**Санкт-Петербургский государственный электротехнический университет**

**“ЛЭТИ”**

**кафедра АСОиУ**

**Конспект лекций по конструированию программ**

**Выполнил : Фадеев Георгий Георгиевич**

**Факультет КТИ**

**Группа № 1372**

**Преподаватель: Денис Олегович Степуленок**

**Санкт-Петербург**

**2013**

ВВЕДЕНИЕ

Термин конструирование программного обеспечения описывает детальное создание рабочей программной системы посредством комбинации кодирования, верификации (проверки), модульного тестирования, интеграционного тестирования и отладки.

Данная область знаний связана с другими областями. Наиболее сильная связь существует с проектированием (Software Design) и тестированием (Software Testing). Причиной этого является то, что сам по себе процесс конструирования программного обеспечения затрагивает важные аспекты деятельности по проектированию и тестированию. Кроме того, конструирование отталкивается от результатов проектирования, а тестирование (в любой своей форме) предполагает работу с результатами конструирования. Достаточно сложно определить границы между проектированием, конструированием и тестированием, так как все они связаны в единый комплекс процессов жизненного цикла и, в зависимости от выбранной модели жизненного цикла и применяемых методов (методологии), такое разделение может выглядеть по разному.

Хотя ряд операций по проектированию детального дизайна может происходить до стадии конструирования, большой объем такого рода проектных работ происходит параллельно с конструированием или как его часть. Это есть суть связи с областью знаний “Проектирование программного обеспечения”.

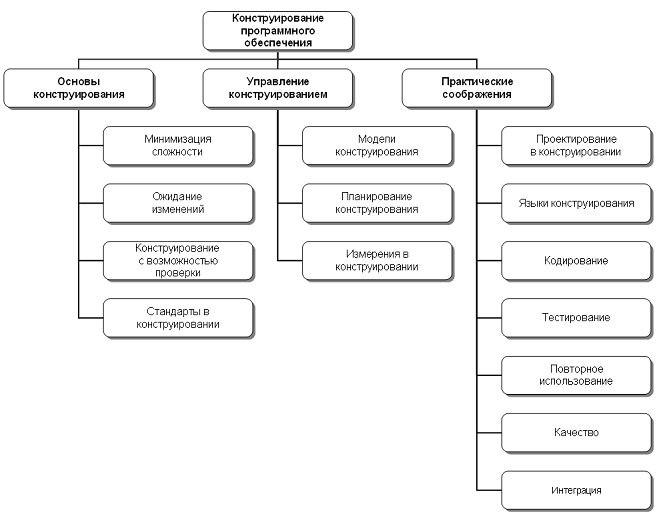
В свою очередь, на протяжении всей деятельности по конструированию, инженеры используют модульное и интеграционное тестирование. Таким образом, данная область знаний связана с “Тестированием программного обеспечения”.

В процессе конструирования обычно создается большая часть активов программного проекта - конфигурационных элементов. Поэтому в реальных проектах просто невозможно рассматривать деятельность по конструированию в отрыве от области знаний “Конфигурационного управления” (Software Configuration Management).

Так как конструирование невозможно без использования соответствующего инструментария и, вероятно, данная деятельность является наиболее инструментально-насыщенной, важную роль в конструировании играет область знаний “Инструменты и методы программной инженерии” (Software Engineering Tools and Methods).

Безусловно, вопросы обеспечения качества значимы для всех областей знаний и этапов жизненного цикла. В то же время, код является основным результирующим элементом программного проекта. Таким образом, явно напрашивается и присутствует связь обсуждаемых вопросов с областью знаний “Качество программного обеспечения” (Software Quality).

Из связанных дисциплин программной наиболее тесная и естественная связь данной области знаний существует с компьютерными науками. Именно в них, обычно, рассматриваются вопросы построения и использования алгоритмов и практик кодирования. Наконец, конструирование касается и управления проектами, причем, в той степени, насколько деятельность по управлению конструированием важна для достижения результатов конструирования.



# Работа с тестирующей системой

Тестирующая система **TestSys** расположена по адресу: [ts.lokos.net](http://ts.lokos.net/).

Для входа в систему следует использовать логин (обычно - 2 цифры) и пароль выданные преподавателем.

## Задача "A+B" на разных языках программирования

Нужно ввести из входного файла два целых числа и вывести их сумму в выходной файл.

### Pascal

**const** FN **=** 'aplusb'**;** *{ Название входного и выходного файла }*

**var** a**,**b **:** **longint;** *{ Входные данные }*

**begin**

assign**(**Input**,**FN**+**'.in'**);** reset**(**input**);**

assign**(**Output**,**FN**+**'.out'**);** rewrite**(**output**);**

read**(**a**,**b**);** *{ Читаем исходные данные из входного файла }*

writeln**(**a**+**b**);** *{ Записываем результат в выходной файл }*

**end.**

### Delphi

*{$APPTYPE CONSOLE}*

**const** FN **=** 'aplusb'**;** *{ Название входного и выходного файла }*

**var** a**,**b **:** **Int64;** *{ Входные данные }*

**begin**

reset**(**Input**,**FN**+**'.in'**);**

rewrite**(**Output**,**FN**+**'.out'**);**

read**(**a**,**b**);** *{ Читаем исходные данные из входного файла }*

writeln**(**a**+**b**);** *{ Записываем результат в выходной файл }*

**end.**

### C/C++

#include <stdio.h>

#define s "aplusb"

**int** main**()** **{**

freopen**(**s".in"**,**"r"**,**stdin**);** *// Стандартный поток ввода из входного файла*

freopen**(**s".out"**,**"w"**,**stdout**);** *// Стандартный поток вывода в выходной файл*

**long** **long** a**,**b**;**

scanf**(**"%lld%lld"**,&**a**,&**b**);**

printf**(**"%lld"**,**a**+**b**);**

**return** 0**;**

**}**

## Сообщения тестирующей системы

* **Accepted -**Все в порядке! Ваша программа принята! Она откомпилировалась без ошибок и прошла все тесты.
* **Presentation Error (PE)** - неправильный формат вывода, проверяющая программа не смогла прочитать ваш выходной файл или ваша программа вообще не создала выходной файл.
* **Wrong Answer (WA)**- неправильный ответ на тест.
* **Compile Error (CE)** - ошибки компиляции программы. Посмотрите что вы отправляете (нажмите view в отправках).
* **Runtime Error (RT)** - ошибка времени выполнения (выход за границы массива, переполнение переменной, деление на ноль, корень из отрицательного числа, ошибка в имени входного файла).
* **Time Limit (TL)** - ваша программа выполнялась на каком-то тесте больше времени по условию задачи.
* **Memory Limit (ML)** - ваша программа использовала больше памяти, чем разрешено по условию задачи.

## Компиляторы

* Pascal: Borland Delphi 7.0, Free Pascal 2.6.0;
* C/C++: Visual C++ 2010 Express Edition, GNU C++ 4.6.1 (MinGW), Code::Blocks 10.05;
* C#: Visual C# 2010 Express Edition;
* Java: Sun JDK 7 update 9, Eclipse 4.2.
* Python: Python 3.3.0, Wing IDE 101 4.1.9.

# Примеры ошибок в решениях:

## Presentation Error (PE)

Неправильное имя выходного файла:

*{$APPTYPE CONSOLE}*

**var** A**,**B **:** **Int64;**

**begin**

Assign**(**Input**,**'aplusb.in'**);** Reset**(**Input**);**

Assign**(**Output**,**'YA\_NE\_POMNU.out'**);** Rewrite**(**Output**);**

Read**(**A**,**B**);**

Write**(**A**+**B**);**

**end.**

Программа выводит на экран вместо файла:

*{$APPTYPE CONSOLE}*

**var** A**,**B **:** **Int64;**

**begin**

Assign**(**Input**,**'aplusb.in'**);** Reset**(**Input**);**

*{ Assign(Output,'aplusb.out'); Rewrite(Output); }*

Read**(**A**,**B**);**

Write**(**A**+**B**);**

**end.**

## Wrong Answer (WA)

Неверное решение:

*{$APPTYPE CONSOLE}*

**var** A**,**B **:** **Int64;**

**begin**

Assign**(**Input**,**'aplusb.in'**);** Reset**(**Input**);**

Assign**(**Output**,**'aplusb.out'**);** Rewrite**(**Output**);**

Read**(**A**,**B**);**

Write**(**A**-**B**);**

**end.**

Точности/разрядности типов данных не хватает:

*{$APPTYPE CONSOLE}*

**var** A**,**B **:** **Extended;**

**begin**

Assign**(**Input**,**'aplusb.in'**);** Reset**(**Input**);**

Assign**(**Output**,**'aplusb.out'**);** Rewrite**(**Output**);**

Read**(**A**,**B**);**

Write**((**A**+**B**):**0**:**0**);**

**end.**

## Compile Error (CE)

*{$APPTYPE CONSOLE}*

**var** A**,**B **:** **Int64;**

**begin**

ЯВНО НЕВЕРНАЯ СТРОКА**!!!**

Assign**(**Input**,**'aplusb.in'**);** Reset**(**Input**);**

Assign**(**Output**,**'aplusb.out'**);** Rewrite**(**Output**);**

Read**(**A**,**B**);**

Write**(**A**+**B**);**

**end.**

Комментарий тестирующей системы:

Borland Delphi Version 15.0

Copyright (c) 1983,2002 Borland Software Corporation

a.dpr(4) Error: Illegal character in input file: 'Я' ($DF)

a.dpr(4) Error: Illegal character in input file: 'А' ($C0)

a.dpr(4) Error: Illegal character in input file: '!' ($21)

a.dpr(9)

## Runtime Error (RE)

Программа завершилась с ненулевым кодом возврата, либо создала исключительную ситуацию (exception) и не обработала ее.

*{$APPTYPE CONSOLE}*

**var** A**,**B **:** **Integer;** *{ Integer переполняется на больших числах }*

**begin**

Assign**(**Input**,**'aplusb.in'**);** Reset**(**Input**);**

Assign**(**Output**,**'aplusb.out'**);** Rewrite**(**Output**);**

Read**(**A**,**B**);**

Write**(**A**+**B**);**

**end.**

*{$APPTYPE CONSOLE}*

**var** A**,**B **:** **Int64;**

**begin**

Assign**(**Input**,**'aplusb.in'**);** Reset**(**Input**);**

Assign**(**Output**,**'aplusb.out'**);** Rewrite**(**Output**);**

Read**(**A**,**B**);**

Write**(**A **div** B**);** *{ Когда B равно 0, тут будет деление на ноль! }*

**end.**

Для поиска RE в программе на Delphi используйте директивы:

*{$O Off}* *{ Выключаем оптимизацию }*

*{$R+}* *{ Включаем проверку границ массивов }*

*{$Q+}* *{ Включаем проверку переполнений }*

## Time Limit (TL)

Программа не завершилась за отведенный период времени.

**Возможные причины:**

* Неэффективное решение;
* Ошибка в программе (например, зацикливание).

*{$APPTYPE CONSOLE}*

**var**

A**,**B **:** **Int64;**

I**,**K **:** **Integer;**

**begin**

Assign**(**Input**,**'aplusb.in'**);** Reset**(**Input**);**

Assign**(**Output**,**'aplusb.out'**);** Rewrite**(**Output**);**

Read**(**A**,**B**);**

*{ Цикл выполняется слишком долго!!! }*

**for** I**:=**Low**(Integer)** **to** High**(Integer)** **do**

K **:=** I**;**

Write**(**A**+**B**);**

**end.**

## Memory Limit (ML)

Программа попыталась использовать больше памяти, чем разрешается.

*{$APPTYPE CONSOLE}*

**var**

A**,**B **:** **Int64;**

T **:** **array** **[**1**..**100000000**]** **of** **Integer;** *{ Слишком большой массив!!! }*

I **:** **Integer;**

**begin**

Assign**(**Input**,**'aplusb.in'**);** Reset**(**Input**);**

Assign**(**Output**,**'aplusb.out'**);** Rewrite**(**Output**);**

**for** I**:=**Low**(**T**)** **to** High**(**T**)** **do** T**[**I**]:=**2135**;**

Read**(**A**,**B**);**

Write**(**A**+**B**);**

**end.**

# Оценка сложности алгоритмов

Существует несколько способов измерения сложности алгоритма. Программисты обычно сосредотачивают внимание на**скорости** алгоритма, но не менее важны и другие показатели – требования к **объёму памяти**, например. Использование быстрого алгоритма не приведёт к ожидаемым результатам, если для его работы понадобится больше памяти, чем есть у компьютера.

При сравнении различных алгоритмов важно знать, как их сложность зависит от объёма входных данных.

*O*(1) - время выполнения алгоритма константа (не зависит от количества элементов во входном множестве).

*O*(*n*) - линейная сложность.

Пример: поиск минимального элемента в массиве (цикл по всем элементам массива).

*O*(*n*2) - квадратичная сложность. Например, сортировки "пузырьком", вставками и т.д.

Пример: поиск двух точек на плоскости с максимальным расстоянием между ними.

*O*(*n*3) - кубическая сложность.

Пример: поиск треугольника максимальной площади с вершинами в точках на плоскости (перебор вершин тройным циклом).

Оценка *O*(*n*) ассимптотическая (при *n* стремящемся к бесконечности).

*O*(ln*n*) - **Бинарный поиск** - логарифмическая сложность.

*O*(*n*log*n*) - быстрая сортировка Хоара (рус) - QuickSort, HeapSort, MergeSort.

При оценке сложности важна только самая старшая степень. Например:

*O*(*n*3+*n*2+4*n*)=*O*(*n*3)

Константа так же отбрасывается, так как нам важно лишь во сколько раз возрастает время выполнения программы при увеличении размера входных данных.

# Язык программирования Pascal (Delphi)

Несмотря на то что большинство школьников сейчас пишет программы на Pascal / Delphi, у них возникают различные вопросы связанные с использованием языка программирования.

## Записи и оператор with

Записи используются для создания своих типов данных

*{ Обьявление записи - свой тип "Точка" }*

**type**

TPoint **=** **Record**

x**,**y **:** **double;**

**end;**

*{ Расстояние между точками }*

**function** dist**(** A**,**B **:** TPoint **):double;**

**begin**

dist **:=** sqrt**(** **(**A**.**x **-** B**.**x**)\*(**A**.**x **-** B**.**x**)** **+** **(**A**.**y **-** B**.**y**)\*(**A**.**y **-** B**.**y**)** **);**

**end;**

*{ Использование записей }*

**var** A**,**B **:** TPoint**;**

**begin**

*{ Инициализируем координаты точек }*

A**.**x **:=** 1**;** A**.**y **:=** 2**;**

B**.**x **:=** 10**;** B**.**y **:=** 11**;**

writeln**(** dist**(**A**,**B**)** **);**

**end.**

Использование with:

*{ Обьявление записи - тип "Персонаж в игре" }*

**type**

TUnit **=** **Record**

x**,**y **:** **integer;** *{ Координаты клетки где стоит персонаж }*

name **:** **string;** *{ Имя персонажа }*

**end;**

**var** Unit1 **:** TUnit**;**

*{ Инициализация без with }*

Unit1**.**x **:=** 2**;**

Unit1**.**y **:=** 3**;**

Unit1**.**name **:=** 'SUPER-HERO'**;**

*{ Инициализация с with }*

**with** Unit1 **do** **begin**

x **:=** 2**;**

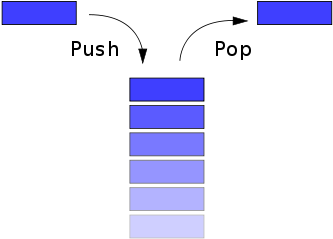
y **:=** 3**;**

name **:=** 'SUPER-HERO'**;**

**end;**

## Реализация Стека и Очереди на базе массива

**Стек** (англ. **stack** — стопка) — структура данных с методом доступа к элементам **LIFO** (англ. Last In — First Out, «последним пришёл — первым вышел»). Чаще всего принцип работы стека сравнивают со стопкой тарелок: чтобы взять вторую сверху, нужно снять верхнюю.



