисел
semant.pfg	sem.all	Общий тест
	sem.calls	Проверка вызовов
	sem.comp	Проверка сравнений
	sem.div	Проверка делений
	sem.f1	Тестовая функция
	sem.f2	Тестовая функция
	sem.idiv	Проверка целочисленных делений
	sem.list	Проверка операций на списках
	sem.minus	Проверка вычитания
	sem.mult	Проверка умножения
	sem.plus	Проверка сложения
	sem.typeofs	Проверка тайпофов
	sem.types	Проверка типов
simple.pfg	sqr	Возведение числа в квадрат
	abs	Модуль числа
	sqrt	Квадратный корень числа
	SqrtR	Вспомогательная функция для sqrt
	square_triangle	Площадь треугольника по трем сторонам (формула Герона)
	sum_of_squares	Вовзрат суммы квадратов двух чисел
sort.pfg	getmind	Вспомогательная функция
	getsort	Основной сортировщик
	mind	Вспомогательная функция
theorsort.pfg	all_compares	Получение "правильного индекса" для каждого из элементов списка
	all_duplicats	X списка из X копий каждого из элементов входного списка где X - длина входного списка
	all_patterns	X копий входного списка где X - длина списка
	BinTreeReduction	Свертка вектора заданной функцией
	GE	Подсчет числа неотсортированных пар в списке
	left_vec	Возврат X-1 первого элемента списка
	replace	Установка элементов списка в порядке, заданном в другом списке
	right_vec	Возврат списка без X левых элементов
	test_all_compares	Проверка all_compares
	test_all_duplicats	Проверка all_duplicats
	test_all_patterns	Проверка all_patterns
	test_left_vec	Проверка left_vec
	test_replace	Проверка replace
	test_right_vec	Проверка right_vec
	test_theorsort	Тест theorsort
	theorsort	Теоретическая сортировка
vecmin.pfg	BinTreeReduction	Свертка вектора заданной функцией
	BinTreeReductionTest	Поиск минимума в векторе через свертку
	Min	Возврат минимального из двух чисел
	VecMin1	Возврат минимального элемента вектора
vector2.pfg	ForAll	Вспомогательная функция для __VVMult
	_VVMult	Перемножение двух векторов поэлементно (через пар-список, в лоб)
	__VVMult	Перемножение двух векторов поэлементно (через ForAll)

ПРИЛОЖЕНИЕ В

Листинг функций на Пифагоре

```
// Внутреняя фукция для вычисления Пи
// ряд Тейлора, за точность берем количество знаков после запятой
Pi.ryad << funcdef param
{
    i << param : 1;
    n << param : 2;
    x << param : 3;
    e << param : 4;
    return << .^[((i,n):[<=,>]):?]^
    (
        {
            block {
            //t << (i,1):dup;
            t << i;
            x << (x,t):*;
            e << (e,(1,x):/):+;
            i << (i,1):+;
            (i,n,x,e):Pi.ryad >> break
            }
        },
        param
    )
}

// Вычисляет число Пи с заданной точностью
Pi.calculate<< funcdef param
{
   n << param;
   x << 1;
   e << 1;
   i << 1;
   return << (i,n,x,e):Pi.ryad
}

// Через ряд Лейбница
Pi.Leib << funcdef
}
    X1<< 4 ;
    X2 << (X1,3):/:- ;
    X3 << (X1,5):/;
    X4 << (X1,7):/:- ;
    X5 << (X1,9):/;
    X6 << (X1,11):/:- ;
    X7 << (X1,13):/;
    X8 << (X1,15):/:- ;
    X9 << (X1,17):/;
    X10 << (X1,19):/:- ;
    X11 << (X1,21):/;
    return <<((((((((((X1,X2):+,X3):+,X4):+,X5):+,X6):+,X7):+,X8):+,X9):+,
X10):+,X11):+;
}

MultOf2 << funcdef x {(x:1,x:2):* >> return}

Count << funcdef x {(x,1):+ >> return}
```

```
Perex<< funcdef { (6,7,8,9,0):. >> return;}

VecMultEl << funcdef Param
{
    len << Param:|;
    return << .^[((len,2):[<,=,>]):?]^
    (
        {Param:[]},
        {Param:*},
        {((Param:1,Param:2):*,((Param:-1):-1))}
        }
    )
   /*
    len << Param:2:|;
    C << 1;
