

Erlang

Бьярн

Дэкер
(Bjarne
Däcker)

Роберт
Вирдинг
(Robert

Virding)³

Настольная книжка по Erlang

Бьярн Дэкер (Björn
Däcker) и Ро-
берт Вирдинг
(Robert Virding)

Версия:

Wed Sep 17

22:30:30 2014
+0200

Последняя вер-
сия этой кни-
ги находится по
адресу:

h t t

4



er⁵r

con

7
er

**Главный редак-
тор**

Омер Килич (Omer
Kilic)

Прочие участни- ки

Перевод на рус-
ский: Дмитрий Ли-
товченко (Dmytro
Lytovchenko)

Список людей, ко-
торые помогли с
изменениями и ис-
правлениями на-
ходится в репо-
зитории проекта.

9

Соглашения
Спецификации син
таксиса задают-

ся с помощью

ПО

Э

10

НО
ШИ
Р

—

—

Н

НО

11



ГО

Та

. В ква

ратные скобки (¹² []) заключаются неопределяемые части. Термы, начинающиеся с заглавной буквы, как например *Integer* должны быть заменены на какое-нибудь подходящее значение. Термы, начинающиеся со стр

13
ной буквы, как на-

er

пример
являются зарезер-
вированными сло-
вами языка Erlang.
Вертикальная чер-
та (|) разделяет аль-
тернативные ва-

14
рианты, как на-
пример Integer |
Float.

Ошибки и улучшения

Это живой документ, пожалуйста
присылайте исправ-
ления и рекоменда-
ции по улучше-
нию содержимо-
го используя си-

15
систему учёта задач
Github по адресу

h t t

/

/

g

c

o

m

17

esl

erl

Вы также може-

18
те создать свою
ветвь репозито-
рия (fork) и при-
слать нам запрос
на соединение вет-
вей (pull request)
с вашими пред-
лагаемыми исправ-
лениями и пред-
ложениями. Но-
вые ревизии это-
го документа бу-
дут публиковать-

19
ся по мере накопления больших изменений.



Этот текст доступен согласно условиям лицензии Creative Commons Attribution-ShareAlike 3.0 License. Вы имеете право копировать, распространять и передавать эту кни-

20
гу по условиям
лицензии, опи-
санным по ад-

ресу **ht**

21



C

O

r

g

lic

by -

3.0²³

Оглавление



1

Вступ- ление

или почему
Erlang такой
как он есть

Erlang — это ре-
зультат проекта

ГЛАВА 1 ВСТУПДЕН
по улучшению про-
граммирования пр
ложений для те-
лекоммуникаций
в Лаборатории Ком
пьютерных Наук
(CSLab или Comput
Science Lab) ком-
пании Ericsson. Кри
тически важным
требованием бы-
ла поддержка ха-

ГЛАВА 1. ВСТУПЛЕ
рактеристик этих
приложений, та-
ких, как:

- **Массивная парал-**
лельность
- **Устойчивость к**
сбоям
- **Изоляция**
- **Динамическое об-**
новление кода во
время его испол-
нения

ГЛАВА 1. ВСТУПЛЕНИЕ

- Транзакционность

В течение всей истории Erlang процесс его разработки был исключительно прагматичным. Характеристики и свойства видов систем, в которых была заинтересована компания Ericsson, пря-

ГЛАВА 1. ВСТУПЛЕНИЕ
мым образом влияли на ход разработки Erlang. Эти свойства считались настолько фундаментальными, что было решено поддержку для них вставить прямо в язык, вместо дополнительных библиотек. По причине

ГЛАВА 1. ВСТУПЛЕНИЕ
прагматичного процесса разработки вместо предварительного планирования, Erlang «структурно функциональным языком» — поскольку свойства функциональных языков очень хорошо подходили к свойствам систем, которые разраба-

ГЛАВА 1. ВСТУПЛЕНИЕ

Содержание



2

Структура Erlang- программ

2.1 Синтаксис модулей

ГЛАВА 2. СТРУКТУ
лей где каждый
модуль является
текстовым фай-
лом с расширени-
ем **.erl**. Для неболь-
ших программ все
модули обычно хра-
нятся в одной ди-
ректории. Модуль
состоит из атри-
бутов модуля и опр-
делений функций.

ГЛАВА 2. СТРУКТУ

— mo

— ex

ГЛАВА 2. СТРУКТУ

дош

ГЛАВА 2. СТРУКТУРА

tim

Показанный мо-

дуль den

ГЛАВА 2. СТРУКТУРА

СОСТОИТ ИЗ ФУНК-

ции

•

т i

которая являет-
ся локальной для
модуля и функ-

ГЛАВА 2. СТРУКТУ

ции do

которая экспор-
тирована и может
быть вызвана сна-
ружи модуля.

ГЛАВА 2. СТРУКТУ

dem

20 (стр

⇒

ка ⇒ читается как
«даёт результат»)

ГЛАВА 2. СТРУКТУ

дои

означает, что перед нами функция «double» с *од-*
ним аргументом.

ГЛАВА 2. СТРУКТУРА

Функция

do

принимает *два* аргумента и считается другой, отличной от первой, функцией. Количество аргументов называется арг-

ГЛАВА 2. СТРУКТУРНОСТЬЮ (arity) данной функции.

2.2 Атрибутируемость модулей

Атрибут модуля определяет некоторое свойство модуля.

ГЛАВА 2. СТРУКТУ-
ДУЛЯ И СОСТОИТ ИЗ
МЕТКИ И ЗНАЧЕ-
НИЯ В СЛЕДУЮЩЕМ
ВИДЕ:

— М е

ГЛАВА 2. СТРУКТУРА

Мет

должна быть ато-
мом, в то время,

как значение

З

Ченение

должно быть термом (смотрите главу ??). Можно указывать любое имя для атрибута мо-

ГЛАВА 2. СТРУКТУ
дуля, также раз-
решены повторы.

Атрибуты хранят-
ся в скомпилиро-
ванном коде и мо-
гут быть извлече-
ны с помощью вы-

зова функции

M

ГЛАВА 2. СТРУКТУ

ДУЛ

ГЛАВА 2. СТРУКТУРА

2.2.1 Предопределённые атрибуты модулей

Предопределённые атрибуты модулей должны быть размещены в модуле до начала первого определения

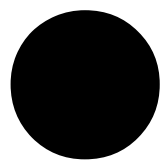
ГЛАВА 2. СТРУКТУРА функции.

• — mo

Этот атрибут обязательен и должен идти первой строкой файла. Он опре

ГЛАВА 2. СТРУКТУ
деляет имя мо-
дуля. Имя Module,
атом (смотрите
раздел ??), долж-
но соответство-
вать имени фай-
ла без расшире-

НИЯ



е

ГЛАВА 2. СТРУКТУ

• — ex

кпн

ГЛАВА 2. СТРУКТУ

НО —

СТЬ

Этот атрибут указывает, какие функции модуля мо-

ГЛАВА 2. СТРУКТУ
гут быть вызва-
ны снаружи мо-
дуля. Каждое имя

функции

Ф

ГЛАВА 2. СТРУКТУРА

— это атом и

НО

—

ГЛАВА 2. СТРУКТУ

СТЬ

функции — це-
лое число.

•

—

•

im

ГЛАВА 2. СТРУКТУ

КНУ

НО —

ГЛАВА 2. СТРУКТУ

СТЬ

Этот атрибут ука-

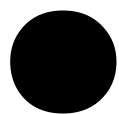
зывает модуль

Л

ГЛАВА 2. СТРУКТУ **ДУЛ**

из которого им-
портируется спи-
сок функций. На-
пример:

ГЛАВА 2. СТРУКТУРА



import

Эта запись означает, что теперь

ГЛАВА 2. СТРУКТУРА

МОЖНО ПИСАТЬ

ВМЕСТО БОЛЕЕ ДЛИННО

ГЛАВА 2. СТРУКТУРА

ной записи

d

которая может
быть непрактич-
на, если функ-
ция используется
часто.

ГЛАВА 2. СТРУКТУРА

• — СО

Параметры компилятора для данного модуля.

ГЛАВА 2. СТРУКТУРА

• — VS

Версия модуля.
Если не указать
этот атрибут, то
по умолчанию бу-
дет использована
на контрольная

ГЛАВА 2. СТРУКТУРА

сумма содержи-
мого модуля.

• — be

Этот атрибут ука-
зывает поведе-
ние модуля, ли-
бо выбранное пол

ГЛАВА 2. СТРУКТУ зователем, либо одно из стандарт- ных поведений

ОТР: **ge**

ГЛАВА 2. СТРУКТУ

gen

gen

ГЛАВА 2. СТРУКТУРА

или **S U**

Принимаются как
британская (behav
так и американ-
ская запись (behav

ГЛАВА 2. СТРУКТУРА

2.2.2 Определ записей и мак- росов

Записи и макро-
сы определяются
так же, как и
другие атрибуты
модуля:

ГЛАВА 2. СТРУКТУ

— r e

— d e

ГЛАВА 2. СТРУКТУ
Записи и определения макросов также позволяют-
ся между функци-
ями, если опре-
деление встреча-
ется раньше, чем
его первое исполь-
зование. (Подроб-
нее о записях чи-
тайте секцию ??,
а о макросах гла-
ву ??.)

ГЛАВА 2. СТРУКТУ

2.2.3 Включе содер- жимо- го фай- лов

Включение содер-
жимого файлов ука-
зывается анало-
гично другим ат-
рибутам модуля:

ГЛАВА 2. СТРУКТУ

— in

— in

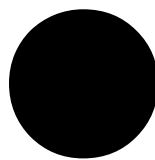
ГЛАВА 2. СТРУКТУРА

File

— это строка, представляющая собой имя файла. Включаемые файлы обычно используются для определений записей и макро-

ГЛАВА 2. СТРУКТУ
сов, которые ис-
пользуются в неско-
ких модулях сра-
зу. По договорён-
ности для вклю-
чаемых файлов ис-
пользуется расши-

рение



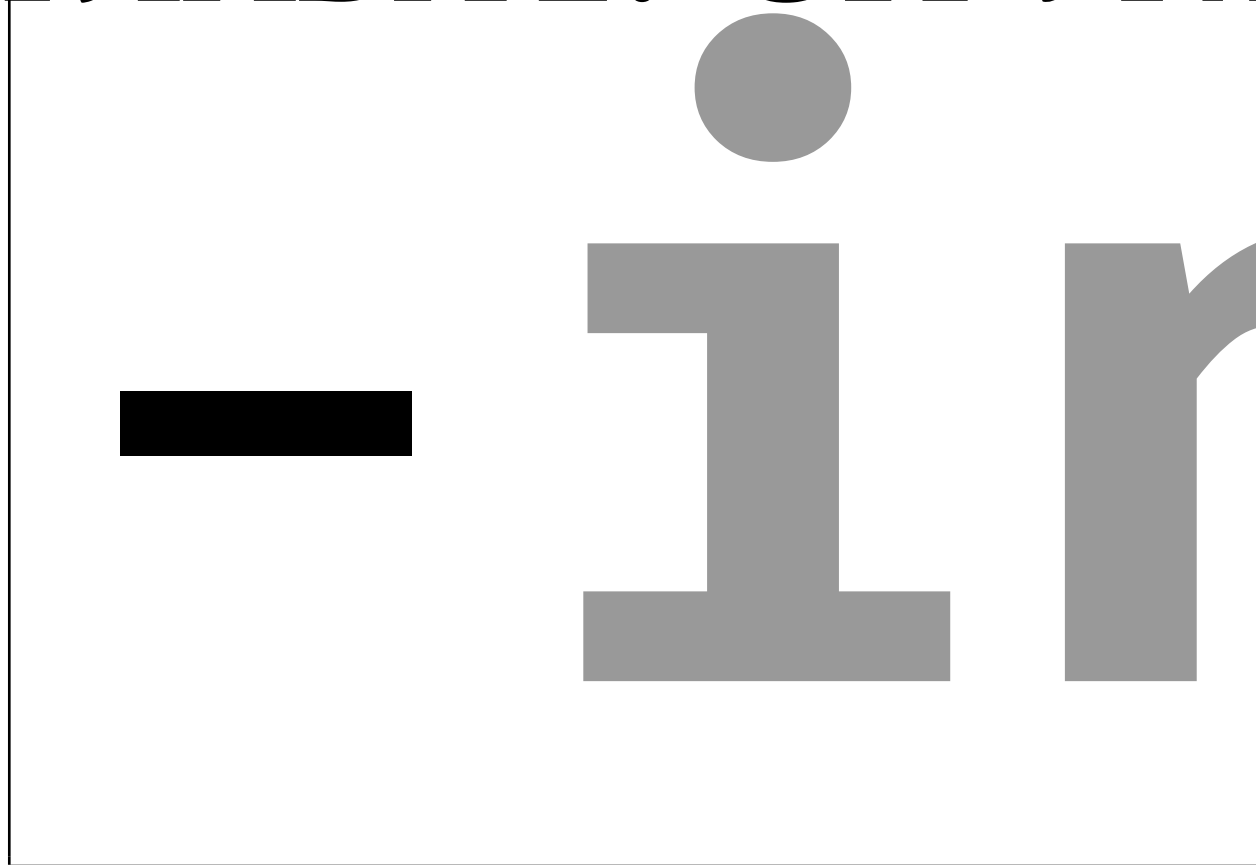
h

ГЛАВА 2. СТРУКТУ

— in

— in

ГЛАВА 2. СТРУКТУ



Если путь к фай-

лу

Fi

ГЛАВА 2. СТРУКТУ
начинается с ком-

понента пути

то значение пе-
ременной окру-

жения

\$

Va

ГЛАВА 2. СТРУКТУ
(которое возвра-
щается функци-

ей OS :