**const** StackSize **=** 10000**;** *{ Размер стека (сколько в него можно положить элементов) }*

*{ === Хранение стека === }*

**var**

Stack **:** **array** **[**1**..**StackSize**]** **of** **Integer;** *{ Массив для хранения стека }*

StackTop **:** **Integer** **=** 0**;** *{ Вершина стека - индекс в массиве Stack }*

*{ === Операции со стеком === }*

*{ Стек пуст? }*

**function** isEmpty **:** **Boolean;**

**begin**

isEmpty **:=** StackTop **=** 0**;**

**end;**

*{ Положить значение на вершину стека }*

**procedure** Push**(** Value **:** **Integer** **);**

**begin**

assert**(** StackTop **<** StackSize**,** 'Стек полон! Больше положить в него нельзя!'**);**

Inc**(**StackTop**);**

Stack**[**StackTop**]** **:=** Value**;**

**end;**

*{ Забрать значение с вершины стека }*

**function** Pop **:** **Integer;**

**begin**

assert**(** **not** isEmpty**,** 'Нельзя извлечь элемент, потому что стек пуст!'**);**

Pop **:=** Stack**[**StackTop**];**

Dec**(** StackTop **);**

**end;**

*{ === Тестирование работы стека === }*

**begin**

Writeln**(**isEmpty**);** *{ Выводит "TRUE" - стек пуст }*

Push**(**2**);** *{ В стеке: 2 }*

Writeln**(**isEmpty**);** *{ Выводит "FALSE" - стек не пуст }*

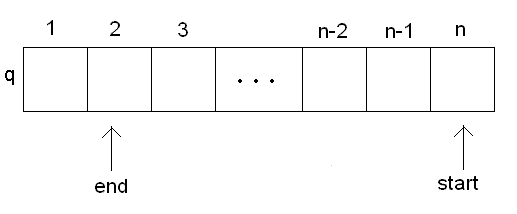
Push**(**5**);** *{ В стеке: 2, 5 }*

Writeln**(**Pop**);** *{ Выводит "5", в стеке: 2 }*

Writeln**(**Pop**);** *{ Выводит "2", в стеке пусто }*

**end.**

**Очередь** — структура данных с дисциплиной доступа к элементам «первый пришёл — первый вышел» (**FIFO**, First In — First Out). Добавление элемента (принято обозначать словом enqueue — поставить в очередь) возможно лишь в конец очереди, выборка — только из начала очереди (что принято называть словом dequeue — убрать из очереди), при этом выбранный элемент из очереди удаляется.



**const** QSize **=** 10000**;** *{ Размер очереди (сколько в неё можно положить элементов) }*

**var**

Q **:** **array** **[**1**..**QSize**]** **of** **Integer;** *{ Массив для хранения очереди }*

Q\_Start **:** **Integer** **=** 1**;** *{ Указывает на голову очереди }*

Q\_End **:** **Integer** **=** 1**;** *{ Указывает на элемент, который заполнится, когда в очередь войдёт новый элемент }*

*{ = Операции с очередью = }*

*{ Очередь пуста? }*

**function** isEmpty **:** **Boolean;**

**begin**

isEmpty **:=** Q\_Start **=** Q\_End**;**

**end;**

*{ Положить значение в конец очереди }*

**procedure** **Put(** Value **:** **Integer** **);**

**begin**

Q**[**Q\_End**]** **:=** Value**;**

Dec**(**Q\_End**);**

*{ Поддержка закольцованности очереди }*

**if** Q\_End **<** 1 **then** Q\_End **:=** QSize**;**

**end;**

*{ Забрать значение с начала очереди }*

**function** **Get** **:** **Integer;**

**begin**

assert**(** **not** isEmpty**,** 'В очереди ничего нет!'**);**

**Get** **:=** Q**[**Q\_Start**];**

Dec**(**Q\_Start**);**

*{ Поддержка закольцованности очереди }*

**if** Q\_Start **<** 1 **then** Q\_Start **:=** QSize**;**

**end;**

**begin**

Writeln**(**isEmpty**);** *{ Выводит "TRUE" - очередь пуста }*

**Put(**2**);** *{ В очереди: 2 }*

Writeln**(**isEmpty**);** *{ Выводит "FALSE" - очередь не пуста }*

**Put(**5**);** *{ В очереди: 5, 2 }*

Writeln**(Get);** *{ Выводит "2", в очереди: 5 }*

Writeln**(Get);** *{ Выводит "5", в очереди пусто }*

**end.**

# Системы счисления

Системы счисления делятся на:

* **позиционные** - одна и та же цифра имеет разное значение в зависимости от позиции 143=1⋅102+4⋅10+3.
* **непозиционные** - число задаётся несколькими знаками с различным значением.
* смешанные - например, римская система счисления *IX*=9 - значение зависит от позиции цифр, но есть ограничения.

# Системы счисления

## Позиционные и непозиционные

Позиционные - значение зависит от позиции цифры.

Например, десятичная система счисления: 343=3∗102+4∗101+3

## Перевод из одной системы счисления в другую

Мы считаем в десятичной системе счисления.

Перевод из **десятичной** в **двоичную** - **деление** в столбик или сравнение по таблице степеней двойки.

Перевод обратно, из **двоичной** в **десятичную** - умножение разрядов на степени двойки и сложение.

### Непозиционная: Римская система счисления

I - 1, V - 5, X - 10, D - 50, L - 500, M - 1000 - значение не зависит от позиции (зависит по-другому).

## Двоичная система счисления (запись чисел)

**Позиционная** система счисления с основанием 2, Числа записываются с помощью двух символов: 0 и 1.

Например: число 3=112.

*x*2,2=*an*−1*an*−2…*a*1*a*0 2,2=∑*n*−1*k*=0*akbk*=∑*n*−1*k*=0*ak*2*k*,  
где:

* *x*2,2 — представляемое число, первый индекс - основание системы счисления (размерность множества цифр *a*=0,1), второй индекс - основание весовой показательной функции *b* (в двоично-десятичном кодировании *b*=10),
* *an*−1*an*−2...*a*1*a*0 — запись числа, строка цифр,
* . . . 2,2 — обозначение основания системы кодирования и основания системы счисления,
* *n* — количество цифр (знаков) в числе *x*2,2,
* *k* — порядковый номер цифры,
* *ak* — цифры числа *x*2,2измножестваa={0,1}$, в двоичной системе счисления, основание системы кодирования равно 2,
* *b*=2 — основание показательной весовой функции, основание системы счисления,
* *bk*=2*k* — весовая показательная функция, создающая весовые коэффициенты.

## Восьмеричная, шестнадцатеричная

Позиционная целочисленная система счисления с основанием 8 (используются цифры от 0 до 7).

Таблица перевода восьмеричных чисел в двоичные:

08 = 0002  
18 = 0012  
28 = 0102  
38 = 0112  
48 = 1002  
58 = 1012  
68 = 1102  
78 = 1112  
Для перевода восьмеричного числа в двоичное необходимо заменить каждую цифру восьмеричного числа на триплет двоичных цифр. Например: 25418 = [ 28 | 58 | 48 | 18 ] = [ 0102 | 1012 | 1002 | 0012 ] = 0101011000012

## Перевод целых чисел из 10 системы счисления в любую

Для перевода надо делить "в столбик" число на основание системы счисления. Каждый очередной остаток будет цифрой числа

Вывод числа в произвольной системе счисления:

*{$APPTYPE CONSOLE}*

**uses** SysUtils**;**

**var**

x**,** b **:** **longint;**

s **:** **string;** *{ Результат преобразования }*

**begin**

Write**(**'Введите число: '**);** Readln**(**x**);**

Write**(**'Введите основание системы счисления: '**);** Readln**(**b**);**

s **:=** ''**;**

**while** x **>** 0 **do** **begin**

s **:=** IntToStr**(**x **mod** b**)** **+** s**;** *{ Остаток - это очередная цифра. Приписываем её спереди к числу }*

x **:=** x **div** b**;** *{ Делим на основание системы счисления }*

**end;**

**if** s **=** '' **then** s **:=** '0'**;**

writeln**(**'Результат: '**,**s**);**

**end.**

# Системы счисления

Системы счисления делятся на:

* **позиционные** - одна и та же цифра имеет разное значение в зависимости от позиции 143=1⋅102+4⋅10+3.
* **непозиционные** - число задаётся несколькими знаками с различным значением.
* смешанные - например, римская система счисления *IX*=9 - значение зависит от позиции цифр, но есть ограничения.

# Системы счисления

## Позиционные и непозиционные

Позиционные - значение зависит от позиции цифры.

Например, десятичная система счисления: 343=3∗102+4∗101+3

## Перевод из одной системы счисления в другую

Мы считаем в десятичной системе счисления.

Перевод из **десятичной** в **двоичную** - **деление** в столбик или сравнение по таблице степеней двойки.

Перевод обратно, из **двоичной** в **десятичную** - умножение разрядов на степени двойки и сложение.

### Непозиционная: Римская система счисления

I - 1, V - 5, X - 10, D - 50, L - 500, M - 1000 - значение не зависит от позиции (зависит по-другому).

## Двоичная система счисления (запись чисел)

**Позиционная** система счисления с основанием 2, Числа записываются с помощью двух символов: 0 и 1.

Например: число 3=112.

*x*2,2=*an*−1*an*−2…*a*1*a*0 2,2=∑*n*−1*k*=0*akbk*=∑*n*−1*k*=0*ak*2*k*,  
где:

* *x*2,2 — представляемое число, первый индекс - основание системы счисления (размерность множества цифр *a*=0,1), второй индекс - основание весовой показательной функции *b* (в двоично-десятичном кодировании *b*=10),
* *an*−1*an*−2...*a*1*a*0 — запись числа, строка цифр,
* . . . 2,2 — обозначение основания системы кодирования и основания системы счисления,
* *n* — количество цифр (знаков) в числе *x*2,2,
* *k* — порядковый номер цифры,
* *ak* — цифры числа *x*2,2измножестваa={0,1}$, в двоичной системе счисления, основание системы кодирования равно 2,
* *b*=2 — основание показательной весовой функции, основание системы счисления,
* *bk*=2*k* — весовая показательная функция, создающая весовые коэффициенты.

## Восьмеричная, шестнадцатеричная

Позиционная целочисленная система счисления с основанием 8 (используются цифры от 0 до 7).

Таблица перевода восьмеричных чисел в двоичные:

08 = 0002  
18 = 0012  
28 = 0102  
38 = 0112  
48 = 1002  
58 = 1012  
68 = 1102  
78 = 1112  
Для перевода восьмеричного числа в двоичное необходимо заменить каждую цифру восьмеричного числа на триплет двоичных цифр. Например: 25418 = [ 28 | 58 | 48 | 18 ] = [ 0102 | 1012 | 1002 | 0012 ] = 0101011000012

## Перевод целых чисел из 10 системы счисления в любую

Для перевода надо делить "в столбик" число на основание системы счисления. Каждый очередной остаток будет цифрой числа

Вывод числа в произвольной системе счисления:

*{$APPTYPE CONSOLE}*

**uses** SysUtils**;**

**var**

x**,** b **:** **longint;**

s **:** **string;** *{ Результат преобразования }*

**begin**

Write**(**'Введите число: '**);** Readln**(**x**);**

Write**(**'Введите основание системы счисления: '**);** Readln**(**b**);**

s **:=** ''**;**

**while** x **>** 0 **do** **begin**

s **:=** IntToStr**(**x **mod** b**)** **+** s**;** *{ Остаток - это очередная цифра. Приписываем её спереди к числу }*

x **:=** x **div** b**;** *{ Делим на основание системы счисления }*

**end;**

**if** s **=** '' **then** s **:=** '0'**;**

writeln**(**'Результат: '**,**s**);**

**end.**

## Перевод целых чисел из 2-й, 8-й, 16-й в 10 сс

Для перевода умножаем каждую последующую цифру на соответствующую степень 2-ки, 8-рки, 16-ти

Для двоичного числа: *x*=*an*⋅2*n*+...+*a*1⋅21+*a*0⋅20

## Римская система счисления

|  |  |  |
| --- | --- | --- |
| 1 | I | лат. unus |
| 5 | V | лат. quinque |
| 10 | X | лат. decem |
| 50 | L | лат. quinquaginta |
| 100 | C | лат. centum |
| 500 | D | лат. quingenti |
| 1000 | M | лат. mille |

*{$APPTYPE CONSOLE}*

*{ Функция для перевода из десятичной системы в римскую }*

**function** to\_roman**(** n **:** **integer** **):string;**

**const** D **:** **array** **[**1**..**13**]** **of** **integer** **=** **(**1**,**4**,**5**,**9**,**10**,**40**,**50**,**90**,**100**,**400**,**500**,**900**,**1000**);**

S **:** **array** **[**1**..**13**]** **of** **string** **=** **(**'I'**,**'IV'**,**'V'**,**'IX'**,**'X'**,**'XL'**,**'L'**,**'XC'**,**'C'**,**'CD'**,**'D'**,**'CM'**,**'M'**);**

**var** k **:** **integer;**

**begin**

result **:=** ''**;**

**for** k **:=** high**(**D**)** **downto** low**(**D**)** **do** *{ Идём от старших разрядов к младшим }*

**while** D**[**k**]** **<=** n **do** **begin**

dec**(**n**,** D**[**k**]);**

result **:=** result **+** S**[**k**];**

**end;**

**end;**

**var** N**,**Tests**,**Test **:** **integer;**

**begin**

assign**(**input**,**'roman.in'**);** reset**(**input**);**

assign**(**output**,**'roman.out'**);** rewrite**(**output**);**

readln**(**Tests**);** *{ Количество тестов }*

**for** Test **:=** 1 **to** Tests **do** **begin** *{ Цикл по тестам }*

readln**(**N**);**

writeLn**(**to\_roman**(**N**));**

**end;**

**end.**

**Реализация через массив с записями**

**const**

T **:** **array** **[**1**..**13**]** **of** **record** D**:integer;** S**:string;** **end** **=** **(**

**(**D**:**1000**;** S**:**'M'**),** **(**D**:**900**;** S**:**'CM'**),**

**(**D**:**500**;** S**:**'D'**),** **(**D**:**400**;** S**:**'CD'**),**

**(**D**:**100**;** S**:**'C'**),** **(**D**:**90**;** S**:**'XC'**),**

**(**D**:**50**;** S**:**'L'**),** **(**D**:**40**;** S**:**'XL'**),**

**(**D**:**10**;** S**:**'X'**),** **(**D**:**9**;** S**:**'IX'**),**

**(**D**:**5**;** S**:**'V'**),** **(**D**:**4**;** S**:**'IV'**),**

**(**D**:**1**;** S**:**'I'**));**

*{ Функция для перевода из десятичной системы в римскую }*

**function** to\_roman**(** N **:** **integer** **):string;**

**var** i **:** **integer;**

**begin**

result **:=** ''**;**

**for** i **:=** 1 **to** 13 **do**

**while** T**[**i**].**D **<=** n **do** **begin**

dec**(**N**,** T**[**i**].**D**);**

result **:=** result **+** T**[**i**].**S**;**

**end;**

**end;**

# Сортировки

Алгоритмы сортировки - это методы для упорядочения элементов массива в каком-либо порядке.

Алгоритмы сортировки оцениваются по скорости выполнения и количествую используемой памяти.

**Время** (скорость выполнения) - основной параметр, он измеряется относительно количества элементов исходного массива. Например: *O*(*n*) - время выполнения растёт пропорционально количеству элементов, *O*(*n*2) - пропорционально квадрату количества элементов.

При описании всех алгоритмов: N или *n* - количество элементов в массиве.

## Квадратичные сортировки, сортировка "Пузырьком"

Это самая простая сортировка, её следует применять когда у вас немного элементов (до десятков тысяч). Её сложность*O*(*n*2), т.е. количество операций растёт как квадрат от количества элементов.

**Шаги алгоритма:**

1. Считываем исходный массив в память.
2. Пока происходят изменения в массиве: сравниваем каждые два соседних элемента, и если они не стоят не в том порядке, меняем их местами.
3. Теперь массив отсортирован, выводим его.

*{$APPTYPE CONSOLE}*

**var**

N **:** **Integer;** *{ Количество элементов в массиве }*

A **:** **array** **[**1**..**5000**]** **of** **Integer;** *{ Сортируемый массив }*

i **:** **Integer;** *{ Переменная цикла }*

temp **:** **Integer;** *{ Переменная для обмена местами двух элементов в массиве }*

Changes **:** **Boolean;** *{ Есть ли изменения? }*

**begin**

*{ Ввод исходного массива из файла }*

Reset**(**Input**,**'bubble.in'**);**

Read**(**N**);** *{ Читаем количество элементов в массиве }*

**for** i**:=**1 **to** N **do** Read**(**A**[**i**]);** *{ Читаем сам массив }*

*{ Сортировка }*

**repeat**

Changes **:=** **false;** *{ Пока изменений нет :) }*

**for** i**:=**1 **to** N-1 **do** *{ Пробегаем по массиву сравнивая соседние элементы }*

**if** A**[**i**]>**A**[**i+1**]** **then** **begin** *{ Если больший элемент слева, а должен быть справа }*

temp **:=** A**[**i**];** *{ Меняем элементы местами при помощи временной переменной temp }*

A**[**i**]** **:=** A**[**i+1**];**

A**[**i+1**]** **:=** temp**;**

Changes **:=** **true;** *{ Изменения произошли! }*

**end;**

**until** **not** Changes**;** *{ Заканчиваем когда нет изменений (значит, все элементы уже по-порядку) }*

*{ Вывод отсортированного массива в файл }*

Rewrite**(**Output**,**'bubble.out'**);**

**for** i**:=**1 **to** N-1 **do** Write**(**A**[**i**],**' '**);**

Writeln**(**A**[**N**])**

**end.**

**Модификация с максимумами:**

В этой сортировке мы пробегаем массив N раз, каждый раз перемещая в конец массива самый большой при сортировке по возрастанию (или самый маленький при сортировке по убыванию) элемент.