    X << Param :1;
    return << .^[((C,len):[<,=>]):?]^
    (
        {
            block {
            C << (C,1):+;
            X << (X,Param:C):*
            (c,x):Mult >> break
            }
        },
        param
    )
*/
}
VecMult<< funcdef Param
{
Len<<Param:|;
[((Len,2):[<,=,>]):?]^
(
 {Param:[]},
 {Param:*},
 {block{
  OddVec<< Param:[(1,Len,2):..]:.;
  EvenVec<< Param:[(2,Len,2):..]:.;
  (([OddVec,EvenVec]:VecMult):*)>>break;
 }}
):.>>return
/*
    len << Param:|;
    C << 1;
    X << Param :1;
    return << .^[((C,len):[<,=>]):?]^
    (
        {
            block {
            C << C:Count;
            X << (X,Param:C):*
            (i,n,x,e):Pi.ryad >> break
            }
        },
        param
    )
*/
}
```

```
//Возведение числа в заданную степень
Pow << funcdef Param
{
    num << Param :1;
    pw << Param :2;
    iter << Param :3;
    V << (num,pw) :dup;
    return << V:VectMultEl
}

//-------------------------------------------------------------------------------
// Функция, осуществляюшая выполнение вычислений
// с использованием хвостовой рекурсии и накопителя
//
Mult.Scal.Vect.Rec.Tail << funcdef Matr2xN_Vec
{
  // Выделение матрицы
  Matr2xN << Matr2xN_Vec:1;
  // Выделение вектора
  vec << Matr2xN_Vec:2;
  // Нахождение длины векторов
  vecLen << Matr2xN:1:|;
  // Проверка длины векторов и переход к альтернативным вычислениям
  [(vecLen,0):(=,>):?]^
  (
    // При ((x),(y))
      vec,
    // Иначе - рекурсивное вычисление
      { ((Matr2xN:[]:-1), (vec:[], (Matr2xN:[]:1):*)):Mult.Scal.Vect.Rec.Tail }
  ):. >> return
}

//-------------------------------------------------------------------------------
// Функция, осуществляюшая запуск функции с хвостовой рекурсией
//
Mult.Scal.Vect.Tail << funcdef Matr2xN
{
  (Matr2xN, ()):Mult.Scal.Vect.Rec.Tail >> return
}


sin << funcdef X
{
arg<<X:1;
x<< ((arg,3.141592653589793):*,180):/;
return<< (x,(1,((x,x):*,(0.16605,(0.00761,(x,x):*):*):-):*):-):*;
};

cos << funcdef X
{
arg<<X:1;
x<< ((arg,3.141592653589793):*,180):/;
return<< (1,((x,x):*,(0.4967,(0.03705,((x,x):*,(x,x):*):*):*):-):*):-;
};

tan << funcdef X
{
arg<<X;
return<< (arg:sin,arg:cos):/;
};
```

```
ctan << funcdef X
{
arg<<X:1;
return<< (arg:cos,arg:sin):/;
};

matrix_element<< funcdef X
{
matr<<X:1;
i<<X:2;
j<<X:3;
i1<<(i,1):+;
j1<<(j,1):+;
return<<matr:i1:j1
}

Trasponirov<< funcdef X //транспонирование матриц обычных и нижнедиоганаьных
{
M<<X:1;
return<<(M:#)
}

Tras<< funcdef X //транпонирование верхне диогональной матрицы
{
M<<X:1;
n<<M:|;
return<<((M,1):Trasponirov_vtr)
}


Trasponirov_vtr<< funcdef X
{
M<<X:1;
n<<M:|;
i<<X:2;
[( (i,n):[>,=<]):?]^
(
                  {.},
 {
   ((M,i,1,i):Trasponirov_vtr_dop,(M,(i,1):+):Trasponirov_vtr)

 }
)
:[]
>>return
}

Trasponirov_vtr_dop<< funcdef X
{
M<<X:1;
//n<<M:|;
n<<X:2;
i<<X:3;
t<<X:4;
[( (i,n):[>,=<]):?]^
(
 {.},
 {
   []^(M:i:t,(M,n,(i,1):+,(t,1):-):Trasponirov_vtr_dop:[])
```

```
 }
):([])
>>return
}

//Метод Гаусса для систем
//Данные - матрица и вектор (((1,1,-1)(2,1,1)(1,-1,-1)),(0,8,-2))

String << funcdef S
{ S:[]:[]:[]:(.)>>return }

VecScalMult << funcdef V
//((вектор),скаляр)
{
X<<V:1;
Y<<(V:2,X:|):dup;
((X,Y):#:[]:*)>>return
}

Podmatrica << funcdef P
//(A,a)
{
return<<.^[((1,P:2):[<=,>]):?]^
  ( {block{
     B<<(P:1):[]:-1;
     A<<(B,(P:2,1):-):Podmatrica;
     A>>break}},{P:1})
}

Socr << funcdef R  //приводим остальные строки к виду (0,x2,..)