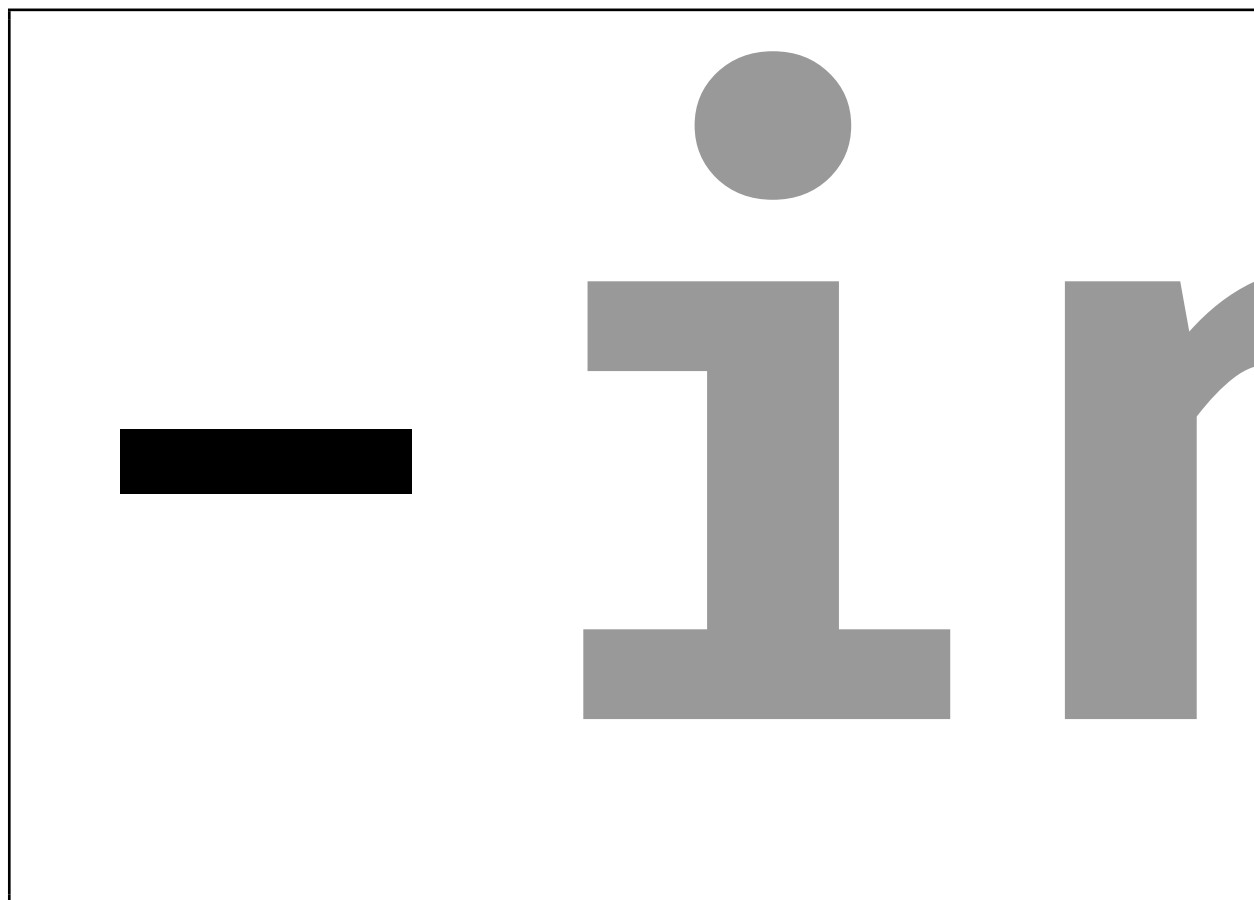
будет подставле-

ГЛАВА 2. СТРУКТУ

НО ВМЕСТО



ГЛАВА 2. СТРУКТУ

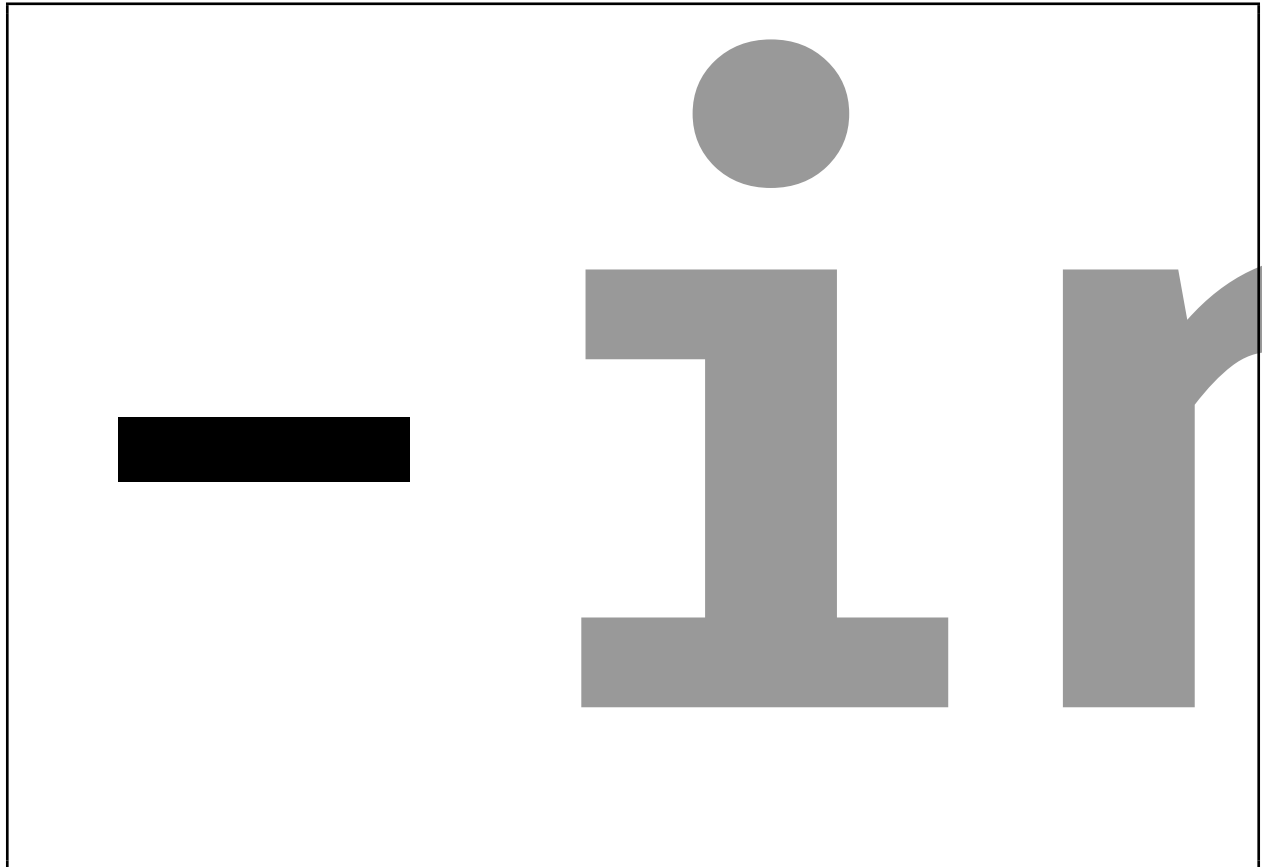


ГЛАВА 2. СТРУКТУРА

Директива 1

подобна
но считается, что
первый компонент

ГЛАВА 2. СТРУКТУРА пути — имя при- ложения.



Сервер кода (спе-
циальный процесс
управляющий за-
грузкой и верси-

ГЛАВА 2. СТРУКТУ- ЯМИ МОДУЛЕЙ В ПА- МЯТИ) ИСПОЛЬЗУ-

ет функцию **С**
для поиска дирек-
тории текущей (све-
жайшей) версии

ГЛАВА 2. СТРУКТУРА

модуля
и затем в подди-

ректории
производится по-
иск нужного фай-

ГЛАВА 2. СТРУКТУ

ла

2.3 Коммме

Комментарии могут появляться в любом месте мо-

ГЛАВА 2. СТРУКТУ
дуля, кроме как
внутри строк и ато-
мов, заключённых
в кавычки. Ком-
ментарий начи-
нается с симво-

ла процента (
и действителен до
конца строки но
не включая сим-

ГЛАВА 2. СТРУКТУ
вол конца стро-
ки. Завершающий
строку символ кон-
ца строки, с точ-
ки зрения компи-
лятора, действу-
ет как пробел.

ГЛАВА 2. СТРУКТУРА

2.4 Кодировка файлов

Erlang работает с полным набором символов Latin-1 (ISO-8859-1). Таким образом, можно использовать и выводить на экран все печатные сим-

ГЛАВА 2. СТРУКТУРА
волы из набора Latin-1 без цитирования с помощью обратной косой черты (\). Атомы и переменные могут использовать все символы из набора Latin-1.

Примечание: Начиная с версии R16 допускается ис-

ГЛАВА 2. СТРУКТУ
пользователь кодиро-
ровку UTF-8 для
исходных файлов,
но этот режим нуж-
но включать па-
раметром команд-
ной строки ком-
пилятора. Начи-
ная с версии R17,
исходные файлы
имеют кодиров-
ку UTF-8 по умол-

ГЛАВА 2. СТРУКТУ- *чанию.*

ГЛАВА 2. СТРУКТУРА	
Классы символов в кодировке	
Водяное рифление	Класс
4032!	Символы
5747	Функ- ции
# \$ % & ,	
60480	Десятич

ГЛАВА 2. СТРУКТУРА

2.5 Зарезервированные слова

Следующие ключевые слова в Erlang зарезервированы, и не могут использоваться в качестве атомов (для использования одного из ключевых слов в качестве ато

ГЛАВА 2. СТРУКТУ
ма, оно должно быть
заключено в оди-
ночные кавычки):

ГЛАВА 2. СТРУКТУ

aft

cat

ГЛАВА 2. СТРУКТУ

rem

3

Типы данных (тер- мы)

3.1 Унарные (одиночные) типы

ГЛАВА 3. ТИПЫ ДА
же известное, как
литерал. Атомы
начинаются со стро
ной латинской бук
вы и могут содер
жать буквенно-циф
символы, подчёр
кивания ()




ГЛАВА 3. ТИПЫ ДА

и символ *at* (

@

Как вариант, ато-
мы могут быть ука-
заны с помощью
заклЮчения в оди-
ночные кавычки

ГЛАВА 3. ТИПЫ ДА



(), это необходимо, если атом начинается с заглавной буквы, или содержит другие символы, кроме подчёркиваний и

ГЛАВА 3. ТИПЫ ДА

at (**@**). Напри-
мер:

ГЛАВА 3. ТИПЫ ДА

h e t
p h o

ГЛАВА 3. ТИПЫ ДА

’

МО

’

рh

ГЛАВА 3. ТИПЫ ДА

'

An

•

ins

ГЛАВА 3. ТИПЫ ДА

quo

\n\

(см. раздел ??)

ГЛАВА 3. ТИПЫ ДАННЫХ

3.1.2 Истина и ложь

В Erlang нет специального типа данных для логических значений. Эту роль выполняют

атомы

tr

ГЛАВА 3. ТИПЫ ДА

f a l

и

2

=

ГЛАВА 3. ТИПЫ ДА

З

t

⇒

t

r

u

ГЛАВА 3. ТИПЫ ДА

**or
tru**



ГЛАВА 3. ТИПЫ ДАННЫХ

3.1.3 Целые числа

В дополнение к обычному способу записи **целых чисел Erlang** предлагает ещё ряд способов записи. За-

ГЛАВА 3. ТИПЫ ДА

пись

\$C

даёт числовое зна-
чение для симво-

ла,

Cha

ГЛАВА 3. ТИПЫ ДА
В кодировке Latin-
1 (это также мо-
жет быть и непе-
чатный символ)
и запись

Bas

— даёт целое чис-
ло, записанное с

ГЛАВА 3. ТИПЫ ДА

В

основанием
основание долж-
но быть целым чис-
лом в диапазоне

2..36 •

42

⇒

ГЛАВА 3. ТИПЫ ДА

4265

\$A

⇒

ГЛАВА 3. ТИПЫ ДАННЫХ

\$\backslash n\$

10

(see

⇒

section ??)

ГЛАВА 3. ТИПЫ ДА

2#1
5

⇒

ГЛАВА 3. ТИПЫ ДА

16#

31

⇒

ГЛАВА 3. ТИПЫ ДАННЫХ

3.1.4 Действия с плавающей точкой

Действительное число с плавающей точкой записывается, как

ГЛАВА 3. ТИПЫ ДА

ОСН

ОС -

где

ГЛАВА 3. ТИПЫ ДА

**НО —
Ва —
нии е**

— это действитель-
ное число в диа-

ГЛАВА 3. ТИПЫ ДА пазоне от 0.01 до

10000 и

П
Д

Э
О
е

К
—
Н

ГЛАВА 3. ТИПЫ ДА

Та (необ.