**Шаги алгоритма:**

1. Считываем исходный массив в память.
2. Переносим в N-ый элемент максимум среди элементов 1..N.
3. Переносим в N-1-ый элемент максимум среди 1..N-1.
4. и так далее (делаем это N раз).
5. Теперь массив отсортирован, выводим его.

*{$APPTYPE CONSOLE}*

**var**

N **:** **Integer;** *{ Количество элементов в массиве }*

A **:** **array** **[**1**..**5000**]** **of** **Integer;** *{ Сортируемый массив }*

i**,**j **:** **Integer;** *{ Переменные цикла }*

temp **:** **Integer;** *{ Переменная для обмена местами двух элементов в массиве }*

**begin**

*{ Ввод исходного массива из файла }*

**...**

*{ Сортировка }*

**for** i**:=**N **downto** 1 **do** *{ A[i] должно быть больше чем все A[j] слева от него }*

**for** j**:=**1 **to** i-1 **do** *{ Проверяем все A[j] }*

**if** A**[**j**]** **>** A**[**i**]** **then** **begin** *{ Если какое-то A[j] больше чем A[i], то меняем их местами }*

temp **:=** A**[**i**];** *{ Меняем элементы местами при помощи временной переменной temp }*

A**[**i**]** **:=** A**[**j**];**

A**[**j**]** **:=** temp**;**

**end;**

*{ Вывод отсортированного массива в файл }*

**...**

**end.**

Другой вариант, легче для запоминания:

*{ Сортировка }*

**for** i **:=** 1 **to** N-1 **do**

**for** j **:=** i+1 **to** N **do**

**if** A**[**j**]** **<** A**[**i**]** **then** **begin**

temp**:=**A**[**i**];** *{ Меняем элементы A[i] и A[j] местами }*

A**[**i**]:=**A**[**j**];**

A**[**j**]:=**temp**;**

**end;**

Ну или можно вообще не запоминать как меняются индексы и писать i от 1 до N и j от 1 до N.

Менять местами элементы можно без использования временной переменной, сложениями/вычитаниями (и - исходное значение):   
*A*′*j*=*Ai*+*Aj*;   
*A*′*i*=*A*′*j*−*Ai*=*Ai*+*Aj*−*Ai*=*Aj*;   
*A*′′*j*=*A*′*j*−*A*′*i*=*Ai*+*Aj*−*Aj*=*Ai*

A**[**j**]** **:=** A**[**i**]** **+** A**[**j**];** *{ A[j] := A[i]\_и + A[j]\_и }*

A**[**i**]** **:=** A**[**j**]** **-** A**[**i**];** *{ A[i] := A[i]\_и + A[j]\_и - A[i]\_и = A[j]\_и }*

A**[**j**]** **:=** A**[**j**]** **-** A**[**i**];** *{ A[j] := A[i]\_и + A[j]\_и - A[j]\_и = A[i]\_и }*

Или при помощи операции XOR (используя *x*⊕*x*=0):   
*A*′*i*=*Ai*⊕*Aj*;   
*A*′*j*=*A*′*i*⊕*Aj*=*Ai*⊕*Aj*⊕*Aj*=*Ai*;   
*A*′′*i*=*A*′*i*⊕*A*′*j*=*Ai*⊕*Aj*⊕*Ai*=*Aj*

A**[**i**]** **:=** A**[**i**]** **xor** A**[**j**];**

A**[**j**]** **:=** A**[**i**]** **xor** A**[**j**];**

A**[**i**]** **:=** A**[**i**]** **xor** A**[**j**];**

## Сортировка подсчётом

Когда диапазон чисел которые нужно отсортировать невелик по сравнению с их количеством, проще (и быстрее всего) посчитать количество элементов каждого вида прямо при чтении входного файла и вывести их "по видам".

*{$APPTYPE CONSOLE}*

**var**

N **:** **Integer;** *{ Количество элементов в массиве }*

Ai **:** **Integer;** *{ Элементы сортируемого массива }*

P **:** **array** **[**1**..**1000**]** **of** **Integer;** *{ P[i] - количество элементов со значением i }*

i **:** **Integer;** *{ Переменная цикла }*

**begin**

*{ Ввод исходного массива из файла }*

Reset**(**Input**,**'sort.in'**);**

Read**(**N**);** *{ Читаем количество элементов в массиве }*

**for** i**:=**1 **to** N **do** **begin**

Read**(**Ai**);** *{ Читаем элементы массива }*

*{ Считаем количество элементов каждого вида прямо при чтении файла }*

P**[**Ai**]** **:=** P**[**Ai**]** **+** 1**;** *{ Увеличиваем P[Ai] - количество элементов равных Ai }*

**end;**

*{ Вывод отсортированного массива в файл }*

Rewrite**(**Output**,**'sort.out'**);**

**for** Ai**:=**1 **to** 1000 **do** *{ Пробегаем по всем возможным значениям }*

**for** i**:=**1 **to** P**[**Ai**]** **do** *{ Выводим P[Ai] элементов Ai }*

Write**(**Ai**,**' '**);**

**end.**

## "Быстрая сортировка" QuickSort

Один из быстрых известных универсальных алгоритмов сортировки массивов (в среднем *O*(*n*log*n*) обменов при упорядочении *n* элементов).

Описание алгоритма:

* выбрать элемент, называемый опорным.
* сравнить все остальные элементы с опорным, на основании сравнения разбить множество на три — «меньшие опорного», «равные» и «большие», расположить их в порядке меньшие-равные-большие.
* повторить рекурсивно для «меньших» и «больших».



**var** A **:** **array** **[**1**..**1000000**]** **of** **Integer;** *{ Сортируемый массив }*

**procedure** QuickSort**(** left**,**right **:** **Integer** **);**

**var** i**,**j **:** **Integer;** m**,**temp**:** **Integer;**

**begin**

i **:=** left**;**

j **:=** right**;**

m **:=** A**[(**left **+** right**)** **div** 2**];** *{ Медиана }*

m **:=** A**[**random**(**right**-**left+1**)** **+** left**];** *{ Рандомизированный вариант QuickSort }*

*{ При выборе разделяющего элемента случайным образом практически невозможно попасть на худший случай }*

**repeat**

**while** A**[**i**]** **<** m **do** inc**(**i**);**

**while** m **<** A**[**j**]** **do** dec**(**j**);**

**if** i**<=**j **then** **begin**

*{ Меняем A[i] и A[j] местами }*

temp **:=** A**[**i**];**

A**[**i**]** **:=** A**[**j**];**

A**[**j**]** **:=** temp**;**

inc**(**i**);** dec**(**j**);**

**end;**

**until** i**>**j**;**

**if** left **<** j **then** QuickSort**(**left**,**j**);** *{ Сортировка левого куска }*

**if** i **<** right **then** QuickSort**(**i**,**right**);** *{ Сортировка правого куска }*

**end;**

**var**

N **:** **Integer;** *{ Количество элементов в массиве }*

i **:** **Integer;** *{ Переменная цикла }*

**begin**

*{ Ввод исходного массива из файла }*

Reset**(**Input**,**'bubble.in'**);**

Read**(**N**);** *{ Читаем количество элементов в массиве }*

**for** i**:=**1 **to** N **do** Read**(**A**[**i**]);** *{ Читаем сам массив }*

*{ Сортировка элементов с 1 по N }*

QuickSort**(**1**,** N**);**

*{ Вывод отсортированного массива в файл }*

Rewrite**(**Output**,**'bubble.out'**);**

**for** i**:=**1 **to** N-1 **do** Write**(**A**[**i**],**' '**);**

Writeln**(**A**[**N**]);**

**end.**

Реализация на C/C++

#include <stdio.h>

#include <iostream>

**using** **namespace** std**;**

*// Сортируемый массив*

**int** A**[**1000000**];**

*// Случайное целое число в заданном диапазоне*

**int** rand**(int** low**,** **int** high**){**

**return** rand**()** **%** **(**high **-** low **+** 1**)** **+** low**;**

**}**

*// Поменять местами 2 элемента массива i-ый и j-ый*

**void** swap**(int** i**,** **int** j**){**

**int** temp **=** A**[**i**];**

A**[**i**]** **=** A**[**j**];**

A**[**j**]** **=** temp**;**

**}**

*// Быстрая сортировка от элемента left до элемента right*

**void** qsort**(int** left**,** **int** right**){**

*// Массивы из 1-ого элемента сортировать незачем*

**if(**left **>=** right**)** **return;**

**int** l **=** left**,** r **=** right**;** *// Левая и правая границы*

*// Выбираем разделяющий элемент - может быть любым,*

*// но выбор случайного элемента устойчив к худшим случаям*

**int** m **=** A**[**rand**(**left**,**right**)];**

**do** **{**

**while(**A**[**l**]** **<** m**)** l**++;** *// Двигаем левую границу*

**while(**A**[**r**]** **>** m**)** r**--;** *// Двигаем правую границу*

**if(**l **<=** r**){** swap**(**l**,**r**);** l**++;** r**--;** **}** *// Меняем элементы местами*

**}** **while** **(**l **<=** r**);**

qsort**(**left**,**r**);** *// Сортируем левую половину массива*

qsort**(**l**,**right**);** *// Сортируем правую половину массива*

**}**

**int** main**(){**

freopen**(**"qsort.in"**,**"r"**,**stdin**);**

freopen**(**"qsort.out"**,**"w"**,**stdout**);**

*// Ввод исходных данных*

**int** N**;**

cin **>>** N**;**

**for(int** i**=**0**;**i<N;i++) cin >> A**[**i**];**

*// Сортировка*

qsort**(**0**,**N-1**);**

*// Вывод отсортированного массива*

**for(int** i**=**0**;**i**<**N**;**i**++)** cout **<<** A**[**i**]** **<<** " "**;**

*// Код возврата*

**return** 0**;**

**}**

## Сортировка слиянием - MergeSort

Алгоритм сортировки, который упорядочивает списки (или другие структуры данных, доступ к элементам которых можно получать только последовательно) в определённом порядке. Эта сортировка — хороший пример использования принципа «разделяй и властвуй». Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью рекурсивного вызова или непосредственно, если их размер достаточно мал. Наконец, их решения комбинируются, и получается решение исходной задачи.

3 этапа алгоритма:

1. Сортируемый массив разбивается на две части примерно одинакового размера;
2. Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом;
3. Два упорядоченных массива половинного размера соединяются в один.

#include <assert.h>

#include <stdlib.h>

#include <iostream>

**using** **namespace** std**;**

**const** **int** N **=** 100000**;** *// Количество элементов в массиве*

**int** A**[**N**];** *// Сортируемый массив*

**int** T**[**N**];** *// Временный массив - буфер*

*// left - левая граница сортируемого участка*

*// right - правая граница*

**void** MergeSort**(** **int** left**,** **int** right **){**

*// Если в массиве один элемент - сортировать нечего => сразу выходим*

**if(**left **>=** right**)** **return;**

*// Делим массив на 2 равные половинки*

**int** m **=** **(**left **+** right**)** **/** 2**;** *// Среднее арифметическое*

MergeSort**(**left**,**m**);** *// Сортируем левую половину*

MergeSort**(**m+1**,**right**);** *// Сортируем правую половину*

*// Объединяем результаты - операция Merge (слияние)*

**int** l **=** left**,** r **=** m+1**;**

**for(int** i **=** left**;** i**<** **=** right**;** i**++){**

**if(**l **>** m**)** *// Если левая половинка кончилась, то берём только из правой*

T**[**i**]** **=** A**[**r**++];**

**else** **if(**r **>** right**)** *// Если правая половинка кончилась, то берём только из правой*

T**[**i**]** **=** A**[**l**++];**

**else** **if(**A**[**l**]** **<** A**[**r**])** *// Если обе не кончились, то берём минимальный элемент*

T**[**i**]** **=** A**[**l**++];**

**else**

T**[**i**]** **=** A**[**r**++];**

**}**

*// Копируем обратно из временно буфера в A*

**for(int** i **=** left**;** i**<=**right**;** i**++)**

A**[**i**]** **=** T**[**i**];**

*// Проверка, что всё верно отсортировано*

**for(int** i **=** left**;** i**<** **=** right-1**;** i**++)**

assert**(** A**[**i**]** **<=** A**[**i+1**]** **);**

**}**

**int** main**()** **{**

*// Заполняем случайным числами массив A*

**for(int** i**=**0**;**i**<**N**;**i**++)** A**[**i**]** **=** rand**()** **%** 1000**;**

*// Вызов сортировки всех элементов*

MergeSort**(**0**,**N-1**);**

*// Вывод отсортированного массива*

**for(int** i**=**0**;**i**<**N**;**i**++)** cout **<<** A**[**i**]** **<<** " "**;**

cout **<<** endl**;**

**return** 0**;**

**}**

*// a - сортируемый массив, его левая граница lb, правая граница ub*

**template<class** T**>**

**void** mergeSort**(**T a**[],** **long** lb**,** **long** ub**)** **{**

**long** split**;** *// индекс, по которому делим массив*

**if** **(**lb **<** ub**)** **{** *// если есть более 1 элемента*

split **=** **(**lb **+** ub**)/**2**;** *// Центр обрабатываемого куска массива*

mergeSort**(**a**,** lb**,** split**);** *// сортировать левую половину*

mergeSort**(**a**,** split+1**,** last**);***// сортировать правую половину*

merge**(**a**,** lb**,** split**,** ub**);** *// слить результаты в общий массив*

**}**

**}**

**template<class** T**>**

**void** merge**(**T a**[],** **long** lb**,** **long** split**,** **long** ub**)** **{**

*// Слияние упорядоченных частей массива в буфер temp*

*// с дальнейшим переносом содержимого temp в a[lb]...a[ub]*

*// текущая позиция чтения из первой последовательности a[lb]...a[split]*

**long** pos1**=**lb**;**

*// текущая позиция чтения из второй последовательности a[split+1]...a[ub]*

**long** pos2**=**split+1**;**

*// текущая позиция записи в temp*

**long** pos3**=**0**;**

T **\***temp **=** **new** T**[**ub**-**lb+1**];**

*// идет слияние, пока есть хоть один элемент в каждой последовательности*

**while** **(**pos1 **<=** split **&&** pos2 **<=** ub**)** **{**

**if** **(**a**[**pos1**]** **<** a**[**pos2**])**

temp**[**pos3**++]** **=** a**[**pos1**++];**

**else**

temp**[**pos3**++]** **=** a**[**pos2**++];**

**}**

*// одна последовательность закончилась -*

*// копировать остаток другой в конец буфера*

**while** **(**pos2 **<=** ub**)** *// пока вторая последовательность непуста*

temp**[**pos3**++]** **=** a**[**pos2**++];**

**while** **(**pos1 **<=** split**)** *// пока первая последовательность непуста*

temp**[**pos3**++]** **=** a**[**pos1**++];**

*// скопировать буфер temp в a[lb]...a[ub]*

**for** **(**pos3 **=** 0**;** pos3 **<** ub**-**lb+1**;** pos3**++)**

a**[**lb**+**pos3**]** **=** temp**[**pos3**];**

**delete** temp**[**ub**-**lb+1**];**

**}**

## Реализация на Delphi

**var**

N **:** **integer;** *{ Количество элементов }*

A**,** T**:** **array** **[**1 **..** 1000000**]** **of** **integer;** *{ Сортируемый массив }*

Inversions**:** **int64** **=** 0**;** *{ Для подсчёта числа инверсий }*

**procedure** MSort**(**l**,** r**:** **integer);**

**var**

e**,** i**,** j**,** k**:** **integer;**

**begin**

**if** l **>=** r **then**

exit**;**

k **:=** **(**l **+** r**)** **div** 2**;** *{ Средний элемент }*

MSort**(**l**,** k**);**

MSort**(**k **+** 1**,** r**);**

i **:=** l**;** *{ Первая половина массива }*

j **:=** k **+** 1**;** *{ Вторая половина массива }*

*{ Слияние }*

**for** e **:=** l **to** r **do** **begin** *{ Куда записываем результат }*

**if** i **>** k **then** **begin**

T**[**e**]** **:=** A**[**j**];**

Inc**(**j**);**

Inc**(**Inversions**,**k**-**i+1**);** *{ Подсчёт инверсий }*

**end** **else** **if** j **>** r **then** **begin**

T**[**e**]** **:=** A**[**i**];**

Inc**(**i**);**

**end** **else** **if** A**[**i**]** **<=** A**[**j**]** **then** **begin**

T**[**e**]** **:=** A**[**i**];**

Inc**(**i**);**

**end** **else** **begin**

T**[**e**]** **:=** A**[**j**];**

Inc**(**j**);**

Inc**(**Inversions**,**k**-**i+1**);** *{ Подсчёт инверсий }*

**end;**

**end;**

*{ Обратное копирование из T }*

**for** e **:=** l **to** r **do**

A**[**e**]** **:=** T**[**e**];**

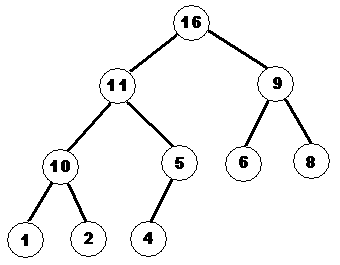
**end;**

## Heap - куча

**Бинарное дерево** - у каждого родителя максимум 2 потомка

Предок: *Parent*=*i*2. Потомки: левый *l*=2*i*, правый *r*=2*i*+1.

