

# Система компьютерной алгебры GAP 4.4

(Brief GAP Guidebook in Russian)

Редакция 3.0.2

7 апреля 2009

**Александр Коновалов**

## Реферат

Данный документ содержит начальные сведения о системе компьютерной алгебры GAP 4.4 (<http://www.gap-system.org/>). Он возник на базе специального курса, читавшегося автором в Запорожском государственном университете в 1999-2005 гг.

## Copyright

© Александр Борисович Коновалов, 1999-2009

## Выходные данные

Первая редакция данного документа была посвящена версии GAP 3.4.4: (*Коновалов А.Б. Система компьютерной алгебры GAP. Методические указания. Запорожье: Запорожский государственный университет, 1999. - 42 с. Рекомендовано к печати научно-методическим советом ЗГУ, протокол N.8 от 24.06.1999 г.*). Вторая редакция, переработанная и дополненная с учетом версии GAP 4.3, была разработана в 2002-2003 гг. В феврале-мае 2005 г. были переработаны и расширены задания для лабораторных работ. В декабре 2005 г. была выполнена проверка на соответствие версии GAP 4.4.6. В третьей редакции, выпущенной в мае 2008 г., исходный текст переведен в формат GAPDoc, позволяющий создавать документацию в форматах PDF и HTML из единого исходного кода с помощью одноименного пакета для системы GAP а также автоматически проверять содержащиеся в документации примеры, которые теперь соответствуют версии GAP 4.4.10. HTML-версия данного документа находится по адресу <http://www.gap-system.org/ukrgap/gapbook/chap0.html>, PDF-версия - по адресу <http://www.gap-system.org/ukrgap/gapbook/manual.pdf>.

# Содержание

<b>1</b>	<b>Введение</b>	<b>5</b>
1.1	Краткая характеристика и история системы GAP	5
1.2	Обзор возможностей GAP	6
1.3	Инсталляция и запуск системы	8
1.4	Первые шаги в GAP	9
<b>2</b>	<b>Язык программирования GAP</b>	<b>13</b>
2.1	Символы и категории слов в GAP	13
2.2	Ключевые слова	14
2.3	Идентификаторы	14
2.4	Выражения	14
2.5	Обращения к функциям	15
2.6	Сравнение выражений	16
2.7	Арифметические операторы	16
2.8	Присваивания	17
2.9	Вызов процедуры	17
2.10	Команда IF	18
2.11	Цикл WHILE	19
2.12	Цикл REPEAT	19
2.13	Цикл FOR	19
2.14	Функции	23
2.15	Команда RETURN	24
<b>3</b>	<b>Структуры данных в GAP</b>	<b>25</b>
3.1	Константы и операторы	25
3.2	Переменные и присваивания	26
3.3	Функции	26
3.4	Списки	27
3.5	Тождественность и равенство списков	30
3.6	Множества	31
3.7	Векторы и матрицы	32
3.8	Записи	34
3.9	Арифметические прогрессии	35
3.10	Использование циклов	35
3.11	Дальнейшие операции со списками	37
3.12	Функции	38

<b>4</b>	<b>Операции над группами и их элементами</b>	<b>41</b>
4.1	Задание группы подстановок . . . . .	41
4.2	Задание подгруппы группы подстановок . . . . .	41
4.3	Простейшие свойства группы. Силовские подгруппы . . . . .	42
4.4	Другие виды подгрупп . . . . .	45
4.5	Факторгруппы . . . . .	46
4.6	Классы сопряженных элементов . . . . .	47
<b>A</b>	<b>Рекомендации по созданию и запуску программ в системе GAP</b>	<b>49</b>
<b>B</b>	<b>Некоторые функции системы GAP для работы с группами</b>	<b>52</b>
<b>C</b>	<b>Лабораторные работы</b>	<b>54</b>
C.1	Основы работы с системой GAP в MS Windows . . . . .	54
C.1.1	Инструкции по выполнению работы . . . . .	54
C.2	Списки. Целые числа . . . . .	58
C.2.1	Инструкции по выполнению работы . . . . .	58
C.2.2	Задания для лабораторных работ . . . . .	59
C.3	Линейные программы. Векторы и матрицы . . . . .	60
C.3.1	Инструкции по выполнению работы . . . . .	60
C.3.2	Задания для лабораторных работ . . . . .	62
C.4	Ветвящиеся программы. Многочлены . . . . .	63
C.4.1	Инструкции по выполнению работы . . . . .	63
C.4.2	Задания для лабораторных работ . . . . .	65
C.5	Циклические программы (цикл FOR). Бинарные отношения . . . . .	66
C.5.1	Инструкции по выполнению работы . . . . .	66
C.5.2	Задания для лабораторных работ . . . . .	68
C.6	Циклические программы (цикл WHILE). Подстановки . . . . .	69
C.6.1	Инструкции по выполнению работы . . . . .	69
C.6.2	Задания для лабораторных работ . . . . .	71
C.7	Циклические программы (цикл REPEAT). Группы подстановок . . . . .	73
C.7.1	Инструкции по выполнению работы . . . . .	73
C.7.2	Задания для лабораторных работ . . . . .	75
C.8	Изучение свойств элементов группы . . . . .	77
C.8.1	Инструкции по выполнению работы . . . . .	77
C.8.2	Задания для лабораторных работ . . . . .	78
C.9	Изучение свойств подгрупп группы . . . . .	79
C.9.1	Инструкции по выполнению работы . . . . .	79
C.9.2	Задания для лабораторных работ . . . . .	80
C.10	Работа с библиотекой конечных групп . . . . .	82
C.10.1	Инструкции по выполнению работы . . . . .	82
C.10.2	Задания для лабораторных работ . . . . .	82
C.11	Дополнительные упражнения различной трудности . . . . .	83

# Глава 1

## Введение

### 1.1 Краткая характеристика и история системы GAP

Разработка системы компьютерной алгебры GAP (<http://www.gap-system.org>), название которой расшифровывается как “*Groups, Algorithms and Programming*”, была начата в 1986 г. в г.Аахен, Германия (<http://www.math.rwth-aachen.de/LDFM/>). В 1997 г. центр координации разработки и технической поддержки пользователей переместился в Университет г.Сент-Эндрюс, Шотландия (<http://www-circa.mcs.st-and.ac.uk/>). В настоящее время GAP является уникальным всемирным совместным научным проектом, объединяющим специалистов в области алгебры, теории чисел, математической логики, информатики и др. наук из различных стран мира. Основные центры разработки системы находятся в университетах г.Сент-Эндрюс (Шотландия), гг. Аахен, Брауншвейг (Германия) и Университете штата Колорадо (США). Текущая версия системы - GAP 4.4.10 - была выпущена в октябре 2007 г.

Изначально система GAP разрабатывалась под Unix, а затем была портирована для работы в других операционных системах. В настоящее время она работает в разнообразных версиях Unix/Linux, а также в Windows и Mac OS. Заметим, что ряд пакетов, расширяющих функциональность системы, работает только в среде Unix/Linux.

GAP является свободно распространяемой, открытой и расширяемой системой. Она распространяется в соответствии с GNU Public License (см. <http://www.gap-system.org/Download/copyright.html>). Система поставляется вместе с исходными текстами, которые написаны на двух языках: ядро системы написано на Си, а библиотека функций - на специальном языке, также называемом GAP, который по синтаксису напоминает Pascal, однако является объектно-ориентированным языком. Пользователи могут создавать свои собственные программы на этом языке, и здесь исходные тексты являются незаменимым наглядным пособием. Наконец, разработчики программ для GAP могут оформить свои разработки в виде пакета для системы GAP и представить их на рассмотрение в Совет GAP. После прохождения процедуры рецензирования и одобрения советом GAP такой пакет включается в приложение к дистрибутиву GAP и распространяется вместе с ним. Процедура рецензирования позволяет приравнивать принятые Советом GAP пакеты к научной публикации, и ссылаться на них наравне с другими источниками.

Помимо уже упомянутых пакетов, система состоит из следующих четырех основных компонент:

- ядра системы, обеспечивающего поддержку языка GAP, работу с системой в программном и интерактивном режиме;
- библиотеки функций, в которой реализованы разнообразные алгебраические алгоритмы (более 4000 пользовательских функций, более 140000 строк программ на языке GAP);
- библиотеки данных, включая, например, библиотеку всех групп порядка не более 2000 (за исключением 49487365422 групп порядка 1024, точное количество которых, кстати, также было определено с помощью системы GAP), библиотеку примитивных групп подстановок, таблицы характеров конечных групп и т.д., что в совокупности составляет эффективное средство для выдвижения и тестирования научных гипотез;
- обширной (около полутора тысяч страниц) документации, доступной в разнообразных форматах (`txt`, `pdf`, `html`), а также через Интернет.

## 1.2 Обзор возможностей GAP

Система GAP была задумана как инструмент комбинаторной теории групп - раздела алгебры, изучающего группы, заданные порождающими элементами и определяющими соотношениями. В дальнейшем, с выходом каждой новой версии системы сфера ее применения охватывала все новые и новые разделы алгебры. В разнообразии областей алгебры, охватываемых GAP сегодня, можно убедиться, даже только лишь прочитав названия разделов обширнейшей документации по системе, занимающей около 1500 страниц (которая, кстати, не только входит в состав дистрибутива, но и доступна через Интернет). Вычислительная мощь системы может быть продемонстрирована находящимся на ее сайте примером определения того, что кубик Рубика имеет 43252003274489856000 различных состояний, и сборки кубика Рубика из произвольного начального состояния в среднем за 100 ходов.

GAP дает возможность производить вычисления с гигантскими целыми и рациональными числами, допустимые значения которых ограничены только объемом доступной памяти. Далее, система работает с циклотомическими полями, конечными полями,  $p$ -адическими числами, многочленами от многих переменных, рациональными функциями, векторами и матрицами. Пользователю доступны различные комбинаторные функции, элементарные теоретико-числовые функции, разнообразные функции для работы с множествами и списками.

Группы могут быть заданы в различной форме, например, как группы подстановок, матричные группы, группы, заданные порождающими элементами и определяющими соотношениями. Более того, построив, например, групповую алгебру, можно вычислить ее мультипликативную группу, и даже задать ее подгруппу, порожденную конкретными обратимыми элементами групповой алгебры. Ряд групп может быть задан непосредственным обращением к библиотечным функциям (например, симметрическая и знакопеременная группы, группа диэдра, циклическая группа и др.).

Функции для работы с группами включают определение порядка группы, вычисление классов сопряженных элементов, центра и коммутанта группы, верхнего и нижнего центрального рядов, ряда коммутантов, Силловских подгрупп, максимальных подгрупп,

нормальных подгрупп, решеток подгрупп, групп автоморфизмов, и т.д. Для ряда конечных групп доступно определение их типа изоморфизма.

Теория представлений групп также входит в область применения системы GAP. Здесь имеются инструменты для вычисления таблиц характеров конкретных групп, действий над характерами и интерактивного построения таблиц характеров, определения теоретико-групповых свойств на основании свойств таблицы характеров группы. Модулярные представления групп (т.е. представления над полем, характеристика которого делит порядок группы) также могут быть исследованы с помощью GAP.

В версии 4.3 были существенным образом расширены возможности для работы с векторными пространствами, алгебрами и модулями. В системе могут быть определены векторные пространства над всеми доступными полями и модули над всеми доступными кольцами. Имеются алгоритмы для вычисления структуры конечномерных алгебр Ли, которые могут быть, например, заданы структурными константами или порождающими элементами, вычисления различных их Лиевских подалгебр и идеалов.

Версия 4.4, заменившая версию 4.3, содержит множество новых особенностей, усовершенствованных алгоритмов и средств программирования, и поэтому мы рекомендуем ее установку всем пользователям предыдущих версий. В частности, в GAP 4.4 появились новые алгоритмы и функции для работы с базисами Гребнера, алгебраическими расширениями полей, группами Галуа, таблицами характеров, векторными пространствами; новые методы для вычисления минимальных нормальных подгрупп конечной группы и цоколя конечной группы; быстрый метод для определения, является ли заданная группа подстановок симметрической или знакопеременной группой в их естественном представлении; разнообразные функции для вычислений с целочисленными матрицами, и др. нововведения.

Кроме новых алгоритмов и функций, в GAP 4.4 усовершенствована производительность многих уже существовавших ранее алгоритмов, в т.ч. для вычисления неприводимых представлений и их характеров, вычисления нормализаторов и сопряженных подгрупп в симметрических группах подстановок, вычисления системы представителей смежных классов в группах подстановок. Разложение подстановки в произведение порождающих элементов теперь возвращает существенно более короткие слова (например, длины около 100 для группы кубика Рубика). Усовершенствования также коснулись списков, многочленов, матриц и матричных групп, расширений конечных групп, конечномерных алгебр.

В вышедшей в мае 2005 г. версии GAP 4.4.5, в библиотеку конечных групп добавлены:

- группы порядков  $p^4$ ,  $p^5$ ,  $p^6$  для произвольных простых  $p$ ;
- группы, порядок которых свободен от квадратов;
- группы, порядок которых свободен от кубов и не превышает 50000.

Также в ней была введена новая функция `StructureDescription` для вычисления структурных описаний конечных групп, например:

Пример

```
gap> l := AllSmallGroups( Size, 8 );;
gap> List( l, StructureDescription );
[ "C8", "C4 x C2", "D8", "Q8", "C2 x C2 x C2" ]
```

Вместе с системой распространяется множество новых версий пакетов для GAP. При этом изменился механизм загрузки пакетов. Для загрузки каждый пакет должен содержать файл `PackageInfo.g` с мета-информацией о нем (все пакеты для GAP 4.4 такие файлы уже содержат). За исключением этой особенности, имеется только несколько незначительных изменений (например, тривиальная группа теперь не является простой), не совместимых с GAP 4.3.

Среди других областей применения системы - теория графов и их автоморфизмов, теория кодирования, теория полугрупп, кристаллография, и многое другое. Существует графический интерфейс XGAP (<http://www.math.rwth-aachen.de/~Max.Neunhoeffler/xgap4>), который работает в среде Unix/Linux и позволяет, например, графически изобразить решетку подгрупп группы (см. пример здесь: <http://ukrgap.exponenta.ru/Examples/Lattice.htm>). Информация о существующих разработках для применения в той или проблемной области может быть найдена на сайте GAP (<http://www.gap-system.org/>).

### 1.3 Инсталляция и запуск системы

При успешном запуске GAP на экране появится эмблема GAP. После нее будет напечатана дополнительная информация о версии системы и установленных компонентах, например:

Пример

```

#####          #####          #####          ###
#####          #####          #####          ###
#####          #####          #####          ###
#####          #####          #####          ###
#####          #          #####          #####
#####          #####          #####          #####
#####          #####          #####          #####
#####          #####          #####          #####
#####          #####          #####          #####
#####          #####          #####          #####
#####          #####          #####          #####
#####          #####          #####          #####
#####          #####          #####          #####
#####          #####          #####          #####
#####          #####          #####          #####
#####          #####          #####          #####
#####          #####          #####          #####
#####          #####          #####          #####
#####          #####          #####          #####
#####          #####          #####          #####
#####          #####          #####          #####

Information at: http://www.gap-system.org
Try 'help' for help. See also '?copyright' and '?authors'

Loading the library. Please be patient, this may take a while.
GAP4, Version: 4.4.10 of 02-Oct-2007, i686-apple-darwin9.1.0-gcc
Components:  small 2.1, small2 2.0, small3 2.0, small4 1.0, small5 1.0,
              small6 1.0, small7 1.0, small8 1.0, small9 1.0, small10 0.2,
              id2 3.0, id3 2.1, id4 1.0, id5 1.0, id6 1.0, id9 1.0, id10 0.1,
              trans 1.0, prim 2.1 loaded.
Packages:    openmath 06.03.02, GAPDoc 1.1, IO 2.3, AClib 1.1, Polycyclic 2.2,
              Alnuth 2.2.5, CrystCat 1.1.2, Cryst 4.1.5, Carat 2.0.2,

```

```
AutPGrp 1.2, CRISP 1.3.1, CTblLib 1.1.3, TomLib 1.1.2,
FactInt 1.5.2, FGA 1.1.0.1, HAP 1.8.2, Homology 1.4.2, EDIM 1.2.3,
IRREDSOL 1.1.2, LAGUNA 3.4, Sophus 1.23, Polenta 1.2.7,
ResClasses 2.5.3 loaded.
```

Точный вид этого сообщения будет зависеть от набора установленных пакетов и их совместимости с операционной системой.

Приглашение системы (командная строка) имеет следующий вид:

Пример

```
gap>
```

Для выхода из системы применяется команда `quit`; (заметим, что ввод любой команды завершается точкой с запятой и нажатием клавиши *Enter*).

*Примечание.* Для дублирования введенных команд и выводимых на экран результатов в текстовом файле используется команда `LogTo("filename.log");`. Ведение файла протокола может быть остановлено командой `LogTo()`; (например, чтобы просмотреть его содержимое в другом окне Windows, не прерывая сеанса работы с GAP).

## 1.4 Первые шаги в GAP

Если Вы впервые работаете с системой, Вы можете попробовать начать читать и вводить (в т.ч. через копирование и вставку) примеры из первых глав “Введения в GAP” (GAP Tutorial) с сайта GAP в формате **HTML** или **PDF**. В частности, там рассказывается, как пользоваться обширнейшей документацией по системе. См. также лабораторную работу “Основы работы с системой GAP в MS Windows” (Приложение C.1).

Вы можете попробовать, например, использовать GAP как простейший калькулятор:

Пример

```
gap> (9 - 7) * (5 + 6);
22
gap> 3^132;
955004950796825236893190701774414011919935138974343129836853841
gap>
```

Определить последние пять из 12978189 цифр 45-го числа Мерсенна  $2^{43112609} - 1$ , найденного в августе 2008 г. в ходе проекта GIMPS (Great Internet Mersenne Prime Search, см. <http://www.mersenne.org>), и являющегося на сегодня самым большим из известных науке простых чисел:

Пример

```
gap> a:=2^43112609-1;
<<an integer too large to be printed>>
gap> a mod 100000;
52511
```

Вычислить наибольший общий делитель двух целых чисел и найти его линейное представление:

Пример

```
gap> Gcd(100, 48);
4
gap> Gcdex(100, 48);
rec( gcd := 4, coeff1 := 1, coeff2 := -2, coeff3 := -12, coeff4 := 25 )
gap> 100*1+48*-2;
4
gap> 100*-12+48*25;
0
```

Разложить целое число на множители с помощью функции `FactorsInt`:

Пример

```
gap> FactorsInt(2^64-1);
[ 3, 5, 17, 257, 641, 65537, 6700417 ]
gap> FactorsInt(2^128-1);
[ 3, 5, 17, 257, 641, 65537, 274177, 6700417, 67280421310721 ]
gap> FactorsInt(2^200-1);
[ 3, 5, 5, 5, 11, 17, 31, 41, 101, 251, 401, 601, 1801, 4051, 8101, 61681,
  268501, 340801, 2787601, 3173389601 ]
```

GAP предоставляет разнообразные возможности для работы с множествами, списками, матрицами. Например, список квадратов натуральных чисел от 1 до 10 можно получить так:

Пример

```
gap> List([1..10], x -> x^2);
[ 1, 4, 9, 16, 25, 36, 49, 64, 81, 100 ]
```

А список натуральных чисел, не превосходящих 100 и являющихся произведением трех простых чисел (не обязательно различных), можно получить следующим образом:

Пример

```
gap> Filtered([1..100], x -> Length(Factors(x))=3 );
[ 8, 12, 18, 20, 27, 28, 30, 42, 44, 45, 50, 52, 63, 66, 68, 70, 75, 76, 78,
  92, 98, 99 ]
```

Теперь зададим матрицу следующим образом:

Пример

```
gap> A:=[[1,2,3,4],[4,2,1,5],[-1,10,0,0],[2,-4,7,0]];
[ [ 1, 2, 3, 4 ], [ 4, 2, 1, 5 ], [ -1, 10, 0, 0 ], [ 2, -4, 7, 0 ] ]
```

Для того, чтобы вывести ее на экран в удобочитаемой форме, используется команда `Display`:

Пример

```
gap> Display(A);
[ [ 1, 2, 3, 4 ],
  [ 4, 2, 1, 5 ],
  [ -1, 10, 0, 0 ],
  [ 2, -4, 7, 0 ] ]
```

Вычислим определитель этой матрицы:

Пример

```
gap> DeterminantMat(A);
-932
```

Таким образом, данная матрица является невырожденной, и система линейных уравнений с данной матрицей должна иметь единственное решение. Его можно найти с помощью функции `SolutionMat( mat, vec )`, которая возвращает вектор-строку  $x$  такую, что  $x * mat = vec$  (обратите внимание, что умножается вектор-строка на матрицу, а не матрица на вектор-столбец!). Пусть  $vec = [1, -1, 0, 3]$ , тогда

Пример

```
gap> v:=SolutionMat(A,[1,-1,0,3]);
[ 519/932, 36/233, -323/932, -243/932 ]
```

Проверим, что действительно было найдено решение системы:

Пример

```
gap> v*A;
[ 1, -1, 0, 3 ]
```

Можно задать подстановки и вычислить их произведение:

Пример

```
gap> (1,2,3);
(1,2,3)
gap> (1,2,3) * (1,2);
(2,3)
```

Найти подстановку, обратную к данной:

Пример

```
gap> (1,2,3)^-1;
(1,3,2)
```

Найти образ точки под действием данной подстановки:

Пример

```
gap> 2^(1,2,3);
3
```

Вычислить подстановку, сопряженную с данной с помощью другой подстановки:

Пример

```
gap> (1,2,3)^(1,2);
(1,3,2)
```

Теперь зададим группу, порожденную двумя подстановками:

Пример

```
gap> s8 := Group( (1,2), (1,2,3,4,5,6,7,8) );
Group([ (1,2), (1,2,3,4,5,6,7,8) ])
```

Как известно, это есть не что иное, как симметрическая группа подстановок 8-й степени. Теперь мы можем вычислить ее коммутант:

Пример

```
gap> a8 := DerivedSubgroup( s8 );
Group([ (1,2,3), (2,3,4), (2,4)(3,5), (2,6,4), (2,4)(5,7), (2,8,6,4)(3,5) ])
```

и исследовать его свойства - найти его порядок, проверить его коммутативность:

Пример

```
gap> Size( a8 ); IsAbelian( a8 );
20160
false
```

Другие примеры применения GAP описаны в следующих разделах данного методического пособия, а также в документации на английском языке, которая не только является частью системы (см. каталог [gap4r4/doc](http://www.gap-system.org/Doc/manuals.html)), но и доступна для просмотра и загрузки в форматах HTML и PDF на сайте системы по адресу <http://www.gap-system.org/Doc/manuals.html>. Кроме того, в разделе “Изучаем алгебру с GAP” сайта Украинской группы пользователей GAP (<http://ukrgap.exponenta.ru/>) содержится серия примеров к курсу алгебры и теории чисел.