зательное) — это
целое число со зна-
ком, указывающее
экспоненту (сте-
пень десятки, на
которую множит-

ГЛАВА 3. ТИПЫ ДА

Ос —
ся
Но —
Ва —

ГЛАВА 3. ТИПЫ ДА **НИЕ**

Например:

2.3

ГЛАВА 3. ТИПЫ ДА

3 2
⇒
(соответ-
ствует $2.3 \cdot 10^{-3}$)

ГЛАВА 3. ТИПЫ ДАННЫХ

3.1.5 Ссылочные значения

Ссылка — это термин, значение которого является уникальным в системе в момент исполнения Erlang, который создаётся встроенной функ-

ГЛАВА 3. ТИПЫ ДА

цией **ma**

(Для дополнительной информации по встроенным функциям, или *BIF*-ам, смотрите раздел ??.)

ГЛАВА 3. ТИПЫ ДА

3.1.6 Порты

Идентификатор порта указывает на открытый в системе порт (смотрите главу ?? про порты).

ГЛАВА 3. ТИПЫ ДАННЫХ

3.1.7 Идентификатор процессов (Pid)

Идентификатор процесса, или *pid*, указывает на процесс в системе (смотрите главу ?? о процессах).

ГЛАВА 3. ТИПЫ ДАННЫХ

3.1.8 Анонимные функции

Тип данных *fun* идентифицирует анонимную функцию или функциональный объект (см. раздел ??).

ГЛАВА 3. ТИПЫ ДАННЫХ

3.2 Составные типы данных

3.2.1 Кортежи

Кортеж — это составной тип данных, который содержит фиксированное количе-

ГЛАВА 3. ТИПЫ ДА-
ство термов, за-
ключённое {в фи-
гурные скобки}.
Например:

{ T e

ГЛАВА 3. ТИПЫ ДА

Каждый из

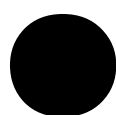
МХ в кор-
теже называется
элементом. Ко-
личество элемен-

ГЛАВА 3. ТИПЫ ДА
ТОВ называется раз
мером данного кор
тежа.

ГЛАВА 3. ТИПЫ ДАННЫХ

ВІФ-
функции
для ра-
боты с
кортежа-
ми

S



1. Размер
размер

К

С

ГЛАВА 3. ТИПЫ ДА

P

=

{

ad

ГЛАВА 3. ТИПЫ ДА

24,

{ j u

ГЛАВА 3. ТИПЫ ДА

29 }

Р

связано



ГЛАВА 3. ТИПЫ ДА

c

{ a d

24,

ГЛАВА 3. ТИПЫ ДА

{

•

j

u

2

9

}

ГЛАВА 3. ТИПЫ ДА

eTe

P)

⇒

ГЛАВА 3. ТИПЫ ДА

ada

ete

ГЛАВА 3. ТИПЫ ДА

Р

)

⇒

{

•

j

u

ГЛАВА 3. ТИПЫ ДА

P2

set

ГЛАВА 3. ТИПЫ ДА

P

,

P'2

СВЯ



ГЛАВА 3. ТИПЫ ДА

зано с

25,

{a

ГЛАВА 3. ТИПЫ ДА

{

•

ј

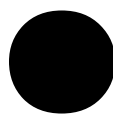
ш

2

9

}

ГЛАВА 3. ТИПЫ ДА

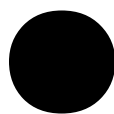


Siz

3



ГЛАВА 3. ТИПЫ ДА



Siz

0



ГЛАВА 3. ТИПЫ ДАННЫХ

3.2.2 Записи

Запись это именованный кортеж, имеющий именованные элементы, которые называются полями. Тип записи определяется в виде атрибута модуля, например:

ГЛАВА 3. ТИПЫ ДА

— r e

ГЛАВА 3. ТИПЫ ДА
Здесь имя запи-

За-
СИ
ПИС

ГЛАВА 3. ТИПЫ ДА

и имена полей

лех

— атомы и каж—

ГЛАВА 3 ТИПЫ ДА

дое **По**

лех

может получить

необязательное зна
чение по умолча-

ГЛАВА 3 ТИПЫ ДА

НИЮ

Зн
че
ни

—

—

ГЛАВА 3. ТИПЫ ДА

еХ. Это определение может быть помещено в любом месте модуля между определениями функций, но обязательно до первого использова-

ГЛАВА 3. ТИПЫ ДА
ния. Если тип за-
писи используется
в нескольких
модулях, рекомен-
дуется поместить
его в отдельный
файл для вклю-
чения.

Новая запись с ти-

пом За

ГЛАВА 3. ТИПЫ ДА

ПИС

создаётся с помощью следующего выражения:

ГЛАВА 3. ТИПЫ ДА

#3a

Поля не обязательно должны идти в том же порядке, как и в определении записи. Пропущенные по-

ГЛАВА 3. ТИПЫ ДА
ля получают свои
соответствующие
значения по умол-
чанию. Если ис-
пользована послед-
няя запись (под-
чёркивание рав-

но **Вы** —

ГЛАВА 3. ТИПЫ ДА

ра

—

же

—

ни

—

ГЛАВА 3 ТИПЫ ДА

е

Л

), то

все оставшиеся по-
ля получают зна-

В

ы

чение

ГЛАВА 3. ТИПЫ ДА

ра

—

же

—

ни

—

ГЛАВА 3. ТИПЫ ДА

е**л**. По-
ля без значений
по умолчанию и
пропущенные по-
ля получают зна-

чение **un**

ГЛАВА 3. ТИПЫ ДА
Значение поля мож
но получить ис-
пользуя выраже-

П е
ние
ре -

ГЛАВА 3. ТИПЫ ДА

Мен

ная

ГЛАВА 3. ТИПЫ ДА

— mo

— ex

ГЛАВА 3. ТИПЫ ДА

— **re**

ГЛАВА 3. ТИПЫ ДА

new

ГЛАВА 3. ТИПЫ ДА

Здесь функция

может быть использо-
вана в другом
модуле, который
также должен вклю-
чить файл, содер-
жащий определе-

ГЛАВА 3. ТИПЫ ДА
НИЕ ИСПОЛЬЗОВАНА
НОЙ ЗДЕСЬ ЗАПИ-

СИ **ре 1**

ГЛАВА 3. ТИПЫ ДА

{ R

emp r

ГЛАВА 3. ТИПЫ ДА

{pre

⇒

ern

ГЛАВА 3. ТИПЫ ДА

4

4

!

e

r

1

ГЛАВА 3. ТИПЫ ДА

Р # р

4 4



ГЛАВА 3. ТИПЫ ДАННЫХ

R # r

e I I

⇒

При работе с за-
писями в интер-

ГЛАВА 3. ТИПЫ ДА претаторе Erlang, МОЖНО ИСПОЛЬЗО- вать функции

rd(

ГЛАВА 3. ТИПЫ ДА

**О пр
де _**

ГЛАВА 3. ТИПЫ ДА

де —

ни —

еза

ГЛАВА 3. ТИПЫ ДА

ПИ —

СИ)

ГЛАВА 3. ТИПЫ ДА

и **И** **И** **(**

для того, чтобы
определить или
загрузить новые
определения за-
писей. Подробнее
читайте в доку-

ГЛАВА 3. ТИПЫ ДАННЫХ

ментации

Erlang Reference Manual

3.2.3 СПИСКИ

Список — это составной тип данных, который содержит *переменное* количество **термов**, заключённое в квадратных скобках.

ГЛАВА 3. ТИПЫ ДА КАХ.

Г Т е

ГЛАВА 3. ТИПЫ ДА

Каждый

МХ в спис-
ке называется эле-
ментом. Количе-
ство элементов в
списке называет-

**ГЛАВА 3. ТИПЫ ДА-
СЯ ДЛИНОЙ СПИС-
КА. Как это при-
нято в функцио-
нальном програм-
мировании, пер-
вый элемент на-
зывается ГОЛОВОЙ
СПИСКА, а ОСТАТОК
(начиная с 2го эле-
мента и до КОН-
ца) называется ХВО-
СТОМ списка. За-
метьте, что отдель-**

ГЛАВА 3. ТИПЫ ДА-
ННЫЕ ЭЛЕМЕНТЫ В
СПИСКЕ НЕ ДОЛЖ-
НЫ БЫТЬ ОДИНА-
КОВОГО ТИПА, ХО-
ТЯ ЧАСТО ПРАКТИ-
КУЕТСЯ (И ЭТО, НА-
ВЕРНОЕ, ДАЖЕ УДОБ-
НО), ИМЕТЬ В СПИС-
КЕ ЭЛЕМЕНТЫ ОДИ-
НАКОВОГО ТИПА —
КОГДА ПРИХОДИТ-
СЯ РАБОТАТЬ С ЭЛЕ-
МЕНТАМИ РАЗНЫХ

ГЛАВА 3. ТИПЫ ДА **ТИПОВ, ОБЫЧНО ИС-** **ПОЛЬЗУЮТСЯ ЗАПИ-** **СИ.**

ГЛАВА 3. ТИПЫ ДАННЫХ

Встроенные
функции
для работы
со списками

len

Возвращает
длину

Список

ГЛАВА 3. ТИПЫ ДА

Оператор «вертикальная черта» (|, также в некоторых книгах по ФП

он называется
отделяет ведущие
элементы списка
(один или более)
от остальных элементов. Например

ГЛАВА 3. ТИПЫ ДА

**Г
Н
Т
Л**

ГЛАВА 3. ТИПЫ ДА

1,
2,

ГЛАВА 3. ТИПЫ ДА

4

,

н

=

1

⇒

ГЛАВА 3. ТИПЫ ДА

и Т =

3,

ГЛАВА 3. ТИПЫ ДА

5

1

Г

Х

,

ГЛАВА 3. ТИПЫ ДА

Y

I

Z

J

ГЛАВА 3. ТИПЫ ДА

Г

а

,

б

,

ГЛАВА 3. ТИПЫ ДА

d
х' = с



ГЛАВА 3. ТИПЫ ДА

Y = b

Z =

И

ГЛАВА 3. ТИПЫ ДАННЫХ

d,

Список — рекурсивная структура. Неявно список завершается ссылкой на пустой

ГЛАВА 3. ТИПЫ ДА

СПИСОК, ТО ЕСТЬ

ЭТО ТО ЖЕ САМОЕ,

ГЛАВА 3. ТИПЫ ДА

как и

Ла

Список, выглядя-

ГЛАВА 3. ТИПЫ ДА

щий как

называется плохо сформированным (badly formed) и такой записи следует избегать (по-

ГЛАВА 3. ТИПЫ ДА

тому что атом ,
завершающий стру
туру списка сам
не является спис-
ком). Списки на-
турально способ-
ствуют рекурсив-
ному функциональ-
ному программи-
рованию.

ГЛАВА 3. ТИПЫ ДАННЫХ

Например, следующая функция

вычисляет сумму списка и функция

`do` умножает каждый элемент списка на заданное значение

ГЛАВА 3. ТИПЫ ДА
элемент списка на
2, при этом кон-
струируя и возвра-
щая новый спи-
сок по ходу вы-
полнения.

ГЛАВА 3. ТИПЫ ДА

sum

sum

ГЛАВА 3. ТИПЫ ДА

дош

дош

ГЛАВА 3. ТИПЫ ДА
Определения функций выше представляют собою сопоставление с образцом, которое описано далее в главе ??.

Образцы в такой записи часто встречаются в рекурсивном программировании, неявно предоставляя

ГЛАВА 3. ТИПЫ ДА
«базовый случай»
(для пустого спис-
ка в ЭТИХ приме-
рах).

Для работы со спис

ками, оператор
соединяет вместе
два списка (при-
соединяет второй

ГЛАВА 3. ТИПЫ ДА

аргумент к пер-
вому) и возвра-
щает список-резул-

Оператор

создаёт спи-
сок, который яв-
ляется копией пер-
вого аргумента, за-
тем исключени-
ем, что для каж-

ГЛАВА 3. ТИПЫ ДА
дого элемента во
втором списке его
первое вхождение
в результат (ес-
ли такое было) уда-
ляется.

ГЛАВА 3. ТИПЫ ДА

[

1

,

+

+

ГЛАВА 3. ТИПЫ ДА

[

1

,

⇒

[

1

,

ГЛАВА 3. ТИПЫ ДА



[

3

я

ГЛАВА 3. ТИПЫ ДА

Подборка функций, работающих со списками может быть найдена в модуле стандартной библиотеки с названи-

ем

lis

ГЛАВА 3. ТИПЫ ДАННЫХ

3.2.4 Строки

Строки — это цепочки символов, заключённые между двойными кавычками, на самом деле они хранятся в памяти, как списки целых чисел — символов.

ГЛАВА 3. ТИПЫ ДА

”

ab

то же самое, как

ГЛАВА 3. ТИПЫ ДА

И

[97

то же самое, как

И

[]

ГЛАВА 3. ТИПЫ ДА
Две строки, записанные подряд без разделительных знаков и операторов будут соединены в одну во время компиляции и не принесут дополнительных затрат по соединению во время исполнения.