Основное свойство кучи, любая функция, которая допускает линейное упорядочивание.



## HeapSort - сортировка при помощи кучи

*{$APPTYPE CONSOLE}*

**uses** SysUtils**;**

**var**

N **:** **integer;** *{ Размер кучи }*

A **:** **array** **[**1**..**1000000**]** **of** **int64;**

size **:** **Integer;**

*{ Обмен местами элементов с индексами i и j }*

**procedure** Swap**(** i**,**j **:** **integer** **);**

**var** T **:** **int64;**

**begin**

T **:=** A**[**i**];**

A**[**i**]** **:=** A**[**j**];**

A**[**j**]** **:=** T**;**

**end;**

**procedure** Heapify**(** i**:integer** **);**

**var** r**,**l **:** **integer;**

**begin**

l **:=** i**\***2**;** *{ Левый потомок }*

r **:=** i**\***2+1**;** *{ Правый потомок }*

**if** l **<=** N **then**

**if** A**[**l**]** **>** A**[**i**]** **then** **begin**

Swap**(**l**,**i**);**

Heapify**(**l**);**

**end;**

**if** r **<=** N **then**

**if** A**[**r**]** **>** A**[**i**]** **then** **begin**

Swap**(**r**,**i**);**

Heapify**(**r**);**

**end;**

**end;**

*{ Построение кучи }*

**procedure** BuildHeap**;**

**var** i**:** **integer;**

**begin**

**for** i **:=** N **downto** 1 **do**

Heapify**(**i**);**

**end;**

*{ Сортировка кучей }*

**procedure** HeapSort**;**

**begin**

BuildHeap**;**

size **:=** N**;**

**while** N **>** 1 **do** **begin**

Swap**(**N**,**1**);** *{ Меняем местами максимальный элемент и элемент в конце кучи }*

Dec**(**N**);** *{ Уменьшаем кучу }*

Heapify**(**1**);** *{ Восстанавливаем основное свойство кучи }*

**end;**

**end;**

**var** i **:** **integer;**

**begin**

Reset**(**Input**,**'heapsort.in'**);**

Rewrite**(**Output**,**'heapsort.out'**);**

*{ Чтение исходных данных }*

Read**(**N**);**

**for** i **:=** 1 **to** N **do**

Read**(**A**[**i**]);**

*{ Вызов сортировки }*

HeapSort**;**

*{ Запись отсортированного массива }*

**for** i **:=** 1 **to** size **do**

Write**(**A**[**i**],**' '**);**

**end.**

#include <assert.h>

#include <stdlib.h>

#include <iostream>

**using** **namespace** std**;**

**const** **int** N **=** 80**;**

*// Массив, который на время станет двоичной кучей, чтобы потом стать отсортированным массивом*

**int** A**[**N**];**

*// Текущий размер кучи, т.е. сколько первых элементов массива A сейчас являются двоичной кучей*

**int** HeapSize**;**

*// Поменять местами 2 элемента массива i-ый и j-ый*

**inline** **void** swap**(int** i**,** **int** j**){**

**int** temp **=** A**[**i**];**

A**[**i**]** **=** A**[**j**];**

A**[**j**]** **=** temp**;**

**}**

*// Выполняется ли основное свойство кучи*

**bool** heap\_function**(int** parent**,** **int** child**){**

**return** A**[**parent**]** **>** A**[**child**];**

**}**

*// Мы хотим сверху получить самый минимум*

*// parent - корневой элемент*

*// Мы обновили элемент parent и хотим чтобы куча снова была кучей*

**void** heap**(int** parent**){**

*// -= Левый потомок =-*

**int** left **=** 2**\***parent **+** 1**;**

**if(**left **<** HeapSize**)** *// Левый потомок есть*

**if(!**heap\_function**(**parent**,**left**)){** *// Проверка основного свойства кучи*

*// Меняем их местами*

swap**(**left**,**parent**);**

heap**(**left**);** *// Проталкиваем значение дальше*

**}**

*// -= Правый потомок =-*

**int** right **=** 2**\***parent **+** 2**;**

**if(**right **<** HeapSize**)** *// Правый потомок есть*

**if(!**heap\_function**(**parent**,**right**)){** *// Проверка основного*

*// Меняем их местами*

swap**(**right**,**parent**);**

heap**(**right**);**

**}**

**}**

**void** HeapSort**(){**

HeapSize **=** N**;**

*// Строим кучу*

**for(int** i**=**N-1**;**i**>=**0**;**i**--)**

heap**(**i**);**

*// Выводим очередной элемент*

**for(int** i**=**0**;**i**<**N**;**i**++){**

*// Ставим элемент с вершины кучи в конец массива*

swap**(**0**,**HeapSize-1**);**

*// Уменьшаем кучу*

HeapSize**--;**

*// Упорядочиваем вершину кучи*

heap**(**0**);**

**}**

*// Проверяем, что массив отсортирован*

**for(int** i**=**0**;**i**<**N-1**;**i**++)**

assert**(** A**[**i**]** **<=** A**[**i+1**]** **);**

**}**

**int** main**()** **{**

*// Заполняем случайным числами массив A*

**for(int** i**=**0**;**i**<**N**;**i**++)** A**[**i**]** **=** rand**()** **%** 1000**;**

*// Вызов сортировки*

HeapSort**();**

*// Вывод отсортированного массива*

**for(int** i**=**0**;**i**<**N**;**i**++)** cout **<<** A**[**i**]** **<<** " "**;**

cout **<<** endl**;**

**return** 0**;**

**}**

## Алгоритмы сортировки в стандартных библиотеках

Использование qsort в C/C++

В C/C++ в STL (Standard Template Library, которую, как правило, можно использовать на олимпиадах) доступна библиотека**algorithm** в которой в числе прочего есть реализованная процедура сортировки **sort**.

#include <stdio.h>

#include <assert.h>

#include <math.h>

#include <string.h>

#include <vector>

#include <algorithm>

#define MaxN 5000

**using** **namespace** std**;**

**int** a**[**MaxN **+** 3**];**

**int** n**;**

**int** main**()** **{**

freopen**(**"bubble.in"**,** "rt"**,** stdin**);** *// Открываем входной файл*

freopen**(**"bubble.out"**,** "wt"**,** stdout**);** *// Открываем выходной файл*

*// Чтение исходных данных*

scanf**(**"%d"**,** **&**n**);** *// Чтение количества элементов*

**int** i**;**

**for(**i **=** 0**;** i **<** n**;** i**++)** scanf **(**"%d"**,** **&**a**[**i**]);**

*// Вызов процедуры сортировки*

sort**(**a**,** a **+** n**);**

*// Вывод отсортированного массива в файл*

**for(**i **=** 0**;** i **<** n **-** 1**;** i**++)**

printf**(**"%d "**,** a**[**i**]);**

printf**(**"%d\n"**,** a**[**n **-** 1**]);**

**return** 0**;**

**}**

В Python у списков (list) есть встроенный метод sort

*# -\*- coding: windows-1251 -\*-*

import sys

sys**.**stdin **=** open**(**'bubble.in'**)** *# Открываем входной файл на чтение*

sys**.**stdout **=** open**(**'bubble.out'**,** 'w'**)** *# Открываем выхолной файл на запись*

*# Ввод исходных данных*

N **=** input**()** *# Считываем из первой строки целое число*

A **=** map**(**int**,**raw\_input**().**split**())** *# Считываем строку, делим её на элементы и каждый приводим к типу целое*

*# Сортировка массива*

A**.**sort**()**

*# Вывод массива*

**for** i **in** A**:**

**print** i**,**

# Двоичный (бинарный) поиск, BinSearch, БинПоиск

Поиск в упорядоченном массиве за *O*(log*n*). Так же его называют - метод деления пополам и дихотомия (деление пополам по-гречески).

Перед применением двоичного поиска нужно отсортировать массив одним из [алгоритмов сортировки](http://ts.lokos.net/lisiynos/s1/sort.html).

### Цель

Найти элемент со значением *x* в отсортированном массиве *A* из *N* элементов или установить, что элемента *x* в массиве *A*нет.

### Идея

Разделить отсортированный массив на две половины, сравнить средний элемент с *x*, понять в какой половине массива может находиться значение *x* и перейти к поиску в этой половине.. И так далее, пока размер массива не уменьшиться до 1 элемента, тогда либо этот элемент равен *x* и мы нашли *x*, либо не равен *x*, и тогда элемента *x* нет в массиве *A*.

**Скорость работы**

Линейный поиск (последовательный просмотр всех элементов массива) выполняется за *O*(*N*) операций, двоичный поиск - за*O*(log*N*).

**Пример реализации на C++**

Шаблонная функция и пример её использования:  
template <typename  T> int binary\_serch (const T \*a,int n ,const  T &elem);  
Пусть а - отсортированный массив, тогда функция за логарифмическое время возвращает *i*-ый индекс элемента массива elem=a[i] и -1, если элемент не найден.

#include <iostream> *// Отсюда будем использовать вывод на экран при потоков (cout)*

#include <algorithm> *// Алгоритм сортировки (функция sort) из STL*

#include <stdlib.h> *// Функция rand() - случайные числа*

#include <ctime> *// Функция time() - время*

#include <conio.h> *// Функция getch() - ожидание нажатия клавиши в конце программы*

*// Шаблонная функция*

**template** **<typename** T**>** **int** binary\_search**(const** T **\***a**,** **int** n**,** **const** T **&**elem**){**

**int** L **=** 0**,** R **=** n-1**;** *// Левая и правая границы поиска*

**while** **(**L **<** R**){** *// Пока в рассматриваемом куске массива больше одного элемента*

**int** m **=** **(**L**+**R**)/**2**;** *// Вычисляем индекс среднего элемента*

**if** **(**elem **<=** a**[**m**])** *// Если искомый элемент меньше центрального*

R **=** m**;** *// Сдвигаем правую границу влево*

**else** *// , а иначе*

L **=** m+1**;** *// Левую границу вправо*

**}**

*// Остался один элемент с индеком L = R,*

*// и это либо искомый элемент elem, либо elem в массиве вообще нет!*

**return** **(**a**[**R**]** **==** elem**)** **?** R **:** **-**1**;** *// возвращаем индекс или -1 если элемент не найден*

**}**

**using** **namespace** std**;** *// Чтобы не писать перед каждым cout "std::"*

*// Основная программа c примером использования*

**int** main**()** **{**

*// Создаём случайный массив из N элементов*

**const** **int** N **=** 20**;**

**int** a**[**N**];**

*// Заполняем массив a случайными целыми числами*

srand**(**time**(**0**));** *// Инициализация генератора случайных чисел*

**for(int** i**=**0**;**i**<**N**;**i**++)**

a**[**i**]** **=** rand**()** **%** 200**;** *// случайные числа в диапазоне 0..199*

**int** el1 **=** a**[**rand**()** **%** N**];** *// Запоминаем случайный элемент массива*

*// он точно будет в отсортированном массиве :)*

*// Сортируем массив a при помощи функции sort из STL*

sort**(**a**,** a**+**N**);** *// Да-да.. вот так просто, одной строкой, можно отсортировать массив ;)*

*// Выводим массив на экран*

**for(int** i**=**0**;**i**<**N**;**i**++)**

cout **<<** "a[" **<<** i **<<** "] = " **<<** a**[**i**]** **<<** endl**;**

*// Поиск элементов массива при помощи функции binary\_search*

*// Ищем элемент массива, который там заведомо есть*

cout **<<** "search " **<<** el1 **<<** " " **<<** binary\_search**(**a**,**N**,**el1**)** **<<** endl**;**

*// Поиск элемента, которого в массиве a заведомо нет, т.к. он больше 199*

cout **<<** "search " **<<** 1000 **<<** " " **<<** binary\_search**(**a**,**N**,**1000**)** **<<** endl**;**

cout **<<** "Press any key..."**;** getch**();** *// Ожидаем нажатия клавиши :)*

**return** 0**;**

**}**

Стандартная STL функция binary\_search возвращает лишь найден элемент или нет.

#include <iostream> *// Вывод на экран*

#include <algorithm> *// Алгоритм сортировки из STL*

#include <stdlib.h> *// Функция rand() - случайные числа*

#include <ctime> *// Функция time() - время*

*// Основная программа для демонстрации функции*

**using** **namespace** std**;** *// Чтобы не писать везде "std::"*

**int** main**()** **{**

*// Создаём случайный массив из N элементов*

**const** **int** N **=** 20**;**

**int** a**[**N**];**

*// Заполняем случайными числами*

srand**(**time**(**0**));** *// Инициализация генератора случаных чисел*

**for(int** i**=**0**;**i**<**N**;**i**++)**

a**[**i**]** **=** rand**()** **%** 200**;**

**int** el1 **=** a**[**rand**()** **%** N**];** *// Запоминаем элемент массива*

*// Сортируем массив a*

sort**(**a**,** a**+**N**);**

*// Выводим массив на экран*

**for(int** i**=**0**;**i**<**N**;**i**++)**

cout **<<** "a[" **<<** i **<<** "] = " **<<** a**[**i**]** **<<** endl**;**

*// Поиск элементов массива*

*// Ищем элемент массива, который там заведомо есть*

cout **<<** "search " **<<** el1 **<<** " " **<<** binary\_search**(**a**,**a**+**N**,**el1**)** **<<** endl**;**

cout **<<** "search " **<<** 1000 **<<** " " **<<** binary\_search**(**a**,**a**+**N**,**1000**)** **<<** endl**;**

**return** 0**;**

**}**

### Реализация на Delphi

Добавить картинки

*{$APPTYPE CONSOLE}*

**uses** SysUtils**;**

**procedure** swap**(var** a**,** b**:** **int64);**

**var** t**:** **int64;**

**begin**

t **:=** a**;** a **:=** b**;** b **:=** t**;**

**end;**

**var**

n**,** i**,** j**,** m**,** k**,** ans**,** l**,** r**:** **integer;**

a**:** **array** **[**1**..**1000000**]** **of** **int64;**

**procedure** qsort**(**l**,** r**:** **integer);**

**var**

i**,** j**:** **integer;**

m**:** **int64;**

**begin**

i **:=** l**;**

j **:=** r**;**

m **:=** a**[**random**(**r **-** l **+** 1**)** **+** l**];**

**while** i **<=** j **do** **begin**

**while** a**[**i**]** **<** m **do** inc**(**i**);**

**while** a**[**j**]** **>** m **do** dec**(**j**);**

**if** i **<=** j **then** **begin**

swap**(**a**[**i**],** a**[**j**]);**

inc**(**i**);**

dec**(**j**);**

**end;**

**end;**

**if** i **<** r **then** qsort**(**i**,** r**);**

**if** l **<** j **then** qsort**(**l**,** j**);**

**end;**

**begin**

assign**(**input**,** 'find.in'**);** reset**(**input**);**

assign**(**output**,** 'find.out'**);** rewrite**(**output**);**

read**(**n**,** l**);**

**for** i **:=** 1 **to** n **do**

read**(**a**[**i**]);**

qsort**(**1**,** n**);**

**for** j **:=** 1 **to** l **do** **begin**

read**(**k**);**

l **:=** 1**;**

r **:=** n **+** 1**;**

**While** r **-** l **>** 1 **do** **begin**

m **:=** **(**r **+** l**)** **div** 2**;**

**if** k **>=** a**[**m**]** **then** l **:=** m **else** r **:=** m**;**

**end;**

**if** k **=** a**[**l**]** **then** writeln**(**'YES'**)** **else** writeln**(**'NO'**);**

**end;**

**end.**

# Представление чисел в памяти и точность вычислений

## Представление целых чисел в памяти компьютера

Микропроцессор рассчитан на определённую разрядность поступающих в него данных, у процессора определённое ограниченное количество ножек, каждая ножка отвечает за определённый двоичный разряд числа. Например, если процессор является 32-битным, то основной размер числа для него 32 бита, т.е. число размером 32 бита (4 байта) может быть передано в память или из памяти за один такт (один цикл обработки данных процессором).

Наиболее эффективным по скорости является использование "родной" для этого процессора разрядности: если процессор 16-битный, то это 16 бит, если 32-битный, то 32 бита. Если число больше по размеру (например: 40 бит для 32-битного процессора), то процессор будет его обрабатывать уже (минимум) в два прохода. Если число будет меньше чем 32 бита, то процессор всё равно не сможет обработать несколько чисел за такт, а если потребуется сложить или вычесть два числа с разной разрядностью, то потребуется ещё и привести их к разрядности большего числа. Так что самое выгодное с точки зрения скорости - использовать для всех целых чисел тип данных с "родной" для процессора разрядностью (например, при программированни на Delphi для 32-битных процессоров использовать тип Integer). С точки зрения экономии памяти выгодно использовать как можно меньше байт памяти (типы как можно меньшего размера).

Какие же числа можно представить в 8 битах? В одном бите можно представить только 2 значения: 0 или 1. В двух битах – 4 значения: 00, 01, 10, 11. В *N* битах можно представить 2*N* различных значений. Для целых неотрицательных чисел логично считать, что когда все разряды 0, то это число 0, а когда все разряды 1, то это максимальное число, которое в этом случае равно 2*N*−1.