Для получения новостей рекомендуем подписаться на рассылку новостей Украинской группы пользователей GAP (<http://subscribe.ru/catalog/science.exact.gap/>), а также на GAP Forum (<http://mail.gap-system.org/mailman/listinfo/forum>) - англоязычный форум для обсуждения связанных с GAP вопросов, сообщений об обновлениях, новых версиях, пакетах, конференциях и др. связанных с GAP событиях.

Желаем Вам приятной работы с системой GAP!

## Глава 2

# Язык программирования GAP

### 2.1 Символы и категории слов в GAP

GAP воспринимает следующие символы: цифры, буквы (верхний и нижний регистры), пробел, символы табуляции и новой строки, а также специальные символы:

"	'	(	)	*	+	,	-	#
.	/	:	;	<	=	>	~	&
[	\	]	^	_	{	}	!	

Составленные из символов слова относятся к следующим категориям:

- ключевые слова (зарезервированные последовательности букв нижнего регистра)
- идентификаторы (последовательности цифр и букв, содержащая не менее одной буквы и не являющаяся ключевым словом)
- строки (последовательности произвольных символов, заключенная в двойные кавычки)
- целые числа (последовательности цифр)
- операторы и ограничители в соответствии со следующим списком:

+	-	*	/	^	~	!.
=	<>	<	<=	>	>=	![
:=	.	..	->	,	;	!{
[	]	{	}	(	)	:

Следует заметить, что пробелы могут быть использованы для повышения удобочитаемости текста, так как любая последовательность пробелов воспринимается GAP как один пробел. Таким образом, команда

```
if i<0 then a:=-i;else a:=i;fi;
```

может быть записана следующим образом:

```
if i < 0 then # если i отрицательное
  a := -i;
else          # иначе
  a := i;
fi;
```

## 2.2 Ключевые слова

Ключевыми словами GAP являются следующие слова:

and	do	elif	else	end	fi
for	function	if	in	local	mod
not	od	or	repeat	return	then
until	while	quit	QUIT	break	rec
continue					

## 2.3 Идентификаторы

Идентификаторы состоят из букв, цифр, символов подчеркивания `_`, и должны содержать не менее одной буквы или символа подчеркивания `_`. При этом регистр является существенным. Примеры идентификаторов:

a	foo	LongIdentifier
hello	Hello	HELLO
x100	100x	_100
underscores_case	MixedCase	

## 2.4 Выражения

Примерами выражений являются: переменные, обращения к функциям, целые числа, перестановки, строки, функции, списки, записи. С помощью операторов из них могут быть составлены более сложные выражения. Операторы разбиты на три класса:

- операторы сравнения:    =    <>    <    <=    >    >=    in
- арифметические операторы:    +    -    \*    /    mod    ^

- логические операторы: `not` `and` `or`

Пример 1:

Пример

```
gap>2*2;; #два знака ";" подавляют вывод на экран
gap>2*2+9=Fibonacci(7) and Fibonacci(13) in Prime;
true
```

Следует различать глобальные и локальные переменные, различия которых можно видеть из следующего примера: Пример 2:

```
g := 0;          # глобальная переменная g
x := function ( a, b, c )
  local y;
  g := c;        # c - аргумент функции x
  y := function ( y )
    local d, e, f;
    d := y; # y - аргумент функции y
    e := b; # b - аргумент функции x
    f := g; # g - глобальная переменная g
    return d + e + f;
  end;
  return y(a); # y-локальная переменная функции x
end;
```

## 2.5 Обращения к функциям

Формат:

```
function-var()
function-var( arg-expr {, arg-expr} )
```

Пример 1:

Пример

```
gap> Fibonacci( 11 ); # обращение к функции "Fibonacci" с аргументом 11
89
```

Пример 2: обращение к операции `RightCosets`, в котором второй аргумент определяется обращением к другой функции

Пример

```
gap> RightCosets( G, Intersection( U, V ) );
```

## 2.6 Сравнение выражений

Формат:

```
left-expr = right-expr
left-expr <> right-expr
left-expr < right-expr
left-expr > right-expr
left-expr <= right-expr
left-expr >= right-expr
```

Операторы = и <> проверяют соответственно равенство и неравенство, возвращая **true** или **false**. Заметьте, что с их помощью можно сравнивать любые объекты, т.е. при использовании = и <> никогда не будет получено сообщение об ошибке. Для каждого типа объектов определение равенства может отличаться и описано в соответствующем разделе справочного руководства. Объекты, относящиеся к различным семействам (*families*) всегда различны, т.е. = приведет к **false**, и <> - к **true**. Кроме того, в некоторых случаях для них может быть определено отношение “меньше”. Операторы сравнения имеют больший приоритет по сравнению с логическими операторами, но меньший по сравнению с арифметическими. Например,  $a*b = c$  and  $d$  интерпретируется как  $((a*b)=c)$  and  $d$ . Еще один пример (сравнение, левая часть которого является выражением):

Пример

```
gap> 2 * 2 + 9 = Fibonacci(7);
true
```

## 2.7 Арифметические операторы

Формат:

```
+ right-expr
- right-expr
left-expr + right-expr
left-expr - right-expr
left-expr * right-expr
left-expr / right-expr
left-expr mod right-expr
left-expr ^ right-expr
```

Значение, как правило, зависит от типа операндов. **mod** определен только для целых и рациональных чисел. Для элемента группы **^** означает возведение в степень, если правый операнд - целое число, а если он - элемент группы, то сопряжение с его помощью. Приоритет операторов (по убыванию):

- ^

- унарные + и -
- \*, /, mod
- + и -

Пример:

```
-2 ^ -2 * 3 + 1
```

означает

```
(-(2 ^ (-2)) * 3) + 1
```

Арифметические операторы имеют наивысший приоритет по сравнению с операторами сравнения и логическими операторами.

## 2.8 Присваивания

Командами в GAP называются: присваивания, вызовы процедур, структуры `if`, `while`, `repeat`, `for`, а также команда `return`. Все команды заканчиваются точкой с запятой “;”. Присваивания имеют формат

```
var := expr;
```

Пример:

Пример

```
gap> data:= rec( numbers:= [ 1, 2, 3 ] );
rec( numbers := [ 1, 2, 3 ] )
gap> data.string:= "string";; data;
rec( numbers := [ 1, 2, 3 ], string := "string" )
gap> data.numbers[2]:= 4;; data;
rec( numbers := [ 1, 4, 3 ], string := "string" )
```

## 2.9 Вызов процедуры

Формат:

```
procedure-var();
procedure-var( arg-expr {, arg-expr} );
```

Различие между процедурами и функциями введено для удобства, GAP же их не различает. Функция возвращает значение, но не производит побочных эффектов. Процедура не возвращает никакого значения, но производит какое-либо действие (например, процедуры `Print`, `Append`, `Sort`).

## 2.10 Команда IF

Формат:

```
if bool-expr1 then statements1
  { elif bool-expr2 then statements2 }
  [ else statements3 ]
fi;
```

При этом частей `elif` может быть произвольное количество или ни одной. Часть `else` также может отсутствовать. Пример 1: в командах

```
if expr1 then
  if expr2 then stats1
  else stats2 fi;
fi;
```

`else` относится ко второму `if`, тогда как в командах

```
if expr1 then
  if expr2 then stats1 fi;
else stats2
fi;
```

`else` относится к первому `if`. Пример 2:

Пример

```
gap> i := 10;;
gap> if 0 < i then
>   s := 1;
>   elif i < 0 then
>     s := -1;
>   else
>     s := 0;
>   fi;
gap> s; # знак i
1
```

## 2.11 Цикл WHILE

Формат:

```
while bool-expr do statements od;
```

Последовательность команд **statements** выполняется, пока истинно условие **bool-expr**. При этом сначала проверяется условие, а затем, если оно истинно, выполняются команды. Если уже при первом обращении условие ложно, то последовательность команд **statements** не выполнится ни разу. Пример:

Пример

```
gap> i := 0;; s := 0;;
gap> while s <= 200 do
>     i := i + 1; s := s + i^2;
>     od;
gap> s;
204
```

## 2.12 Цикл REPEAT

Формат:

```
repeat statements until bool-expr;
```

Последовательность команд **statements** выполняется до тех пор, пока не будет выполняться условие **bool-expr**. При этом сначала выполняются команды, а затем проверяется условие. Таким образом, при любом начальном значении условия набор команд **statements** выполнится, по крайней мере, один раз. Пример (вычисление наименьшей суммы квадратов первых  $n$  последовательных натуральных чисел, превышающей 200):

Пример

```
gap> i := 0;; s := 0;;
gap> repeat
>     i := i + 1; s := s + i^2;
>     until s > 200;
gap> s;
204
```

## 2.13 Цикл FOR

Форматы:

```

for simple-var in list-expr do statements od;
for variable in iterator do statements od;
for variable in object do statements od;

```

В первом варианте последовательность команд **statements** выполняется для каждого элемента из списка **list-expr**. Цикл **for** эквивалентен циклу **while**:

```

loop-list := list;
loop-index:= 1;
while loop-index <= Length(loop-list) do
  variable := loop-list[loop-index];
  ...
  statements
  ...
  loop-index := loop-index+1;
od;

```

Список **list** часто является последовательностью. Команда

```

for variable in [from..to] do statements od;

```

соответствует распространенной в других языках команде

```

for variable from from to to do statements od;

```

Пример:

Пример

```

gap> s := 0;;
gap> for i in [1..100] do
>   s := s + i;
> od;
gap> s;
5050

```

В следующем примере изменение списка приводит к выполнению команд для новых его элементов:

Пример

```

gap> l := [ 1, 2, 3, 4, 5, 6 ];;
gap> for i in l do

```

```

>      Print( i, " " );
>      if i mod 2 = 0 then Add( l, 3*i/2 ); fi;
> od; Print( "\n" );
1 2 3 4 5 6 3 6 9 9
gap> l;
[ 1, 2, 3, 4, 5, 6, 3, 6, 9, 9 ]

```

А в следующем - не приводит:

Пример

```

gap> l := [ 1, 2, 3, 4, 5, 6 ];;
gap> for i in l do
>      Print( i, " " );
>      l := [];
> od; Print( "\n" );
1 2 3 4 5 6
gap> l;
[ ]

```

Теперь рассмотрим оставшиеся два варианта цикла FOR:

```

for variable in iterator do statements od;
for variable in object do statements od;

```

Для этого необходимо сначала ввести некоторые понятия, характеризующие объекты, с которыми работает GAP. Каждый объект имеет свой *тип*. Тип объекта - это информация, которая используется для того, чтобы решить, применима ли к объекту данная операция, и, для выбора метода ее применения в случае положительного ответа. Например, тип двух объектов определяет, могут ли они быть умножены друг на друга, и если да, то каким способом. Аналогично, тип объекта определяет, может ли быть вычислен порядок (размер) этого объекта, и если да, то как. Тип объекта состоит из двух основных частей, характеризующих разные аспекты объекта, а именно из *семейства*, к которому он принадлежит, и из набора *фильтров*. Семейство определяет отношение данного объекта к другим объектам. Например, семейство образуют все подстановки. Другой пример семейства - семейство всех коллекций (collections), подстановок. Последнее семейство содержит множество всех групп подстановок в качестве подмножества. Третий пример - семейство всех рациональных функций с коэффициентами из некоторого семейства. Фильтр можно описать как последовательность `true` и `false`, характеризующую соответствующие параметры данного типа объектов (например, `IsAbelian`, `IsPrime`, `IsGroup` и т.п.). Хотя при выборе метода фильтры не различаются по их происхождению и применению, их можно классифицировать (с небольшими исключениями) на *категории*, *представления*, *свойства*, а также *тестеры атрибутов*. Набор элементов, лежащих в одном семействе, называется *коллекцией*. Примерами коллекций являются однородные списки, а также *домены* (domain) - так в GAP называются множества с дополнительной структурой, например, группа, класс сопряженных элементов, векторное пространство и т.д. Пусть теперь C

- коллекция. Тогда *итератор* `Iterator(C)` позволяет организовать перебор всех элементов этой коллекции без повторов. В этом случае цикл FOR эквивалентен следующему циклу:

```
while not IsDoneIterator(iterator) do
  variable := NextIterator(iterator)
  statements
od;
```

Если же `D` - домен, то результат выполнения цикла вида

```
for variable in domain do
```

должен быть таким же, как результат выполнения цикла вида

```
for variable in AsList(D) do
```

Если в цикле FOR объект `object` не является списком или итератором, то будет вычисляться `Iterator(object)`. Если это вычисление будет успешным, то будет реализован перебор по итератору.

Пример

```
gap> g := Group((1,2,3,4,5), (1,2)(3,4)(5,6));
Group([ (1,2,3,4,5), (1,2)(3,4)(5,6) ])
gap> count := 0;; sumord := 0;;
gap> for x in g do
> count := count + 1; sumord := sumord + Order(x); od;
gap> count;
120
gap> sumord;
471
```

Такая конструкция использует намного меньше памяти, так как итератор может быть намного более компактным, чем список всех элементов. Из цикла FOR можно выйти до его выполнения, используя оператор `break`. Это особенно эффективно в комбинации с циклом по итератору, например, для поиска элемента с нужным свойством в некотором домене. Этот оператор может быть использован только внутри цикла.

Пример

```
gap> g := Group((1,2,3,4,5), (1,2)(3,4)(5,6));
Group([ (1,2,3,4,5), (1,2)(3,4)(5,6) ])
gap> for x in g do
> if Order(x) = 3 then
> break;
```

```
> fi; od;
gap> x;
(1,4,3)(2,6,5)
```

Оператор `continue` отменяет выполнение оставшейся части цикла и переходит сразу к выполнению следующей итерации. Этот оператор также может быть использован только внутри цикла.

Пример

```
gap> g := Group((1,2,3),(1,2));
Group([ (1,2,3), (1,2) ])
gap> for x in g do
> if Order(x) = 3 then
> continue;
> fi; Print(x,"\n"); od;
()
(2,3)
(1,3)
(1,2)
```

## 2.14 Функции

Формат:

```
function ( [ arg-ident {, arg-ident} ] )
  [ local loc-ident {, loc-ident} ; ]
  statements
end
```

Пример функции, которая определяет  $n$ -е число Фибоначчи:

Пример

```
gap> fib := function ( n )
>   local f1, f2, f3, i;
>   f1 := 1; f2 := 1;
>   for i in [3..n] do
>     f3 := f1 + f2; f1 := f2; f2 := f3;
>   od;
>   return f2;
> end;;
gap> List( [1..10], fib );
[ 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 ]
```

Ту же функцию можно определить рекурсивно:

Пример

```

gap> fib := function ( n )
>   if n < 3 then
>     return 1;
>   else
>     return fib(n-1) + fib(n-2);
>   fi;
> end;;
gap> List( [1..10], fib );
[ 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 ]

```

Заметим, что рекурсивная версия требует  $2fib(n) - 1$  шагов для вычисления  $fib(n)$ , тогда как итеративная требует только  $n - 2$  шага. Обе функции, однако, не являются оптимальными, так как библиотечная функция `Fibonacci` требует порядка  $\log(n)$  шагов. Запись `arg-ident -> expr` является краткой записью для функции

```
function ( arg-ident ) return expr; end
```

Здесь `arg-ident` - один идентификатор, т.е. таким образом нельзя задать функцию от нескольких переменных. Пример типичного использования такой записи:

Пример

```

gap> Sum( List( [1..100], x -> x^2 ) );
338350

```

## 2.15 Команда RETURN

Формат:

```

return;
return expr;

```

Первая форма прерывает выполнение внутренней (при вызове одной функции из другой) функции и передает управление вызывающей функции, не возвращая при этом никакого значения. Вторая, кроме того, возвращает значение выражения `expr`.

## Глава 3

# Структуры данных в GAP

### 3.1 Константы и операторы

Основные принципы задания констант и действий над ними видны из следующих примеров:

Арифметические вычисления:

Пример

```
gap> 12345/25;  
2469/5  
gap> -3; 17 - 23;  
-3  
-6  
gap> 3^132;  
955004950796825236893190701774414011919935138974343129836853841
```

Действия с подстановками:

Пример

```
gap> (1,2,3);  
(1,2,3)  
gap> (1,2,3) * (1,2);  
(2,3)  
gap> (1,2,3)^-1;  
(1,3,2)  
gap> 2^(1,2,3);  
3  
gap> (1,2,3)^(1,2);  
(1,3,2)
```

Задание символьных констант:

Пример

```
gap> 'a';  
'a'  
gap> "abc";
```

```
"abc"
```

## 3.2 Переменные и присваивания

Порядок присваивания демонстрируется следующим примером:

Пример

```
gap> a:= (9 - 7) * (5 + 6);
22
gap> a;
22
gap> a:= 10;
10
gap> a * (a + 1);
110
```

Примечание 1. После присваивания присвоенное значение отображается в следующей строке вывода. Это можно подавить, если завершить команду двумя знаками ; вместо одного:

Пример

```
gap> w:= 2;;
```

Примечание 2. Всякий раз, когда GAP возвращает значение, печатая его в следующей после команды строке, это значение присваивается переменной с именем last:

Пример

```
gap> (9 - 7) * (5 + 6);
22
gap> a:= last;
22
```

Аналогичным образом определяются переменные last2 и last3.

## 3.3 Функции

GAP содержит более 4000 стандартных функций. Пример обращения к нескольким из них приведен ниже:

Пример

```
gap> Factorial(17);
355687428096000
gap> Gcd(1234, 5678);
2
gap> Print(1234, "\n");
1234
```

Кроме того, пользователь может вводить собственные функции. Наиболее просто это делается так:

Пример

```
gap> cubed:= x -> x^3;
function( x ) ... end
gap> cubed(5);
125
```

Другой способ определения функций и процедур изложен в пунктах 2.14 и 3.12. Рекомендации по разработке программ на языке GAP содержатся в Приложении А.