ГЛАВА 3. ТИПЫ ДА

”

S t

4

2

”

ГЛАВА 3. ТИПЫ ДАННЫХ

”

St

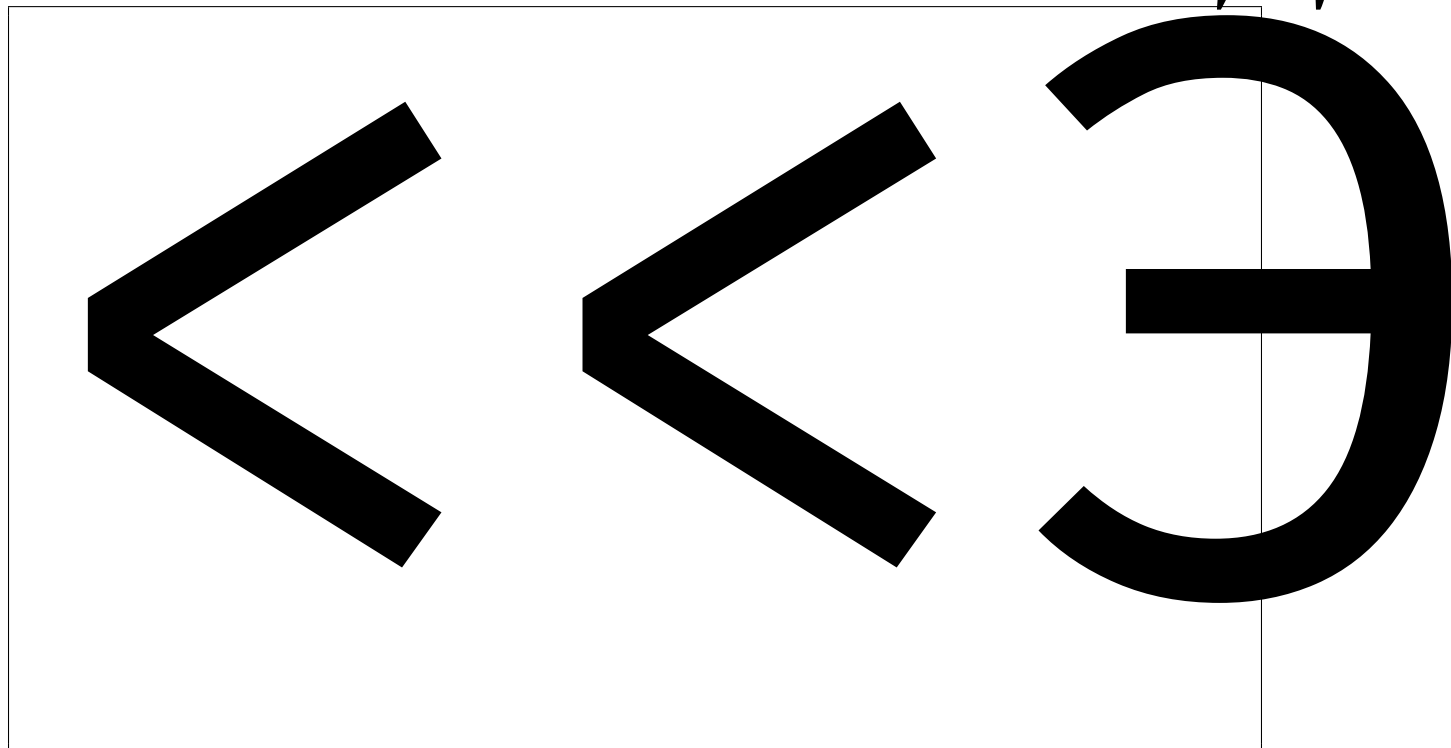
⇒

3.2.5 Двоичные данные

Двоичные данные — это блок

ГЛАВА 3. ТИПЫ ДА-
нетипизирован-
ной памяти, по умо-
чанию двоичные
данные являют-
ся последователь-
ностью 8-битных
элементов (бай-
тов).

ГЛАВА 3. ТИПЫ ДА



ГЛАВА 3. ТИПЫ ДА

Каждый

Мен

ТХ ука-

ГЛАВА 3. ТИПЫ ДА
зывается в виде

зна

ГЛАВА 3. ТИПЫ ДА

Спецификация элемента двоичных данных

ЗРЖа

Выражения, состоящие из
двоичных данных, необязательно
вычисляются спецификацией

ГЛАВА 3. ТИПЫ ДАННЫХ

Спецификаторы типов

Тип
дан-
ных

По
умол-
чанию

ГЛАВА 3. ТИПЫ ДА

Значение

ме
ра

умно-

ГЛАВА 3. ТИПЫ ДА
жаётся на едини-
цу измерения и
даёт число бит, ко-
торые может за-
нять данный сег-
мент двоичных дан-
ных. Каждый сег-
мент может состо-
ять из нуля или
более битов, но
общая сумма би-
тов должна делить-
ся нацело на 8, ина-

ГЛАВА 3. ТИПЫ ДА
че во время ис-
полнения возник-

нет ошибка

в

Также сегмент с

ГЛАВА 3. ТИПЫ ДА

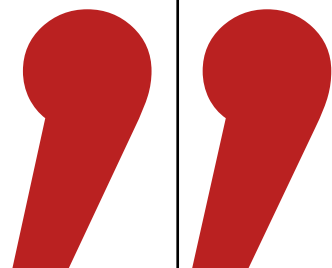
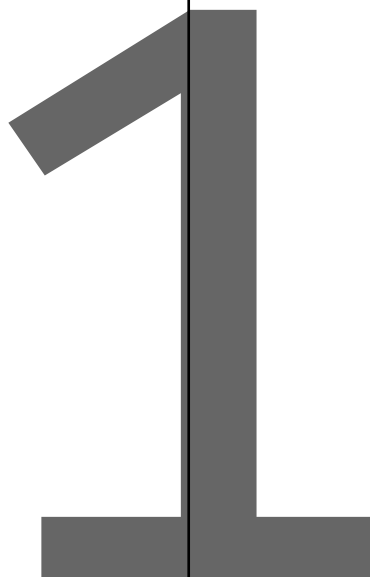
ТИПОМ

в 1

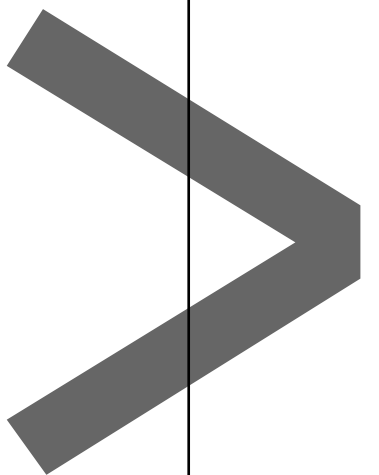
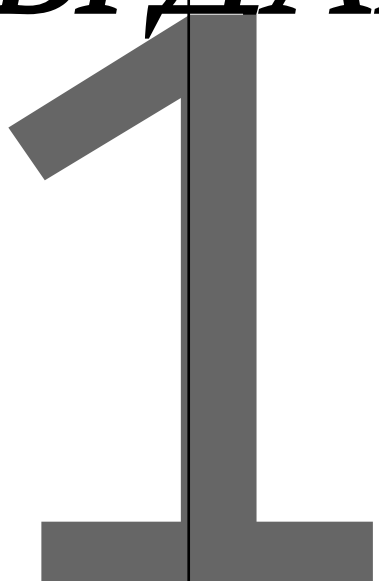
должен иметь размер, делящийся нацело на 8.

Двоичные данные не могут вкладываться друг в друга.

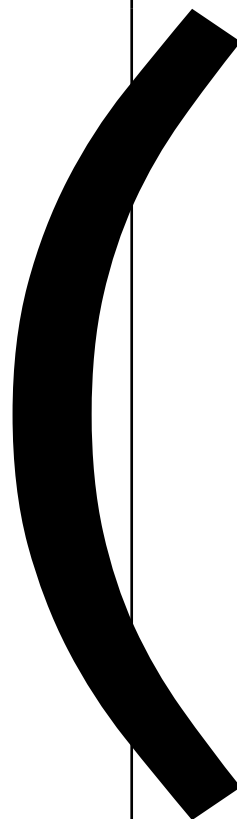
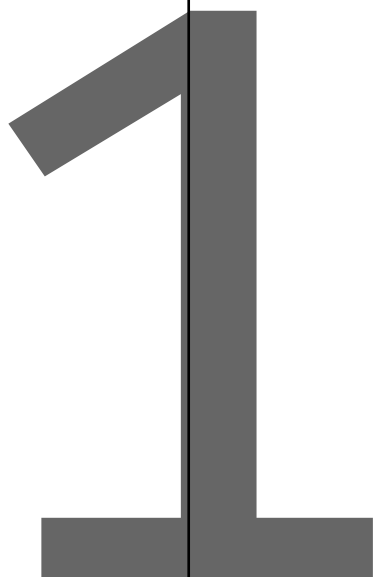
ГЛАВА 3. ТИПЫ ДА



ГЛАВА 3. ТИПЫ ДА



ГЛАВА 3. ТИПЫ ДА



ГЛАВА 3. ТИПЫ ДАННЫХ

3.3. Escape-последовательности

Escape-последовательности разрешено использовать в строках и атомах, которые заключены в кавычки, они позволяют ввести в исходный текст программы символ,

ГЛАВА 3. ТИПЫ ДА КОТОРЫИ ДРУГИМ СПОСОБОМ ВВЕСТИ ТРУДНО ИЛИ НЕВОЗ- МОЖНО.

ГЛАВА 3. ТИПЫ ДАННЫХ	
Escapes-последовательно	
\b	Backspace (удаление слева)
\d	Delete (удаление справа)
\e	Escape
\f	Новая страница (при

ГЛАВА 3. ТИПЫ ДА

3.4 Преобр ТИПОВ

Erlang — строго типизированный язык, то есть неявные автоматические преобразования типов в нём не происходят. Но есть ряд встроенных в стандарт-

ГЛАВА 3. ТИПЫ ДАННЫХ

ную библиотеку функций, предназначенных для преобразования между типами данных при участии программиста:

ГЛАВА 3. ТИПЫ ДА Преобразования ТИПОВ

	atomic	integer	float	binary
atom	-	-	-	XX
integer	X	-	-	XX
float	X	-	-	XX
pid	-	-	-	XX
fun	-	-	-	XX
tuple	-	-	-	XX
list	XX	XXXX		X
binary	X	XXXX		

ГЛАВА 3. ТИПЫ ДАННЫХ

Встроенная функция

ция

float

переводит целые
числа в числа с пла-
вающей точкой.

Встроенные функ-

ГЛАВА 3. ТИПЫ ДА

ции **to**

tr
и

переводят числа

ГЛАВА 3. ТИПЫ ДА С ПЛАВАЮЩЕЙ ТОЧ- КОЙ ОБРАТНО В це- лые, округляя или отбрасывая дроб- ную часть.

ГЛАВА 3. ТИПЫ ДА

Функции

и

ГЛАВА 3. ТИПЫ ДА переводят различ- ные типы в спис- ки (строки) и из СПИСКОВ.

ГЛАВА 3. ТИПЫ ДА

Функции

и b i n

ГЛАВА 3. ТИПЫ ДА
переводят любое
значение в зако-
дированную дво-
ичную форму и об-
ратно (Подробнее:

h t t

ГЛАВА 3. ТИПЫ ДА

/ / e

o r g

ГЛАВА 3. ТИПЫ ДА

dos

арр

ГЛАВА 3. ТИПЫ ДА

e r t

e r t

ГЛАВА 3. ТИПЫ ДА

ext

dis

ГЛАВА 3. ТИПЫ ДА

html

ГЛАВА 3. ТИПЫ ДА

ato

lis

ГЛАВА 3. ТИПЫ ДА

f l o

l i s

ГЛАВА 3. ТИПЫ ДА

int

lis

ГЛАВА 3. ТИПЫ ДА

тцр

lis

ГЛАВА 3. ТИПЫ ДА



р

і

d

t

e

r

ГЛАВА 3. ТИПЫ ДА

teir

ГЛАВА 3. ТИПЫ ДА

bin

ГЛАВА 3. ТИПЫ ДА

teir



4

264

Сопо- ставле- ние с образ- цом

4.1 Переме

Переменные пред-
ставлены, как ар-

ГЛАВА 4. СОПОСТА
гументы функции
или как резуль-
тат сопоставления
с образцом. Пе-
ременные начи-
наются с заглав-
ной буквы или сим-
вола подчёркива-
ния () и мо-


гут содержать букв
цифровые симво-

ГЛАВА 4...СОПОСТА лы, подчёркива- ния и символы *at*

(*a*). Перемен-
ные могут быть
связаны со зна-
чением (присво-
ены) только один
раз.