//(-Osn,A:i)
{
Mnoj<<(R:2):1;
(((R:1,Mnoj):VecScalMult,R:2):#:[]:+)>>return
}

Stupen << funcdef S
//(((a11,..,a1m,b1),..,(an1,..,anm,bn)),X,a,b)
{
M<<(S:1,S:3):Podmatrica;     //выделили приводимую подматрицу
OsnM<<(((S:2,-1):VecScalMult,M:|):dup,M):#:[]:Socr;  //приводим остальные строки
к виду (0,x2,..)
N<<(S,X:|,S:3):Podmatrica;
(N,OsnM)>>return //заменям строки старой матрицы на приведенные к виду (0,x2,..)
}

Privedenie << funcdef F
//(((a11,..,a1m,b1),..,(an1,..,anm,bn)),n,m,a,b,i,j )
//где a,b - индексы, с которых начинать приведение
{
i<<F:6;
j<<F:7;
G<<F:1;    //A - матрица,
return<<.^[(j,F:3):(<,=>):?]^
(
      {
     {[(i,n):(<=,>):?]^                                  //j<=n
       (
         { [(G:i:j,0):(!=,=):?]^                           // =1 при A[a][b]!=0
и =2 при A[a][b]=0
           (
```

```
             {                                          //A[a][b]!=0
              block
               {
               Norm<<(1,G:i:j):/;
               X<<(G:i,Norm):VecScalMult;    //Строка G[i] приведена к виду
X=(1, x2,.., xn)-основная
               B<<(G,X,F:4,F:5):Stupen;
                (B,n,m,(a,1):+,(b,1):+,(a,1):+,(b,1):+):Privedenie  //переходим
на след ступень приведения
              >>break
               }
             },
             {(G,n,m,a,(i,1):+,j):Privedenie}             // G[a][b]=0 -
переходим на след строку
           )
          },
          {(G,n,m,a,b,i,(j,1):+):Privedrnie}
       )
      },               //i>n переходим на след столбец
    {G}
  }
 ) //возвращ привед матрицу          j>n
}

SolvI << funcdef I
//(A[i],X,Y,n,j)
{
j<<I:5;
return<<.^[(j,I:4):(<,=>):?]^
               ( {block
                     { Z<<(I:2,(Y:j,A:j):*):+;
                       (I:1,Z,I:2,I:3,(j,1):+):SolvI >>break}},
                {I:2})
}

Solve << funcdef H
// (ступенчатая матрица,X,n,m,i,j)
{
i<<H:5;
j<<H:6;
Y<<H:2;  //копим решение
return<<.^[(i,0):(>=,<):?]^
     (
      {block{
             X<<H:1:i:m;    //B[i]
             Z<<(H:1:i,0,Y,n,i):SolvI;
             Yi<<(X,Z):-;
             V<<(0,(i,-1):+):dup;
             YY<<((V,Yi),Y):#:[]:+;
             (H,YY,n,m,1,(i,1):-):Solve>>break}},
       {Y})
}

SLU << funcdef A
//((Матрица),(Вектор))
{
C<<((A:1,A:2):#);
B<<C:[]:[]:String:(.);
n<<B:|;           //строк в матрице
m<<B:1:|;         //столбцов (с учетом свободного вектора)
BB<<(B,n,m,1,1,1,1):Privedenie;  //приведение к ступенчатому виду
```

```
Y<<(0,(n,-1):+):dup;
X<<(Y,BB:n:m);
H<<(BB,X,n,m,n,(n,1):-):Solve;
H>>return

}
//здесь СЛУ должна быть совместна!!! Размер n*n

Simpson << funcdef param //для медота сипсона
{

n<<param:1;
x<<param:2;
h<<param:3;

[ ((n,1):[=,>]):?]^

(
   {s<<(4,x:f):*},
{
                      [((n:chet,1):[=, !=]):?]^
                                          (
                                  {((2,x:f):*, ((n,1):-, (x,h):+, h):Simpson):+ },
             {((4,x:f):*, ((n,1):-, (x,h):+, h):Simpson):+ }
            )
}
):.