### 3.4 Списки

Список является заключенным в квадратные скобки набором объектов, разделенных запятыми. Например, список из первых десяти простых чисел можно задать следующим образом:

Пример

```
gap> primes:=[2, 3, 5, 7, 11, 13, 17, 19, 23, 29];
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 ]
```

Затем к нему можно добавить следующие два простых числа:

Пример

```
gap> Append(primes, [31, 37]);
gap> primes;
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37 ]
```

Если добавляется только один элемент, это можно сделать и по-другому:

Пример

```
gap> Add(primes, 41);
gap> primes;
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41 ]
```

Указать отдельный элемент списка можно по его номеру в списке:

Пример

```
gap> primes[7];
17
```

Этот же механизм позволяет присвоить значение существующему или новому элементу списка (функция `Length` определяет длину списка):

Пример

```
gap> Length(primes);
13
gap> primes[14]:= 43;
43
gap> primes;
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43 ]
```

При этом значение не обязательно должно присваиваться следующему элементу списка. Например, если двадцатым простым числом является 71, мы можем сразу присвоить значение 71 двадцатому элементу списка `primes`, пропуская недостающие элементы. Полученный список будет иметь длину 20:

Пример

```
gap> primes[20]:= 71;
71
gap> primes;
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,,,,, 71 ]
gap> Length(primes);
20
```

Список должен быть создан перед заданием его элемента (например, быть пустым списком `[ ]`):

Пример

```
gap> l11[1]:= 2;
Variable: 'l11' must have a value

gap> l11:= [];
gap> l11[1]:= 2;
2
```

Функция `Position` возвращает номер первого элемента списка, имеющего заданное значение. Если в списке нет элемента с заданным значением, функция возвращает `false`:

Пример

```
gap> Position(primes, 17);
7
gap> Position(primes, 20);
fail
```

Заметим, что при всех приведенных выше изменениях списка `primes` длина списка изменялась автоматически. Функция `IsBound` для списков показывает, содержит ли список элемент с заданным номером (для записей - содержит ли запись указанное поле):

Пример

```
gap> k:= [ , 2, 3, , 5, , 7, , , 11 ];;
gap> IsBound(k[7]); IsBound(k[4]); IsBound(k[20]);
true
false
false
```

Список может состоять из объектов различных типов, например:

Пример

```
gap> l11:= [true, "This is a String",,, 3];
[ true, "This is a String",,, 3 ]
```

Далее, список может являться частью другого списка:

Пример

```
gap> l11[3]:= [4,5,6];;
gap> l11;
[ true, "This is a String", [ 4, 5, 6 ],, 3 ]
```

и даже частью самого себя:

Пример

```
gap> l11[4]:= l11;
[ true, "This is a String", [ 4, 5, 6 ], ~, 3 ]
```

Здесь знак ~ в четвертой позиции обозначает объект, вывод которого на экран (печать) производится в данный момент. Строки являются частным случаем списков, и печатаются без разделителей. Примеры задания строк и операций над ними:

Пример

```
gap> s1:=['H','e','l','l','o',' ','w','o','r','l','d','.'];
"Hello world."
gap> s2 := "Hello world.";
"Hello world."
gap> s1 = s2;
true
gap> s2[7];
'w'
```

Извлечение и изменение подмножеств списка производит оператор { }:

Пример

```
gap> s1 := l11{ [ 1, 2, 3 ] };
[ true, "This is a String", [ 4, 5, 6 ] ]
```

```
gap> sl{ [ 2, 3 ] } := [ "New String", false ];
[ "New String", false ]
gap> sl;
[ true, "New String", false ]
```

### 3.5 Тожественность и равенство списков

Для изучения способов управления сложными структурами данных в GAP важно понимать различия между тождественными и равными объектами. В данном разделе это различие демонстрируется на примере списков. Аналогичные примеры могут быть подобраны и для записей.

Два списка равны (т.е. оператор сравнения = возвращает true) тогда и только тогда, когда они имеют одинаковую длину и их соответствующие элементы равны.

Пример

```
gap> numbers:= primes;
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,,,,, 71 ]
gap> numbers = primes;
true
```

Теперь изменим список `numbers` и снова сравним его с `primes`:

Пример

```
gap> primes[3]:= 4;
4
gap> numbers = primes;
true
```

Оказывается, что списки `numbers` и `primes` снова равны, а распечатав список `primes`, мы увидим, что `primes[3]=4`. Это объясняется тем, что списки `primes` и `numbers` не только равны, но и идентичны. Идентификаторы `primes` и `numbers` указывают на один и тот же список, и изменения в нем происходят при указании любого из его имен. Присваивание `numbers:= primes` создает не новый список, а только новое имя для уже существующего списка.

Если необходимо изменить список, совпадающий по содержанию с `primes` таким образом, чтобы сам список `primes` при этом не изменился, необходимо создать копию списка `primes` с помощью функции `ShallowCopy` (в следующем примере предварительно восстановим старое значения `primes`):

Пример

```
gap> primes[3]:= 5;
5
gap> primes;
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,,,,, 71 ]
gap> numbers:= ShallowCopy(primes);
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,,,,, 71 ]
```

```
gap> numbers = primes;
true
gap> numbers[3] := 4;
4
gap> numbers = primes;
false
```

Примечание. Единственными объектами, которые могут быть изменены таким способом, являются списки и записи, т.к. только эти объекты в GAP могут состоять из других объектов. Например, после выполнения следующих команд значения  $i$  и  $j$  будут соответственно равны 2 и 1:

Пример

```
gap> i:= 1;; j:= i;; i:= i+1;;
```

Упражнение. Объяснить, что происходит в результате выполнения команд:

Пример

```
gap> l:= [];
[ ]
gap> l:= [1];
[ [ ] ]
gap> l[1]:= 1;
[ ~ ]
```

### 3.6 Множества

Множествами в GAP называются списки специального вида. Элементы множества расположены последовательно (т.е. не содержат пробелов, как, например, список [ 2, 3, 5, , , , , , , 31, 37, 41 ]), упорядочены (порядок сортировки GAP определяет самостоятельно) и встречаются в списке только один раз. Множества, как и списки, могут содержать объекты различных типов.

Проверить, является ли объект множеством, можно с помощью функции `IsSet`. Для каждого списка существует соответствующее ему множество, получаемое с помощью функции `Set`.

Пример

```
gap> fruits:=["apple", "strawberry", "cherry", "plum", "apple"];
gap> IsSet(fruits);
false
gap> fruits:= Set(fruits);
[ "apple", "cherry", "plum", "strawberry" ]
```

Заметим, что при этом исходный список `fruits` был изменен.

Для проверки принадлежности объекта множеству используется оператор `in`. Его также можно использовать для проверки принадлежности к списку, однако в первом

случае проверка выполняется быстрее, т.к. сортировка позволяет использовать двоичный поиск вместо последовательного перебора.

Пример

```
gap> "apple" in fruits;
true
gap> "banana" in fruits;
false
```

Добавить к множеству новый элемент можно с помощью функции `AddSet` (обратите внимание на порядок следования элементов):

Пример

```
gap> AddSet(fruits, "banana");
gap> fruits;
[ "apple", "banana", "cherry", "plum", "strawberry" ]
gap> AddSet(fruits, "apple");
gap> fruits;      # 'fruits' не изменилось
[ "apple", "banana", "cherry", "plum", "strawberry" ]
```

Пересечение, объединение и разность множеств определяются с помощью функций `Intersection`, `Union` и `Difference`. При этом аргументы могут быть обычными списками, тогда как результат всегда будет являться множеством.

Пример

```
gap> breakfast:= ["tea", "apple", "egg"];
[ "tea", "apple", "egg" ]
gap> Intersection(breakfast, fruits);
[ "apple" ]
gap> Difference(breakfast,fruits);
[ "egg", "tea" ]
```

Те же операции над множествами производят функции `IntersectSet`, `UniteSet` и `RemoveSet`, но они не возвращают результат, а заменяют им первый аргумент.

### 3.7 Векторы и матрицы

Вектор является не содержащим пробелов списком элементов, принадлежащих общему полю.

Пример

```
gap> v:= [3, 6, 2, 5/2];
[ 3, 6, 2, 5/2 ]
gap> IsRowVector(v);
true
```

Векторы умножаются на скаляры из любого поля, содержащего данное. Умножение двух векторов равной длины дает их скалярное произведение.

Пример

```
gap> 2 * v;
[ 6, 12, 4, 5 ]
gap> v * 1/3; # это эквивалентно команде v/3;
[ 1, 2, 2/3, 5/6 ]
gap> v * v; # скалярное произведение v на себя
221/4
```

Матрица - список векторов одинаковой длины, не содержащий пробелов:

Пример

```
gap> m:= [[1, -1, 1],
>        [2, 0, -1],
>        [1, 1, 1]];
[ [ 1, -1, 1 ], [ 2, 0, -1 ], [ 1, 1, 1 ] ]
gap> m[2][1];
2
```

Матрицы можно умножать на скаляры, векторы и другие матрицы (при этом умножение обобщается и возможно также при несоответствии размеров):

Пример

```
gap> m:= [[1, 2, 3, 4],
>        [5, 6, 7, 8],
>        [9,10,11,12]];
[ [ 1, 2, 3, 4 ], [ 5, 6, 7, 8 ], [ 9, 10, 11, 12 ] ]
gap> Display(m);
[ [ 1, 2, 3, 4 ],
  [ 5, 6, 7, 8 ],
  [ 9, 10, 11, 12 ] ]
gap> [1, 0, 0, 0] * m;
[ 1, 2, 3, 4 ]
gap> [1, 0, 0] * m;
[ 1, 2, 3, 4 ]
gap> m * [1, 0, 0];
[ 1, 5, 9 ]
gap> m * [1, 0, 0, 0];
[ 1, 5, 9 ]
gap> m * [0, 1, 0, 0];
[ 2, 6, 10 ]
```

Заметим, что умножение вектора на матрицу приводит к линейной комбинации строк матрицы, тогда как умножение матрицы на вектор приводит к линейной комбинации ее столбцов. В последнем случае вектор рассматривается как вектор-столбец.

Подматрицы извлекаются или изменяются с помощью фигурных скобок:

Пример

```
gap> sm := m{ [ 1, 2 ] }{ [ 3, 4 ] };
[ [ 3, 4 ], [ 7, 8 ] ]
gap> sm{ [ 1, 2 ] }{ [2] } := [[1],[-1]];
[ [ 1 ], [ -1 ] ]
gap> sm;
[ [ 3, 1 ], [ 7, -1 ] ]
```

Первая пара скобок указывает выбранные строки, вторая - столбцы.

### 3.8 Записи

Другой способ создания новых структур данных - записи. Как и списки, записи - наборы других объектов (которые называются компонентами, или полями), обращение к которым происходит не по номеру, а по имени.

Пример

```
gap> date:= rec(year:=1992, month:="Jan", day:=13);
rec( year := 1992, month := "Jan", day := 13 )
```

Изначально запись определяется как разделенный запятыми список присваиваний значений ее полям. Для обращения к значению соответствующего поля записи необходимо указать имя записи и имя поля, разделив их точкой. Определив запись, в дальнейшем можно добавлять к ней новые поля.

Пример

```
gap> date.year;
1992
gap> date.time:=rec(hour:=19,minute:=23,second:=12);
rec( hour := 19, minute := 23, second := 12 )
gap> date;
rec( year := 1992, month := "Jan", day := 13,
      time := rec( hour := 19, minute := 23, second := 12 ) )
```

Для определения, является ли объект записью, применяется функция `IsRecord`. Структуру записи можно получить с помощью функции `RecNames`:

Пример

```
gap> RecNames(date);
[ "year", "month", "day", "time" ]
```

Упражнение. Что происходит в результате выполнения команд:

Пример

```
gap> r:= rec();
```

```

rec(
  )
gap> r:= rec(r:= r);
rec(
  r := rec(
    ) )
gap> r.r:= r;
rec(
  r := ~ )

```

### 3.9 Арифметические прогрессии

Другим специальным видом списков являются целочисленные конечные арифметические прогрессии. Они описываются первым, вторым и последним элементами, разделенными соответственно запятой или двумя точками, и заключенными в квадратные скобки. Если прогрессия состоит из последовательных чисел, второй элемент может быть опущен.

Пример

```

gap> [1..999999]; #натуральные числа от 1 до 999999
[ 1 .. 999999 ]
gap> [1,2..999999];#эквивалентно предыдущей команде
[ 1 .. 999999 ]
gap> [1,3..999999]; # здесь шаг равен 2
[ 1, 3 .. 999999 ]
gap> Length( last );
500000
gap> [ 999999, 999997 .. 1 ];
[ 999999, 999997 .. 1 ]

```

### 3.10 Использование циклов

Пример 1: вычислить произведение подстановок, являющихся элементами списка.

Пример

```

gap> pp:=[(1,3,2,6,8)(4,5,9), (1,6)(2,7,8)(4,9),
> (1,5,7)(2,3,8,6), (1,8,9)(2,3,5,6,4),
> (1,9,8,6,3,4,7,2) ];;
gap> prod:= ();
()
gap> for p in pp do
>   prod:= prod * p;
> od;
gap> prod;
(1,8,4,2,3,6,5)

```

Пример 2: вычисление  $n!$  для  $n = 15$ .

Пример

```

gap> ff:= 1;
1
gap> for i in [1..15] do
>     ff:= ff * i;
>   od;
gap> ff;
1307674368000

```

Пример 3: разложить на простые множители число 1333, используя список простых чисел `primes`.

Пример

```

gap> n:= 1333;
1333
gap> factors:= [];
[ ]
gap> for p in primes do
>     while n mod p = 0 do
>         n:= n/p;
>         Add(factors, p);
>     od;
> od;
gap> factors;
[ 31, 43 ]
gap> n;
1

```

Так как  $n=1$ , то процесс завершен (легко проверить, умножив 31 на 43).

Пример 4: составить список простых чисел, не превышающих 1000 (функция `Unbind` исключает элемент из списка).

Пример

```

gap> primes:= [];
[ ]
gap> numbers:= [2..1000];
[ 2 .. 1000 ]
gap> for p in numbers do
>     Add(primes, p);
>     for n in numbers do
>         if n mod p = 0 then
>             Unbind(numbers[n-1]);
>         fi;
>     od;
> od;

```

### 3.11 Дальнейшие операции со списками

Существует более удобный способ умножения элементов списка из чисел или подстановок.

Пример

```
gap> Product([1..15]);
1307674368000
gap> Product(pp);
(1,8,4,2,3,6,5)
```

Аналогичным образом работает функция `Sum`.

Пример 1:

Аргументами функции `List` является список и имя функции. В результате будут создан список значений заданной функции на элементах заданного списка. Например, для нахождения куба числа ранее была определена функция `cubed`. Составим с ее помощью список кубов чисел от 2 до 10.

Пример

```
gap> List([2..10], cubed);
[ 8, 27, 64, 125, 216, 343, 512, 729, 1000 ]
```

Чтобы сложить все эти величины, мы можем применить функцию `Sum` к последнему списку. Это же можно сделать, используя функцию `cubed` в качестве дополнительного аргумента функции `Sum`:

Пример

```
gap> Sum(last) = Sum([2..10], cubed);
true
```

Пример 2: получение списка простых чисел, меньших 30, из списка `primes` с помощью функции `Filtered`:

Пример

```
gap> Filtered(primes, x-> x < 30);
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 ]
```

Пример 3: оператор `{ }` извлекает часть списка, определяемую номерами начального и конечного элементов списка:

Пример

```
gap> primes{ [1 .. 10] };
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 ]
```

### 3.12 Функции

Ранее было показано, как обратиться к библиотечным, т.е. стандартным функциям GAP. Данный раздел посвящен разработке новых функций.

Пример 1: задать простейшую функцию, которая печатает на экране текст “hello, world!”.

Пример

```
gap> sayhello:= function()
> Print("hello, world!\n");
> end;
function( ) ... end
```

При этом добавление к строке вывода символа `\n` приведет к тому, что следующее приглашение GAP появится на новой строке, а не непосредственно после напечатанного текста.

Определение функции начинается с ключевого слова `function`, после которого в скобках указываются формальные параметры. Скобки необходимы и в том случае, если параметры отсутствуют. Следует обратить внимание на отсутствие точки с запятой после первой команды. Определение функции завершается ключевым словом `end`.

Функция после ее определения является такой же переменной, как и целые числа, суммы и списки, и может быть присвоена другой переменной. Завершающий знак “;” в приведенном примере не принадлежит к определению функции, а завершает ее присвоение имени `sayhello`. После этого, в отличие от других присваиваний, значение функции `sayhello` отображается в сокращенной форме `function( ) ... end`, отображающей только ее формальные параметры, как наиболее интересную часть.

Полное значение `sayhello` может быть получено с помощью функции `Print`:

Пример

```
gap> Print(sayhello, "\n");
function ( )
  Print( "hello, world!\n" );
  return;
end
```

Обращение к данной функции произойдет по команде `sayhello()`:

Пример

```
gap> sayhello();
hello, world!
```

Однако данный пример не является типичным, так как введенная нами функция не возвращает ни одно значение, а только печатает текст.

Пример 2: задание функции, определяющей знак числа.

Пример

```
gap> sign:= function(n)
>   if n < 0 then
```

```

>         return -1;
>     elif n = 0 then
>         return 0;
>     else
>         return 1;
>     fi;
> end;
function( n ) ... end
gap> sign(0); sign(-99); sign(11);
0
-1
1

```

Пример 3: Числа Фибоначчи определяются рекурсивно:  $f(1) = f(2) = 1$ ,  $f(n) = f(n-1) + f(n-2)$ .

Так как функция в GAP может обращаться сама к себе, то функция для вычисления  $n$ -го числа Фибоначчи может быть задана следующим образом:

Пример

```

gap> fib:= function(n)
>     if n in [1, 2] then
>         return 1;
>     else
>         return fib(n-1) + fib(n-2);
>     fi;
> end;
function( n ) ... end
gap> fib(15);
610

```

Упражнение: Добавить к данной функции проверку того, что  $n$  является натуральным числом.

Пример 4: Функция gcd, вычисляющая наибольший общий делитель двух целых чисел по алгоритму Евклида, требует создания локальных переменных в дополнение к формальным параметрам. Описание локальных переменных, если они есть, должно предшествовать всем операторам, входящим в определение функции.

Пример

```

gap> gcd:= function(a, b)
>     local c;
>     while b <> 0 do
>         c:= b;
>         b:= a mod b;
>         a:= c;
>     od;
>     return c;
> end;
function( a, b ) ... end
gap> gcd(30, 63);

```

3

Пример 5: Составим функцию, которая определяет количество разложений натурального числа (разложением данного числа называется невозрастающая последовательность натуральных чисел, сумма которых равна данному числу). Все множество разложений для данного числа  $n$  может быть разделено на подмножества в зависимости от максимального элемента разложения. Тогда количество разложений для  $n$  равняется сумме по всем возможным  $i$  количества разложений для  $n-i$ , элементы которых меньше, чем  $i$ . Обобщая это, получаем, что количество разложений числа  $n$ , элементы которых меньше, чем  $m$ , является суммой (по  $i < m, n$ ) количества разложений для  $n-i$  с элементами, меньшими, чем  $i$ . Отсюда получаем следующую функцию:

Пример

```
gap> nrparts:= function(n)
>   local np;
>   np:= function(n, m)
>     local i, res;
>     if n = 0 then
>       return 1;
>     fi;
>     res:= 0;
>     for i in [1..Minimum(n,m)] do
>       res:= res + np(n-i, i);
>     od;
>     return res;
>   end;
>   return np(n,n);
> end;
function( n ) ... end
```

Желая составить функцию, которая имеет один аргумент, мы решили поставленную задачу с помощью рекурсивной процедуры с двумя аргументами. Поэтому понадобилось фактически ввести две функции. Единственной задачей одной из них является вызов другой с двумя равными аргументами.

При этом функция `np` является локальной по отношению к `nrparts`. Она могла бы быть определена и независимо, но тогда идентификатор `np` уже не мог бы быть использован для других целей, а если бы это все-таки произошло, функция `nrparts` не могла бы обратиться к функции `np`.

Теперь рассмотрим функцию `np`, имеющую две локальные переменные `res` и `i`. Переменная `res` используется для суммирования, `i` - параметр цикла. Внутри цикла опять происходит обращение к функции `np`, но уже с другими аргументами. Однако для быстрого действия программы предпочтительнее избегать рекурсивных процедур, если только можно без них обойтись.

Упражнение: Предложить решение последней задачи, не используя рекурсивные процедуры.

## Глава 4

# Операции над группами и их элементами

### 4.1 Задание группы подстановок

Данный раздел посвящен операциям над группами и их элементами. Приведенные ниже примеры используют группы подстановок, но большинство используемых в них функций (в т.ч. `Group`, `Size`, `SylowSubgroup`) применяется и к другим используемым в GAP видам групп, для каждого из которых вычисления производятся по специальному алгоритму.