Если число не укладывается ни в один из стандартных типов, то надо реализовывать ["Длинную арифметику" (ссылка на урок по "Длинной арифметике")](http://ts.lokos.net/lisiynos/s1/long_ar.html).

## Целые числа и дополнительный код

**Дополнительный код**- самый распространённый способ представления отрицательных целых чисел в компьютерах. Он позволяет заменить операцию вычитания на операцию сложения (используя свойства переполнения) и сделать операции сложения и вычитания одинаковыми для знаковых и беззнаковых чисел, чем упрощает архитектуру ЭВМ. Дополнительный код отрицательного числа можно получить инвертированием модуля двоичного числа (первое дополнение) и прибавлением к инверсии единицы (второе дополнение). Либо вычитанием числа из нуля.

Примеры:

* 1 => 00000001
* 0 => 00000000
* -1 => 11111111

## Инструмент исследования представления чисел в памяти

Для того чтобы исследовать формат представления переменных различных типов в памяти можно использовать следующий приём: создать указатель на массив байт и сделать, чтобы он ссылался на нужную нам исследуемую переменную и вывести побайтно содержимое памяти (столько байт, сколько занимает эта переменная).

*{ Массив для вывода шестнадцатеричных цифр }*

**const** Hex **:** **array** **[**0**..**15**]** **of** **Char** **=**

**(**'0'**,**'1'**,**'2'**,**'3'**,**'4'**,**'5'**,**'6'**,**'7'**,**'8'**,**'9'**,**'A'**,**'B'**,**'C'**,**'D'**,**'E'**,**'F'**);**

**var**

X **:** **Integer;** *{ X - исследуемая переменная, Integer - исследуемый тип (замените на интересующий) }*

P **:** **array** **of** **Byte** **=** @X**;** *{ P - указатель на массив байт, который указывает на переменную X }*

i**,**j **:** **Integer;** *{ Переменные циклов }*

**begin**

*{ Считываем исходное значение }*

Readln**(**X**);**

*{ Вывод в двоичной системе счисления }*

**for** i **:=** sizeof**(**x**)** **-** 1 **downto** 0 **do** **begin** *{ Цикл по байтам (начиная со старшего) }*

**for** j **:=** 7 **downto** 0 **do** *{ Цикл по битам (начиная со старшего) }*

write**((**P**[**i**]** **shr** j**)** **and** 1**);** *{ Получаем бит j: сдвигаем на j-бит вправо и берём последний бит }*

**if** i **>** 0 **then** write**(**' '**);** *{ Между байтами выводим пробел }*

**end;**

writeln**;**

*{ Вывод в шестнадцатеричной системе счисления }*

**for** i **:=** sizeof**(**x**)** **-** 1 **downto** 0 **do** **begin**

*{ Вывод 2-х шестнадцатеричных цифр - содержимое i-ого байта }*

write**(**Hex**[**P**[**i**]** **div** 16**],** Hex**[**P**[**i**]** **mod** 16**]);**

**if** i **>** 0 **then** write**(**' '**);**

**end;**

writeln**;**

**end.**

Вместо массива шестнадцатеричных символов можно использовать функцию (в ней труднее ошибиться):

**procedure** out\_hex\_digit**(**b**:** **Byte);**

**begin**

**if** b **<** 10 **then** write**(**chr**(**b **+** ord**(**'0'**)))**

**else** write**(**chr**(**b **+** ord**(**'A'**)** **-** 10**));**

**end;**

В этом примере были использованы побитные операции:

* A **shr** B - сдвиг числа A вправо на B бит (младшие биты при этом теряются).
* A **and** B - побитная операция "И" в C := A and B бит будет установлен в 1, только если он 1 и в A и B.

Теперь, когда у нас есть программа для проведения экспериментов, мы можем изучить как представляются различные типы данных в памяти.

## Неточность представления чисел с плавающей запятой

Как хранить нецелые числа в компьютере? Вариантов много. Можно, например, хранить их в виде рациональных дробей (тогда мы можем использовать механизм хранения целых чисел).

Любые типы данных в компьютере хранятся как последовательность бит. Для записи действительных чисел хранится порядок и мантисса. Т.е. число хранится как двоичная дробь. Вначале стоит точка, затем мантисса, а затем порядок (двоичная степень).

Любое действительное число не кратное 2 хранится неточно. С погрешностью в последнем бите мантиссы. Задача про сравнение это наглядно показывает.

**uses** SysUtils**;**

**var** a**,**b**,**c**:extended;**

**begin**

reset**(**input**,**'eq.in'**);**

rewrite**(**output**,**'eq.out'**);**

readln**(**a**,**b**,**c**);**

**if** **(**abs**(**a**+**b**-**c**)** **<** 0.0000001**)** **then** writeln**(**'YES'**)** **else** writeln**(**'NO'**);**

**end.**

## Эксперимент на Delphi на точность сравнения

*{$APPTYPE CONSOLE}*

*{ Сравнение без учёта погрешности машинного представления чисел }*

**function** eq1**(**a**,**b**,**c**:extended):boolean;**

**begin**

result **:=** a**+**b **=** c**;**

**end;**

*{ С учётом погрешности }*

**function** eq2**(**a**,**b**,**c**:extended):boolean;**

**begin**

result **:=** abs**(**a**+**b**-**c**)** **<** 1e-17**;**

**end;**

**var** a**,**b**,**c **:** **extended;**

**begin**

a **:=** 0.1**;**

b **:=** 1.2**;**

c **:=** 1.3**;**

writeln**(**a**+**b**=**c**);** **//** **FALSE**

writeln**(**0.1+1.2**=**1.3**);** **//** **TRUE,** хотя написано**,** вроде бы**,** то же самое

writeln**(**0.1+0.1+0.1-0.3**=**0.0**);** **//** **TRUE**

writeln**(**1+0.1-1**=**0.1**);** **//** **FALSE,** как ни странно **:)**

**end.**

Для предоления проблем с точностью вычислений в языке программирования Python есть специальный тип Decimal, этот тип основан на модели чисел с плавающей точкой, которая разработана по принципу: "компьютер должен обеспечивать арифметические действия, которые работают так же, как людей обучают в школе".

Например: 0.1+0.1+0.1-0.3 в точности равно 0.

from decimal import **\***

## Действительные числа (с плавающей точкой, floating point)

Задачки с трудностями с точностью результата бывают на олимпиадах любого уровня. Запоминать битовое представление чисел - необязательно, но знать что могут быть проблемы с точностью и как их решать - нужно.

*{$apptype console}*

**function** convert**(**x **:** **extended)** **:** **int64;**

**begin**

result **:=** trunc**(**x**);**

assert**(**abs**(**result**)** **<** **int64(**1**)** **\*** 1000000 **\***

1000000 **\***

1000000**);**

**end;**

**var** a**,** b **:** **extended;**

**begin**

a **:=** **-**706378499182879656**;**

b **:=** **-**513623273852583522**;**

writeln**((**a**+**b**):**0**:**0**);**

writeln**(**convert**(**a**)** **+** convert**(**b**));**

**end.**

# Рекурсивные алгоритмы и их построение

Помните детское стихотворение-считалку про 10 негритят? Оно может служить эпиграфом к любой работе на тему "Рекурсия".

«10 негритят пошли купаться в море,

10 негритят резвились на просторе,

Один из них пропал – и вот вам результат:

9 негритят пошли купаться в море,

9 негритят резвились на просторе,

Один из них пропал – и вот вам результат:

… … … … …

1 (из) негритят пошли(ел) купаться в море,

1 (из) негритят резвились(ся)на просторе,

Один из них пропал – и вот вам результат:

Нет больше негритят!»

Первые три строчки этого стихотворения повторяются 10 раз с небольшим изменением - число негритят уменьшается с каждым разом на единицу. И только, когда число негритят уменьшилось до нуля, стихотворение заканчивается единственной строчкой «Нет больше негритят!». Напишем процедуру, печатающую это стихотворение.

**procedure** Negr**(**k**:** **integer);** *{k - число негритят, параметр процедуры}*

**begin**

**if** k**=**0 **then** *{проверка, что число негритят равно нулю}*

Writeln**(**'Нет больше негритят!'**)** *{выход из рекурсии }*

**else** **begin**

Writeln**(**k**,**' негритят пошли купаться в море,'**);**

Writeln**(**k**,**' негритят резвились на просторе,'**);**

Writeln**(**'Один и них пропал - и вот вам результат:'**);**

Negr**(**k-1**);** *{Вызов процедуры с уменьшенным на 1 параметром}*

**end**

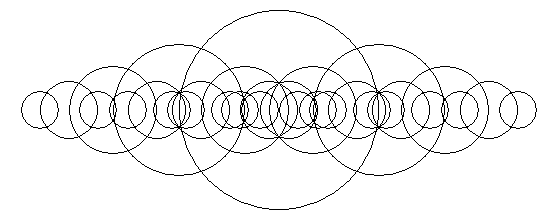
**end;**

Вызов этой процедуры в основной программе будет выглядеть так: Negr(10).

Интересным является применение рекурсии при создании рисунков.

***Пример 1*.**

Рассмотрим простейший рисунок из окружностей разных радиусов.



Если вглядеться в него внимательно, то можно заметить, что рисунок начинается с центральной окружности самого большого радиуса. Затем осуществляется переход на концы горизонтального диаметра окружности, которые должны играть роль центров двух окружностей меньшего радиуса (примерно в полтора раза). Этот же процесс повторяется и с этими двумя окружностями, и с полученными четырьмя новыми, и так далее до тех пор, пока уменьшающийся радиус окружности не станет меньше первоначального в 1,5\*1,5\*1,5\*1,5 раз (если посчитать, то должно выполниться четыре вложенных вызова рекурсивной процедуры).

Рекурсивная процедура, выполняющая такой рисунок, должна иметь в качестве передаваемых параметров координаты центра окружности и величину радиуса.

Программа, при помощи которой будет нарисован этот рисунок, может быть написана так:

**uses** Graph**;**

**procedure** Ris**(**x**,**y**,**r**:integer);**

**begin**

**If** r**>=**10 **then** **begin**

Circle**(**x**,**y**,**r**);** *{ Рисуем окружность }*

*{ Вызываем 2 раза рекурсивно саму себя }*

Ris**(**x**-**r**,**y**,**r**\***2 **div** 3**);**

Ris**(**x**+**r**,**y**,**r**\***2 **div** 3**)**

**end**

**end;**

**var** GD**,**GM **:** **integer;**

**begin**

GD **:=** detect**;**

GM **:=** 1**;**

InitGraph**(**GD**,**GM**,**''**);**

Ris**(**320**,**240**,**100**);** *{ Рисуем по центру экрана, самая большая окружность - 100 пикселей }*

Readln**;**

CloseGraph**;**

**end.**

Реализация на Delphi с сохранением в файл.

**unit** MainUnit**;**

**interface**

**uses**

Winapi**.**Windows**,** Winapi**.**Messages**,** System**.**SysUtils**,** System**.**Variants**,**

System**.**Classes**,** Vcl**.**Graphics**,** Vcl**.**Controls**,** Vcl**.**Forms**,** Vcl**.**Dialogs**,**

Vcl**.**ExtCtrls**,** PngImage**;**

**type**

TDrawForm **=** **class(**TForm**)**

Image**:** TImage**;**

**procedure** FormCreate**(**Sender**:** TObject**);**

**private**

**procedure** SaveToPNG**(**FileName**:** **string);**

**procedure** Circle**(**x**,** y**,** r**:** **integer);**

**procedure** Ris**(**x**,** y**,** r**:** **integer);**

**public**

**end;**

**var**

DrawForm**:** TDrawForm**;**

**implementation**

*{$R \*.dfm}*

**procedure** TDrawForm**.**FormCreate**(**Sender**:** TObject**);**

**begin**

*{ Не закрашивать фигуру }*

Image**.**Canvas**.**Brush**.**Style **:=** bsClear**;**

*{ Подстройка ширины и высоты картинки }*

Image**.**Width **:=** 559**;**

Image**.**Height **:=** 220**;**

Image**.**Picture**.**Bitmap**.**Width **:=** Image**.**Width**;**

Image**.**Picture**.**Bitmap**.**Height **:=** Image**.**Height**;**

*{ Вызов рекурсивной процедуры }*

Ris**(**Image**.**Width **div** 2**,** Image**.**Height **div** 2**,** 100**);**

*{ Сохранение в формате .png }*

SaveToPNG**(**'ris.png'**);**

**end;**

*{ Сохранение в формате .png: FileName - имя файла }*

**procedure** TDrawForm**.**SaveToPNG**(**FileName**:** **string);**

**var**

png**:** TPngImage**;**

**begin**

png **:=** TPngImage**.**Create**;**

png**.**Assign**(**Image**.**Picture**.**Bitmap**);**

png**.**SaveToFile**(**FileName**);**

png**.**Free**;**

**end;**

*{ Окружность с центром: x, y - центр окружности, r - радиус }*

**procedure** TDrawForm**.**Circle**(**x**,** y**,** r**:** **integer);**

**begin**

Image**.**Canvas**.**Ellipse**(**x **-** r**,** y **-** r**,** x **+** r**,** y **+** r**);**

**end;**

*{ Рекурсивная процедура }*

**procedure** TDrawForm**.**Ris**(**x**,** y**,** r**:** **integer);**

**begin**

**if** r **>=** 15 **then**

**begin**

Circle**(**x**,** y**,** r**);**

Ris**(**x **-** r**,** y**,** r **\*** 2 **div** 3**);**

Ris**(**x **+** r**,** y**,** r **\*** 2 **div** 3**)**

**end;**

**end;**

**end.**

***Пример 2.***

На входном потоке находится слово (последовательность литер), заканчивающаяся пробелом. Написать программу, которая позволит напечатать это слово "задом наперед", т.е. выписывая буквы слова с конца.

В этом примере мы опишем процедуру, у которой не будет параметров, т.к. при каждом ее вызове будет вводиться одна литера, которая будет сравниваться с пробелом. Сама же программа будет состоять только из вызова этой процедуры.

**Program** REVERSE;

**Procedure** REV;

**Var** c:char;

**Begin** read(c);

**If** c<>' '

**Then begin** REV; write(c) **end**

**End**;

**BEGIN** REV

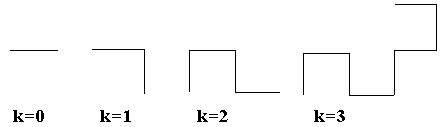
**END**.

Теперь переходим к более сложным задачам.

*Пример 3.*

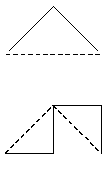
**Фракталами** называются множества, части которых являются повторением образов самих множеств. Изображения фракталов вызывают обычно у всех большой интерес.

Рассмотрим процесс сгибания бумажной полоски: если взять полоску бумаги, согнуть ее пополам К раз и развернуть полоску так, чтобы углы на сгибах стали равны 90°, то, посмотрев на торец полоски, можно увидеть ломаную, которая называется "драконовой ломаной К-го порядка", где K-количество сгибов. Схема этого процесса изображена ниже.



http://ts.lokos.net/lisiynos/s2/img/2.2_html_6f56b3c3.pnghttp://ts.lokos.net/lisiynos/s2/img/2.2_html_66fd6fc8.pnghttp://ts.lokos.net/lisiynos/s2/img/2.2_html_66fd6fc8.pnghttp://ts.lokos.net/lisiynos/s2/img/2.2_html_66fd6fc8.png

http://ts.lokos.net/lisiynos/s2/img/2.2_html_m25cbe199.png

Опишем способ создания «драконовой» ломаной.

На ломаной нулевого порядка, как на гипотенузе, строим прямой угол, на полученных сторонах прямого угла строим тоже прямые углы, но один развернут в правую сторону, а другой - в левую. Данный процесс повторяется К раз.

Напишем соответствующую программу, которая по заданному числу К рисует драконову ломаную К-го порядка.

**Program** DRACON;

**Uses** Graph;

**Var** Gd,Gm,k:Integer;

**Procedure** st (x1, y1, x2, y2, k:Integer);

**Var** xn,yn:Integer;

**Begin**

**If** k > 0

**then begin**

xn:=(x1+x2)**div** 2 + (y2-y1)**div** 2;

yn:=(y1+y2)**div** 2 - (x2-x1)**div** 2;

st(x1,y1,xn,yn,k-1);

st(x2,y2,xn,yn,k-1);

**end**

**else** Line(x1,y1,x2,y2);

**End**;

**Begin**

Gd:=Detect; Gm:=1;

InitGraph(Gd,Gm,'D:\BP\BGI');

WriteLn ('Введите номер уровня ');

Readln( k);

st(200, 300, 500, 300, k);

Readln;

CloseGraph;

**End**.

Если с помощью этой программы построить ломаную дракона 14-го порядка, то мы получим образ множества, называемого фракталом Хартера-Хейтуэя, рисунок которого приведен ниже.