>>return

}

SimpsonIter << funcdef p
{
  a << p:1;
  b << p:2;
  n << p:3;
  e << p:4;


  r1 << ( a, b, n ):Simp;
  n1 << (n,10):+;
  r2 << ( a, b, n1 ):Simp;
  delta << ( (r1, r2):- ):abs;
  [((delta,e):[>, <=]):?]^
                                        (
       { ( a, b, n1, e ):SimpsonIter },
       {r2}
     ):.>> return
}
```

ПРИЛОЖЕНИЕ Г

Листинг модуля работы с репозиторием

```
//////////////////////////
//    pif_ide.h         //
//////////////////////////
#ifndef PIF_IDE_H
#define PIF_IDE_H

#include <QMainWindow>
...
#include <QVBoxLayout>
#include "ui_pif_ide.h"
#include "text_edit.h"
#include "rep_list.h"
#include "general.h"
...
#include "findfuncrepos.h"

class Pif_IDE : public QMainWindow
{
      Q_OBJECT
public:
      Pif_IDE(QWidget *parent = 0);
      ~Pif_IDE();
      bool openFile(const QString &fileName);
protected:
      virtual void closeEvent(QCloseEvent *event);
private slots:
      ...
      void about();
      void aboutQt();
    void findFunction();

private:
      Ui::Pif_IDEClass ui;
      Log_Out *log_Out;
      Res_Out *res_Out;
      Console *console;
      Arg_List *argList;
      Func_List *funcList;
      Set_Path *setPath;
      Rep_List *repList;
    FindFuncRepos *findFuncRepos;
      ...
};

#endif // PIF_IDE_H

//////////////////////////
//    pif_ide.cpp       //
//////////////////////////

#include "pif_ide.h"

Pif_IDE::Pif_IDE(QWidget *parent)
      : QMainWindow(parent)
{
```

```
      ui.setupUi(this);
      ...
      setPath = new Set_Path;
      repList = new Rep_List;
      findFuncRepos = new FindFuncRepos;
      mdiArea = new QMdiArea;
      ...
      connect(findFuncRepos->findLineEdit, SIGNAL(textChanged(QString)), this,
SLOT(findFunction()));
      connect(repList, SIGNAL(itemDoubleClicked(QTreeWidgetItem*, int)), this,
SLOT(openFuncToRep(QTreeWidgetItem*, int)));
      ...
      setWindowTitle(tr("Pifagor IDE 0.1"));
}
...
void Pif_IDE::createDockWidget()
{
      setDockOptions(QMainWindow::AllowNestedDocks | QMainWindow::AnimatedDocks
| QMainWindow::AllowTabbedDocks);

      ...
      funcDock = new QDockWidget(tr("Function"));
      funcDock->setObjectName("funcDock");
      funcDock->setWidget(funcList);
      addDockWidget(Qt::RightDockWidgetArea, funcDock);

      repDock = new QDockWidget(tr("Repository"));
    repDock->setObjectName("repDock");
    QVBoxLayout *headLayout = new QVBoxLayout();
    headLayout->addWidget(findFuncRepos);
    headLayout->addWidget(repList);
    headLayout->setMargin(0);
    headLayout->setSpacing(0);
    QWidget *wgt = new QWidget();
    wgt->setLayout(headLayout);
    repDock->setWidget(wgt);
      addDockWidget(Qt::RightDockWidgetArea, repDock);
}
...