Зададим группу подстановок, которая порождается (записанными в виде произведения независимых циклов) подстановками  $(12)$  и  $(12345678)$ . Эта группа есть не что иное, как симметрическая группа  $S_8$ :

Пример

```
gap> s8:= Group( (1,2), (1,2,3,4,5,6,7,8) );
Group([ (1,2), (1,2,3,4,5,6,7,8) ])
```

### 4.2 Задание подгруппы группы подстановок

Группа  $S_8$  содержит знакопеременную группу  $A_8$ , которая может быть задана как подгруппа, состоящая из четных подстановок, или как ее коммутант:

Пример

```
gap> a8 := DerivedSubgroup( s8 );
Group([ (1,2,3), (2,3,4), (3,4,5), (2,6)(3,4), (3,7)(4,5), (3,5,6,7)(4,8) ])
```

Если обращение к объекту происходит часто, удобно присвоить ему имя. В этом случае при следующих обращениях к объекту вместо его представления будет выводиться на печать его имя:

Пример

```
gap> SetName(a8, "A8");
gap> a8;
```

A8

Несмотря на отсутствие связи между именем и идентификатором, их желательно выбирать согласованно.

### 4.3 Простейшие свойства группы. Силовские подгруппы

Изучим группу `a8`. Она является объектом, список известных свойств и атрибутов которого можно получить следующим образом:

Пример

```
gap> KnownPropertiesOfObject(a8);
[ "IsFinite", "CanEasilyCompareElements", "CanEasilySortElements",
  "IsDuplicateFree", "IsGeneratorsOfMagmaWithInverses", "IsAssociative",
  "IsSimpleSemigroup", "IsFinitelyGeneratedGroup",
  "IsSubsetLocallyFiniteGroup", "KnowsHowToDecompose" ]
gap> KnownAttributesOfObject(a8);
[ "Name", "OneImmutable", "ParentAttr", "LargestMovedPoint",
  "GeneratorsOfMagmaWithInverses", "MultiplicativeNeutralElement",
  "StabChainMutable", "StabChainOptions" ]
```

Этот список может расширяться в процессе работы с объектом, так как многие функции сохраняют информацию о нем в новых атрибутах и свойствах, что позволяет эффективно избегать повторных вычислений. Например, найдем порядок группы. Видно, что скорость его повторного определения существенно выше, так как вместо его вычисления просто выводится на печать его уже сохраненное значение.

Пример

```
gap> Size( a8 );
20160
gap> time;
6
gap> Size( a8 );
20160
gap> time;
6
```

Теперь проверим, является ли группа абелевой и совершенной:

Пример

```
gap> IsAbelian( a8 ); IsPerfect( a8 );
false
true
```

Сравним теперь список известных свойств и атрибутов с первоначальным и увидим, что добавились к ним новые:

Пример

```
gap> KnownPropertiesOfObject(a8);
[ "IsEmpty", "IsTrivial", "IsNonTrivial", "IsFinite",
  "CanEasilyCompareElements", "CanEasilySortElements", "IsDuplicateFree",
  "IsGeneratorsOfMagmaWithInverses", "IsAssociative", "IsCommutative",
  "IsSimpleSemigroup", "IsFinitelyGeneratedGroup",
  "IsSubsetLocallyFiniteGroup", "KnowsHowToDecompose", "IsPerfectGroup" ]
gap> KnownAttributesOfObject(a8);
[ "Name", "Size", "OneImmutable", "ParentAttr", "LargestMovedPoint",
  "GeneratorsOfMagmaWithInverses", "TrivialSubmagmaWithOne",
  "MultiplicativeNeutralElement", "DerivedSubgroup", "StabChainMutable",
  "StabChainOptions" ]
```

Теперь получим список простых делителей порядка группы:

Пример

```
gap> Set( Factors( Size( a8 ) ) );
[ 2, 3, 5, 7 ]
```

Для каждого из простых делителей  $p$  вычислим силовскую  $p$ -подгруппу и напечатаем ее порядок:

Пример

```
gap> for p in last do
> Print(p, " : ", Size(SylowSubgroup(a8,p)), "\n");
> od;
2 : 64
3 : 9
5 : 5
7 : 7
```

Исследуем силовскую 2-подгруппу. Обозначим ее `sy12`:

Пример

```
gap> sy12:=SylowSubgroup(a8,2);
Group([ (1,5)(4,8), (1,5)(2,7), (3,6)(4,8), (2,3)(6,7), (1,4)(2,3)(5,8)(6,7),
  (1,2)(3,4)(5,7)(6,8) ])
```

Теперь вычислим ее нормализатор в `a8`:

Пример

```
gap> Normalizer( a8, sy12 );
Group([ (1,5)(4,8), (1,5)(3,6), (2,7)(4,8), (2,3)(6,7), (1,4)(2,3)(5,8)(6,7),
  (1,6)(2,8)(3,5)(4,7) ])
```

Проверим, что он совпадает с ней самой:

Пример

```
gap> last = syl2;
true
```

Вычислим центр подгруппы `syl2`:

Пример

```
gap> Centre(syl2);
Group([ (1,5)(2,7)(3,6)(4,8) ])
```

Найдем централизатор `cent` последней подгруппы в `a8`, т.е. подгруппу элементов `a8`, перестановочных с каждым элементом центра группы `syl2`:

Пример

```
gap> cent:= Centralizer( a8, last );
Group([ (1,5)(2,7)(3,6)(4,8), (3,4)(6,8), (3,6)(4,8), (2,3)(6,7), (1,2)(5,7)
])
```

Найдем его порядок:

Пример

```
gap> Size( cent );
192
```

Вычислим ряд коммутантов `cent`:

Пример

```
gap> DerivedSeries(cent);
[ Group([ (1,5)(2,7)(3,6)(4,8), (3,4)(6,8), (3,6)(4,8), (2,3)(6,7),
(1,2)(5,7) ]), Group([ (1,6,7)(2,5,3), (2,3,8)(4,7,6), (1,5)(3,6) ]),
Group([ (1,7)(2,5)(3,8)(4,6), (1,5)(2,7), (3,6)(4,8), (1,3)(2,8)(4,7)(5,6),
(1,3)(2,4)(5,6)(7,8) ]), Group([ (1,5)(2,7)(3,6)(4,8) ]), Group(()) ]
```

Последний элемент полученного списка - тривиальная подгруппа, поэтому `cent` - разрешимая группа. Порядки подгрупп, входящих в ряд коммутантов (производный ряд) удобно получить следующим образом:

Пример

```
gap> List( last, Size );
[ 192, 96, 32, 2, 1 ]
```

Вычислим теперь нижний центральный ряд группы `cent`:

Пример

```
gap> LowerCentralSeries(cent);
[ Group([ (1,5)(2,7)(3,6)(4,8), (3,4)(6,8), (3,6)(4,8), (2,3)(6,7),
          (1,2)(5,7) ]), Group([ (1,6,7)(2,5,3), (2,3,8)(4,7,6), (1,5)(3,6) ]) ]
```

## 4.4 Другие виды подгрупп

Теперь покажем, как найти стабилизатор некоторого элемента множества, на котором действует группа подстановок. Как видно из следующего примера, стабилизатором единицы является подгруппа порядка 2520 и индекса 8, порожденная пятью подстановками:

Пример

```
gap> stab:= Stabilizer( a8, 1 );
Group([ (2,3,4), (3,4,5), (2,6)(3,4), (3,7)(4,5), (3,5,6,7)(4,8) ])
gap> Size(stab);
2520
gap> Index(a8,stab);
8
```

С помощью функции `Random` получим случайный элемент из `a8`:

Пример

```
gap> x:=Random( a8 );
(1,7,6,5)(4,8)
```

Новые подгруппы могут быть теперь получены путем поиска его централизатора, а затем комбинаций сопряжения и пересечения уже известных подгрупп.

Пример

```
gap> cent:=Centralizer(a8,x);
Group([ (1,7,6,5)(4,8), (2,3)(4,8) ])
gap> Size(cent);
8
gap> conj:= ConjugateSubgroup( cent, (2,3,4) );
Group([ (1,7,6,5)(2,8), (2,8)(3,4) ])
gap> inter:= Intersection( cent, conj );
Group([ (1,6)(5,7) ])
```

В следующем примере мы вычислим подгруппу группы `a8`, затем ее нормализатор и в итоге определим структуру факторгруппы. Сначала создадим элементарную абелеву подгруппу порядка 8:

Пример

```
gap> elab := Group( (1,2)(3,4)(5,6)(7,8), (1,3)(2,4)(5,7)(6,8),
> (1,5)(2,6)(3,7)(4,8) );;
```

```
gap> Size( elab );
8
gap> IsElementaryAbelian( elab );
true
```

Теперь присвоим ей имя и вычислим ее нормализатор:

Пример

```
gap> SetName( elab, "2^3" ); elab;
2^3
gap> norm := Normalizer( a8, elab );; Size( norm );
1344
```

## 4.5 Факторгруппы

Теперь мы имеем подгруппу `norm` порядка 1344 и ее подгруппу `elab`, и желаем построить факторгруппу. Поскольку мы также жеедем найти прообразы элементов факторгруппы в `norm`, нам также понадобится естественный гомоморфизм из `norm` в факторгруппу с ядром `elab`.

Пример

```
gap> hom := NaturalHomomorphismByNormalSubgroup( norm, elab );
<action epimorphism>
gap> f := Image( hom );
Group([ (), (), (), (4,5)(6,7), (4,6)(5,7), (2,3)(6,7), (2,4)(3,5),
(1,2)(5,6) ])
gap> Size( f );
168
```

Полученная факторгруппа `f` также является группой подстановок. Однако множество, на котором она действует, не имеет ничего общего с множеством точек, на котором действует `norm` (это просто совпадение, что оба множества являются подмножествами множества натуральных чисел). Теперь мы можем найти образы и прообразы относительно естественного гомоморфизма. Множество прообразов элемента является смежным классом по подгруппе `elab`. Мы используем функцию `PreImages`, так как `hom` не является взаимно однозначным отображением.

Пример

```
gap> ker:= Kernel( hom );
2^3
gap> x := (1,8,3,5,7,6,2);; Image( hom, x );
(1,7,5,6,2,3,4)
gap> coset := PreImages( hom, last );
RightCoset(2^3,(2,8,6,7,3,4,5))
```

Заметьте, что GAP может выбирать любой из представителей смежного класса прообразов. Конечно же, частное двух представителей одного смежного класса лежит в ядре гомоморфизма:

Пример

```
gap> rep:= Representative( coset );
(2,8,6,7,3,4,5)
gap> x * rep^-1 in ker;
true
```

Факторгруппа  $f$  является простой группой, т.е. она не имеет нетривиальных нормальных подгрупп:

Пример

```
gap> IsSimple( f );
true
```

Группа `norm` действует на 8 элементах своей нормальной подгруппы `elab` с помощью сопряжения, что приводит к представлению группы  $f$  в  $S_8$ , оставляющему неподвижной только лишь точку 1. Образ этого представления может быть вычислен с помощью функции `Action`; более того, он даже содежится в группе `norm`, и мы можем показать, что `norm` в действительности является расщепимым расширением элементарной абелевой группы `elab` с помощью группы  $f$ .

Пример

```
gap> op := Action( norm, elab );
Group([ (), (), (), (5,6)(7,8), (5,7)(6,8), (3,4)(7,8), (3,5)(4,6),
(2,3)(6,7) ])
gap> IsSubgroup( a8, op ); IsSubgroup( norm, op );
true
true
gap> IsTrivial( Intersection( elab, op ) );
true
```

Примечание. Нельзя использовать знак “<” вместо `IsSubgroup`. Так, не приводит к ошибке команда:

Пример

```
gap> elab < a8;
false
```

Оператор же равенства “=” фактически проверяет равенство групп.

## 4.6 Классы сопряженных элементов

Другим источником информации о группе `a8` будет являться ее разбиение на классы сопряженных элементов. Получим список классов сопряженности:

Пример

```

gap> ccl:=ConjugacyClasses(a8);
[ ()^G, (1,2)(3,4)^G, (1,2)(3,4)(5,6)(7,8)^G, (1,2,3)^G, (1,2,3)(4,5)(6,7)^G,
  (1,2,3)(4,5,6)^G, (1,2,3,4)(5,6)^G, (1,2,3,4)(5,6,7,8)^G, (1,2,3,4,5)^G,
  (1,2,3,4,5)(6,7,8)^G, (1,2,3,4,5)(6,8,7)^G, (1,2,3,4,5,6)(7,8)^G,
  (1,2,3,4,5,6,7)^G, (1,2,3,4,5,6,8)^G ]
gap> Length(last);
14

```

Теперь определим порядки представителей классов сопряженности, взяв в каждом классе по одному представителю:

Пример

```

gap> reps:= List( ccl, Representative );
[ (), (1,2)(3,4), (1,2)(3,4)(5,6)(7,8), (1,2,3), (1,2,3)(4,5)(6,7),
  (1,2,3)(4,5,6), (1,2,3,4)(5,6), (1,2,3,4)(5,6,7,8), (1,2,3,4,5),
  (1,2,3,4,5)(6,7,8), (1,2,3,4,5)(6,8,7), (1,2,3,4,5,6)(7,8), (1,2,3,4,5,6,7),
  (1,2,3,4,5,6,8) ]
gap> List( reps, r -> Order( r ) );
[ 1, 2, 2, 3, 6, 3, 4, 4, 5, 15, 6, 7, 7 ]

```

Определим, сколько элементов содержится в каждом классе:

Пример

```

gap> List(ccl,Size);
[ 1, 210, 105, 112, 1680, 1120, 2520, 1260, 1344, 1344, 1344, 3360, 2880,
  2880 ]

```

Примечание: следует различать функции `Order` (порядок элемента), `Size` (порядок группы, класса сопряженности и т.п.) и `Length` (длина списка). Построив классы сопряженных элементов, мы можем рассматривать их функции, т.е. отображения, принимающие одинаковые значения на всем классе сопряженных элементов. Примером может являться число неподвижных точек:

Пример

```

gap> nrfixedpoints:= function( perm, support )
> return Number( [1 .. support], x -> x^perm = x);
> end;
function( perm, support ) ... end

```

Вычислим его для группы `a8`:

Пример

```

gap> permchar1:= List(reps, x->nrfixedpoints(x,8));
[ 8, 4, 0, 5, 1, 2, 2, 0, 3, 0, 0, 0, 1, 1 ]

```

## Приложение А

# Рекомендации по созданию и запуску программ в системе GAP

GAP позволяет не только производить вычисления в интерактивном режиме, но и сохранять программы для дальнейшего их применения. Программы создаются и сохраняются в формате текстовых файлов (которым обычно присваивают расширение “g”), и редактируются с помощью любого текстового редактора.

Как правило, программа состоит из главной части и набора функций. При чтении программы (с помощью команды **Read**) команды, содержащиеся в главной части, выполняются непосредственно, а функции только лишь задаются, и делают возможным последующее обращение к ним. В случае синтаксических ошибок при чтении файла с программой будут выданы соответствующие сообщения.

При разработке программ удобно одновременно запускать:

- GAP
- текстовый редактор, в котором открыта разрабатываемая программа;
- средство просмотра HTML-файлов (например, Firefox или Internet Explorer) для чтения описания GAP в гипертекстовом формате (начальный файл - GAP\HTM\Index.htm).

Кроме того, если требуется оформить в виде программы последовательность команд, которая была введена в ходе работы с системой в диалоговом режиме, то это удобно сделать, редактируя файл протокола, который для этого должен быть уже открыт на момент начала ввода этих команд, с помощью команды вида `LogTo("file.log");`.

Пример: составить программу, которая определяет, является ли группа  $G$  конечной  $p$ -группой для некоторого  $p$ , и возвращает список, первый элемент которого - **true** или **false** в зависимости от результата проверки, а второй - соответствующее значение  $p$ , если  $G$  -  $p$ -группа, и **false** - иначе.

1. Создаем с помощью текстового редактора файл `prog.g` следующего содержания:

```
Print(" Loading IsFinitePGroup()", "\n");
IsFinitePGroup:=function(G)
local divisors; # список простых делителей
if IsFinite(G)=false then
```

```

return [false, false];
else
divisors:=Set(Factors(Size(G)));
if Length(divisors)=1 then
return [true, divisors[1] ];
else
return [false, false];
fi;
fi;
end;

```

2. Сохраняем этот файл в каталоге, выбранном с учетом рекомендаций пункта 1.3.
3. Запустим GAP и определим файл протокола `log.txt`:

Пример

```
gap> LogTo("log.txt");
```

Теперь зададим группу диэдра порядка 8:

Пример

```
gap> G:=DihedralGroup(8);
<pc group of size 8 with 3 generators>
```

Попробуем обратиться к функции из файла `prog.g`:

Пример

```
gap> IsFinitePGroup(G);
Error, Variable: 'IsFinitePGroup' must have a value
```

Ошибка вызвана тем, что для использования функции этот файл сначала необходимо прочитать. При этом, если он содержит синтаксические ошибки, то будут выданы сообщения о них. Чтение производится командой `Read`:

Пример

```
gap> Read("prog.g");
Loading IsFinitePGroup()
```

Ошибки обнаружены не были. Было выдано сообщение, включенное для удобства в файл `prog.g`.

Теперь проверим работу программы для группы диэдра порядка 8, а также для симметрической группы  $S_3$ .

Пример

```
gap> IsFinitePGroup(G);
[ true, 2 ]
gap> H:=SymmetricGroup(3);
```

```
Sym( [ 1 .. 8 ] )
gap> IsFinitePGroup(H);
[ false, false ]
```

Очевидно, что программа работает корректно.

Заметим, что в GAP имеются стандартные функции `IsPGroup`, определяющая, является ли группа  $G$  конечной  $p$ -группой для некоторого  $p$ , и `PrimePGroup`, возвращающая соответствующее значение  $p$ .

## Приложение В

# Некоторые функции системы GAP для работы с группами

$g*h$	произведение элементов $g$ и $h$
$g/h$	произведение элементов $g$ и $h^{-1}$
$g^h$	вычисление $h^{-1}gh$ ( $g, h$ - элементы группы)
$g^i$	вычисление $i$ -й степени элемента $g$ ( $i$ - целое)
$list*g$	умножение списка $list$ на элемент $g$ справа
$g*list$	умножение списка $list$ на элемент $g$ слева
$list/g$	умножение списка $list$ на элемент $g^{-1}$ справа
$Comm( g, h )$	коммутатор $g^{-1}h^{-1}gh$
$IsGroup( obj )$	проверка, является ли $obj$ группой
$Order( g )$	порядок элемента $g$
$Subgroup( G, gens )$	подгруппа группы $G$ , порожденная списком элементов $gens$
$AsSubgroup( G, U )$	подгруппа группы $G$ , порожденная порождающими элементами ранее независимо созданной группы $U$ (если они лежат в $G$ )
$Agemo( G, p )$	подгруппа, порожденная $p$ -ми степенями элементов $p$ -группы $G$
$Centralizer( G, x )$	централизатор элемента $x$ в группе $G$
$Centralizer( G, U )$	централизатор подгруппы $U$ в группе $G$
$Centre( G )$	центр группы $G$
$ClosureGroup( U, g )$	подгруппа, порожденная подгруппой $U$ и элементом $g$
$ClosureGroup( U, S )$	подгруппа, порожденная подгруппами $U$ и $S$
$CommutatorSubgroup(G,H)$	коммутатор подгрупп $G$ и $H$
$ConjugateSubgroup(U,g)$	подгруппа, сопряженная с подгруппой $U$ с помощью элемента $g$
$DerivedSubgroup( G )$	коммутант группы $G$
$FittingSubgroup( G )$	подгруппа Фиттинга группы $G$
$FrattniSubgroup( G )$	подгруппа Фраттини группы $G$

Normalizer( S, U )	нормализатор подгруппы $U$ в подгруппе $S$
SylowSubgroup( G, p )	Силовская $p$ -подгруппа конечной группы $G$
TrivialSubgroup( U )	тривиальная подгруппа группы $U$
FactorGroup( G, N )	факторгруппа группы $G$ по нормальной подгруппе $N$ (то же, что $G/N$ )
CommutatorFactorGroup(G)	факторгруппа группы $G$ по ее коммутанту
DerivedSeries( G )	ряд коммутантов группы $G$
LowerCentralSeries( G )	нижний центральный ряд группы $G$
UpperCentralSeries( G )	верхний центральный ряд группы $G$
AbelianInvariants( G )	инварианты абелевой группы $G$ (если $G$ - неабелева - инварианты факторгруппы группы $G$ по ее коммутанту)
Exponent( G )	показатель (экспонента) группы $G$
Index( G, U )	индекс подгруппы $U$ в группе $G$
IsAbelian( G )	проверка, является ли группа $G$ абелевой
IsCyclic( G )	проверка, является ли группа $G$ циклической
IsNilpotent( G )	проверка, является ли группа $G$ нильпотентной
IsElementaryAbelian( G )	проверка, является ли $G$ элементарной абелевой
IsConjugate( G, x, y )	проверка, сопряжены ли элементы $x$ и $y$ в группе $G$
IsNormal( G, U )	проверка, нормальна ли подгруппа $U$ в группе $G$
IsSimple( G )	проверка, является ли группа $G$ простой
IsSolvable( G )	проверка, является ли группа $G$ разрешимой
IsSubgroup( G, U )	проверка, является ли $U$ подгруппой группы $G$
IdGroup( G )	идентификация группы $G$
ConjugacyClasses( G )	классы сопряженных элементов группы $G$
ConjugacyClass( G, g )	класс сопряженности, содержащий элемент $g$
NormalSubgroups( G )	список нормальных подгрупп группы $G$
AsList( G )	список элементов группы $G$

## Приложение С

# Лабораторные работы

### С.1 Основы работы с системой GAP в MS Windows

#### С.1.1 Инструкции по выполнению работы

В данной лабораторной работе необходимо последовательно выполнить все этапы.