ГЛАВА 4. СОПОСТА

Abc

A_ c

ГЛАВА 4. СОПОСТА

Апо

Не

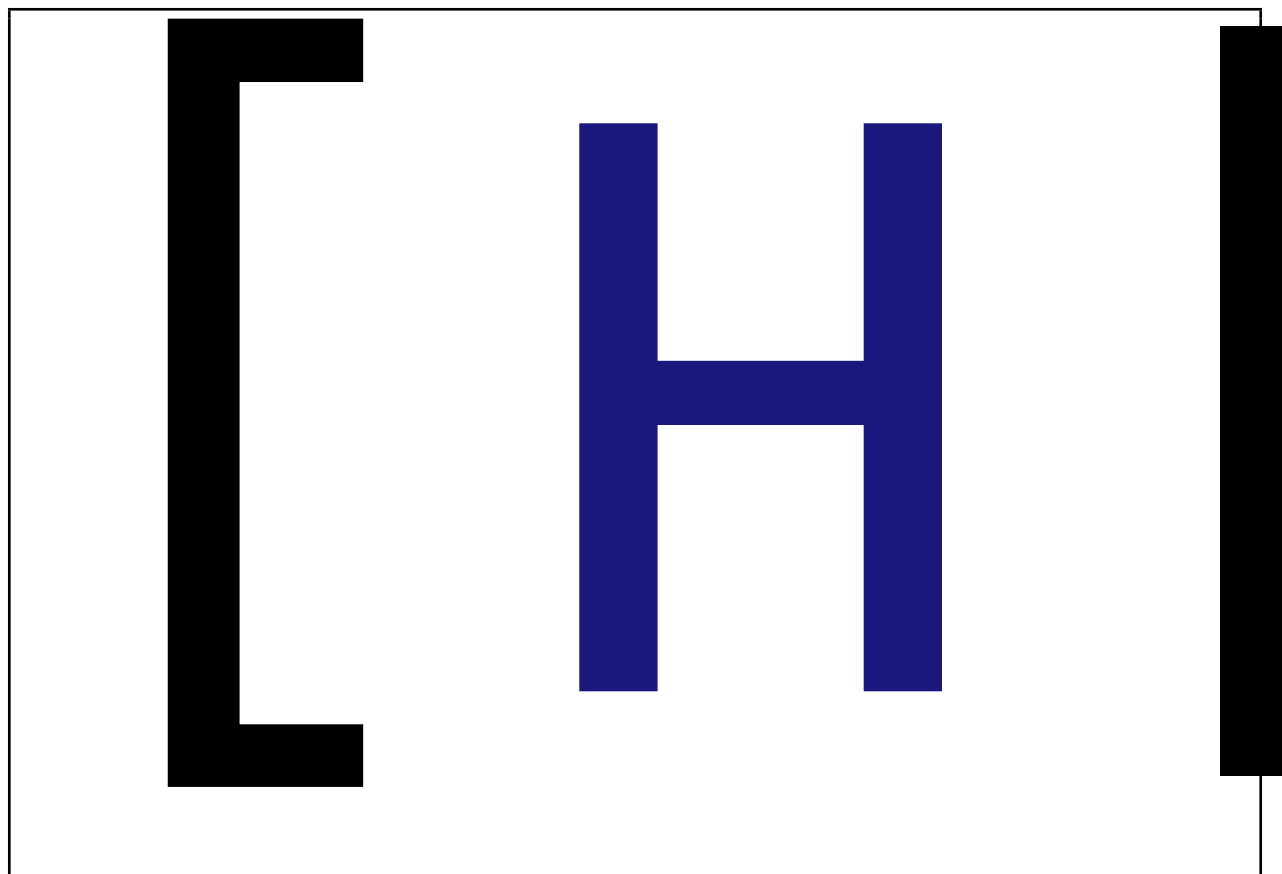


ГЛАВА 4. СОПОСТА

Анонимная переменная объявляется с помощью одного символа подчёркивания (

и может использоваться там, где требуется переменная, но её значение нас не интересует и может быть проигнорировано

ГЛАВА 4. СОПОСТАВ- НО.



Переменные, на-
чинающиеся с сим-
вола подчёркива-
ния, как, напри-

ГЛАВА 4. СОПОСТА

Н

мер,



являются обыч-
ными не аноним-
ными переменны-
ми. Однако они
игнорируются ком-
пилятором в том
смысле, что они

ГЛАВА 4. СОПОСТА-
не производят пре-
дупреждений ком-
пилятора о неис-
пользуемых пе-
ременных. Таким
образом, возмож-
на следующая за-
пись:

ГЛАВА 4. СОПОСТА

mem

ВМЕСТО:

ГЛАВА 4. СОПОСТА

т е т

ГЛАВА 4. СОПОСТА
что улучшает чи-
таемость кода.

*Область видимо-
сти* для перемен-
ной — это её урав-
нение функции.

Переменные, свя-
занные со значе-

нием в ветке

1

ГЛАВА 4. СОПОСТА

cas

tree

или

должны быть свя-
заны с чем-нибудь
во всех ветвях это-
го оператора, что-

ГЛАВА 4. СОПОСТА
бы иметь значе-
ние за предела-
ми выражения, ина-
че компилятор бу-
дет считать это зна-
чение *небезопас-*
ным (unsafe) (ве-
роятно, не при-
своенным) за пре-
делами этого вы-
ражения, и выдаст
соответствующее
предупреждение.

ГЛАВА 4 СОПОСТА

4.2 Сопоста с образ- цом

**Образец имеет та-
кую же структу-
ру, как и терм, но
может содержать
новые свободные
переменные. На-
пример:**

ГЛАВА 4. СОПОСТА

Нам

Гни

ГЛАВА 4. СОПОСТА

{ e r

Образцы могут вст-
чаться в заголов-
ках функций, вы-
ражениях *case*, *case*
и *try* и в выраже-
ниях оператора со-

ГЛАВА 4. СОПОСТА

поставления (
Образцы вычис-
ляются посредство
сопоставления об
разца с выраже-
нием, и таким об-
разом новые пе-
ременные опре-
деляются и свя-
зываются со зна-
чением.

ГЛАВА 4. СОПОСТА

Обр

Обе стороны выражения должны иметь одинаковую структуру. Если сопоставление проходит успешно, то

ГЛАВА 4. СОПОСТА
все свободные пе-
ременные (если
такие были) в об-
разце слева ста-
новятся связан-
ными. Если сопо-
ставление не про-
ходит, то возни-
кает ошибка вре-
мени исполнения

ГЛАВА 4. СОПОСТА

bad

ГЛАВА 4. СОПОСТА

1 >

{an

ГЛАВА 4. СОПОСТА

2

>

a

n

s

3

>

ГЛАВА 4. СОПОСТА

42

ГЛАВА 4. СОПОСТА

4.2.1 Операто

сопостав

ления (

в образ-

цах

06

Если

ГЛАВА 4. СОПОСТА

З е щ

и 06 —

ГЛАВА 4. СОПОСТА-

ра-

зец

являются действи-
тельными образ-
цами, тогда сле-

ГЛАВА 4. СОПОСТА
дующая запись то-
же действитель-
ный образец:

Обр

ГЛАВА 4. СОПОСТА

Оператор
представляет со-
бой **подмену** (alias
при сопоставле-
нии которой с вы-
ражением, оба и

ГЛАВА 4. СОПОСТА

**Об-
ра-**

—

—

ГЛАВА 4. СОПОСТА

ЗеЩ

и 06 —

ГЛАВА 4. СОПОСТАВ-

ра-

же

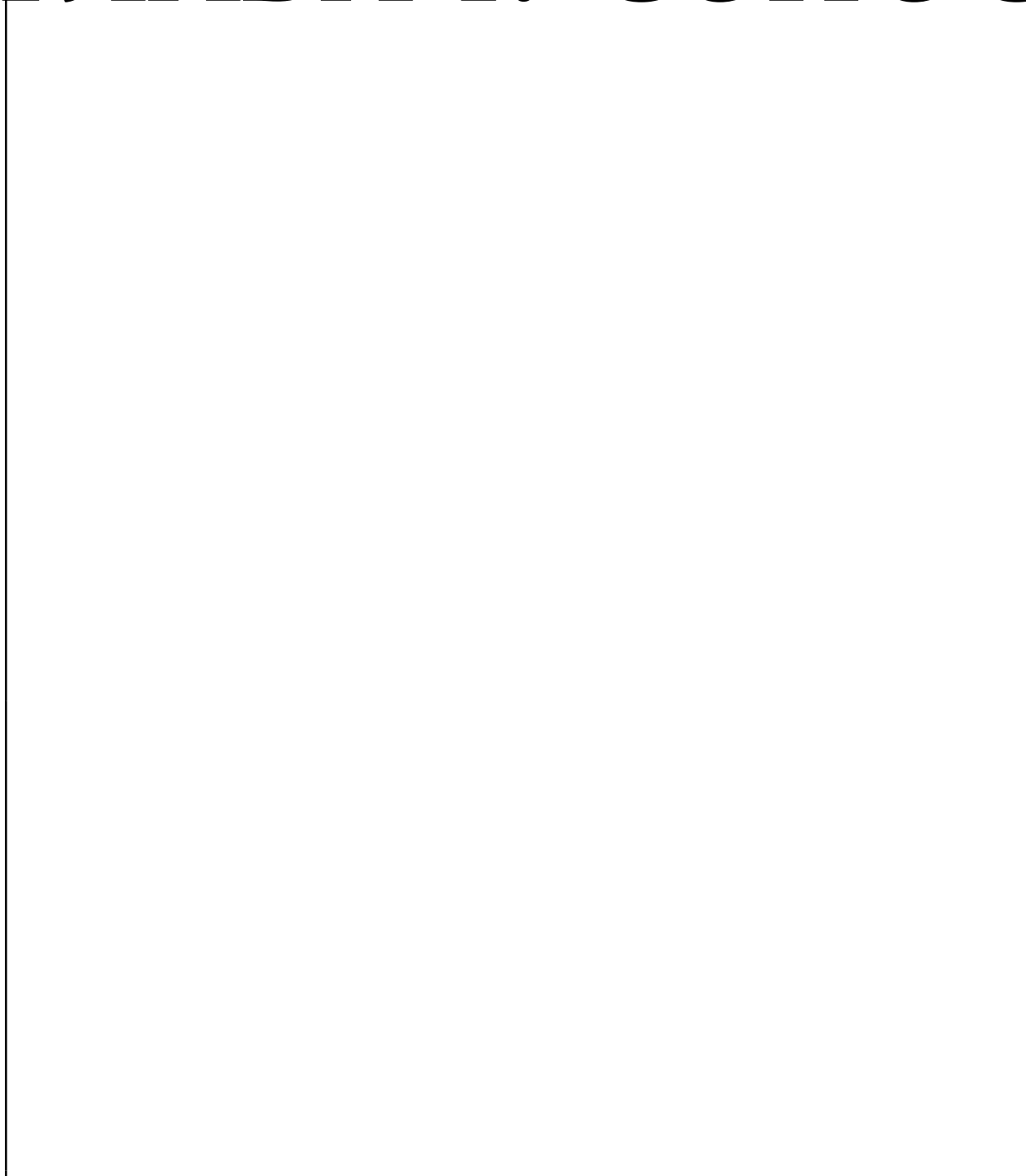
также сопостав-
ляются с ней. Цель
этого — избежать

ГЛАВА 4. СОПОСТА
необходимости по-
вторно строить тер
мы, которые бы-
ли разобраны на
составляющие в
сопоставлении.

ГЛАВА 4. СОПОСТА

f o o

ГЛАВА 4. СОПОСТА



ГЛАВА 4. СОПОСТА
можно более эф-
фективно записать
как:

ГЛАВА 4. СОПОСТА

foo

ГЛАВА 4. СОПОСТА

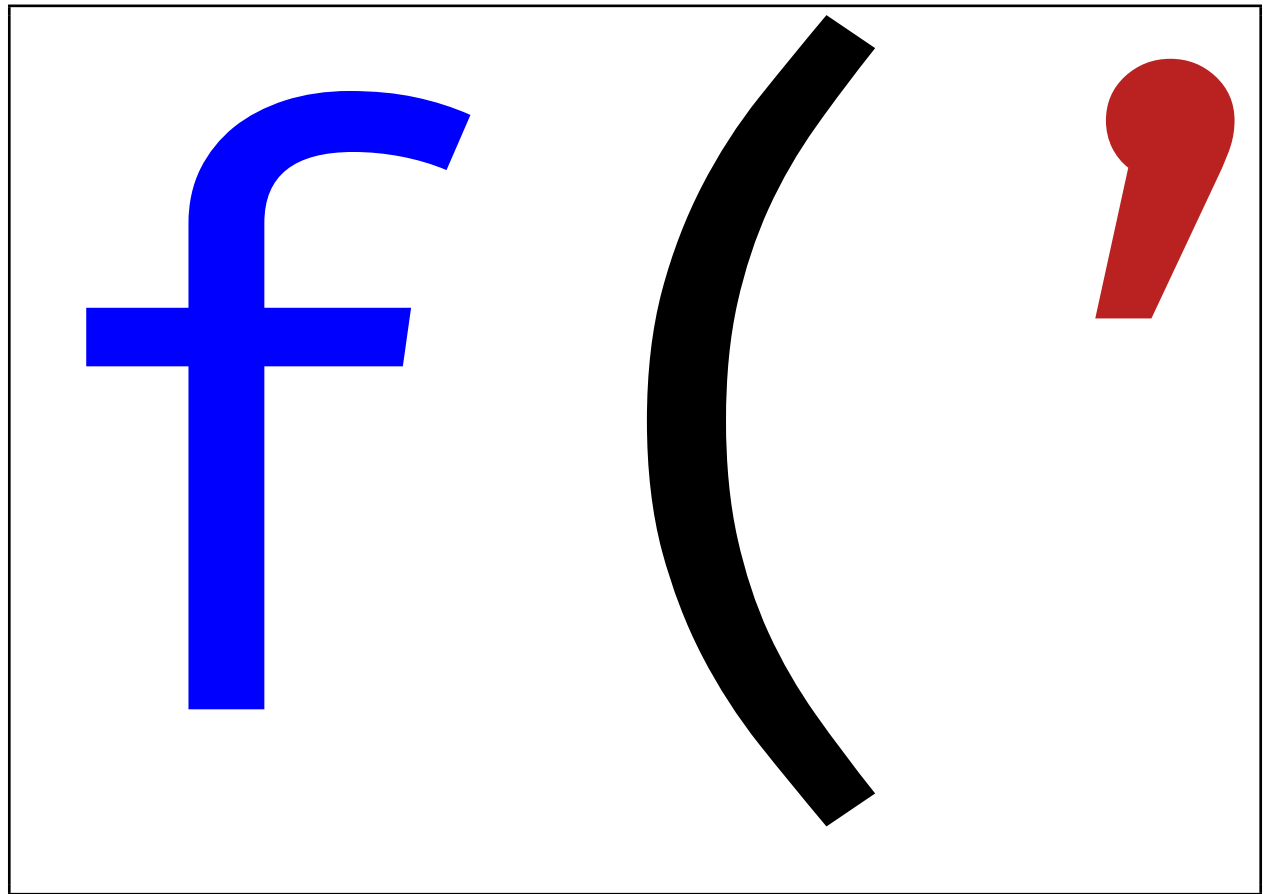


ГЛАВА 4. СОПОСТА

4.2.2 Строков префикс в образ- цах

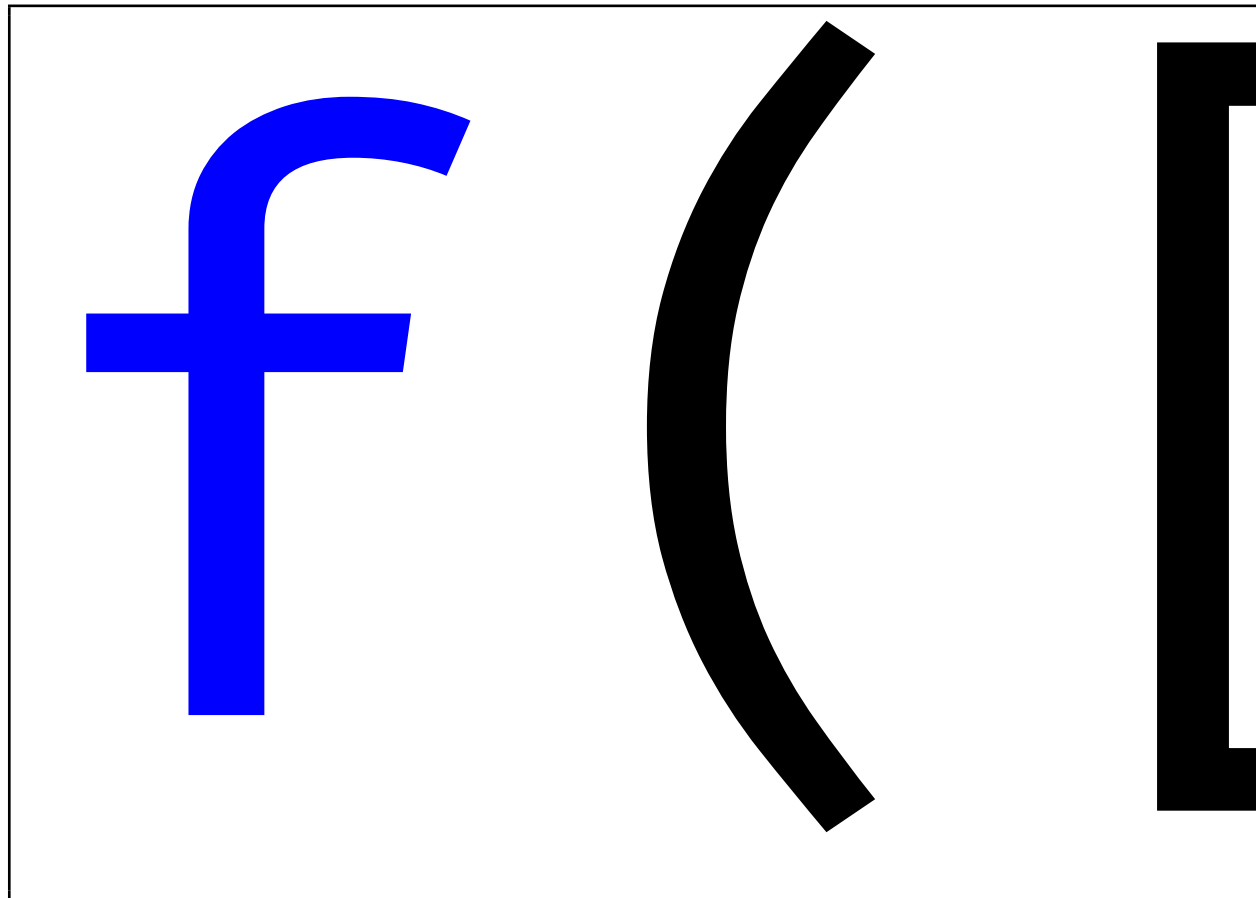
При сопоставле-
нии строк с об-
разцом, следую-
щая запись явля-
ется действитель-
ным образцом:

ГЛАВА 4. СОПОСТА



что эквивалентно и легче читается, чем следующая запись:

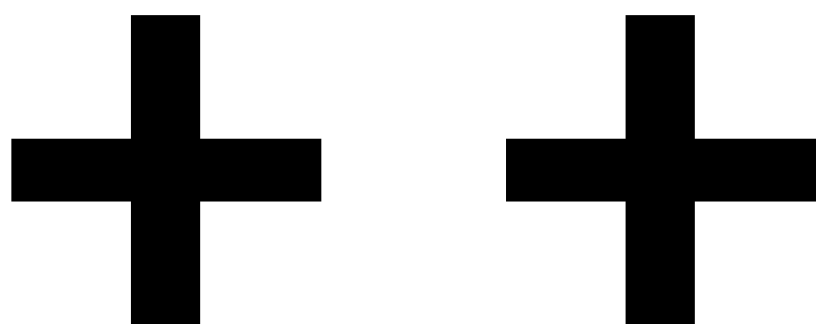
ГЛАВА 4. СОПОСТА



Вы можете использовать строки только как префикс; варианты с постфиксом для образ-

ГЛАВА 4. СОПОСТА

цов, такие как



не разрешаются.

ГЛАВА 4. СОПОСТА

4.2.3 Выраже

в образ- цах

Арифметическое выражение может быть использовано внутри образца, если оно использует только числовые, битовые операторы, и

ГЛАВА 4. СОПОСТА
его значение яв-
ляется констан-
той, которая мо-
жет быть вычис-
лена во время ком-
пиляции.

ГЛАВА 4. СОПОСТА

cas

ГЛАВА 4. СОПОСТА

ГЛАВА 4. СОПОСТА

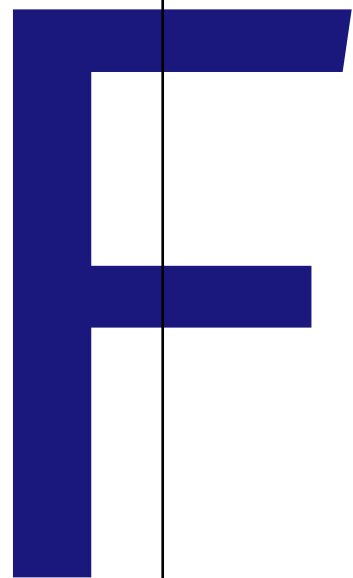
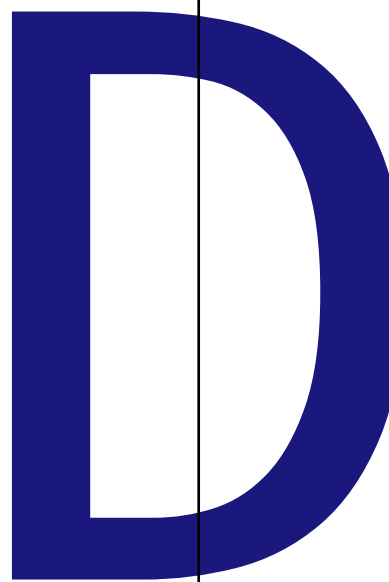
4.2.4 Сопоста

ДВОИЧ- НЫХ дан НЫХ

Bin

< < Δ

ГЛАВА 4. СОПОСТА



ГЛАВА 4. СОПОСТА

В последней стро-

ке переменная
неуказанного раз-
мера сопоставля-
ется с остатком двс

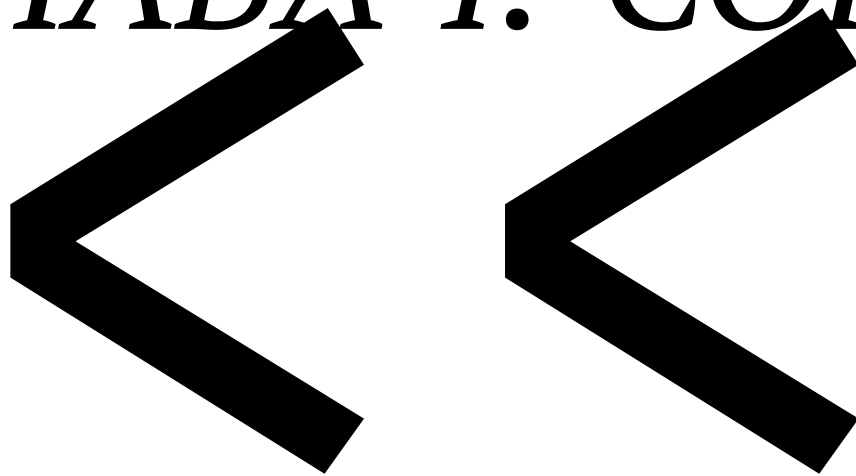
ГЛАВА 4. СОПОСТА

ичных данных

Всегда ставьте про-
бел между опера-

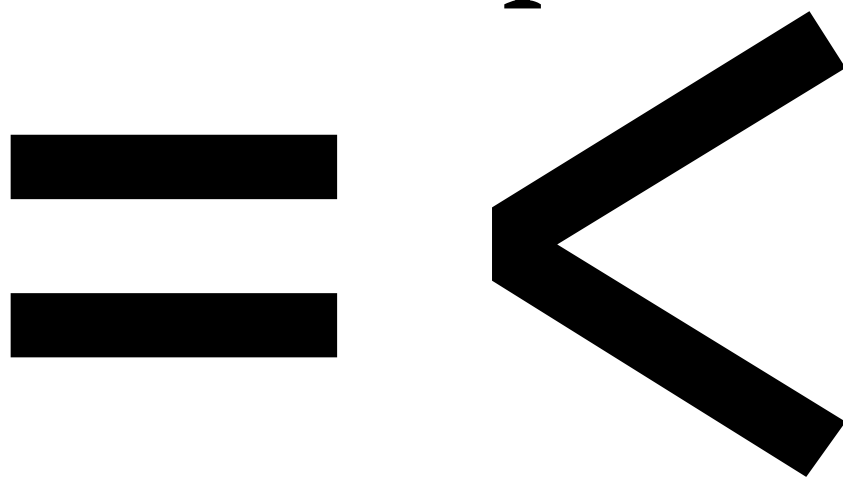
тором () и

ГЛАВА 4. СОПОСТА



(), что

бы избежать воз-
можной путани-
цы с оператором



().

5

Функ- ции

5.1 Опреде функ- ции

Функция опреде-
ляется, как после-

ГЛАВА 5. ФУНКЦИИ ДОВАТЕЛЬНОСТЬ ИЗ ОДНОГО ИЛИ НЕСКОЛЬ КИХ **уравнений** фу нкции. Имя функ- ции должно быть АТОМОМ.

ГЛАВА 5. ФУНКЦИИ

Функция

ГЛАВА 5. ФУНКЦИИ

Уравнения функции разделены точками с запятой (

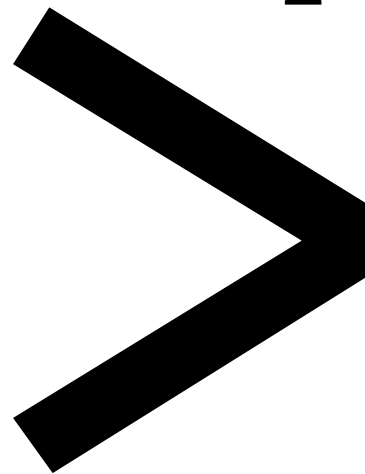
камаи с запятой (

и последнее уравнение завершается

точкой (●).

Уравнение функции состоит из **заголовка уравне-**

ГЛАВА 5. ФУНКЦИИ и тела урав- нения функции, разделённых стрел



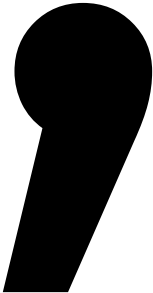
кой ().

Заголовок урав-
нения состоит из
имени функции
(атома), списка ар-
гументов, заклю-

ГЛАВА 5. ФУНКЦИИ
ченного в скоб-
ки, и необязатель-
ного списка охран-
ных выражений,
начинающихся с
ключевого слова

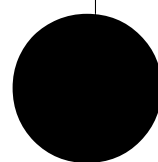
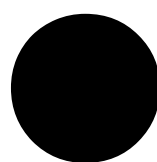
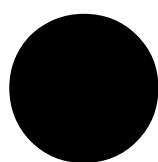
while

Каждый аргумент
функции — обра-

ГЛАВА 5. ФУНКЦИИ
зец. Тело функ-
ции состоит из по-
следовательности
выражений, раз-
делённых запяты-
ми ().

ГЛАВА 5. ФУНКЦИИ

В ы і р



В ы і р

ГЛАВА 5. ФУНКЦИИ

Количество аргу-

ментов **Н** ещё
называется **арно-**
стью функции. Фун-
кция уникально опре-
деляется именем
модуля, именем
функции и своей

ГЛАВА 5. ФУНКЦИИ
арностью. Две раз-
ные функции в од-
ном модуле, име-
ющие разную ар-
ность, могут иметь
одно и то же имя.