void Pif_IDE::findFunction()
{
    QString str = findFuncRepos->Find();
    repList->FindFunc(str);
}

Pif_IDE::~Pif_IDE() {}

//////////////////////////
//    rep_list.h        //
//////////////////////////

#ifndef REP_LIST_H
#define REP_LIST_H

#include <QTreeWidget>
#include <QListView>
#include <QDir>
#include <QString>

class Rep_List : public QTreeWidget
{
```

```
      Q_OBJECT
public:
      Rep_List(QWidget *parent = 0);
      void createList();
    void FindFunc(QString str);
      ~Rep_List();
private:
    void workDir(QString pathPrefix, QString currentPath, QTreeWidgetItem
*parent);
      void AddGroup(QTreeWidgetItem *parent, QStringList strList);
      void AddFunc(QTreeWidgetItem *parent, QString str);
      virtual void contextMenuEvent(QContextMenuEvent * e);
public:
      QAction *showAction;
    QAction *newFileAction;
      QAction *showRIGjpg;
      QAction *showCGjpg;
};

#endif // REP_LIST_H

//////////////////////////
//    findfuncrepos.h   //
//////////////////////////

#ifndef FINDFUNCREPOS_H
#define FINDFUNCREPOS_H

#include <QWidget>

#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QLabel>
#include <QLineEdit>
#include <QString>

#include "rep_list.h"

class FindFuncRepos : public QWidget
{
    Q_OBJECT

public:
    FindFuncRepos(QWidget *parent = 0);
    QLineEdit *findLineEdit;
    Rep_List *f_RepList;
    QString Find();

private:

};

#endif // FINDFUNCREPOS_H

//////////////////////////
//    findfuncrepos.cpp //
//////////////////////////

#include "findfuncrepos.h"

FindFuncRepos::FindFuncRepos(QWidget *parent)
```

```
    : QWidget(parent)
{
    QLabel *lbl = new QLabel("Search");
    findLineEdit = new QLineEdit();
    //f_RepList = new Rep_List();

    QHBoxLayout *mainLayout = new QHBoxLayout();
    mainLayout->addWidget(lbl);
    mainLayout->addWidget(findLineEdit);
    mainLayout->setMargin(2);
    //headLayout->addWidget(f_RepList);

    this->setLayout(mainLayout);

}

QString FindFuncRepos::Find()
{
    QString str = findLineEdit->text();
    return str;
}

//////////////////////////
//    rep_list.cpp      //
//////////////////////////

#include <QtWidgets>

#include "rep_list.h"
#include "general.h"

Rep_List::Rep_List(QWidget *parent) : QTreeWidget(parent)
{
      setColumnCount(1);
    setHeaderLabel("Repository");

      showAction = new QAction(QIcon("./images/file.png"), tr("Open File"),
this);
      showAction->setStatusTip(tr("Open original file"));

    newFileAction = new QAction(QIcon("./images/new.png"), tr("Open as new
file"), this);
    newFileAction->setStatusTip(tr("Open as new file"));

      showRIGjpg = new QAction(QIcon("./images/graf.png"), tr("Open RIG"),
this);
      showRIGjpg->setStatusTip(tr("Open RIG.jpg"));

      showCGjpg = new QAction(QIcon("./images/graf.png"), tr("Open CG"), this);
      showCGjpg->setStatusTip(tr("Open CG.jpg"));
}
void Rep_List::createList()
{
      clear();
    workDir(stdValue.pathRepository,"",NULL);
}
void Rep_List::workDir(QString pathPrefix, QString subPath, QTreeWidgetItem
*parent)
{
    QString fullPath=pathPrefix+"\\"+subPath;
    QDir dir;
```

```
    dir.setFilter(QDir::AllDirs | QDir::NoSymLinks);
    dir.setSorting(QDir::Name);
    dir.setPath(fullPath);
    QFileInfoList list = dir.entryInfoList();
    QString currentPath;

    QTreeWidgetItem *item;

    for(int i = 0; i < list.count(); i++)
    {
        currentPath=list.at(i).fileName();
        if(currentPath != "." && currentPath != "..")
        {
            if(parent == NULL)
                addTopLevelItem(item = new QTreeWidgetItem());
            else
                item = new QTreeWidgetItem(parent);
            item->setText(0, currentPath);
            workDir(fullPath,currentPath,item);
        }
    }
}
...

void Rep_List::FindFunc(QString str)
{
    QList<QTreeWidgetItem *> found = findItems(str, Qt::MatchRecursive);
    if (found.size() != 0)
    {
        setCurrentItem(found[0]);
    }
    //setHeaderLabel(str);
}

Rep_List::~Rep_List() { }
```