1. Найдите каталог `gap4r4`, в котором инсталлирована система GAP на локальном или сетевом диске (например, с помощью FAR или Проводника). Если инсталляция системы GAP не выполнена, то Вы можете произвести ее самостоятельно в соответствии с разделами “Инсталляция” или “GAP для Windows” сайта Украинской группы пользователей GAP. Для учебных целей намного быстрее можно инсталлировать мини-дистрибутив из раздела “Мини-тест” этого же сайта. Адрес сайта - <http://ukrgap.exponenta.ru/>.

2. Найдите в каталоге `gap4r4\bin` командные файлы `gap.bat` и `gaprxvt.bat`. Теперь Вы уже можете запускать систему GAP с их помощью. Запустите сначала файл `gap.bat` для работы в окне командной строки Windows (окне MS-DOS). После появления приглашения вида `gap>` введите команду `quit`; для выхода из системы. После этого запустите файл `gaprxvt.bat` для работы в окне оболочки RXVT. После завершения загрузки системы также выйдите из нее с помощью команды `quit`; (помните, что команды завершаются точкой с запятой, после чего необходимо нажать клавишу *Enter*).

3. Простейшие вычисления можно выполнять, запуская систему так, как указано в п.2. Однако, в этом случае при чтении и записи файлов нужно будет указывать полный путь к ним. Эффективнее будет создать рабочий каталог в том разделе диска, где Вы имеете соответствующие права доступа, и скопировать туда файлы `gap.bat` и `gaprxvt.bat`. Выполните эти инструкции, создав свой рабочий каталог (который можно назвать, например, `gap`) и ознакомьтесь с содержанием этих файлов (например, с помощью FAR или Блокнота). В дальнейшем Вы также сможете создать ярлыки для запуска этих файлов и поместить их в главное меню и на рабочий стол.

4. Теперь Вам нужно освоить работу с системой в обоих вариантах - как в окне MS-DOS, так и в окне RXVT. Для этого снова запустите систему, но теперь уже из только что созданного Вами рабочего каталога. Если производительность компьютера позволяет, Вы можете одновременно запустить оба файла `gap.bat` и `gaprxvt.bat`. В противном случае нижеследующие пункты нужно будет сначала выполнить в одном окне, а потом повторить в другом.

5. Выполните простейшие вычисления, введя следующие команды:

Пример

```

352/182;
2*(15+256)/17;
2^64;
2^20000 mod 100;
3 in [1,2,3];
2*2 >= 4;

```

Одна команда может занимать несколько строк, последняя из которых заканчивается точкой с запятой. Таким образом, если Вы забыли поставить точку с запятой в конце строки и уже нажали клавишу *Enter*, Вы можете поставить точку с запятой в следующей строке, а затем нажать *Enter* еще раз. Попробуйте ввести следующую многострочную команду:

Пример

```

155/4545+
1234*5678+
Factorial(100)+
Sum([1..100]);

```

Помните при этом, что в GAP имеет значение регистр текста. Например, следующая команда приводит к ошибке:

Пример

```

gap> factorial(100);
Variable: 'factorial' must have a value
gap>

```

При некоторых ошибках на экран выводится промежуточное приглашение системы вида *brk*>. Для выхода из него нужно ввести команду *quit*; (в этом случае она не приводит к завершению работы системы). Например:

Пример

```

gap> Factorial(1/2);
Range: <last> must be an integer less than 2^28 (not a rational) at
return Product( [ 1 .. n ] );
called from
<function>( <arguments> ) called from read-eval-loop
Entering break read-eval-print loop ...
you can 'quit;' to quit to outer loop, or
you can replace <last> via 'return <last>;' to continue
brk> quit;
gap>

```

6. Теперь нужно освоить работу с историей команд. Нажимайте клавиши перемещения курсора вверх и вниз для просмотра истории команд. Теперь наберите в командной строке цифру 2, а затем нажимайте те же клавиши управления курсором. При этом Вы будете видеть только те из ранее введенных команд, которые начинались с цифры 2.

7. Вы можете перемещаться по содержимому командной строки с помощью клавиш перемещения курсора влево и вправо, и можете удалять символы с помощью клавиш *Delete* и *Backspace*. Например, наберите в командной строке **F** и найдите в истории команд ранее введенную строку **Factorial(100)+**. Теперь переместитесь в конец строки и отредактируйте ее так, чтобы вычислить 500!.

Для быстрого перемещения в конец и начало строки можно также использовать клавиши *Home* и *End* в окне MS-DOS, и комбинации клавиш *Ctrl-A* и *Ctrl-B* как в окне MS-DOS, так и в окне RXVT (о других полезных при редактировании содержимого командной строки сочетаниях клавиш Вы можете прочитать в документации по системе, для чего введите команду `?options!command line`, а затем следуйте выведенным на экран инструкциям).

8. Научитесь выделять, копировать и вставлять текст. Попробуйте выделить в окне браузера (т.е. MS Internet Explorer, Netscape и т.п.) и скопировать в буфер обмена команды, приведенные выше, а затем перейти в окно GAP и вставить их в командную строку (в окне MS-DOS используйте стандартные способы вставки, а в окне RXVT используйте сочетание клавиш *Shift-Ins*). Затем попробуйте выделить и скопировать текст из окна GAP (в окне MS-DOS используйте стандартные средства, в окне RXVT выделяйте текст мышью, а для копирования используйте *Ctrl-Ins*) и вставить его в текстовый файл (редактируемый, например, с помощью FAR или Блокнота).

9. Если имеются, используйте полосы прокрутки для просмотра информации, которая в процессе работы с системой переместилась за верхний край экрана.

10. Одной из составных частей системы GAP является ее документация. С помощью Проводника откройте каталог `gap4r4/doc`. В нем Вы обнаружите подкаталог `htm`, в котором нужно открыть файл `index.htm` - это стартовый файл для просмотра документации в HTML-формате. В зависимости от выбранного варианта инсталляции, возможно также наличие каталога `htmie` - в нем та же документация, оптимизированная для просмотра с помощью MS Internet Explorer). Для быстрого обращения к документации создайте в своем рабочем каталоге ярлык, указывающий на файл `index.htm` в одном из каталогов `htm` или `htmie`, после чего откройте его с помощью данного ярлыка и ознакомьтесь с названиями пяти основных разделов документации. Перейдите в раздел "Индекс" и найдите с его помощью описание функций `Factorial` и `Sum`. Вы можете копировать приведенные в документации примеры и выполнять их в GAP так, как это было описано в п.8.

При полной инсталляции системы каталог `gap4r4/doc` также содержит документацию и в других форматах. В частности, он содержит другие подкаталоги, наименования которых соответствуют пяти основным разделам документации, в которых можно найти эти разделы в формате PDF, более удобном при печати документации (учтите, что центральный раздел документации - Reference Manual - занимает в формате PDF почти тысячу страниц!).

11. Альтернативным вариантом использования документации является подстрочная справка, которую можно вызвать прямо из командной строки GAP. Это удобно, если в дальнейшем не предвидится активное перемещение по гиперссылкам в документации, а также может быть полезно при удаленном подключении или в случае, когда ресурсы компьютера ограничены. Например, наберите в командной строке `?Factorial` (без точки с запятой) для отображения справки по данной функции.

Если ввести два знака вопроса, то производится полный поиск по документации и возвращается список всех вхождений данного термина в нее. Например, наберите `??Sum` для вывода списка всех имеющих отношение к запросу разделов, а затем наберите `?1`

для перехода к первому из предложенных разделов.

12. Удобным свойством системы является автодополнение имен переменных при их вводе. Это означает, что если по первым буквам введенного имени переменной (в т.ч. имени функции) можно однозначно определить его окончание, то при нажатии клавиши *Tab* это окончание будет добавлено автоматически. Если же однозначности нет, то при повторном нажатии клавиши *Tab* будет предложен список всех возможных вариантов. Например, наберите в командной строке **Fib** и нажмите *Tab*. Вычислите теперь 100-й элемент последовательности Фибоначчи, указав 100 в скобках в качестве аргумента. Затем наберите в командной строке **Factor** и нажмите *Tab* дважды для вывода на экран имен всех переменных, начинающихся с этого сочетания букв.

13. Историю работы с системой можно сохранить в текстовом файле (т.наз. файле протокола). Выберите одно из окон GAP, работа в котором Вам показалась более удобной - окно MS-DOS или окно RXVT. Введите команду

Пример

```
LogTo("logfile.txt");
```

После этого все введенные Вами команды и результаты их работы, отображаемые на экране, будут дублироваться в файле с именем `logfile.txt`, который содержится в Вашем рабочем каталоге.

Теперь выполните следующие вычисления: задайте сначала переменную **n**, в которой сохраните номер своего варианта, например:

Пример

```
n:=20;
```

Затем последовательно введите следующие команды:

Пример

```
a:=2^(n+1)-1;
IsPrime(a);
Factors(a);
x:=n+10;
Factors(Factorial(x));
Phi(x);
Sigma(x);
Tau(x);
```

Теперь закройте файл протокола с помощью команды

Пример

```
LogTo();
```

и просмотрите его с помощью, например, FAR или Проводника.

## С.2 Списки. Целые числа

### С.2.1 Инструкции по выполнению работы

Данная лабораторная работа предназначена для изучения приемов работы со списками на примере действий над целыми числами. Подробные сведения по данным темам содержатся:

- в разделе 3 данного методического пособия;
- в разделе “Целые числа, алгоритм Евклида и разложение на множители” учебных материалов к курсу алгебры и теории чисел (<http://ukrgap.exponenta.ru/Examples/integers.htm>);
- в разделе “Lists and Records” введения в систему GAP (<http://www.gap-system.org/Manuals/doc/htm/tut/chapters.htm>);
- в разделах “Integers”, “Number theory” и “Lists” справочного руководства по системе GAP (<http://www.gap-system.org/Manuals/doc/htm/ref/chapters.htm>).

Задания для лабораторной работы взяты из сборника задач [1], и после номера варианта указан номер соответствующей задачи в [1]. Студентам рекомендуется ознакомиться с приведенным в [1] решением данных задач, чтобы увидеть, что теоретическое их решение более эффективно, чем простой перебор, выполненный ими в учебных целях в данной лабораторной работе.

В зависимости от конкретной задачи, при выполнении работы полезными могут оказаться следующие функции и операции (детальное их описание см. в документации):

- `Collected(list)` возвращает новый список `newlist`, который для каждого элемента `x` исходного списка `list` содержит соответствующий ему список из двух элементов, первый из которых является самим элементом `x`, а второй показывает кратность его вхождения в список `list`.

Пример

```
gap> Collected([ 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 5, 5, 7 ]);
[ [ 2, 8 ], [ 3, 4 ], [ 5, 2 ], [ 7, 1 ] ]
```

- `Combinations( list [, k ] )` возвращает множество всевозможных комбинаций (неупорядоченных наборов без повторов), составленных из `k` элементов списка `list` (который может даже содержать одинаковые элементы несколько раз). Если `k` не указано, возвращаются все возможные комбинации, составленные из элементов списка `list`.

Пример

```
gap> Combinations( [1,2,2,3] );
[ [ ], [ 1 ], [ 1, 2 ], [ 1, 2, 2 ], [ 1, 2, 2, 3 ], [ 1, 2, 3 ], [ 1, 3 ],
  [ 2 ], [ 2, 2 ], [ 2, 2, 3 ], [ 2, 3 ], [ 3 ] ]
```

- `DivisorsInt(n)` возвращает список натуральных делителей целого числа `n`.

- `FactorsInt(n)` возвращает разложение целого числа  $n$  на простые множители в виде их списка.
- `Filtered(list, x->f(x))` возвращает список тех элементов  $x$  из списка  $list$ , для которых выполняется условие  $f(x)=true$ .
- `ForAll(list, x->f(x))` проверяет, что для каждого элемента  $x$  из списка  $list$  выполняется условие  $f(x)=true$ .
- `ForAny(list, x->f(x))` проверяет, что существует хотя бы один элемент  $x$  из списка  $list$ , для которого выполняется условие  $f(x)=true$ .
- `Gcd(list)` или `Gcd(a1,a2,...,aN)` вычисляет наибольший общий делитель целых чисел  $a_1, a_2, \dots$  или целых чисел из списка  $list$ .
- `Length(list)` определяет длину списка  $list$ .
- `a mod b` возвращает остаток от деления  $a$  на  $b$ .
- `Phi(n)` вычисляет функцию Эйлера  $\phi(n)$ .
- `Product(list)` вычисляет произведение всех элементов списка  $list$
- `Sum(list)` вычисляет сумму всех элементов списка  $list$

*Пример* (ГТ 77): Сколько чисел в интервале от 1 до 120 делится на одно и только одно из чисел 2, 3 или 5 ?

Сначала мы получим список этих чисел, а затем определим их количество:

Пример

```
gap> l := Filtered( [ 1 .. 120 ], i ->
>                 Length( Filtered( [2,3,5], j -> i mod j = 0 ) ) = 1 );
[ 2, 3, 4, 5, 8, 9, 14, 16, 21, 22, 25, 26, 27, 28, 32, 33, 34, 35, 38, 39,
  44, 46, 51, 52, 55, 56, 57, 58, 62, 63, 64, 65, 68, 69, 74, 76, 81, 82, 85,
  86, 87, 88, 92, 93, 94, 95, 98, 99, 104, 106, 111, 112, 115, 116, 117, 118 ]
gap> Length(l);
56
```

Этот же результат можно было бы получить и с помощью одной команды:

Пример

```
gap> Length( Filtered( [ 1 .. 120 ], i ->
>                 Length( Filtered( [2,3,5], j -> i mod j = 0 ) ) = 1 ) );
56
```

## С.2.2 Задания для лабораторных работ

*Вариант 1* (ГТ 65). Найти показатель степени числа 3 в каноническом разложении числа 100!.

*Вариант* (ГТ 66). Найти показатель степени числа 11 в каноническом разложении числа  $1000!$ .

*Вариант 3* (ГТ 67). Сколькими нулями оканчивается число  $100!$  (предложить два способа решения данной задачи, не сводящихся к выводу  $100!$  на экран и ручному подсчету нулей) ?

*Вариант 4* (ГТ 68). Разложить на простые множители числа  $10!$ ,  $15!$ ,  $20!$ ,  $25!$ ,  $30!$ .

*Вариант 5* (ГТ 69). Найти количество целых положительных чисел, не превосходящих 180 и не делящихся ни на одно из простых чисел 5, 7, 11.

*Вариант 6* (ГТ 70). Найти количество целых положительных чисел, не превосходящих 2311 и не делящихся ни на одно из простых чисел 5, 7, 11, 13.

*Вариант 7* (ГТ 71). Найти количество целых положительных чисел, не превосходящих 100 и взаимно простых с 36.

*Вариант 8* (ГТ 72). Найти количество целых положительных чисел, не превосходящих 12317 и взаимно простых с 1575.

*Вариант 9* (ГТ 73). Найти количество целых положительных чисел, не превосходящих 1000 и не взаимно простых с 363.

*Вариант 10* (ГТ 74). В ряду натуральных чисел 1, ... , 1800, начиная с единицы, вычеркивается каждое пятое, каждое восьмое и каждое девятое число. Сколько чисел не будет вычеркнуто?

*Вариант 11* (ГТ 78). В урне 5000 шаров, перенумерованных от 1 до 5000. Какова вероятность того, что вынутый наудачу шар будет иметь номер, кратный какому-нибудь из чисел 14, 21, 10 ?

*Вариант 12* (ГТ 84). Найти все делители чисел 360, 375, 957, 988 (предложить два способа решения, сравнить результаты).

*Вариант 13* (ГТ 85). Найти целое положительное число, зная, что оно имеет только два простых делителя, число всех делителей равно 6, а сумма всех делителей равна 28.

*Вариант 14* (ГТ 91). Найти целое положительное число, произведение всех делителей которого равно 5832.

*Вариант 15* (ГТ 101). Сколько чисел в интервале от 1 до 120, не взаимно простых с 30 (предложить два способа решения, сравнить результаты)?

*Вариант 16* (ГТ 113). Найти количество натуральных чисел, меньших числа 300 и имеющих с ним наибольшим общим делителем число 20.

*Вариант 17* (ГТ 114). Найти количество натуральных чисел, меньших числа 1665 и имеющих с ним наибольшим общим делителем число 37.

*Вариант 18* (ГТ 115). Найти количество натуральных чисел, меньших числа 1476 и имеющих с ним наибольшим общим делителем число 41.

## С.3 Линейные программы. Векторы и матрицы

### С.3.1 Инструкции по выполнению работы

Данная лабораторная работа предназначена для обучения разработке простейших линейных программ, оформленных в виде функций, на примере работы с векторами и матрицами. Подробные сведения по данным темам содержатся:

- в разделе 3 данного методического пособия (о матрицах и векторах);
- в приложении А (о создании и запуске программ);

- в разделе “Lists and Records” введения в систему GAP (<http://www.gap-system.org/Manuals/doc/htm/tut/chapters.htm>);
- в разделах “Row vectors” и “Matrices” справочного руководства по системе GAP (<http://www.gap-system.org/Manuals/doc/htm/ref/chapters.htm>).

*Пример:* Разработать функцию для вычисления суммы элементов побочной диагонали матрицы  $A$  третьего порядка, входным параметром которой является матрица  $A$ , а выходным - указанная сумма.

Для этого создадим в рабочем каталоге (созданном в соответствии с инструкциями лабораторной работы С.1) файл `lab3.g` следующего содержания:

```
SumDiag:=function(A)
local s;
s := A[1][3] + A[2][2] + A[3][1];
return s;
end;
```

Теперь прочитаем его с помощью следующей команды (если система GAP запущена из другого каталога, то нужно будет указать полный путь к файлу):

Пример

```
Read("lab3.g");
```

Теперь мы можем обращаться к функции `SumDiag`, например:

Пример

```
gap> M:=[[1,2,3],[-1,-2,-3],[1,1,1]];
[ [ 1, 2, 3 ], [ -1, -2, -3 ], [ 1, 1, 1 ] ]
gap> SumDiag(M);
2
```

Данную функцию можно было бы разработать и более универсальным образом для работы с матрицами любого порядка (ее доработку для проверки того, что матрица является квадратной, мы оставляем читателю):

```
BetterSumDiag:=function(A)
local i, nrows, ncols;
nrows := Length( A ); # количество строк
ncols := Length( A[1] ); # количество столбцов
return Sum( List( [ 1 .. nrows ], i -> A[i][ncols-i+1] ) );
end;
```

При разработке функций обратите внимание на следующее:

- отсутствие точки с запятой в первой строке
- обязательность точки с запятой после команды `end`
- необходимость указания локальных переменных с помощью команды `local`
- сообщение об ошибке, если имя Вашей функции совпадет с именем одной из библиотечных функций GAP

### С.3.2 Задания для лабораторных работ

*Вариант 1.* Разработать функцию для вычисления объема треугольной пирамиды, заданной координатами своих вершин, с помощью смешанного произведения векторов.