Ф у н

ГЛАВА 5. ФУНКЦИИ

ЩИЯ

МО —

В

ГЛАВА 5. ФУНКЦИИ

ДУ

ле с ар-

НОСТЬЮ ча-
Н

ГЛАВА 5. ФУНКЦИИ
кто может заши-

сываться так:

дул

ГЛАВА 5. ФУНКЦИИ

— mo

— ex

ГЛАВА 5. ФУНКЦИИ

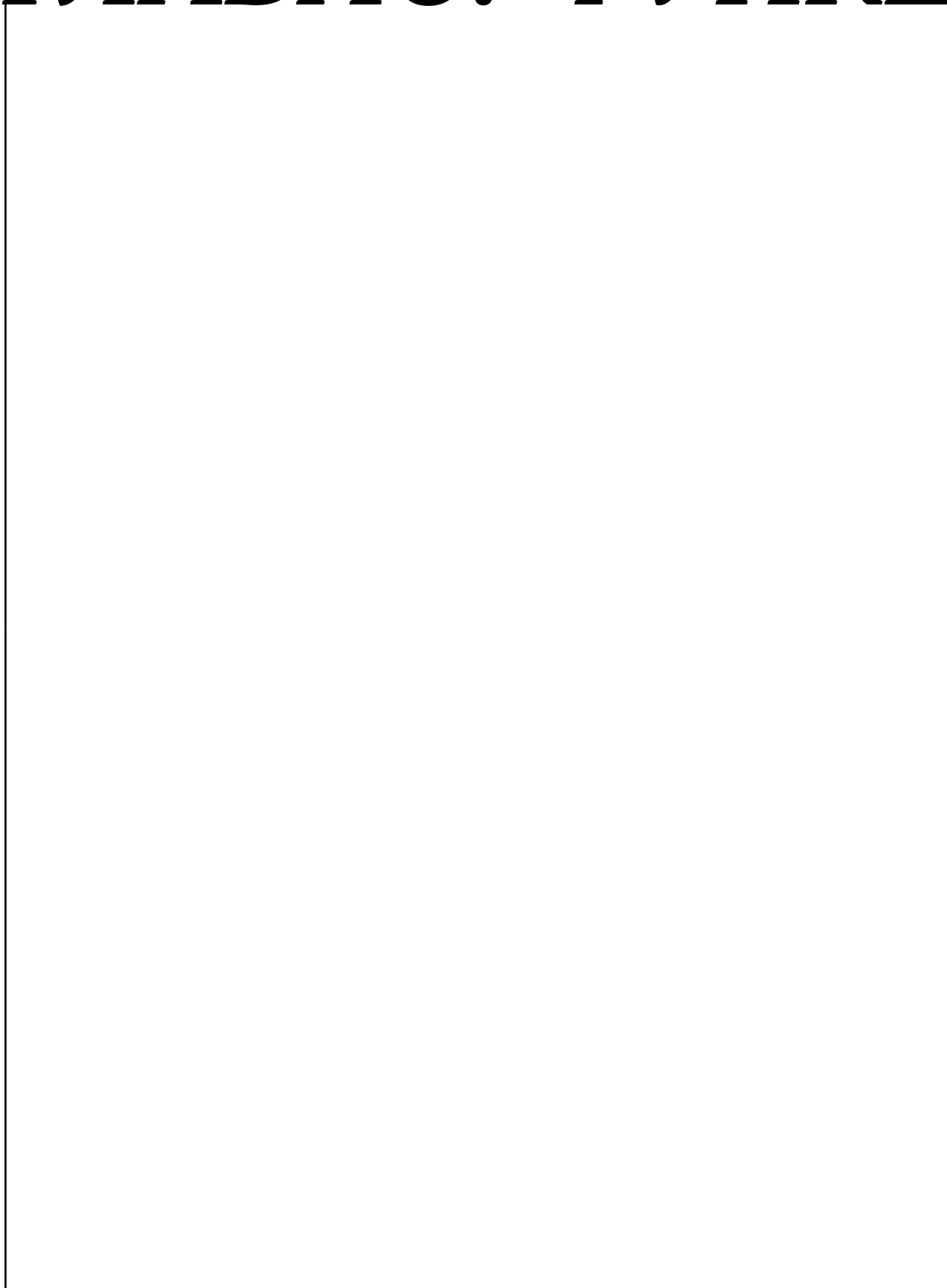
are

ГЛАВА 5. ФУНКЦИИ

are

are

ГЛАВА 5. ФУНКЦИИ



ГЛАВА 5 ФУНКЦИИ

5.2 ВЫЗОВЪ функ- ций

Функция вызывается с помощью записи:

ГЛАВА 5. ФУНКЦИИ

ГМО

ГЛАВА 5. ФУНКЦИИ

Выражение

М
ДУЛ

ДОЛЖНО ВЫЧИСЛЯТЬСЯ
В ИМЯ МОДУЛЯ
(или быть атомом)

ГЛАВА 5. ФУНКЦИИ

и выражение

Ф

ЦИЯ

должно вычислять-
ся в имя функции
или в анонимную

ГЛАВА 5. ФУНКЦИИ
функцию. При вы-
зове функции в дру-
гом модуле, сле-
дует указать имя
модуля и функ-
ция должна быть
экспортирована.
Такой вызов бу-
дет называться пол-
ностью опреде-
лённым вызовом
функции.

ГЛАВА 5. ФУНКЦИИ

lis

Имя модуля может быть опущено

ГЛАВА 5. ФУНКЦИИ

но, если

ФУ

ЩИЯ

вычисляется в имя
локальной функ-
ции, импортиро-

ГЛАВА 5. ФУНКЦИИ
ванной функции
или авто-импорти
встроенной (BIF)
функции. Такой
вызов называется
неявно опре-
делённым вызо-
вом функции.

Перед тем, как вы-
звать функцию, вы-
числяются аргу-

ГЛАВА 5 ФУНКЦИИ

МЕНТЫ

EX

Если функция не
может быть най-
дена, то возника-
ет ошибка време-

ГЛАВА 5. ФУНКЦИИ

ни исполнения
Если уравнений
функции несколько,
они последовательно
сканируются до тех пор,
пока не будет найдено
подходящее уравнение,
такое,

ГЛАВА 5. ФУНКЦИИ
что образцы в за-
головке уравне-
ния успешно мо-
гут быть сопостав-
лены с данными
аргументами и что
охранное выраже-
ние (если оно за-

дано) равно

t

ГЛАВА 5. ФУНКЦИИ

Если такое уравнение не может быть найдено, возникает ошибка времени исполнения

fun

ГЛАВА 5. ФУНКЦИИ
Если совпадающее
уравнение найде-
но, то вычисля-
ется соответству-
ющее тело
функции, то есть
выражения в те-
ле функции вы-
числяются одно
за другим и воз-
вращается резуль-
тат последнего вы-
ражения.

ГЛАВА 5. ФУНКЦИИ
Полностью определённое имя функции должно быть использовано, если вызывается встроенная функция с таким же именем (см. секцию ?? о встроенных VIF функциях). Компилятор не разрешит определить функ-

ГЛАВА 5. ФУНКЦИИ
цию с таким же
именем, как дру-
гая импортирован-
ная функция. При
вызове локальной
функции есть раз-
ница между ис-
пользованием неяв-
но или полностью
определённого име-
ни функции, по-
скольку второе все-
гда относится к по-

ГЛАВА 5. ФУНКЦИИ
следней версии модуля (см. главу ??
О модулях и версиях).

5.3 Выраже

Выражение это
терм либо вызов
оператора, резуль-

ГЛАВА 5. ФУНКЦИИ
татом которого будет терм, например:

ГЛАВА 5. ФУНКЦИИ

Теперь

опре

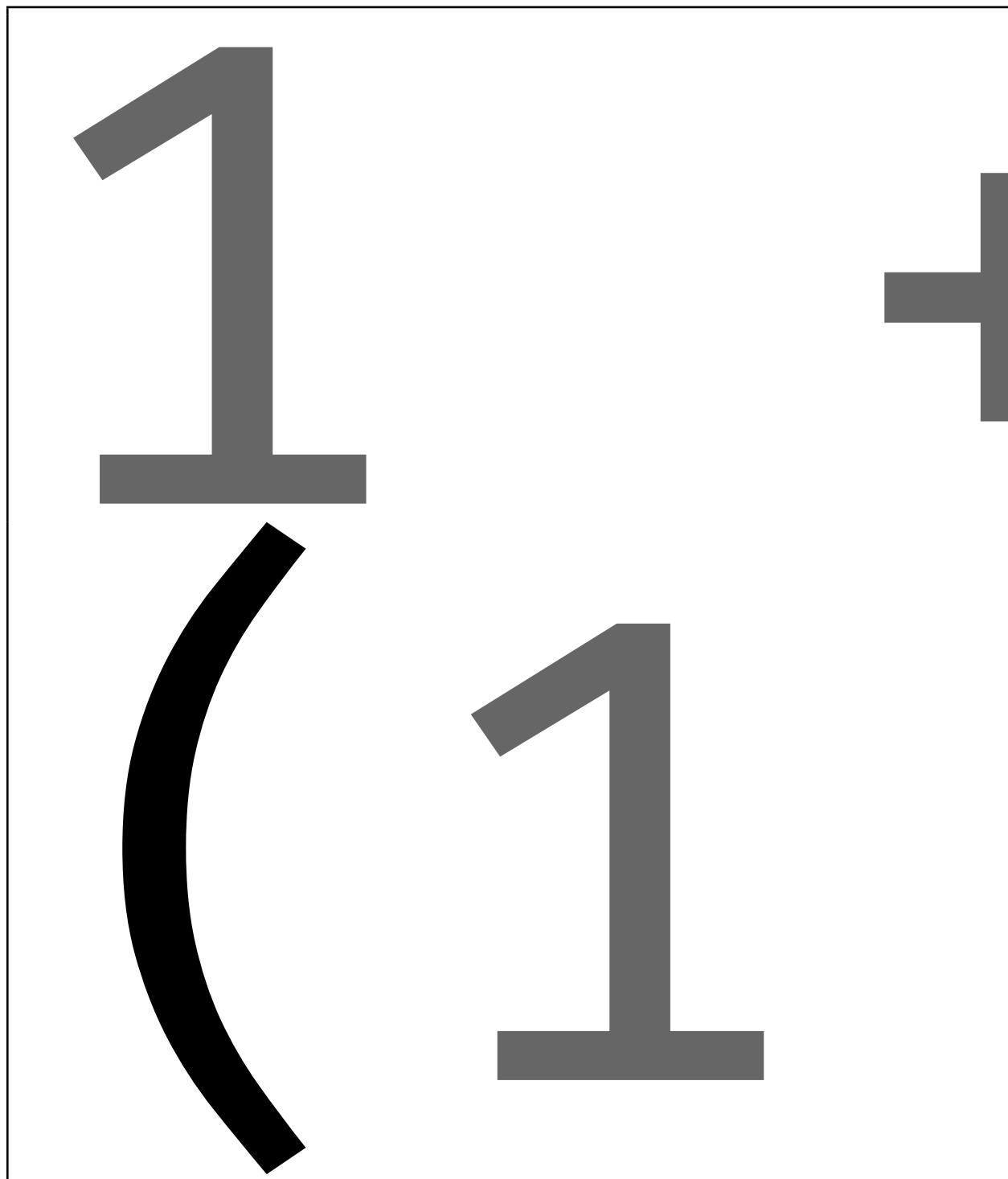
выр

ГЛАВА 5. ФУНКЦИИ
В наличии имеются как *унарные* так и *бинарные* операторы. Простейшая форма выражения — это терм, например *целое число*, *число с плавающей точкой*, *атом строка*, *список* или *кортеж*, и возвращаемое оператором значение то-

ГЛАВА 5. ФУНКЦИИ
же терм. Выраже-
ние может содер-
жать *макрос* или
операций над за-
писью, которые бу-
дут развёрнуты во
время компиля-
ции.

Выражения в скоб-
ках полезны для
изменения поряд-
ка вычисления опе-
раторов (см. раз-

ГЛАВА 5. ФУНКЦИИ дел ??):



ГЛАВА 5. ФУНКЦИИ

Блочные выражения, заключённые в операторные скобки

ки между

be

могут использоваться для группировки последовательности вы-

ГЛАВА 5. ФУНКЦИИ
ражений и возвра-
щают значение,
равное значению
последнего выра-

жения внутри

ра -

В

-

ГЛАВА 5. ФУНКЦИИ

ЖЕ

—

Н

И

—

е

М.

ГЛАВА 5. ФУНКЦИИ

Все вложенные подвыражения вычисляются до главного выражения, но порядок, в котором происходит вычисление вложенных, не определён стандартом.

Многие операторы могут применяться только к аргументам определённого типа.