## http://ts.lokos.net/lisiynos/s2/img/2.2_html_m14572a1e.png

**Длинная арифметика**

Числа, для представления которых в стандартных типах данных не хватает количества разрядов, называются**длинными**. Реализация арифметических операций над такими «длинными» числами называется **длинной арифметикой**. Задачи на «длинную» арифметику возникают, когда разрядности стандартных типов данных (целые, длинные целые, вещественные числа) не хватает чтобы хранить результат (настолько он велик).

Например, вычислить 30! (30 факториал) = 265252859812191058636308480000000, это число больше чем, например, максимальное для типа **int64** - 9223372036854775807. В этом случае используется прием хранения длинных чисел в виде строки или массива цифр, а чтобы выполнять арифметические действия с такими числами, необходимо написать специальные процедуры сложения, умножения и деления длинных чисел, которые основаны на правилах вычисления "в столбик".

Итак**Длинная арифметика** — набор структур данных и алгоритмов, которые позволяют работать с числами гораздо большими, чем это позволяют стандартные типы данных.

**Длинная арифметика используется:**

* При решении олимпиадных задач.
* В компьютерах низкой разрядности, микроконтроллерах (например, процессор умеет работать только с числами длиной 8 бит, 8 двоичных разрядов, в 8 битах можно представить только числа от 0 до 28−1=255, а требуется обрабатывать большие числа).
* Криптография.
* Математическое и финансовое ПО, требующее, чтобы результат вычисления на компьютере совпал до последнего разряда с результатом вычисления на бумаге. В частности, калькулятор Windows (начиная с 95)
* «Спортивные» вычисления знаменитых трансцендентных чисел ("число Пи", "число e" и т. д.) с высокой точностью. **Вещественное число** - число, которое может возникать как результат измерения (Например: 4,31; 5,23432; корень из 2-х). Множество вещественных чисел больше чем множество рациональных дробей (чисел представляющихся в виде дроби), но меньше чем множество комплексных чисел. **Комплексное число** - расширение множества вещественных чисел за счёт добавления мнимой компоненты числа. Комплексное число представляется в виде: *x+iy*, где x и y - вещественные числа, а i-мнимая единица.
* Высококачественные изображения фракталов.

**Представление в компьютере длинных чисел**

Реализация операций с длинными числами во многом определяется тем, как представить их в памяти компьютера. "Длинное" число можно записать, например, с помощью **массива** десятичных цифр, количество элементов в таком массиве равно количеству значащих цифр в "длинном" числе. При этом размер массива должен быть достаточным, чтобы разместить в нем и результат, например, умножения.

**Классификация реализации длинной арифметики (преимущества и недостатки различных способов)**Количество знаков может быть фиксированным (знаки хранятся в массиве с индексом от 0 до количества знаков минус 1) и с переменной длиной (отдельно ещё хравнится длина числа).

| **Количество цифр** | **Только целые числа** | **Действительные числа** |
| --- | --- | --- |
| Фиксированное | "+" Простота реализации |  |
| Переменное | "+" Экономия памяти |  |

Разряды в позиционной системе счисления.

**Рассмотрим реализацию операций в простейшем случае - с целыми положительными числами**

Существуют и другие представления "длинных" чисел. Рассмотрим одно из них. Представим наше число  
30! = 265252859812191058636308480000000 в виде: 30! = 2 \* (104)8 + 6525 \* (104)7 + 2859 \* (104)6 + 8121 \* (104)5+ 9105 \* (104)4 + 8636 \* (104)3 + 3084 \* (104)2 + 8000 \* (104)1 + 0000 \* (104)0.

Для вычислений его удобно хранить в виде массива:

| **Номер элемента в массиве А** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Значение | 9 | 0000 | 8000 | 3084 | 8636 | 9105 | 8121 | 2859 | 6525 | 2 |

Наше "длинное" число представлено в 10000-10 системе счисления (десятитысячно-десятичная система счисления, приведите аналогию с восьмерично-десятичной системой счисления), а "цифрами" числа являются четырехзначные числа. 9 в А [0] - длина числа. Число хранится "задом наперед", начиная с младшего разряда.

**Алгоритмы чтения и записи длинных чисел**

**Ввод длинного числа из файла.** Решение задачи начнем с описания данных.

**Const** MaxDig **=** 1000**;** *{ Максимальное количество цифр — четырехзначных! }*

Osn **=** 10000**;** *{ Основание нашей системы счисления, в элементах массива храним четырехзначные числа }*

**Type** TLong **=** **Array** **[**0**..**MaxDig**]** **Of** **Integer;** *{ Десятичных цифр в нашем числе }*

*{ В А[0] храним количество задействованных (ненулевых) элементов массива А. }*

При обработке каждой очередной цифры входного числа старшая цифра элемента массива с номером i становится младшей цифрой числа в элементе i + 1, а вводимая цифра будет младшей цифрой числа из А[1]. В результате работы нашего алгоритма мы получили число, записанное "задом наперед".

**procedure** ReadLong**(Var** A **:** TLong**);**

**Var** ch **:** **char;** i **:** **Integer;**

**begin**

FillChar**(**A**,** SizeOf**(**A**),** 0**);** *{ Заполнение нулями массива }*

Read**(**ch**);**

**While** **Not(**ch **In** **[**'0'**..**'9'**])** **Do** Read**(**ch**);** *{пропуск не цифр во входном файле}*

**While** ch **In** **[**'0'**..**'9'**]** **do** **begin**

**For** i **:=** A**[**0**]** **DownTo** 1 **do** **begin**

*{"протаскивание" старшей цифры в числе из A[i] в младшую цифру числа из A[i+l]}*

A**[**i**+**l**]** **:=** A**[**i**+**l**]** **+** **(LongInt(**A**[**i**])** **\*** 10**)** **Div** Osn**;**

A**[**i**]** **:=** **(LongInt(**A**[**i**])** **\*** 10**)** **Mod** Osn

**end;**

A**[**1**]** **:=** A**[**l**]** **+** Ord**(**ch**)** **-** Ord**(**'0'**);**

*{добавляем младшую цифру к числу из А[1]}*

**If** A**[**A**[**0**]+**1**]** **>** 0 **Then** Inc**(**A**[**0**]);**

*{изменяем длину, число задействованных элементов массива А}*

Read**(**ch**)**

**end**

**end;**

**Задачи на длинную арифметику**

**Задача 1.**(Районная олимпиада 1997).  
Числа Фибоначчи выписываются подряд, начиная с Ф(1). Какая цифра будет на N-ом месте (N < 5000)?

Указание: Ф(n+1) = Ф(n) + Ф(n-1); Ф(1) = Ф(2) = 1

Необходимо написать процедуру сложения «длинных» чисел (представленных в виде массивов отдельных разрядов A и B). В первой ячейке массива будем хранить «длину» числа (количество разрядов). Массив B содержит предыдущее вычисленное число Фибоначчи, а массив A – текущее число. Для удобства сложения полагаем Ф(0) = 0.

Программа на Basic для сложения 2 "длинных чисел":

DECLARE SUB LongAdd **()**

DEFLNG A**-**Z

**DIM** SHARED A**(**10000**),** B**(**10000**)**

'Длинные числа. A(0)-длина числа A, A(i)-i-ый разряд,

A**(**0**)** **=** 1**:** A**(**1**)** **=** 1 'A=Ф(1)=1

B**(**0**)** **=** 1**:** B**(**1**)** **=** 0 'B=Ф(0)=0

L **=** 1 'L - общая длина строки, состоящей из чисел Фибоначчи

**INPUT** "N="**,** N

**WHILE** N **>** L 'Пока N-номер искомой цифры больше длины строки...

LongAdd 'найти очередное число Фибоначчи

L **=** L **+** A**(**0**)** 'и прибавить его длину к общей длине строки

**WEND**

**PRINT** A**(**L **-** N **+** 1**)** 'Как только N-ая цифра получена в явном виде, печатаем ее

'Процедура сложения двух "длинных" чисел A и B "в столбик"

'Результат сложения помещается в A, прежнее значение А переносится в B

SUB LongAdd

r **=** 0

**FOR** i **=** 1 **TO** A**(**0**)**

k **=** A**(**i**)** 'Запоминается i-ая цифра числа A

r **=** A**(**i**)** **+** B**(**i**)** **+** r 'Сумма i-ых разрядов чисел A и B с учетом переноса

A**(**i**)** **=** r **MOD** 10

r **=** r **\** 10 'r - перенос в следующий разряд

B**(**i**)** **=** k 'Переносим запомненную i-ую цифру числа A в число B

**NEXT**

**IF** r **<>** 0 **THEN** 'Если в результате сложения старших разрядов оказался

A**(**0**)** **=** A**(**0**)** **+** 1 'ненулевой перенос, то "удлиняем" результат на 1 цифру

A**(**A**(**0**))** **=** r

**END** **IF**

**END** SUB

**Реализация знаковой длинной целой арифметики для Delphi**: **cложение, вычитание, умножение и деление**.

*{ Самая простая знаковая длинная целая арифметика }*

*{ Реализация вычислений "в столбик": сложения, вычитания, умножения, деления, остатка от деления }*

**Const** MaxLen **=** 3000**;** *{ Максимальная длина длинного числа }*

**var** Base **:** **Integer** **=** 10**;** *{ Основание системы счисления (можно установить в начале программы) }*

**Type** Long **=** **array** **[-**1**..**MaxLen**]** **of** **Integer;**

*{ В -1-ой позиции находится знак, -1 - отрицательное число, +1 - положительное, 0 - ноль }*

*{ Получить цифру в виде символа по её значению }*

*{ 0 -> '0', 1 -> '1', 2 -> '2', ... 9 -> '9', 'A' -> 10, ... 'F' -> 15 }*

**function** Digit**(** Dig**:Integer** **):Char;**

**begin**

**case** Dig **of**

0**..**9**:** Result **:=** Chr**(**Ord**(**'0'**)+**Dig**);**

10**..**35**:** Result **:=** Chr**(**Ord**(**'A'**)+**Dig-10**);**

**else**

raise EOverflow**.**Create**(**'Слишком большая цифра. Невозможно вывести!'**);**

**end;**

**end;**

*{ Получаем из "короткого" числа int64 длинное (используется для инициализации длинных чисел) }*

**function** ToLong**(** N**:Int64** **):**Long**;**

**var** i **:** **integer;**

**begin**

fillChar**(**Result**,**sizeOf**(**Result**),**0**);**

*{ Знак }*

**if** N**=**0 **then** Result**[-**1**]:=**0 *{ Число равно 0 }*

**else** **if** N**>**0 **then** Result**[-**1**]:=**1 *{ Число положительное }*

**else** **begin** Result**[-**1**]** **:=** **-**1**;** N **:=** **-**N **;** **end;** *{ Число отрицательное, дальше будем }*

i **:=** 0**;**

**while** N **>** 0 **do** **begin**

Result**[**i**]** **:=** N **mod** Base**;**

N **:=** N **div** Base**;**

inc**(**i**);**

**end;**

**end;**

*{ Получаем из "длинного" короткое если это возможно }*

**function** ToInt**(** L**:**Long **):Int64;**

**var**

i **:** **Integer;**

X **:** **Int64;**

**begin**

X **:=** 1**;** *{ Base в нужной степени }*

Result **:=** 0**;**

**for** i**:=**0 **to** MaxLen **do** **begin**

assert**(** L**[**i**]** **>=** 0 **);**

Inc**(** Result**,** X**\***L**[**i**]** **);**

X **:=** X **\*** Base**;** *{ Получаем Base в следующей степени }*

**end;**

Result **:=** Result **\*** L**[-**1**];** *{ Умножаем на знак }*

**end;**

*{ Перевод "длинного числа" в строку (для вывода) }*

**function** ToString**(** **Var** L**:**Long **):String;**

**var** i **:** **Integer;**

**begin**

*{ Сначала учитываем знак }*

**Case** L**[-**1**]** **of**

**-**1**:** Result **:=** '-'**;**

0**,**1**:** Result **:=** ''**;**

**end;**

*{ Выводим число по цифрам }*

**for** i**:=**Len**(**L**)** **downto** 0 **do**

Result **:=** Result **+** Digit**(**L**[**i**]);**

**end;**

*{ Приведение нуля в нормальный вид }*

**procedure** FixZero**(** **Var** L**:**Long **);**

**var**

i **:** **Integer;**

**begin**

**for** i**:=**0 **to** MaxLen **do**

**if** L**[**i**]<>**0 **then**

exit**;**

L**[-**1**]:=**0**;**

**end;**

*{ Учёт переноса в старшие разряды }*

**procedure** FixUp**(** **Var** L**:**Long **);**

**var** i **:** **Integer;**

**begin**

**for** i**:=**0 **to** MaxLen-1 **do** **begin**

inc**(** L**[**i+1**],** L**[**i**]** **div** Base **);**

L**[**i**]** **:=** L**[**i**]** **mod** Base**;**

**end;**

FixZero**(**L**);**

**end;**

*{ Учёт заёма из старших разрядов }*

**procedure** FixDown**(** **Var** L**:**Long **);**

**var** i **:** **Integer;**

**begin**

**for** i**:=**0 **to** MaxLen-1 **do**

**while** L**[**i**]** **<** 0 **do** **begin**

inc**(** L**[**i**],** Base **);**

dec**(** L**[**i+1**],** 1 **);**

**end;**

FixZero**(**L**);**

**end;**

*{ Больше или равно по абсолютному значению с учётом сдвига B на Sdvig\_B цифр влево }*

**function** isGreatEq\_Abs**(** A**,**B**:**Long**;** Sdvig\_B**:Integer=**0 **):boolean;**

**var** i **:** **Integer;**

**begin**

**for** i**:=**MaxLen**-**Sdvig\_B **downto** 0 **do** **begin**

**if** A**[**i**+**Sdvig\_B**]>**B**[**i**]** **then** **begin** Result **:=** **true;** exit**;** **end;**

**if** A**[**i**+**Sdvig\_B**]<** B**[**i**]** **then** **begin** Result **:=** **false;** exit**;** **end;**

**end;**

Result **:=** **true;** *{ A и B равны }*

**end;**

*{ Больше или равно с учётом знака }*

**function** isGreatEq**(** A**,**B**:**Long **):boolean;** *{ Больше ли A B }*

**begin**

**case** A**[-**1**]** **of**

**-**1**:** **case** B**[-**1**]** **of**

**-**1**:** Result **:=** isGreatEq\_Abs**(**B**,**A**);** *{ A отрицательно B отрицательно }*

0**:** Result **:=** **false;** *{ A отрицательно B ноль }*

**+**1**:** Result **:=** **false;** *{ A отрицательно B положительно }*

**end;**

0**:** **case** B**[-**1**]** **of**

**-**1**:** Result **:=** **true;** *{ A ноль B отрицательно }*

0**:** Result **:=** **true;** *{ A ноль B ноль }*

**+**1**:** Result **:=** isGreatEq\_Abs**(**A**,**B**);** *{ A ноль B положительно }*

**end;**

**+**1**:** **case** B**[-**1**]** **of**

**-**1**:** Result **:=** **true;** *{ A положительно B отрицательно }*

0**:** Result **:=** **true;** *{ A положительно B ноль }*

**+**1**:** Result **:=** isGreatEq\_Abs**(**A**,**B**);** *{ A положительно B положительно }*

**end;**

**end;**

**end;**

*{ Сложение по абсолютному значению }*

**function** Add\_Abs**(** A**,**B**:**Long **):**Long**;**

**var** i **:** **Integer;**

**begin**

**for** i**:=**0 **to** MaxLen **do** Result**[**i**]** **:=** A**[**i**]** **+** B**[**i**];**

FixUp**(**Result**);**

**end;**

*{ Вычитание по абсолютному значению с возможным сдвигом (сдвиг нужен при делении) }*

**function** Sub\_Abs**(** A**,**B**:**Long**;** Sdvig\_B**:Integer=**0 **):**Long**;**

**var** i **:** **Integer;**

**begin**

assert**(** isGreatEq\_Abs**(**A**,**B**)** **);**

**for** i**:=**0 **to** Sdvig\_B-1 **do** Result**[**i**]** **:=** A**[**i**];**

**for** i**:=**Sdvig\_B **to** MaxLen **do** Result**[**i**]** **:=** A**[**i**]** **-** B**[**i**-**Sdvig\_B**];**

FixDown**(**Result**);**

**end;**

*{ Сложение "длинных" с учётом знака }*

**function** Add**(** A**,**B**:**Long **):**Long**;**

**begin**

*{ Рассмотрим 2 случая: }*

**if** A**[-**1**]** **=** B**[-**1**]** **then** **begin** *{ A и B одного знака }*

Result **:=** Add\_Abs**(**A**,**B**);**

Result**[-**1**]** **:=** A**[-**1**];** *{ Тогда знак их суммы такой же как у A и B }*

**end** **else** **begin** *{ A и B с разными знаками }*

*{ Тогда знак суммы равен знаку наибольшего из них по абсолютному значению,*

*а значение - разности абсолютных значений }*

**if** isGreatEq\_Abs**(**A**,**B**)** **then** **begin**

REsult **:=** Sub\_Abs**(**A**,**B**);** *{ Если A больше по абсолютному значению - вычитаем из A B }*