*Вариант 2.* Разработать функцию для вычисления главных миноров определителя матрицы  $A$  третьего порядка, входным параметром которой является матрица  $A$ , а выходным - список ее главных миноров.

*Вариант 3.* Разработать функцию для вычисления определителя матрицы  $A$  второго порядка, входным параметром которой является матрица  $A$ , а выходным - ее определитель, вычисленный по формуле  $\det A = a_{11}a_{22} - a_{12}a_{21}$ .

*Вариант 4.* Разработать функцию для вычисления определителя матрицы  $A$  третьего порядка, входным параметром которой является матрица  $A$ , а выходным - ее определитель, вычисленный по правилу Саррюса.

*Вариант 5.* Разработать функцию для вычисления векторного произведения двух векторов, заданных своими координатами в ПДСК, входными параметрами которой являются два вектора  $u$  и  $v$ , а выходным - вектор, являющийся их векторным произведением.

*Вариант 6.* Разработать функцию для вычисления векторного произведения двух векторов, заданных координатами их начал и концов в ПДСК, входными параметрами которой являются координаты четырех точек пространства, соответствующих началу и концу векторов  $u$  и  $v$ , а выходным - вектор, являющийся их векторным произведением.

*Вариант 7.* Разработать функцию для вычисления произведения двух матриц второго порядка непосредственно по определению произведения матриц (т.е. не используя имеющуюся в GAP операцию умножения матриц), входными параметрами которой являются матрицы  $A$  и  $B$ , а выходным - матрица  $C = AB$ .

*Вариант 8.* Разработать функцию для транспонирования матрицы третьего порядка непосредственно по определению транспонирования матриц (т.е. не используя имеющуюся в GAP операцию транспонирования), входным параметром которой является матрица  $A$ , а выходным - матрица  $A^T$ .

*Вариант 9.* Разработать функцию для вычисления скалярного произведения двух векторов, заданных своими координатами в ПДСК (не используя имеющуюся в GAP операцию скалярного умножения векторов), входными параметрами которой являются два вектора  $u$  и  $v$ , а выходным - их скалярное произведение.

*Вариант 10.* Разработать функцию для вычисления скалярного произведения двух векторов, заданных координатами их начал и концов в ПДСК (не используя имеющуюся в GAP операцию скалярного умножения векторов), входными параметрами которой являются координаты четырех точек пространства, соответствующих началу и концу векторов  $u$  и  $v$ , а выходным - их скалярное произведение.

*Вариант 11.* Разработать функцию для вычисления перманента матрицы  $A$  второго порядка, входным параметром которой является матрица  $A$ , а выходным - ее перманент, равный  $a_{11}a_{22} + a_{12}a_{21}$ .

*Вариант 12.* Разработать функцию для вычисления перманента матрицы  $A$  третьего порядка, входным параметром которой является матрица  $A$ , а выходным - ее перманент, равный сумме шести слагаемых, каждое из которых является произведением трех элементов матрицы  $A$ , взятых по одному из каждой строки и каждого столбца.

*Вариант 13.* Разработать функцию для вычисления следа матрицы второго порядка, входным параметром которой является матрица  $A$ , а выходным ее след, равный сумме элементов ее главной диагонали.

*Вариант 14.* Разработать функцию для вычисления следа матрицы третьего порядка, входным параметром которой является матрица  $A$ , а выходным ее след, равный сумме элементов ее главной диагонали.

*Вариант 15.* Разработать функцию, входным параметром которой являются координаты двух точек плоскости  $A$  и  $B$ , а выходным - координаты середины отрезка  $AB$ .

*Вариант 16.* Разработать функцию, входным параметром которой являются координаты двух точек пространства  $A$  и  $B$ , а выходным - координаты середины отрезка  $AB$ .

*Вариант 17.* Разработать функцию для вычисления образа вектора  $u = (u_1, u_2, u_3)$  под действием линейного оператора  $f$ , заданного матрицей  $A$  (не используя имеющуюся в GAP операцию умножения матрицы на вектор), входными параметрами которой являются матрица  $A$  и координаты вектора  $u$ , а выходным - координаты вектора  $Au$ .

*Вариант 18.* Разработать функцию для вычисления Лиевского коммутатора  $[A, B]$  двух квадратных матриц  $A$  и  $B$  произвольного порядка  $n$ , входными параметрами которой являются матрицы  $A$  и  $B$ , а выходным - матрица  $D = AB - BA$ .

## С.4 Ветвящиеся программы. Многочлены

### С.4.1 Инструкции по выполнению работы

Данная лабораторная работа предназначена для изучения оператора условного перехода на примере работы с многочленами от одной переменной.

Подробные сведения по данным темам содержатся:

- в разделе 2 данного методического пособия;
- в разделе “Применение GAP для изучения теории многочленов” учебных материалов к курсу алгебры и теории чисел (<http://ukrgap.exponenta.ru/Examples/Polynoms.htm>);
- в разделе “Polynomials and Rational Functions” справочного руководства по системе GAP (<http://www.gap-system.org/Manuals/doc/htm/ref/chapters.htm>).

*Пример:* Разработать функцию, которая определяет, являются ли коэффициенты заданного многочлена взаимно простыми, и в случае положительного ответа возвращает исходный многочлен, а в случае отрицательного - исходный многочлен, разделенный на НОД своих коэффициентов.

Перед тем, как приступить к разработке функции, попробуем понять схему ее работы в диалоговом режиме. Зададим переменную  $x$  и многочлен  $f$  от этой переменной, затем

получим список коэффициентов многочлена  $f$ , найдем их наибольший общий делитель  $d$  и разделим на него многочлен  $f$  :

Пример

```
gap> x:=Indeterminate(Rationals,"x");
x
gap> f:=20*x^3+15*x^2+10*x+5;
20*x^3+15*x^2+10*x+5
gap> CoefficientsOfUnivariatePolynomial(f);
[ 5, 10, 15, 20 ]
gap> d:=Gcd(last);
5
gap> f/5;
4*x^3+3*x^2+2*x+1
```

Теперь в рабочем каталоге создадим текстовый файл следующего содержания (подробные инструкции по работе с файлами см. в лабораторной работе [С.3](#)):

```
CoeffCancellation := function( f )
local d;
d := Gcd( CoefficientsOfUnivariatePolynomial( f ) );
if d<>1 then
  return f/d;
else
  return f;
fi;
end;
```

После чтения данной программы (см. пример из лабораторной работы [С.3](#) мы можем обращаться к функции `CoeffCancellation`. Для проверки правильности работы обеих алгоритма сначала применим ее к ранее заданному многочлену  $f$ , а затем к многочлену  $g$ , коэффициенты которого уже являются взаимно простыми:

Пример

```
gap> CoeffCancellation(f);
4*x^3+3*x^2+2*x+1
gap> g:=x^3+x^2+10*x+5;
x^3+x^2+10*x+5
gap> CoeffCancellation(g);
x^3+x^2+10*x+5</C>
```

Очевидно, что программа работает корректно. При разработке функций обратите внимание на следующее:

- необходимость тестирования обеих ветвей алгоритма
- использование отступов при форматировании текста программы
- опциональность операторов `else` и `elif` в зависимости от конкретной задачи

### С.4.2 Задания для лабораторных работ

*Вариант 1.* Разработать функцию, которая определяет, имеет ли заданный многочлен  $f$  корни кратности выше первой, и в случае отрицательного ответа возвращает исходный многочлен, а в случае положительного - многочлен  $g$ , имеющий те же корни, что и исходный, но только первой кратности (при этом для нахождения многочлена  $g$  необходимо разделить многочлен  $f$  на наибольший общий делитель многочлена  $f$  и его производной  $f'$ ). *Примечание:* использовать функции `Gcd`, `Derivative`, `IsOne`.

*Вариант 2.* Разработать функцию, которая определяет, являются ли два заданных многочлена  $f$  и  $g$  взаимно простыми, и в случае положительного ответа возвращает их произведение, а в случае отрицательного - их произведение, разделенное на их наибольший общий делитель. *Примечание:* использовать функции `Gcd`, `IsOne`.

*Вариант 3.* Разработать функцию, которая определяет, имеет ли заданный многочлен  $f$  разные знаки на концах заданного интервала  $[a, b]$ , и в случае положительного ответа возвращает среднее арифметическое значений многочлена на концах интервала, а в случае отрицательного - наибольшее из значений многочлена на концах интервала. *Примечание:* использовать функции `Value`, `Maximum`.

*Вариант 4.* Разработать функцию, которая определяет, все ли коэффициенты заданного многочлена  $f$  являются четными, и в случае отрицательного ответа возвращает исходный многочлен, а в случае положительного - многочлен  $(1/2)*f$ .

*Вариант 5.* Разработать функцию, которая определяет, разлагается ли заданный многочлен  $f$  в произведение линейных множителей, и в случае положительного ответа возвращает их произведение, а в случае отрицательного - единицу. *Примечание:* использовать функции `Factors`, `Degree`.

*Вариант 6.* Разработать функцию, которая определяет, совпадают ли степени двух заданных многочленов  $f$  и  $g$ , и в случае положительного ответа возвращает их сумму, а в случае отрицательного - произведение. *Примечание:* использовать функцию `Degree`.

*Вариант 7.* Разработать функцию, которая определяет, имеет ли заданный многочлен  $f$  четную степень, и в случае отрицательного ответа возвращает исходный многочлен, а в случае положительного - многочлен  $x*f$ . *Примечание:* использовать функции `Degree`, `IndeterminateOfUnivariateRationalFunction`.

*Вариант 8.* Разработать функцию, которая определяет, равен ли нулю свободный член заданного многочлена, и в случае отрицательного ответа возвращает исходный многочлен, а в случае положительного - многочлен  $f/x$ . *Примечание:* использовать функцию `IndeterminateOfUnivariateRationalFunction`.

*Вариант 9.* Разработать функцию, которая определяет, равна ли единице сумма коэффициентов заданного многочлена  $f$ , и в случае положительного ответа возвращает исходный многочлен, а в случае отрицательного - многочлен, полученный из многочлена  $f$  делением его на сумму своих коэффициентов.

*Вариант 10.* Разработать функцию, которая определяет, равен ли единице старший коэффициент заданного многочлена  $f$ , и в случае положительного ответа возвращает исходный многочлен, а в случае отрицательного - многочлен, полученный из многочлена  $f$  делением его на свой старший коэффициент.

*Вариант 11.* Разработать функцию, которая определяет, равна ли нулю сумма коэффициентов заданного многочлена  $f$ , и в случае положительного ответа возвращает исходный многочлен, а в случае отрицательного - многочлен, полученный из многочлена  $f$  вычитанием из него суммы своих коэффициентов.

*Вариант 12.* Разработать функцию, которая определяет, равна ли единице сумма коэффициентов заданного многочлена  $f$ , и в случае положительного ответа возвращает исходный многочлен, а в случае отрицательного - многочлен, полученный из многочлена  $f$  вычитанием из него суммы своих коэффициентов и прибавлением единицы.

*Вариант 13.* Разработать функцию, которая определяет, имеет ли заданный многочлен  $f$  нечетную степень, и в случае отрицательного ответа возвращает исходный многочлен, а в случае положительного - многочлен  $x * f$ . *Примечание:* использовать функции Degree, IndeterminateOfUnivariateRationalFunction.

*Вариант 14.* Разработать функцию, которая определяет, равна ли нулю производная заданного многочлена  $f$  в заданной точке  $a$ , и в случае отрицательного ответа возвращает значение производной многочлена  $f$  в точке  $a$ , а в случае положительного - значение многочлена  $f$  в точке  $a$ . *Примечание:* использовать функции Derivative, Value.

*Вариант 15.* Разработать функцию, которая определяет, положительно ли значение заданного многочлена  $f$  в заданной точке  $a$ , и в случае отрицательного ответа возвращает значение производной многочлена  $f$  в точке  $a$ , а в случае положительного - значение многочлена  $f$  в точке  $a$ . *Примечание:* использовать функции Derivative, Value.

*Вариант 16.* Разработать функцию, которая определяет, отрицательно ли значение заданного многочлена  $f$  в заданной точке  $a$ , и в случае отрицательного ответа возвращает значение производной многочлена  $f$  в точке  $a$ , а в случае положительного - значение многочлена  $f$  в точке  $a$ . *Примечание:* использовать функции Derivative, Value.

*Вариант 17.* Разработать функцию, которая определяет, все ли коэффициенты заданного многочлена  $f$  делятся на три, и в случае отрицательного ответа возвращает исходный многочлен, а в случае положительного - многочлен  $f/3$ .

*Вариант 18.* Разработать функцию, которая определяет, равен ли единице свободный член заданного многочлена  $f$ , и в случае положительного ответа возвращает исходный многочлен, а в случае отрицательного - многочлен, полученный из многочлена  $f$  делением его на свой свободный член.

## С.5 Циклические программы (цикл FOR). Бинарные отношения

### С.5.1 Инструкции по выполнению работы

Данная лабораторная работа предназначена для изучения оператора цикла FOR на примере работы с бинарными отношениями. Подробные сведения по данным темам содержатся:

- в разделе 2 данного методического пособия;
- в разделе “Бинарные отношения в GAP” учебных материалов к курсу алгебры и теории чисел (<http://ukrgap.exponenta.ru/Examples/relation.htm>);
- в разделе “Relations” справочного руководства по системе GAP (<http://www.gap-system.org/Manuals/doc/htm/ref/chapters.htm>).

*Пример:* Разработать функцию, входным параметром которой является бинарное отношение  $r$ , заданное на множестве из  $n$  элементов, а выходным - квадратная матрица  $M$  порядка  $n \times n$ , в которой  $m_{ij} = 1$ , если  $i$  и  $j$  находятся в бинарном отношении  $\rho$ , и  $m_{ij} = 0$

в противном случае (таким образом,  $M$  - матрица смежности ориентированного графа, соответствующего бинарному отношению  $\rho$ ).

Данная задача решается следующим образом. Сначала мы определяем порядок множества  $N$ , на котором задано бинарное отношение  $r$ , и строим нулевую матрицу  $m$  соответствующего размера с помощью функции `NullMat`. Затем с помощью функции `Successors` мы получаем список  $s$ ,  $i$ -й элемент которого является списком номеров тех элементов множества  $N$ , которые находятся в бинарном отношении  $\rho$  с  $i$ -м элементом. Для замены нужных элементов нулевой матрицы единицами мы перебираем элементы полученного списка  $s$ . При этом  $i$ -й элемент списка  $s$  определяет, какие элементы  $i$ -й строки матрицы  $m$  должны быть равны единице - на единицу заменяются те элементы  $i$ -й строки матрицы  $m$ , которые лежат в столбцах с номерами из списка  $s[i]$ :

```
MatrixOfBinaryRelation:=function(r)
local s, n, m, i, j;
n := DegreeOfBinaryRelation( r );
m := NullMat( n, n );
s := Successors( r );
for i in [ 1 .. Length(s) ] do
  for j in s[i] do
    m[i][j]:=1;
  od;
od;
return m;
end;
```

Обратите внимание на форматирование программы (выделение циклов с помощью отступов), а также на то, что перебирать можно не только числа из заданного диапазона (как во внешнем цикле), но и все элементы из заданного списка (как во внутреннем цикле).

После ввода данной программы зададим случайным образом бинарное отношение на множестве из пяти элементов:

Пример

```
gap> r:=RandomBinaryRelationOnPoints(5);
Binary Relation on 5 points
```

Получим его описание с помощью функции `Successors`:

Пример

```
gap> Successors(r);
[ [ 1, 2, 4 ], [ 1, 3, 4, 5 ], [ 3, 5 ], [ 2, 4, 5 ], [ 2, 3, 4 ] ]
```

Теперь вычислим его матрицу смежности и убедимся в правильности полученного результата, сопоставив номера строк и столбцов, в которых находятся единицы, с результатом `Successors(r)`:

Пример

```
gap> Display( MatrixOfBinaryRelation( r ) );
```

```
[ [ 1, 1, 0, 1, 0 ],
  [ 1, 0, 1, 1, 1 ],
  [ 0, 0, 1, 0, 1 ],
  [ 0, 1, 0, 1, 1 ],
  [ 0, 1, 1, 1, 0 ] ]
```

Вычислим матрицу смежности для минимального симметричного бинарного отношения, содержащего заданное, и убедимся в ее симметричности относительно главной диагонали:

Пример

```
gap> Display( MatrixOfBinaryRelation( SymmetricClosureBinaryRelation( r ) ) );
[ [ 1, 1, 0, 1, 0 ],
  [ 1, 0, 1, 1, 1 ],
  [ 0, 1, 1, 0, 1 ],
  [ 1, 1, 0, 1, 1 ],
  [ 0, 1, 1, 1, 0 ] ]
```

Вычислим матрицу смежности для минимального рефлексивного бинарного отношения, содержащего заданное, и убедимся в том, что все элементы на ее главной диагонали равны единице:

Пример

```
gap> Display( MatrixOfBinaryRelation( ReflexiveClosureBinaryRelation( r ) ) );
[ [ 1, 1, 0, 1, 0 ],
  [ 1, 1, 1, 1, 1 ],
  [ 0, 0, 1, 0, 1 ],
  [ 0, 1, 0, 1, 1 ],
  [ 0, 1, 1, 1, 1 ] ]
```

Теперь проверим, что матрица отношения тождества является единичной:

Пример

```
gap> Display( MatrixOfBinaryRelation( IdentityBinaryRelation( 5 ) ) );
[ [ 1, 0, 0, 0, 0 ],
  [ 0, 1, 0, 0, 0 ],
  [ 0, 0, 1, 0, 0 ],
  [ 0, 0, 0, 1, 0 ],
  [ 0, 0, 0, 0, 1 ] ]
```

Таким образом, разработанная функция работает корректно.

### С.5.2 Задания для лабораторных работ

*Вариант 1.* Разработать функцию, которая для заданного бинарного отношения возвращает бинарное отношение, являющееся его отрицанием.

*Вариант 2.* Разработать функцию, которая для двух заданных бинарных отношений возвращает бинарное отношение, являющееся их пересечением.

*Вариант 3.* Разработать функцию, которая для двух заданных бинарных отношений возвращает бинарное отношение, являющееся их объединением.

*Вариант 4.* Разработать функцию, которая для двух заданных бинарных отношений возвращает бинарное отношение, являющееся разностью первого и второго отношений.

*Вариант 5.* Разработать функцию, которая для двух заданных бинарных отношений возвращает бинарное отношение, являющееся композицией первого и второго отношений.

*Вариант 6.* Разработать функцию, которая для заданного бинарного отношения возвращает обратное к нему бинарное отношение.

*Вариант 7.* Разработать функцию, которая для двух заданных бинарных отношений проверяет, является ли первое из них подмножеством второго.

*Вариант 8.* Разработать функцию, которая возвращает бинарное отношение, соответствующее заданному разбиению множества первых  $n$  натуральных чисел.

*Вариант 9.* Разработать функцию, которая возвращает список всех упорядоченных пар, принадлежащих заданному бинарному отношению.

*Вариант 10.* Разработать функцию, которая для заданного бинарного отношения  $\rho$  возвращает список элементов  $x$ , для которых выполняется условие  $x\rho x$ .

*Вариант 11.* Разработать функцию, которая для заданного бинарного отношения  $\rho$  возвращает список всех упорядоченных пар элементов  $(x, y)$ , для которых одновременно выполняются условия  $x\rho y$  и  $y\rho x$ .

*Вариант 12.* Разработать функцию, которая для заданного бинарного отношения  $r$  возвращает список всех упорядоченных пар элементов  $(x, y)$ , таких что которых выполняется условие  $x r y$ , но не выполняется условие  $y r x$ .

*Вариант 13.* Разработать функцию, которая для заданного бинарного отношения  $r$  возвращает список всех упорядоченных пар элементов  $(x, y)$ , таких что которых выполняется условие  $x r y$ , и элементы  $x$  и  $y$  не совпадают.

*Вариант 14.* Разработать функцию, которая для заданного бинарного отношения определяет минимальный набор упорядоченных пар, которые нужно исключить из бинарного отношения для того, чтобы оно стало антирефлексивным.

*Вариант 15.* Разработать функцию, которая для заданного бинарного отношения определяет минимальный набор упорядоченных пар, которые нужно исключить из бинарного отношения для того, чтобы оно стало асимметричным.

*Вариант 16.* Разработать функцию, которая для заданного бинарного отношения определяет минимальный набор упорядоченных пар, которые нужно исключить из бинарного отношения для того, чтобы оно стало антисимметричным.

*Вариант 17.* Разработать функцию, которая возвращает область определения заданного бинарного отношения.

*Вариант 18.* Разработать функцию, которая возвращает область значения заданного бинарного отношения.

## С.6 Циклические программы (цикл WHILE). Подстановки

### С.6.1 Инструкции по выполнению работы

Данная лабораторная работа предназначена для изучения оператора цикла WHILE на примере работы с подстановками. Подробные сведения по данным темам содержатся:

- в разделе 2 данного методического пособия;

- в разделах “Подстановки в системе GAP” и “Алгоритм умножения подстановок” учебных материалов к курсу алгебры и теории чисел (<http://ukrgap.exponenta.ru/Examples/examples.htm>);
- в разделе “Permutations” справочного руководства по системе GAP (<http://www.gap-system.org/Manuals/doc/htm/ref/chapters.htm>).

*Пример:* Разработать функцию, которая определяет порядок заданной подстановки, определяя минимальную степень, в которую нужно ее возвести, чтобы получить тождественную подстановку.

Данная задача решается с помощью следующей функции:

```
OrderOfPermutation:=function(s)
local k, t;
k:=1;
t:=s;
while t <> () do
  t:=t*s;
  k:=k+1;
od;
return k;
end;
```

Имя функции `OrderOfPermutation` было выбрано таким образом, чтобы оно не совпадало с именем имеющейся в GAP стандартной функции `OrderPerm` - иначе при ее вводе было бы получено сообщение об ошибке, т.к. библиотечные функции защищены от переопределения пользователем.

После ввода данной программы протестируем ее на различных подстановках:

Пример

```
gap> OrderOfPermutation( () );
1
gap> OrderOfPermutation( (1,2) );
2
gap> OrderOfPermutation( (1,2,3) );
3
gap> OrderOfPermutation( (1,2,3)(4,5) );
6
```

Заметим, что порядок подстановки равен наименьшему общему кратному длин независимых циклов в ее разложении (см. последний пример), и использование этого факта позволило бы более эффективно найти порядок подстановки, избежав ее возведения в степень. Поэтому данный пример применим только в учебных целях. Сравним быстродействие нашей функции со стандартной:

Пример

```
gap> for i in [1..1000] do
>   k:=OrderPerm( (1,2,3,4,5)(6,7,8)(9,10)(11,12,13,14,15)(16,17,18) );
```

```

> od;
gap> time;
384
gap> for i in [1..1000] do
>   k:=OrderOfPermutation( (1,2,3,4,5)(6,7,8)(9,10)(11,12,13,14,15)(16,17,18) );
>   od;
gap> time;
743

```

Таким образом, мы видим, что стандартная функция более эффективна.

Получим с помощью нашей функции список порядков всех элементов симметрической группы подстановок третьей степени:

Пример

```

gap> List( SymmetricGroup(3), OrderOfPermutation );
[ 1, 2, 3, 2, 3, 2 ]

```

Повторим то же самое для симметрической группы подстановок пятой степени, сгруппировав результаты:

Пример

```

gap> Collected( List( SymmetricGroup(5), OrderOfPermutation ) );
[ [ 1, 1 ], [ 2, 25 ], [ 3, 20 ], [ 4, 30 ], [ 5, 24 ], [ 6, 20 ] ]

```

Таким образом, в ней 25 элементов второго порядка, 20 - третьего, 30 - четвертого, 24 - пятого, и 20 - шестого порядка.

## С.6.2 Задания для лабораторных работ

*Примечание:* необходимо разработать собственную функцию с обязательным применением цикла WHILE, независимо от того, решается ли задача прямым применением стандартной функции системы GAP или нет. *Вариант 1.* Разработать функцию, которая для заданной подстановки  $s$  определяет минимальное натуральное число  $k$ , такое что  $s^k$  коммутирует с заданной подстановкой  $t$ .

*Вариант 2.* Разработать функцию, которая для заданной подстановки  $s$  определяет минимальное натуральное число  $k$ , такое что  $s^k$  переводит заданное натуральное число  $n$  в заданное натуральное число  $m$ , и возвращает **fail**, если такого числа  $k$  не существует.

*Вариант 3.* Разработать функцию, которая для заданной подстановки  $s$  определяет минимальное натуральное число  $k$ , которое она оставляет на месте.

*Вариант 4.* Разработать функцию, которая для заданной подстановки  $s$  определяет минимальное натуральное число  $k$ , такое что  $s^k$  оставляет на месте заданное натуральное число  $n$ .

*Вариант 5.* Разработать функцию, которая для заданной подстановки  $s$  определяет минимальное натуральное число  $k$ , такое что количество натуральных чисел, перемещаемых подстановкой  $s^k$ , не превосходит заданного натурального числа  $n$ .

*Вариант 6.* Разработать функцию, которая для заданной подстановки  $s$  определяет минимальное натуральное число  $k$ , такое что  $s^k$  оставляет на месте единицу.

*Вариант 7.* Разработать функцию, которая для заданной подстановки  $s$  определяет минимальное натуральное число  $k$ , такое что  $s^k$  коммутирует с подстановкой (12).

*Вариант 8.* Разработать функцию, которая для заданной подстановки  $s$  определяет минимальное натуральное число  $k$ , такое что  $s^k$  переводит 1 в 2, и возвращает **fail**, если такого числа  $k$  не существует.

*Вариант 9.* Разработать функцию, которая для заданной подстановки  $s$  вычисляет орбиту числа 1, т.е. множество всех чисел, в которые единицу можно перевести с помощью некоторой степени  $s^k$  исходной подстановки.

*Вариант 10.* Разработать функцию, которая для заданной подстановки  $s$  определяет минимальное натуральное число  $k$ , такое что количество натуральных чисел, перемещаемых подстановкой  $s^k$ , не превосходит 5.

*Вариант 11.* Разработать функцию, которая для заданной подстановки  $s$  определяет минимальное натуральное число  $k$ , такое что  $s^k$  оставляет на месте наибольшее число, перемещаемое данной подстановкой.

*Вариант 12.* Разработать функцию, которая для заданной подстановки  $s$  определяет максимальное натуральное число  $k$ , такое что  $s^k$  не коммутирует с подстановкой (1 2).

*Вариант 13.* Разработать функцию, которая для заданной подстановки  $s$  определяет максимальное натуральное число  $k$ , такое что количество натуральных чисел, перемещаемых подстановкой  $s^k$ , превосходит заданное натуральное число  $n$ , и возвращает **fail**, если такого числа  $k$  не существует.

*Вариант 14.* Разработать функцию, которая для заданной подстановки  $s$  вычисляет орбиту наименьшего числа, перемещаемого данной подстановкой, т.е. множество всех чисел, в которые его можно перевести с помощью некоторой степени  $s^k$  исходной подстановки.

*Вариант 15.* Разработать функцию, которая для заданной подстановки  $s$  возвращает множество орбит чисел, перемещаемых данной подстановкой.

*Вариант 16.* Разработать функцию, которая для заданной подстановки  $s$  определяет минимальное натуральное число  $k$ , такое что  $s^k$  оставляет на месте наименьшее число, перемещаемое данной подстановкой.

*Вариант 17.* Разработать функцию, которая для заданной подстановки  $s$  вычисляет орбиту наибольшего числа, перемещаемого данной подстановкой, т.е. множество всех чисел, в которые его можно перевести с помощью некоторой степени  $s^k$  исходной подстановки.

*Вариант 18.* Разработать функцию, которая для заданной подстановки  $s$  определяет максимальное натуральное число  $k$ , такое что количество натуральных чисел, перемещаемых подстановкой  $s^k$ , превосходит 5, и возвращает **fail**, если такого числа  $k$  не существует.

## С.7 Циклические программы (цикл REPEAT). Группы подстановок

### С.7.1 Инструкции по выполнению работы

Данная лабораторная работа предназначена для изучения оператора цикла REPEAT на примере работы с подстановками и группами подстановок. Подробные сведения по данным темам содержатся:

- в разделе 2 данного методического пособия;
- в разделах “Подстановки в системе GAP”, “Построение таблицы умножения для конечной группы подстановок” и “Алгоритм умножения подстановок” учебных материалов к курсу алгебры и теории чисел (<http://ukrgap.exponenta.ru/Examples/examples.htm>);
- в разделах “Permutations” и “Permutation groups” справочного руководства по системе GAP (<http://www.gap-system.org/Manuals/doc/htm/ref/chapters.htm>).

*Пример:* Разработать функцию, которая выбирает случайным образом элемент заданной группы подстановок, действующей на множестве  $M = \{1, \dots, n\}$ , до тех пор, пока не обнаружит подстановку, которая переставляет все точки множества  $M$ , и возвращает найденную подстановку.

Сначала нужно понять, как определить порядок множества  $M$ , и как понять, что заданная подстановка не имеет неподвижных точек. Для этого поэкспериментируем в диалоговом режиме с некоторыми функциями из раздела “Подстановки в системе GAP” (<http://ukrgap.exponenta.ru/Examples/permutat.htm>):

Пример

```
gap> G:=SymmetricGroup(3);
Sym( [ 1 .. 3 ] )
gap> LargestMovedPoint(G);
3
gap> s:=(1,2);
(1,2)
gap> NrMovedPoints(s);
2
gap> t:=(1,2,3);
(1,2,3)
gap> NrMovedPoints(t);
3
```

Теперь задача стала более ясной, и можно разработать следующую функцию:

```
FindFixedPointFreePermutation:=function(G)
local n,s;
n:=LargestMovedPoint(G);
repeat
  s := Random(G);
```

```

until NrMovedPoints(s) = n;
return s;
end;

```

Протестируем ее на различных группах:

Пример

```

gap> G:=SymmetricGroup(3);
Sym( [ 1 .. 3 ] )
gap> FindFixedPointFreePermutation(G);
(1,3,2)
gap> FindFixedPointFreePermutation(AlternatingGroup(4));
(1,2)(3,4)
gap> FindFixedPointFreePermutation(AlternatingGroup(5));
(1,3,5,4,2)
gap> G:=SymmetricGroup(30);
Sym( [ 1 .. 30 ] )
gap> Size(G);
26525285981219105863630848000000
gap> FindFixedPointFreePermutation(G);
(1,23)(2,19,10,8,3)(4,12,5,28,6,13,14,29,11,21,20,24,26,18,16,17)(7,27,15,9,25,30,22)

```

Заметим, что условие  $M=\{1,\dots,n\}$  в постановке задачи существенно, так как программа закликивается в следующем примере:

Пример

```

gap> G:=Group((2,3));
Group([ (2,3) ])
gap> FindFixedPointFreePermutation(G);

```

Такое поведение объясняется тем, что в этом случае `LargestMovedPoint(G)` возвращает 3. Естественно, что никакая подстановка из группы  $G$  не перемещает три точки, и поэтому нужное условие никогда не достигается. С другой стороны, функция `NrMovedPoints(G)` возвращает 2, и нужно использовать именно ее. Поэтому для большей универсальности нашу функцию нужно модифицировать следующим образом:

```

FindFixedPointFreePermutation:=function(G)
local n,s;
n:=NrMovedPoints(G);
repeat
s := Random(G);
until NrMovedPoints(s) = n;
return s;
end;

```

Проверим теперь ее работу:

```

gap> G:=Group((2,3));
Group([ (2,3) ])
gap> FindFixedPointFreePermutation(G);
(2,3)
gap> FindFixedPointFreePermutation(SymmetricGroup(10));
(1,2)(3,5,9,8)(4,10)(6,7)
gap> FindFixedPointFreePermutation(SymmetricGroup(100));
(1,11,24,8,37,73,62,38,97,54,88,47,2,60,50,56,19,84,67,65,32,69,14,4,10,6,33,
52,83,70,5,29,91,86,77,78,43,61,35,3,22,57,95,85,23)(7,98,16,28,34,13,48,39,
46,90,45,40,21,42,36,18,17,58,76,74)(9,96,92,15,89,41,93,59,75)(12,79,30,94,
99,51,53,31,81,63,44,55)(20,82,71)(25,80,49,68,26,87,64,100,27,72,66)

```

Мы видим, что она работает корректно как с группами, оставляющими единицу на месте, так и с симметрическими группами подстановок.

Иногда при тестировании гипотез интересно отображать динамику расчета на экране и проследить, сколько попыток выбора случайного элемента из группы было сделано, прежде чем был найден элемент с необходимыми свойствами. Для этого можно вставить в функцию счетчик и отображать его текущее значение на экране. При этом управляющая строка “\r” используется для стирания результата предыдущего вывода на экран и печати новой информации в той же строке:

```

FindFixedPointFreePermutation:=function(G)
local n,s,k;
n:=NrMovedPoints(G);
k:=0;
repeat
  s := Random(G);
  k := k+1;
  Print(k, "\r");
until NrMovedPoints(s) = n;
Print("\n");
return s;
end;

```

Теперь скопируйте в окно GAP последний вариант нашей функции и обратитесь к ней несколько раз для какой-нибудь довольно большой группы. Посмотрите, сколько шагов в среднем требуется для нахождения нужной подстановки.

### С.7.2 Задания для лабораторных работ

В данной лабораторной работе Вам необходимо решить ту же задачу, что и в работе С.6, но только с применением цикла REPEAT вместо цикла WHILE. Кроме того, при тестировании разработанной Вами функции Вы должны вместо непосредственного задания подстановки сначала задать некоторую группу подстановок (например, симметрическую,

знакопеременную, или порожденную указанным Вами списком подстановок), после чего выбрать случайным образом ее элемент(ы) для использования в качестве аргумента Вашей функции.

*Вариант 1.* Разработать функцию, которая для заданной подстановки  $s$  определяет минимальное натуральное число  $k$ , такое что  $s^k$  коммутирует с заданной подстановкой  $t$ .

*Вариант 2.* Разработать функцию, которая для заданной подстановки  $s$  определяет минимальное натуральное число  $k$ , такое что  $s^k$  переводит заданное натуральное число  $n$  в заданное натуральное число  $m$ , и возвращает **fail**, если такого числа  $k$  не существует.

*Вариант 3.* Разработать функцию, которая для заданной подстановки  $s$  определяет минимальное натуральное число  $k$ , которое она оставляет на месте.

*Вариант 4.* Разработать функцию, которая для заданной подстановки  $s$  определяет минимальное натуральное число  $k$ , такое что  $s^k$  оставляет на месте заданное натуральное число  $n$ .

*Вариант 5.* Разработать функцию, которая для заданной подстановки  $s$  определяет минимальное натуральное число  $k$ , такое что количество натуральных чисел, перемещаемых подстановкой  $s^k$ , не превосходит заданного натурального числа  $n$ .

*Вариант 6.* Разработать функцию, которая для заданной подстановки  $s$  определяет минимальное натуральное число  $k$ , такое что  $s^k$  оставляет на месте единицу.

*Вариант 7.* Разработать функцию, которая для заданной подстановки  $s$  определяет минимальное натуральное число  $k$ , такое что  $s^k$  коммутирует с подстановкой  $(1\ 2)$ .

*Вариант 8.* Разработать функцию, которая для заданной подстановки  $s$  определяет минимальное натуральное число  $k$ , такое что  $s^k$  переводит 1 в 2, и возвращает **fail**, если такого числа  $k$  не существует.

*Вариант 9.* Разработать функцию, которая для заданной подстановки  $s$  вычисляет орбиту числа 1, т.е. множество всех чисел, в которые единицу можно перевести с помощью некоторой степени  $s^k$  исходной подстановки.

*Вариант 10.* Разработать функцию, которая для заданной подстановки  $s$  определяет минимальное натуральное число  $k$ , такое что количество натуральных чисел, перемещаемых подстановкой  $s^k$ , не превосходит 5.

*Вариант 11.* Разработать функцию, которая для заданной подстановки  $s$  определяет минимальное натуральное число  $k$ , такое что  $s^k$  оставляет на месте наибольшее число, перемещаемое данной подстановкой.

*Вариант 12.* Разработать функцию, которая для заданной подстановки  $s$  определяет максимальное натуральное число  $k$ , такое что  $s^k$  не коммутирует с подстановкой  $(12)$ .

*Вариант 13.* Разработать функцию, которая для заданной подстановки  $s$  определяет максимальное натуральное число  $k$ , такое что количество натуральных чисел, перемещаемых подстановкой  $s^k$ , превосходит заданное натуральное число  $n$ , и возвращает **fail**, если такого числа  $k$  не существует.

*Вариант 14.* Разработать функцию, которая для заданной подстановки  $s$  вычисляет орбиту наименьшего числа, перемещаемого данной подстановкой, т.е. множество всех чисел, в которые его можно перевести с помощью некоторой степени  $s^k$  исходной подстановки.

*Вариант 15.* Разработать функцию, которая для заданной подстановки  $s$  возвращает множество орбит чисел, перемещаемых данной подстановкой.

*Вариант 16.* Разработать функцию, которая для заданной подстановки  $s$  определяет минимальное натуральное число  $k$ , такое что  $s^k$  оставляет на месте наименьшее число, перемещаемое данной подстановкой.

*Вариант 17.* Разработать функцию, которая для заданной подстановки  $s$  вычисляет орбиту наибольшего числа, перемещаемого данной подстановкой, т.е. множество всех чисел, в которые его можно перевести с помощью некоторой степени  $s^k$  исходной подстановки.

*Вариант 18.* Разработать функцию, которая для заданной подстановки  $s$  определяет максимальное натуральное число  $k$ , такое что количество натуральных чисел, перемещаемых подстановкой  $s^k$ , превосходит 5, и возвращает `fail`, если такого числа  $k$  не существует.

## С.8 Изучение свойств элементов группы

### С.8.1 Инструкции по выполнению работы

Данная лабораторная работа предназначена для изучения некоторых приемов работы с элементами групп.

Подробные сведения по данным темам содержатся:

- в разделе 4 и приложении В данного методического пособия;
- в разделе “Groups” справочного руководства по системе GAP (<http://www.gap-system.org/Manuals/doc/htm/ref/chapters.htm>) и других соответствующих его разделах.

*Пример:* Найти все 2-элементные множества, порождающие симметрическую группу  $S_3$ . В интерактивном режиме эту задачу можно решить следующим образом:

Пример

```
gap> S:=SymmetricGroup(3); # исходная группа
Sym( [ 1 .. 3 ] )
gap> g:=AsList(S); # список ее элементов
[ (), (2,3), (1,2), (1,2,3), (1,3,2), (1,3) ]
gap> m:=Combinations(g,2); # всевозможные неупорядоченные пары
[ [ (), (2,3) ], [ (), (1,2) ], [ (), (1,2,3) ], [ (), (1,3,2) ],
  [ (), (1,3) ], [ (2,3), (1,2) ], [ (2,3), (1,2,3) ], [ (2,3), (1,3,2) ],
  [ (2,3), (1,3) ], [ (1,2), (1,2,3) ], [ (1,2), (1,3,2) ], [ (1,2), (1,3) ],
  [ (1,2,3), (1,3,2) ], [ (1,2,3), (1,3) ], [ (1,3,2), (1,3) ] ]
gap> Length(m); # всего 15 пар элементов
15
gap> Filtered(m, t -> S=Subgroup(S,t) ); # отбираем пары, порождающие всю группу
[ [ (2,3), (1,2) ], [ (2,3), (1,2,3) ], [ (2,3), (1,3,2) ], [ (2,3), (1,3) ],
  [ (1,2), (1,2,3) ], [ (1,2), (1,3,2) ], [ (1,2), (1,3) ],
  [ (1,2,3), (1,3) ], [ (1,3,2), (1,3) ] ]
gap> Length(last); # осталось только 9 пар элементов
9
```

При разработке функции аналогичного назначения можно организовать двойной перебор элементов группы для того, чтобы более оптимально расходовать память и не держать в ней одновременно весь список всех неупорядоченных пар элементов группы (еще одно упрощение может быть получено исключением из рассмотрения нейтрального элемента

группы - сделайте это самостоятельно). Такая функция будет выглядеть следующим образом:

```

FindGeneratingPairs:=function(G)
local g, n, i, j, s;
g := AsList( G );
n := Size( G );
s := [ ];
for i in [ 1 .. n-1 ] do
  for j in [ i+1 .. n ] do
    if G = Subgroup( G, [ g[i], g[j] ] ) then
      Add(s, [ g[i], g[j] ]);
    fi;
  od;
od;
return s;
end;

```

Протестируем ее и проверим, что результат совпадает с полученным ранее:

Пример

```

gap> S:=SymmetricGroup(3);
Sym( [ 1 .. 3 ] )
gap> FindGeneratingPairs(S);
[ [ (2,3), (1,2) ], [ (2,3), (1,2,3) ], [ (2,3), (1,3,2) ], [ (2,3), (1,3) ],
  [ (1,2), (1,2,3) ], [ (1,2), (1,3,2) ], [ (1,2), (1,3) ],
  [ (1,2,3), (1,3) ], [ (1,3,2), (1,3) ] ]

```

Заметьте, что любое 2-элементное подмножество группы  $S_3$ , не равное  $\{ (1\ 2\ 3), (1\ 3\ 2) \}$  и не содержащее тождественной подстановки, порождает всю группу  $S_3$ .