Result**[-**1**]** **:=** A**[-**1**];**

**end** **else** **begin** *{ иначе из B A }*

Result **:=** Sub\_Abs**(**B**,**A**);**

Result**[-**1**]** **:=** B**[-**1**];**

**end;**

**end;**

**end;**

*{ Вычитание "длинных" с учётом знака }*

**function** Sub**(** A**,**B**:**Long **):**Long**;**

**begin**

B**[-**1**]** **:=** **-**B**[-**1**];** *{ Изменяем знак и складываем }*

Result **:=** Add**(**A**,**B**);**

**end;**

*{ Перемножение длинных }*

**function** Mul**(** A**,**B**:**Long **):**Long**;**

**var** i**,**j **:** **Integer;**

**begin**

fillChar**(**Result**,**sizeOf**(**Result**),**0**);**

**for** i**:=**0 **to** MaxLen **do**

**for** j**:=**0 **to** MaxLen**-**i **do**

Inc**(** Result**[**i**+**j**],** A**[**i**]\***B**[**j**]** **);**

Result**[-**1**]** **:=** A**[-**1**]** **\*** B**[-**1**];** *{ Знаки перемножаются }*

FixUp**(**Result**);**

**end;**

*{ Возведение в квадрат }*

**function** Sqr**(** A**:**Long **):**Long**;**

**begin**

Result **:=** Mul**(**A**,**A**);**

**end;**

*{ Длинное деление }*

**function** LDiv**(** A**,**B**:**Long **):**Long**;**

**var** Sdvig\_B **:** **Integer;**

**begin**

fillChar**(**Result**,**sizeOf**(**Result**),**0**);**

Result**[-**1**]** **:=** A**[-**1**]** **\*** B**[-**1**];** *{ Знаки перемножаются }*

**for** Sdvig\_B**:=**Len**(**A**)-**Len**(**B**)** **downto** 0 **do**

**while** isGreatEq\_Abs**(**A**,**B**,**Sdvig\_B**)** **do** **begin**

A **:=** Sub\_Abs**(**A**,**B**,**Sdvig\_B**);**

Inc**(** Result**[**Sdvig\_B**]** **);**

**end;**

FixUp**(**Result**);**

**end;**

*{ Остаток при целочисленном делении }*

**function** LMod**(** A**,**B**:**Long **):**Long**;**

**var** Sdvig\_B **:** **Integer;**

**begin**

**for** Sdvig\_B**:=**Len**(**A**)-**Len**(**B**)** **downto** 0 **do**

**while** isGreatEq\_Abs**(**A**,**B**,**Sdvig\_B**)** **do**

A **:=** Sub\_Abs**(**A**,**B**,**Sdvig\_B**);**

Result **:=** A**;**

Result**[-**1**]** **:=** 1**;**

**end;**

# Структуры данных: стеки и очереди

## Записи и оператор with

Записи используются для создания своих типов данных

*{ Обьявление записи - тип "Точка" }*

**type**

TPoint **=** **Record**

x**,**y **:** **double;**

**end;**

*{ Функция, вычисляющая расстояние между точками }*

**function** dist**(** A**,**B **:** TPoint **):double;**

**begin**

dist **:=** sqrt**(** **(**A**.**x **-** B**.**x**)\*(**A**.**x **-** B**.**x**)** **+** **(**A**.**y **-** B**.**y**)\*(**A**.**y **-** B**.**y**)** **);**

**end;**

*{ Использование записей }*

**var** A**,**B **:** TPoint**;**

**begin**

*{ Инициализируем координаты точек }*

A**.**x **:=** 1**;** A**.**y **:=** 2**;**

B**.**x **:=** 10**;** B**.**y **:=** 11**;**

writeln**(** dist**(**A**,**B**)** **);**

**end.**

Использование with:

*{ Обьявление записи - тип "Персонаж в игре" }*

**type**

TUnit **=** **Record**

x**,**y **:** **integer;** *{ Координаты клетки где стоит персонаж }*

name **:** **string;** *{ Имя персонажа }*

**end;**

**var** Unit1 **:** TUnit**;**

*{ Инициализация без with }*

Unit1**.**x **:=** 2**;**

Unit1**.**y **:=** 3**;**

Unit1**.**name **:=** 'SUPER-HERO'**;**

*{ Инициализация с with }*

**with** Unit1 **do** **begin**

x **:=** 2**;**

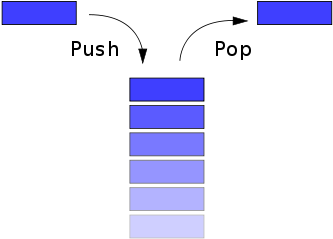
y **:=** 3**;**

name **:=** 'SUPER-HERO'**;**

**end;**

## Реализация Стека и Очереди на базе массива

**Стек** (англ. **stack** — стопка) — структура данных с методом доступа к элементам **LIFO** (англ. Last In — First Out, «последним пришёл — первым вышел»). Чаще всего принцип работы стека сравнивают со стопкой тарелок: чтобы взять вторую сверху, нужно снять верхнюю.



**const** StackSize **=** 10000**;** *{ Размер стека (сколько в него можно положить элементов) }*

*{ === Хранение стека === }*

**var**

Stack **:** **array** **[**1**..**StackSize**]** **of** **Integer;** *{ Массив для хранения стека }*

StackTop **:** **Integer** **=** 0**;** *{ Вершина стека - индекс в массиве Stack }*

*{ === Операции со стеком === }*

*{ Стек пуст? }*

**function** isEmpty **:** **Boolean;**

**begin**

isEmpty **:=** StackTop **=** 0**;**

**end;**

*{ Положить значение на вершину стека }*

**procedure** Push**(** Value **:** **Integer** **);**

**begin**

assert**(** StackTop **<** StackSize**,** 'Стек полон! Больше положить в него нельзя!'**);**

Inc**(**StackTop**);**

Stack**[**StackTop**]** **:=** Value**;**

**end;**

*{ Забрать значение с вершины стека }*

**function** Pop **:** **Integer;**

**begin**

assert**(** **not** isEmpty**,** 'Нельзя извлечь элемент, потому что стек пуст!'**);**

Pop **:=** Stack**[**StackTop**];**

Dec**(** StackTop **);**

**end;**

*{ === Тестирование работы стека === }*

**begin**

Writeln**(**isEmpty**);** *{ Выводит "TRUE" - стек пуст }*

Push**(**2**);** *{ В стеке: 2 }*

Writeln**(**isEmpty**);** *{ Выводит "FALSE" - стек не пуст }*

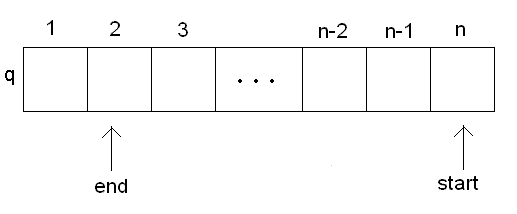
Push**(**5**);** *{ В стеке: 2, 5 }*

Writeln**(**Pop**);** *{ Выводит "5", в стеке: 2 }*

Writeln**(**Pop**);** *{ Выводит "2", в стеке пусто }*

**end.**

**Очередь** — структура данных с дисциплиной доступа к элементам «первый пришёл — первый вышел» (**FIFO**, First In — First Out). Добавление элемента (принято обозначать словом enqueue — поставить в очередь) возможно лишь в конец очереди, выборка — только из начала очереди (что принято называть словом dequeue — убрать из очереди), при этом выбранный элемент из очереди удаляется.



**const** QSize **=** 10000**;** *{ Размер очереди (сколько в неё можно положить элементов) }*

**var**

Q **:** **array** **[**1**..**QSize**]** **of** **Integer;** *{ Массив для хранения очереди }*

Q\_Start **:** **Integer** **=** 1**;** *{ Указывает на голову очереди }*

Q\_End **:** **Integer** **=** 1**;** *{ Указывает на элемент, который заполнится, когда в очередь войдёт новый элемент }*

*{ = Операции с очередью = }*

*{ Очередь пуста? }*

**function** isEmpty **:** **Boolean;**

**begin**

isEmpty **:=** Q\_Start **=** Q\_End**;**

**end;**

*{ Положить значение в конец очереди }*

**procedure** **Put(** Value **:** **Integer** **);**

**begin**

Q**[**Q\_End**]** **:=** Value**;**

Dec**(**Q\_End**);**

*{ Поддержка закольцованности очереди }*

**if** Q\_End **<** 1 **then** Q\_End **:=** QSize**;**

**end;**

*{ Забрать значение с начала очереди }*

**function** **Get** **:** **Integer;**

**begin**

assert**(** **not** isEmpty**,** 'В очереди ничего нет!'**);**

**Get** **:=** Q**[**Q\_Start**];**

Dec**(**Q\_Start**);**

*{ Поддержка закольцованности очереди }*

**if** Q\_Start **<** 1 **then** Q\_Start **:=** QSize**;**

**end;**

**begin**

Writeln**(**isEmpty**);** *{ Выводит "TRUE" - очередь пуста }*

**Put(**2**);** *{ В очереди: 2 }*

Writeln**(**isEmpty**);** *{ Выводит "FALSE" - очередь не пуста }*

**Put(**5**);** *{ В очереди: 5, 2 }*

Writeln**(Get);** *{ Выводит "2", в очереди: 5 }*

Writeln**(Get);** *{ Выводит "5", в очереди пусто }*

**end.**

# Реализация на C/C++

Реализация Стека

#include <iostream>

#include <assert.h>

**using** **namespace** std**;**

*// Определение класса Стек*

*// Шаблон с двумя параметрами:*

*// T - тип элемента (например: int, char, char\*)*

*// size - размер стека (целое число)*

**template** **<typename** T**,int** size**>**

**class** Stack**{**

T data**[**size**];** *// Данные стека*

**int** count**;** *// Количество элементов в стеке*

*// count указывает на ячейку после последнего элемента в стеке*

*// Например, если count = 1, то последний элемент это data[0]*

**public:**

*// Конструктор*

Stack**(){** count **=** 0**;** **};**

*// Стек пуст?*

**bool** isEmpty**(){**

**return** count **<=** 0**;**

**};**

*// Стек полон?*

**bool** isFull**(){**

**return** count **>=** size**;**

**};**

*// Добавить на вершину стека*

**void** push**(**T value**){**

assert**(!**isFull**());** *// Можно добавить только если стек ещё не полон*

data**[**count**++]** **=** value**;** *// Записываем значение в массив и сдвигаем счётчик вправо*

**}**

*// Снять с вершины стека*

T pop**(){**

assert**(!**isEmpty**());** *// Можно снять только если стек не пуст*

**return** data**[--**count**];** *// Уменьшаем счётчик (сдвигаем влево) и возвращаем значение*

**}**

**};**

*// Основная программа - тестирование стека*

**int** main**()** **{**

Stack<int,5> s**;** *// Создаём пример стека*

assert**(**s**.**isEmpty**());** *// Сейчас стек должен быть пустым*

s**.**push**(**5**);** *// Добавляем один элемент*

assert**(!**s**.**isEmpty**());**

assert**(**s**.**pop**()** **==** 5**);**

assert**(**s**.**isEmpty**());**

assert**(!**s**.**isFull**());**

s**.**push**(**1**);** s**.**push**(**2**);** s**.**push**(**3**);**

s**.**push**(**4**);** assert**(!**s**.**isFull**());** *// Стек ещё не полон после добавления четвёртого элемента*

s**.**push**(**5**);** assert**(**s**.**isFull**());**

Stack<char,2> cs**;** *// Заводим другой пример стека - с символами в качестве элементов*

cs**.**push**(**'a'**);**

**return** 0**;**

**}**

Реализация очереди (циклической очереди)

#include <iostream>

#include <assert.h>

**using** **namespace** std**;**

*// Определение класса Очередь*

*// Шаблон с двумя параметрами:*

*// T - тип элемента (например: int, char, char\*)*

*// size - размер очереди (целое число)*

**template** **<typename** T**,int** size**>**

**class** Queue**{**

T data**[**size**];** *// Данные очереди*

**int** head**;** *// Голова очереди (первый элемент)*

**int** tail**;** *// Хвост очереди (последний элемент)*

**public:**

*// Конструктор*

Queue**(){** head **=** **-**1**;** tail **=** 0**;** **};**

*// Очередь пуста?*

**bool** isEmpty**(){**

**return** head **<** tail**;**

**};**

*// Очередь полна?*

**bool** isFull**(){**

**return** len**()** **>=** size**;**

**};**

*// Текущая длина очереди*

**int** len**(){**

**return** head**-**tail+1**;**

**};**

*// Добавить в начало очереди*

**void** put**(**T value**){**

assert**(!**isFull**());** *// Можно добавить только если очередь не полна*

**++**head**;** *// Сдвигаем голову вправо*

data**[**head **%** size**]** **=** value**;** *// Записываем в ячейку с номером по модулю*

*// -----------------------------------------*

*// | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |*

*// -----------------------------------------*

*// Количество ячеек - 10, максимальный размер очереди - 10*

*// Куда должен пойти 10-ый элемент? На 0-ую позицию*

*// Куда должен пойти 20-ый элемент? Тоже на 0-ую позицию*

*// Цикличность очереди обепечивается тем, что мы индекс элемента всегда берём по модулю максимального размера очереди*

**}**

*// Взять из конца очереди*

T get**(){**

assert**(!**isEmpty**());** *// Можно снять только если очередь не полна*

**return** data**[**tail**++** **%** size**];** *// Забираем значение из хвоста и двигаем хвост вправо*

**}**

**};**

*// Основная программа - тестирование стека*

**int** main**()** **{**

Queue<int,5> s**;** *// Создаём пример стека*

assert**(**s**.**isEmpty**());** *// Сейчас очередь должна быть пуста*

s**.**put**(**5**);** *// Добавляем один элемент*

assert**(!**s**.**isEmpty**());**

assert**(**s**.**get**()** **==** 5**);**

assert**(**s**.**isEmpty**());**

assert**(!**s**.**isFull**());**

s**.**put**(**1**);** s**.**put**(**2**);** s**.**put**(**3**);**

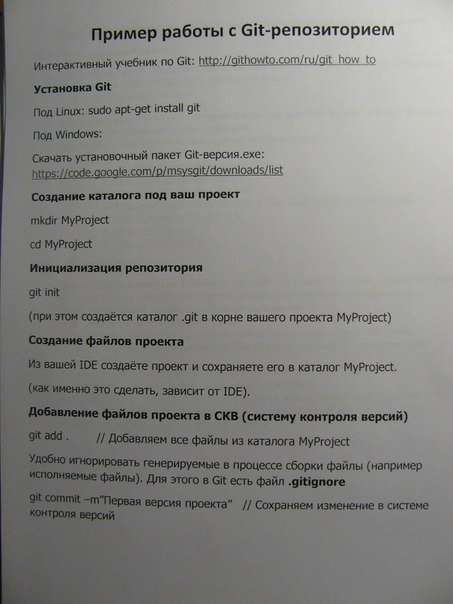
s**.**put**(**4**);** assert**(!**s**.**isFull**());**

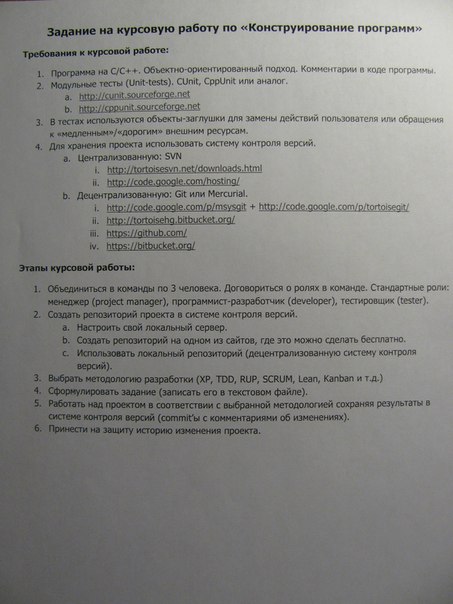
s**.**put**(**5**);** assert**(**s**.**isFull**());**

**return** 0**;**

**}**

**Работа с Git-репозиторием**

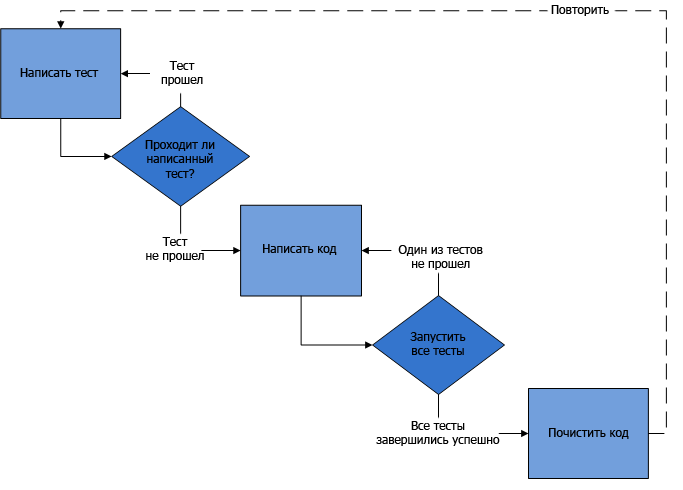




Пример модульных тестов на Java

TDD (Test-driven development), разработка через тестирование - техника разработки программного обеспечения, которая основывается на повторении очень коротких циклов разработки:

* сначала пишется тест, покрывающий желаемое изменение,
* затем пишется код, который позволит пройти тест,
* и под конец проводится рефакторинг нового кода к соответствующим стандартам.