### С.8.2 Задания для лабораторных работ

*Вариант 1.* Разработать функцию для непосредственной проверки того, что заданная группа является коммутативной, путем перемножения всевозможных пар ее порождающих элементов. *Указание:* использовать функцию `GeneratorsOfGroups`.

*Вариант 2.* Разработать функцию для проверки того, что знакопеременная группа  $A_n$  порождается всевозможными циклами длины 3, и проверить с ее помощью данное утверждение для всех  $n$ , не превосходящих 7.

*Вариант 3.* Разработать функцию, которая возвращает множество элементов заданной группы, имеющих порядок, равный заданному числу  $k$ . *Указание:* использовать функции `AsList`, `Order`.

*Вариант 4.* Разработать функцию, которая возвращает множество порядков элементов заданной группы. *Указание:* использовать функции `AsList`, `Order`, `Set`.

*Вариант 5.* Проверить, выполняется ли в группе подстановок  $S_3$  тождество  $((x, y), z) = 1$ , где  $(a, b) = a^{-1}b^{-1}ab$ .

*Вариант 6.* Известно, что при четном  $n > 4$  знакопеременная группа  $A_n$  порождается двумя подстановками:  $(12)(n-1n)$  и  $(12\dots n-1)$ . Проверить это утверждение для всех  $n$ , не превосходящих 10.

*Вариант 7.* Проверить, выполняется ли в группе подстановок  $S_3$  тождество  $x^6 = 1$ .

*Вариант 8.* Проверить, что знакопеременная группа  $A_5$  порождается подстановками  $(2\ 5\ 4)$  и  $(1\ 2\ 3\ 4\ 5)$ .

*Вариант 9.* Проверить, выполняется ли в группе подстановок  $S_3$  тождество  $(x^2, y^2) = 1$ , где  $(a, b) = a^{-1}b^{-1}ab$ .

*Вариант 10.* Известно, что при нечетном  $n > 5$  знакопеременная группа  $A_n$  порождается двумя подстановками:  $(1n)(2n-1)$  и  $(12\dots n-2)$ . Проверить это утверждение для всех  $n$ , не превосходящих 10.

*Вариант 11.* Проверить что группа подстановок  $S_n$  порождается транспозицией  $(12)$  и циклом  $(12\dots n)$ , для всех  $n$ , не превосходящих 10.

*Вариант 12.* Разработать функцию, которая возвращает множество порядков классов сопряженных элементов заданной группы. *Указание:* использовать функцию `ConjugacyClasses`.

*Вариант 13.* Разработать функцию, которая для заданной конечной группы определяет множество индексов циклических подгрупп, порождаемых ее элементами.

*Вариант 14.* Разработать функцию, печатающую для заданного натурального  $n$  таблицу Кэли для симметрической группы  $S_n$ .

*Вариант 15.* Разработать функцию, которая вычисляет подгруппу заданной  $p$ -группы, порожденную  $p$ -ми степенями ее элементов.

*Вариант 16.* Разработать функцию, которая вычисляет подгруппу заданной  $p$ -группы, порожденную всеми ее элементами порядка  $p$ .

*Вариант 17.* Разработать функцию для вычисления показателя (экспоненты) конечной группы как наименьшего общего кратного порядков представителей классов сопряженных элементов данной группы. *Указание:* использовать функцию `ConjugacyClasses`.

*Вариант 18.* Разработать функцию для вычисления количества элементов каждого порядка в заданной группе.

## С.9 Изучение свойств подгрупп группы

### С.9.1 Инструкции по выполнению работы

Данная лабораторная работа предназначена для изучения работы с подгруппами. Подробные сведения по данным темам содержатся:

- в разделе 4 и приложении В данного методического пособия;
- в разделе “Groups” справочного руководства по системе GAP (<http://www.gap-system.org/Manuals/doc/htm/ref/chapters.htm>) и других соответствующих его разделах.

*Пример.* В каких Силовских 2-подгруппах группы  $S_4$  содержатся подстановки  $(1\ 3\ 2\ 4)$ ,  $(1\ 3)$  и  $(12)(34)$  ?

Данную задачу можно решить в интерактивном режиме следующим образом. Сначала зададим исходную группу:

Пример

```
gap> S := SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
```

Вычислим ее Силовские 2-подгруппы. Поскольку по второй теореме Силова все Силовские  $p$ -подгруппы сопряжены, то функция `SylowSubgroup` возвращает только одну подгруппу, которая является представителем некоторого класса сопряженных подгрупп.

Пример

```
gap> P2 := SylowSubgroup( S, 2 );
Group([ (1,2), (3,4), (1,3)(2,4) ])
```

Для того, чтобы получить остальные подгруппы из этого класса, сначала нужно создать класс сопряженных подгрупп данной группы с заданным представителем, а затем получить список содержащихся в нем подгрупп:

Пример

```
gap> c2 := ConjugacyClassSubgroups( S, P2 );
Group([ (1,2), (3,4), (1,3)(2,4) ])^G
gap> l2 := AsList( c2 );
[ Group([ (1,2), (3,4), (1,3)(2,4) ]), Group([ (1,3), (2,4), (1,2)(3,4) ]),
  Group([ (1,4), (2,3), (1,2)(3,4) ])]
```

Теперь мы можем получить из него списки подгрупп, которые содержат указанные подстановки:

Пример

```
gap> Filtered( l2, g -> (1,3,2,4) in g );
[ Group([ (1,2), (3,4), (1,3)(2,4) ])]
gap> Filtered( l2, g -> (1,3) in g );
[ Group([ (1,3), (2,4), (1,2)(3,4) ])]
gap> Filtered( l2, g -> (1,2)(3,4) in g );
[ Group([ (1,2), (3,4), (1,3)(2,4) ]), Group([ (1,3), (2,4), (1,2)(3,4) ]),
  Group([ (1,4), (2,3), (1,2)(3,4) ])]
```

## С.9.2 Задания для лабораторных работ

*Вариант 1.* Разработать функцию, которая для заданной группы возвращает множество порядков всех ее подгрупп, полученное как множество порядков представителей ее классов сопряженных подгрупп. *Указание:* использовать функции `ConjugacyClassesSubgroups`, `Representative`.

*Вариант 2.* Разработать функцию, которая для заданной группы возвращает список простых делителей порядка группы с указанием порядка и количества соответствующих Силовских  $p$ -подгрупп. *Указание:* использовать функции `SylowSubgroup`, `ConjugacyClassSubgroups`.

*Вариант 3.* Разработать функцию, которая для заданной группы возвращает множество порядков ее максимальных подгрупп. *Указание:* использовать функции `Size`, `MaximalSubgroups`.

*Вариант 4.* Разработать функцию, которая для заданной группы возвращает множество порядков ее нормальных подгрупп. *Указание:* использовать функции `Size`, `NormalSubgroups`.

*Вариант 5.* Разработать функцию, которая для заданной группы определяет список порядков факторов ее нижнего центрального ряда. *Указание:* использовать функции `Size`, `LowerCentralSeries`.

*Вариант 6.* На примере знакопеременной группы  $A_4$  показать, что нормальная подгруппа  $K$  нормальной подгруппы  $H$  группы  $G$  не обязательно является нормальной во всей группе  $G$ . *Указание:* использовать функции `NormalSubgroups`, `IsNormal`.

*Вариант 7.* Найти все подгруппы в циклической группе порядка 360. Проверить, что они образуют цепь подгрупп. *Указание:* использовать функции `NormalSubgroups`, `IsSubgroup`.

*Вариант 8.* Проверить, что группы  $S_3$ ,  $A_4$ ,  $S_4$  являются разрешимыми. *Указание:* использовать функцию `DerivedSubgroup`.

*Вариант 9.* Составить функцию, которая для заданной группы вычисляет подгруппу Фраттини, т.е. пересечение всех ее максимальных подгрупп. *Указание:* использовать функции `Intersection`, `MaximalSubgroups`.

*Вариант 10.* Сколько различных силовских  $p$ -подгрупп содержится в группе  $A_5$  для  $p = 2, 3, 5$ ? *Указание:* использовать функцию `SylowSubgroup`.

*Вариант 11.* Разработать функцию, которая для заданной группы возвращает список представителей классов сопряженности ее Силовских  $p$ -подгрупп. *Указание:* использовать функцию `SylowSubgroup`.

*Вариант 12.* Разработать функцию, которая для заданной группы определяет порядок ее фактор-группы по коммутанту. *Указание:* использовать функции `Size` и `DerivedSubgroup`.

*Вариант 13.* Проверить, что знакопеременная группа  $A_5$  является простой, т.е. не содержит нетривиальных нормальных подгрупп. *Указание:* использовать функцию `NormalSubgroups`.

*Вариант 14.* Проверить, что подгруппа группы  $S_7$ , порожденная подстановками  $(1\ 2\ 3)$  и  $(1\ 4\ 5\ 6\ 7)$ , не является разрешимой. *Указание:* использовать функцию `DerivedSubgroup`.

*Вариант 15.* Разработать функцию, которая для заданной группы определяет список показателей (экспонент) элементов ее нижнего центрального ряда. *Указание:* использовать функции `Exponent`, `LowerCentralSeries`.

*Вариант 16.* Составить функцию, которая для заданной группы вычисляет список порядков элементов ее нижнего центрального ряда. *Указание:* использовать функции `Size`, `LowerCentralSeries`.

*Вариант 17.* Составить функцию, которая для заданной группы определяет множество индексов ее максимальных подгрупп. *Указание:* использовать функции `Index`, `MaximalSubgroups`.

*Вариант 18.* Найти все силовские  $p$ -подгруппы в группах  $S_3$ ,  $A_4$ . *Указание:* использовать функцию `SylowSubgroup`.

## С.10 Работа с библиотекой конечных групп

### С.10.1 Инструкции по выполнению работы

Данная лабораторная работа предназначена для изучения работы с библиотекой конечных групп системы GAP. Необходимо сначала выбрать группы из библиотеки по указанному критерию (порядок группы в сочетании с некоторым свойством), а затем для каждой из этих групп определить указанное свойство, и вывести результаты на экран в виде таблицы с указанием номера соответствующей группы в библиотеке конечных групп системы GAP.

Подробные сведения по данным темам содержатся:

- в разделе 2 данного методического пособия;
- в разделе “Библиотеки групп в системе GAP” (<http://ukrgap.exponenta.ru/Examples/Groups.htm>) учебных материалов к курсу алгебры и теории чисел;
- в разделе “Group libraries” справочного руководства по системе GAP (<http://www.gap-system.org/Manuals/doc/htm/ref/chapters.htm>).

### С.10.2 Задания для лабораторных работ

*Вариант 1.* Выбрать все нециклические 2-группы порядка 32. Определить их количество. Каждую из них идентифицировать с помощью функции `GroupId` и вычислить ее класс нильпотентности с помощью функции `LowerCentralSeries`. Результат вывести в виде таблицы.

*Вариант 2.* Выбрать все неабелевы 2-группы порядка 32. Определить их количество. Идентифицировать каждую из них с помощью функции `GroupId` и вычислить порядок ее центра. Результат вывести в виде таблицы.

*Вариант 3.* Выбрать все неабелевы 2-группы порядка 32. Определить их количество. Идентифицировать каждую из них с помощью функции `GroupId`, вычислить и идентифицировать ее фактор-группу по коммутанту. Результат вывести в виде таблицы.

*Вариант 4.* Выбрать все неабелевы 2-группы порядка 32. Определить их количество. Идентифицировать каждую из них с помощью функции `GroupId` и вычислить порядок ее коммутанта. Результат вывести в виде таблицы.

*Вариант 5.* Выбрать все 2-группы порядка 32, порядок коммутанта которых равен 8. Определить их количество. Идентифицировать каждую с помощью функции `GroupId` и вычислить длину ее ряда коммутантов с помощью функции `DerivedSeries`. Результат вывести в виде таблицы.

*Вариант 6.* Выбрать все нециклические 3-группы порядка 27. Определить их количество. Каждую из них идентифицировать с помощью функции `GroupId` и вычислить ее класс нильпотентности с помощью функции `LowerCentralSeries`. Результат вывести в виде таблицы.

*Вариант 7.* Выбрать все неабелевы 3-группы порядка 27. Определить их количество. Идентифицировать каждую из них с помощью функции `GroupId` и вычислить порядок ее центра. Результат вывести в виде таблицы.

*Вариант 8.* Выбрать все неабелевы 3-группы порядка 27. Определить их количество. Идентифицировать каждую из них с помощью функции `GroupId`, вычислить и идентифицировать ее фактор-группу по коммутанту. Результат вывести в виде таблицы.

*Вариант 9.* Выбрать все неабелевы 3-группы порядка 27. Определить их количество. Идентифицировать каждую из них с помощью функции `GroupId` и вычислить порядок ее коммутанта. Результат вывести в виде таблицы.

*Вариант 10.* Выбрать все 3-группы порядка 81 с коммутантом порядка 9. Определить их количество. Идентифицировать каждую из них с помощью функции `GroupId` и вычислить длину ее ряда коммутантов с помощью функции `DerivedSeries`. Результат вывести в виде таблицы.

*Вариант 11.* Выбрать все неабелевы 2-группы порядка 32. Определить их количество. Каждую из них идентифицировать с помощью функции `GroupId` и вычислить ее класс нильпотентности с помощью функции `LowerCentralSeries`. Результат вывести в виде таблицы.

*Вариант 12.* Выбрать все неабелевы 2-группы порядка 64. Определить их количество. Идентифицировать каждую из них с помощью функции `GroupId` и вычислить порядок ее центра. Результат вывести в виде таблицы.

*Вариант 13.* Выбрать все неабелевы 2-группы порядка 64. Определить их количество. Идентифицировать каждую из них с помощью функции `GroupId`, вычислить и идентифицировать ее фактор-группу по коммутанту. Результат вывести в виде таблицы.

*Вариант 14.* Выбрать все неабелевы 2-группы порядка 64. Определить их количество. Идентифицировать каждую из них с помощью функции `GroupId` и вычислить порядок ее коммутанта. Результат вывести в виде таблицы.

*Вариант 15.* Выбрать все 2-группы порядка 64, порядок коммутанта которых равен 8. Определить их количество. Идентифицировать каждую с помощью функции `GroupId` и вычислить длину ее ряда коммутантов с помощью функции `DerivedSeries`. Результат вывести в виде таблицы.

*Вариант 16.* Выбрать все нециклические 3-группы порядка 81. Определить их количество. Каждую из них идентифицировать с помощью функции `GroupId` и вычислить ее класс нильпотентности с помощью функции `LowerCentralSeries`. Результат вывести в виде таблицы.

*Вариант 17.* Выбрать все неабелевы 3-группы порядка 81. Определить их количество. Идентифицировать каждую из них с помощью функции `GroupId` и вычислить порядок ее центра. Результат вывести в виде таблицы.

*Вариант 18.* Выбрать все неабелевы 3-группы порядка 81. Определить их количество. Идентифицировать каждую из них с помощью функции `GroupId`, вычислить и идентифицировать ее фактор-группу по коммутанту. Результат вывести в виде таблицы.

## С.11 **Дополнительные упражнения различной трудности**

1. Разработать функцию для вычисления  $n!! = n \cdot (n - 2) \cdot (n - 4) \cdot \dots \cdot 1$ .

2. Разработать функцию для вычисления  $n$ -го числа Фибоначчи, где  $f(1) = f(2) = 1$ ,  $f(n) = f(n - 1) + f(n - 2)$ .

3. Разработать функцию для проверки того, чётно или нечётно заданное натуральное число.

4. Разработать функцию для проверки того, делится ли заданное натуральное число на три.
5. Разработать функцию для проверки того, сравнимы ли два целых числа по заданному модулю.
6. Разработать функцию для вычисления суммы нечетных натуральных чисел, не превосходящих заданное натуральное число.
7. Разработать функцию для вычисления произведения нечетных натуральных чисел, не превосходящих заданное натуральное число.
8. Разработать функцию для вычисления суммы четных натуральных чисел, не превосходящих заданное натуральное число.
9. Разработать функцию для вычисления произведения четных натуральных чисел, не превосходящих заданное натуральное число.
10. Разработать собственную функцию для проверки того, что заданное натуральное число является простым с помощью одного из известных методов.
11. Разработать функцию для вычисления НОД двух натуральных чисел  $a$  и  $b$  по алгоритму Евклида:  $a = bq_1 + r_1$ ,  $b = r_1q_2 + r_2$ ,  $r_1 = r_2q_3 + r_3$ , ...,  $r_{n-2} = r_{n-1}q_n + r_n$ ,  $r_{n-1} = r_nq_n$ .
12. Разработать функцию для определения произведения всех простых делителей натурального числа. *Указание:* использовать функции **Factors**, **Set**, **Product**.
13. Разработать функцию для определения суммы натуральных делителей натурального числа. *Указание:* использовать функции **Factors** и **Collected**.
14. Разработать функцию для определения количества натуральных делителей натурального числа. *Указание:* использовать функции **Factors** и **Collected**.
15. Разработать функцию для вычисления для натурального  $n = p_1^{k_1} \cdot \dots \cdot p_s^{k_s}$  функции Эйлера  $\phi(n)$  по формуле  $\phi(n) = n(1-1/p_1) \cdot \dots \cdot (1-1/p_s)$ . *Указание:* использовать функции **Factors** и **Collected**.
16. Известна гипотеза о том, что любое четное число  $n$ , большее чем 2, можно представить в виде суммы двух простых чисел. Проверьте ее для всех четных чисел  $n$ , не превышающих 10000.
17. Гипотеза Гольдбаха спрашивает, верно ли то, что любое нечетное число  $n$ , где  $n > 5$ , можно представить в виде суммы трех простых чисел. Проверьте ее для всех нечетных чисел  $n$ , не превышающих 1000.
18. Пусть  $t$  - произвольное натуральное число. Рассмотрим последовательность  $\{n_i\}$ , в которой  $n_1 = t$ , а остальные элементы определяются рекурсивно:  $n_{k+1} = n_k/2$ , если  $n_k$  - четное, и  $n_{k+1} = 3n_k + 1$ , если  $n_k$  - нечетное. Известна так называемая "гипотеза  $3k+1$ ", согласно которой такая последовательность достигает единицы для любого начального  $t$ . Проверьте ее для всех натуральных чисел  $t$ , не превышающих 10000. В вывод результатов включите количество шагов, которые требуются для достижения единицы.
19. Натуральное число  $n$  называется совершенным, если оно равняется сумме всех своих собственных делителей. Например,  $6 = 1+2+3$ ,  $28 = 1+2+4+7+14$ . Найдите все совершенные числа, не превышающие 10000.
20. Натуральные числа  $m$  и  $n$  называются дружественными, если каждое из них равняется сумме всех собственных делителей другого. Например, такими являются числа 220 и 284. Выясните, существуют ли другие пары дружественных чисел, не превышающих 1000.
21. Пусть  $t$  - произвольное натуральное число. Рассмотрим последовательность  $\{n_i\}$ , в которой  $n_1 = t$ , а остальные элементы определяются рекурсивно:  $n_{k+1}$  равняется сум-

ме всех собственных делителей числа  $n_k$ . Исследуйте поведение этой последовательности для всех натуральных чисел  $t$ , не превышающих 1000: когда она достигает единицы, стабилизируется, зацикливается; когда можно выдвинуть гипотезу о том, что она неограниченно возрастает?

## ССЫЛКИ

- [1] В.У. Грибанов and П.И. Титов. *Сборник упражнений по теории чисел*. Просвещение, М., 1964. [58](#)

# Индекс

`:=`, 17  
`;;`, 15  
`break`, 22  
`continue`, 23  
`do`, 19  
`elif`, 18  
`else`, 18  
`end`, 23  
`fi`, 18  
`for`, 19  
`function`, 23  
`if`, 18  
`in`, 19  
`local`, 15, 23  
`LogTo`, 9  
`mod`, 16  
`od`, 19  
`quit`, 9  
`Read`, 49  
`rec`, 34  
`repeat`, 19  
`return`, 24  
`then`, 18  
`until`, 19  
`while`, 19