Конечный результат:

*/\*\**

*\* Реализация функций*

*\*/*

**public** **class** A **{**

**public** **static** **int** add**(int** a**,** **int** b**)** **{**

**return** a **+** b**;**

**}**

**public** **static** **int** fact**(int** N**)** **throws** Exception **{**

**if** **(**N **<** 0**)**

**throw** **new** Exception**(**"Факториал от отрицательного числа не существует"**);**

**if** **(**N **==** 0**)**

**return** 1**;**

**if** **(**N **>=** 2**)**

**return** fact**(**N **-** 1**)** **\*** N**;**

**return** N**;**

**}**

**}**

Модульные тесты:

import org**.**junit**.**Assert**;**

import org**.**junit**.**Test**;**

import **static** junit**.**framework**.**Assert**.**assertEquals**;**

*/\*\**

*\* Тестирование класса A*

*\*/*

**public** **class** ATest **{**

@Test

**public** **void** add**()** **{**

assertEquals**(**7**,** A**.**add**(**2**,** 5**));**

assertEquals**(**12**,** A**.**add**(**3**,** 9**));**

**}**

@Test

**public** **void** fact**()** **throws** Exception **{**

assertEquals**(**1**,** A**.**fact**(**1**));**

assertEquals**(**1 **\*** 2**,** A**.**fact**(**2**));**

assertEquals**(**1 **\*** 2 **\*** 3**,** A**.**fact**(**3**));**

assertEquals**(**1 **\*** 2 **\*** 3 **\*** 4**,** A**.**fact**(**4**));**

assertEquals**(**1 **\*** 2 **\*** 3 **\*** 4 **\*** 5**,** A**.**fact**(**5**));**

assertEquals**(**1 **\*** 2 **\*** 3 **\*** 4 **\*** 5 **\*** 6**,** A**.**fact**(**6**));**

assertEquals**(**1 **\*** 2 **\*** 3 **\*** 4 **\*** 5 **\*** 6 **\*** 7**,** A**.**fact**(**7**));**

assertEquals**(**1**,** A**.**fact**(**0**));**

**try** **{**

A**.**fact**(-**1**);**

Assert**.**fail**(**"Не должно работать!"**);**

**}** **catch** **(**Exception ex**)** **{**

assertEquals**(**"Факториал от отрицательного числа не существует"**,**

ex**.**getMessage**());**

**}**

**}**

**}**

Конечно же, TDD следует рассматривать как разворачивающийся во времени процесс.

Рассмотрим написание кода по шагам:

Начинаем с теста, создаём тест для тестирования класса A.

import junit**.**framework**.**Assert**;**

import org**.**junit**.**Test**;**

*// Тестирование класса A*

**public** **class** ATest **{**

@Test *// Аннотация, показывающая, что этот метод - тест.*

**public** **void** testAdd**()** **{**

Assert**.**assertEquals**(**7**,** A**.**add**(**2**,** 5**));**

**}**

**}**

Запускаем тест, код даже не компилируется. Значит надо реализовать класс A, чтобы тест выполнялся.

Создаём минимальную реализацию, чтобы код вообще компилировался.

*/\*\**

*\* Реализация функций - JavaDoc комментарий*

*\*/*

**public** **class** A **{**

**public** **static** **int** add**(int** a**,** **int** b**)** **{**

**return** 0**;**

**}**

**}**

Код компилируется, но тест не срабатывает. Ожидается 7, а результат 0.

Исправляем код простейшим из пришедших в голову способов.

**public** **class** A **{**

**public** **static** **int** add**(int** a**,** **int** b**)** **{**

**return** 7**;**

**}**

**}**

Запускаем тест, он проходит (Зелёная полоска). Добавляем новый тест.

import junit**.**framework**.**Assert**;**

import org**.**junit**.**Test**;**

import **static** junit**.**framework**.**Assert**.\*;**

**public** **class** ATest **{**

@Test

**public** **void** testAdd**()** **{**

assertEquals**(**7**,** A**.**add**(**2**,** 5**));** *// Первый тест*

assertEquals**(**12**,** A**.**add**(**3**,** 9**));** *// Второй тест*

**}**

**}**

Теперь исправляем код, и сразу обобщаем его (рефакторинг).

**public** **class** A **{**

**public** **static** **int** add**(int** a**,** **int** b**)** **{**

**return** a **+** b**;**

**}**

**}**

Со сложением вроде бы всё понятно, перейдём к реализации более сложной фукнции - факториал. Факториал - это произведение всех чисел от 1 до N. $N! = 1\*2\*3\*...\*N$.

import junit**.**framework**.**Assert**;**

import org**.**junit**.**Test**;**

import **static** junit**.**framework**.**Assert**.\*;**

**public** **class** ATest **{**

@Test *// Тестируем сложение*

**public** **void** add**()** **{**

assertEquals**(**7**,** A**.**add**(**2**,** 5**));**

assertEquals**(**12**,** A**.**add**(**3**,** 9**));**

**}**

@Test *// Тестируем факториал*

**public** **void** fact**(){**

assertEquals**(**1**,** A**.**fact**(**1**));**

assertEquals**(**2**,** A**.**fact**(**2**));**

**}**

**}**

Первая "наивная" реализация факториала:

**public** **class** A **{**

**public** **static** **int** add**(int** a**,** **int** b**)** **{**

**return** a **+** b**;**

**}**

**public** **static** **int** fact**(int** N**)** **{**

**return** N**;**

**}**

**}**

Добавляем ещё тест:

@Test

**public** **void** fact**()** **{**

assertEquals**(**1**,** A**.**fact**(**1**));**

assertEquals**(**1 **\*** 2**,** A**.**fact**(**2**));**

assertEquals**(**1 **\*** 2 **\*** 3**,** A**.**fact**(**3**));**

**}**

Тест не срабатывает - красная полоска. Добавляем if, обрабатывающий ещё и число 3.

**public** **static** **int** fact**(int** N**)** **{**

**if** **(**N **==** 3**)**

**return** 6**;**

**return** N**;**

**}**

Очередной шаг, добавляем тест. Двигаемся маленькими-маленькими шажками, чтобы отработать технику TDD.

@Test

**public** **void** fact**()** **{**

assertEquals**(**1**,** A**.**fact**(**1**));**

assertEquals**(**1 **\*** 2**,** A**.**fact**(**2**));**

assertEquals**(**1 **\*** 2 **\*** 3**,** A**.**fact**(**3**));**

assertEquals**(**1 **\*** 2 **\*** 3 **\*** 4**,** A**.**fact**(**4**));**

**}**

Тест опять не проходит - Красная полоска. Исправляем код, добавляем ещё один if.

**public** **static** **int** fact**(int** N**)** **{**

**if** **(**N **==** 4**)**

**return** 24**;**

**if** **(**N **==** 3**)**

**return** 6**;**

**return** N**;**

**}**

Добавляем ещё один тест, с числом 5.

@Test

**public** **void** fact**()** **{**

assertEquals**(**1**,** A**.**fact**(**1**));**

assertEquals**(**1 **\*** 2**,** A**.**fact**(**2**));**

assertEquals**(**1 **\*** 2 **\*** 3**,** A**.**fact**(**3**));**

assertEquals**(**1 **\*** 2 **\*** 3 **\*** 4**,** A**.**fact**(**4**));**

assertEquals**(**1 **\*** 2 **\*** 3 **\*** 4 **\*** 5**,** A**.**fact**(**5**));**

**}**

Снова добавляем if.

**public** **static** **int** fact**(int** N**)** **{**

**if** **(**N **==** 5**)**

**return** 1**\***2**\***3**\***4**\***5**;**

**if** **(**N **==** 4**)**

**return** 24**;**

**if** **(**N **==** 3**)**

**return** 6**;**

**return** N**;**

**}**

Смотрим на череду if'ов и думаем как их обобщить (рефакторинг), а то код становится "скучным", однообразным.

Обобщение код - каждый раз творчество, каждый раз озарение, инсайт.

**public** **static** **int** fact**(int** N**)** **{**

**if** **(**N **>=** 3**)**

**return** fact**(**N **-** 1**)** **\*** N**;**

**return** N**;**

**}**

Отлично! Всё работает (полоска зелёная). Добавляем ещё тест.

@Test

**public** **void** fact**()** **{**

assertEquals**(**1**,** A**.**fact**(**1**));**

assertEquals**(**1 **\*** 2**,** A**.**fact**(**2**));**

assertEquals**(**1 **\*** 2 **\*** 3**,** A**.**fact**(**3**));**

assertEquals**(**1 **\*** 2 **\*** 3 **\*** 4**,** A**.**fact**(**4**));**

assertEquals**(**1 **\*** 2 **\*** 3 **\*** 4 **\*** 5**,** A**.**fact**(**5**));**

assertEquals**(**1 **\*** 2 **\*** 3 **\*** 4 **\*** 5 **\*** 6**,** A**.**fact**(**6**));**

**}**

Тест срабатывает сразу, даже без модификации кода. Добавляем ещё тест.

@Test

**public** **void** fact**()** **{**

assertEquals**(**1**,** A**.**fact**(**1**));**

assertEquals**(**1 **\*** 2**,** A**.**fact**(**2**));**

assertEquals**(**1 **\*** 2 **\*** 3**,** A**.**fact**(**3**));**

assertEquals**(**1 **\*** 2 **\*** 3 **\*** 4**,** A**.**fact**(**4**));**

assertEquals**(**1 **\*** 2 **\*** 3 **\*** 4 **\*** 5**,** A**.**fact**(**5**));**

assertEquals**(**1 **\*** 2 **\*** 3 **\*** 4 **\*** 5 **\*** 6**,** A**.**fact**(**6**));**

assertEquals**(**1 **\*** 2 **\*** 3 **\*** 4 **\*** 5 **\*** 6 **\*** 7**,** A**.**fact**(**7**));**

**}**

Тест опять срабатывает сразу, без модификации кода. Это скучно! Скорее всего, тест будет правильно работать и на последующих тестах. Надо придумать необычный тест, который не укладывается в наши условия. Добавим факториал от 0.

@Test

**public** **void** fact**()** **{**

assertEquals**(**1**,** A**.**fact**(**1**));**

assertEquals**(**1 **\*** 2**,** A**.**fact**(**2**));**

assertEquals**(**1 **\*** 2 **\*** 3**,** A**.**fact**(**3**));**

assertEquals**(**1 **\*** 2 **\*** 3 **\*** 4**,** A**.**fact**(**4**));**

assertEquals**(**1 **\*** 2 **\*** 3 **\*** 4 **\*** 5**,** A**.**fact**(**5**));**

assertEquals**(**1 **\*** 2 **\*** 3 **\*** 4 **\*** 5 **\*** 6**,** A**.**fact**(**6**));**

assertEquals**(**1 **\*** 2 **\*** 3 **\*** 4 **\*** 5 **\*** 6 **\*** 7**,** A**.**fact**(**7**));**

assertEquals**(**1**,** A**.**fact**(**0**));**

**}**

И сразу же реализуем его отдельным if'ом.

**public** **static** **int** fact**(int** N**)** **{**

**if** **(**N **==** 0**)**

**return** 1**;**

**if** **(**N **>=** 2**)**

**return** fact**(**N **-** 1**)** **\*** N**;**

**return** N**;**

**}**

Подумаем над обработкой ошибок, если в фукцию вычисления факториала передадут отрицательное число, надо выкинуть исключение. Пишем соответствующий тест.

@Test

**public** **void** fact**()** **throws** Exception **{**

assertEquals**(**1**,** A**.**fact**(**1**));**

assertEquals**(**1 **\*** 2**,** A**.**fact**(**2**));**

assertEquals**(**1 **\*** 2 **\*** 3**,** A**.**fact**(**3**));**

assertEquals**(**1 **\*** 2 **\*** 3 **\*** 4**,** A**.**fact**(**4**));**

assertEquals**(**1 **\*** 2 **\*** 3 **\*** 4 **\*** 5**,** A**.**fact**(**5**));**

assertEquals**(**1 **\*** 2 **\*** 3 **\*** 4 **\*** 5 **\*** 6**,** A**.**fact**(**6**));**

assertEquals**(**1 **\*** 2 **\*** 3 **\*** 4 **\*** 5 **\*** 6 **\*** 7**,** A**.**fact**(**7**));**

assertEquals**(**1**,** A**.**fact**(**0**));**

**try** **{**

A**.**fact**(-**1**);**

Assert**.**fail**(**"Не должно работать!"**);**

**}** **catch** **(**Exception ex**)** **{**

assertEquals**(**"Факториал от отрицательного числа не существует!"**,**

ex**.**getMessage**());**

**}**

**}**

Реализуем последний случай:

**public** **static** **int** fact**(int** N**)** **throws** Exception **{**

**if** **(**N **<** 0**)**

**throw** **new** Exception**(**"Факториал от отрицательного числа не существует!"**);**

**if** **(**N **==** 0**)**

**return** 1**;**

**if** **(**N **>=** 2**)**

**return** fact**(**N **-** 1**)** **\*** N**;**

**return** N**;**

**}**

**Вопросы по курсу "Конструирование программ"**

1. Жизненный цикл ПО: сбор требований, разработка, реализация, эксплуатация, поддержка, утилизация.
2. Методологии (виды) разработки программного обеспечения (гибкие - agile, жёсткие).
3. Модель разработки как последовательного перевода и уточнения требований.
4. Водопадная (каскадная) модель разработки ПО.
5. XP – Экстремальное программирование: принципы, особенности.
6. Короткий цикл обратной связи: разработка через тестирование, игра в планирование, заказчик всегда рядом, парное программирование.
7. Непрерывность процесса разработки: рефакторинг, непрерывная интеграция (Continuous Integration), частые небольшие релизы.
8. Понимание, разделяемое всеми: простота архитектуры, метафора работы системы, коллективное владение кодом, стандарты кодирования.
9. RUP - Rational Unified Process: принципы, жизненный цикл ПО (как он описан в RUP).
10. Техническое задание на программный продукт (структура, оценки, обязательства). Формализация требований, исполняемые спецификации.
11. Проектирование пользовательского интерфейса с учётом требований эргономичности (компоненты, события, сигналы, слоты в Qt).
12. Событийно-ориентированная архитектура.
13. Сервис-ориентированная архитектура (сервер, клиент, сервис).
14. Прототипирование как средство для получения обратной связи и уточнения требований.
15. Языки высокого и низкого уровней (признаки, применение).
16. История развития языков программирования: ассемблер, FORTRAN, структурное программирование, ООП, дальнейшее развитие.
17. Трансляция, интерпретация и компиляция.
18. Перечислите основные концепции языков программирования.
19. Императивные и декларативные языки программирования.
20. Машинный код и язык ассемблера (достоинства и недостатки).
21. Структурное программирование: FORTRAN, Algol.
22. Объектно-ориентированное программирование (ООП): абстракция, инкапсуляция, наследование и полиморфизм.
23. ООП: конструкторы и деструкторы.
24. ООП: перегрузка операторов в C++.
25. Отличие C и C++ (ссылки, в приведении типов, перегрузка функций).
26. Функциональное программирование (LISP). Логическое программирование (Prolog).
27. Аспектно-ориентированное программирование (АОП): аспект, совет, точка соединения, срез, внедрение.
28. Предметно-ориентированное программирование (предметно-ориентированный язык). DSL - domain-specific language. Пример разработки языка программирования для конкретной задачи.
29. Метапрограммирование (шаблоны, генераторы кода, интроспекция - Reflection API, интерпретация строк).
30. Виды отладки: отладчик, трассировка (логгирование), модульные тесты, визуализаторы процесса выполнения.
31. Виды тестов: модульные, приёмочные (общее и отличия). Ручное тестирование. Модульные тесты в С/C++: CUnit, CppUnit.
32. Test Driven Development - разработка через тестирование ("красная" и "зелёная" полоса, цикл: тест - разработка - рефакторинг).
33. Объекты заглушки (MockObjects) и их использование (цели, применимость).
34. Работа в команде: распределение ролей, ответственность. Инструменты (системы планирования, системы контроля версий и т.д.).
35. Системы контроля версий: централизованные, распределённые.
36. Алгоритмы. Классификация.
37. Сложность алгоритмов. Оценка по времени выполнения и памяти.
38. Алгоритмы сортировки и поиска.
39. Алгоритмы поиска в графах.
40. Алгоритмы целочисленной арифметики. НОД. Простые числа.
41. Стиль оформления исходных тестов программ. Отступы, "лесенка", пробелы. Преимущества и недостатки использования утилит для автоматического форматирования исходного текста программы.
42. Системы автоматической генерации программной документации по комментариям в коде (JavaDoc, DoxyGen и др.).