ГЛАВА 5. ФУНКЦИИ
Например, арифметические операторы могут только применяться к целым числам или числам с плавающей точкой. Аргумент неверного типа вызовет ошибку во времени выполнения

ГЛАВА 5. ФУНКЦИИ

bad

ГЛАВА 5. ФУНКЦИИ
**5.3.1 Сравнение
термов**

В ы р

**Сравнение тер-
мов** возвращает
булево (логическое)

ГЛАВА 5. ФУНКЦИИ

значение, в фор-

ме атомов

или

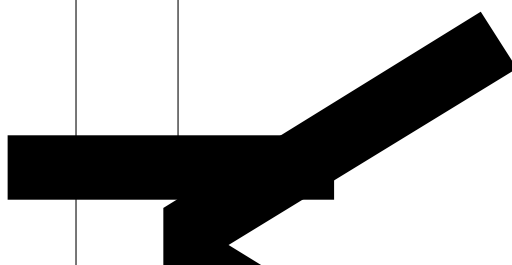
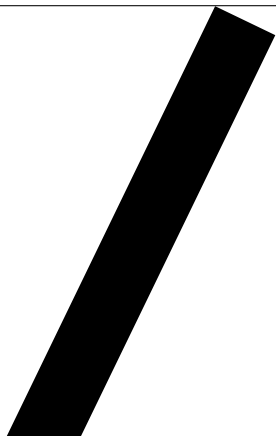
ГЛАВА 5. ФУНКЦИИ

Операторы сравне- ния

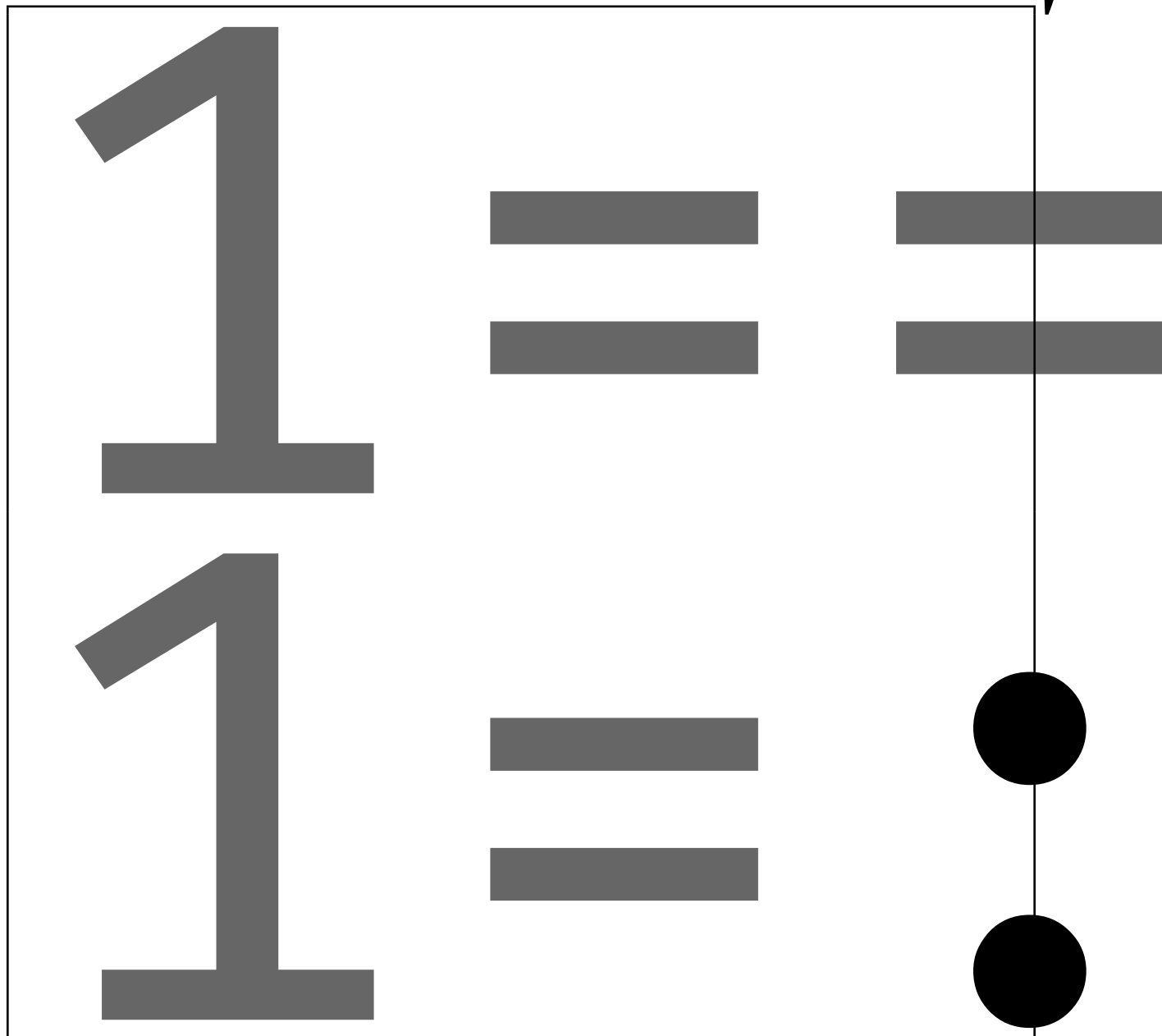


Равно
(с
при-
веде-
нием
типа)

Меньше
или
рав-
но

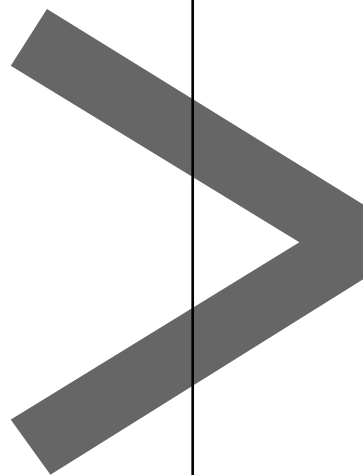


ГЛАВА 5. ФУНКЦИИ



ГЛАВА 5. ФУНКЦИИ

1



Аргументы оператора сравнения могут иметь разные типы данных. В таком случае действует следующий порядок сравнения:

ГЛАВА 5 ФУНКЦИИ

ЧИС

<

a

<

c

ГЛАВА 5. ФУНКЦИИ

<

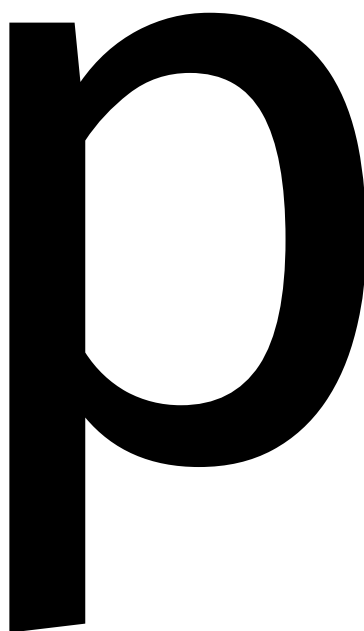
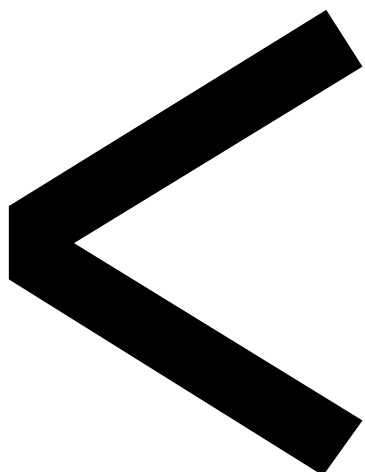
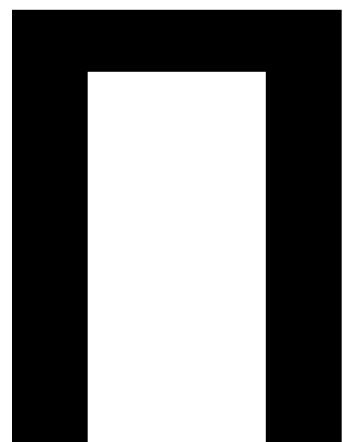
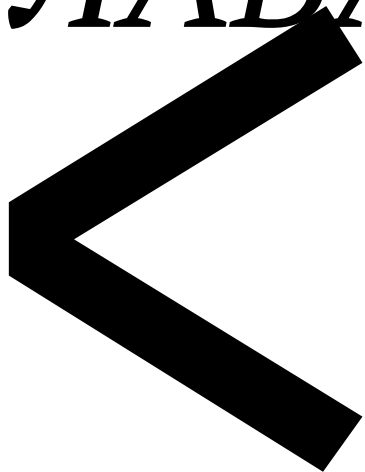
a

ф

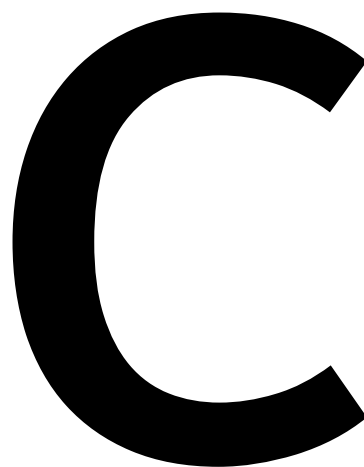
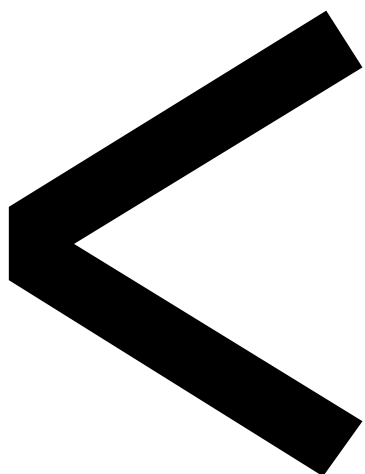
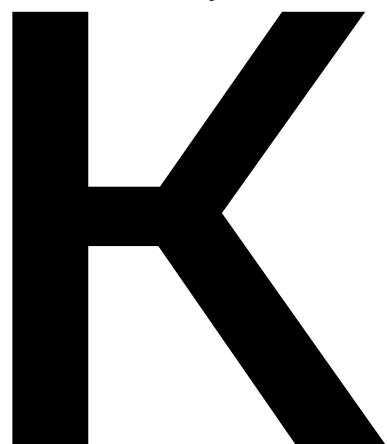
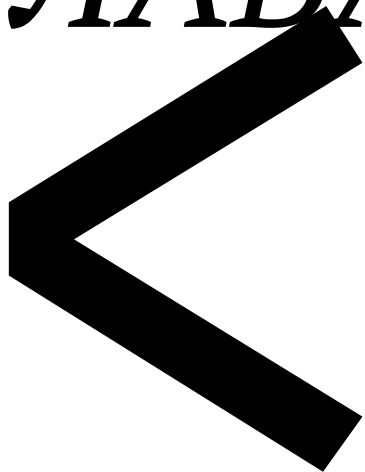
у

н

ГЛАВА 5. ФУНКЦИИ



ГЛАВА 5. ФУНКЦИИ



ГЛАВА 5. ФУНКЦИИ

<

д

дан

Списки сравниваются поэлементно. Кортежи сравниваются по раз-

ГЛАВА 5. ФУНКЦИИ
меру, два кортежа одного размера сравниваются поэлементно. При сравнении целого числа и числа с плавающей точкой, целое сначала приводится к числу с плавающей точкой. В случае использования точного ра-

ГЛАВА 5. ФУНКЦИИ

венства

или

тип числа не изменяется и учитывается в равенстве.

ГЛАВА 5. ФУНКЦИИ
**5.3.2 Арифметические
выражения**

О п е

В ы р

ГЛАВА 5. ФУНКЦИИ

Арифметическое выражение воз- вращает резуль- тат после приме- нения оператора.

ГЛАВА 5. ФУНКЦИИ

Арифметические операторы

+

Унарный
плюс

↑

↑

|

↓

↓

ГЛАВА 5. ФУНКЦИИ

+

1

4

/

2

ГЛАВА 5. ФУНКЦИИ

55

dr

ГЛАВА 5. ФУНКЦИИ

2

#

1

2

#

1

ГЛАВА 5. ФУНКЦИИ

5.3.3 Логические (булевы выраже- ния

О п е

В ы р

ГЛАВА 5. ФУНКЦИИ

Логическое выражение возвращает значение

и

fa

или
после применения оператора.

ГЛАВА 5. ФУНКЦИИ

Логические
(булевы)
операторы

по

унарное
логиче-
ское НЕ
(отри-
цание)

ан

ГЛАВА 5. ФУНКЦИИ

not

trw

ГЛАВА 5. ФУНКЦИИ

t r u

ГЛАВА 5. ФУНКЦИИ
**5.3.4 УМНЫЕ
ЛОГИЧЕ-
СКИЕ ВЫ-
РАЖЕНИЯ**

В ы р

В ы р

ГЛАВА 5. ФУНКЦИИ
В ЭТИХ ЛОГИЧЕСКИЕ
ВЫРАЖЕНИЯХ ВТО-
рой операнд вы-
числяется только
в том случае, ес-
ли его значение
необходимо для
конечного резуль-
тата. В случае с

ГЛАВА 5. ФУНКЦИИ

Оте
Выір

будет вычислено,

ГЛАВА 5. ФУНКЦИИ

только если

равняется

B

f

ГЛАВА 5. ФУНКЦИИ

В случае с \sin

Выір

будет вычислено,

ГЛАВА 5. ФУНКЦИИ

только если

равняется

В

т.

ГЛАВА 5. ФУНКЦИИ

A large, stylized logo in a vibrant green color. It consists of a lowercase 'i' followed by a lowercase 'f'. The 'i' has a solid circular dot above it. The 'f' is formed by a thick vertical stem and a horizontal crossbar that curves upwards at its right end. The entire logo is contained within a thin black rectangular border.

ГЛАВА 5. ФУНКЦИИ

A large, stylized green logo consisting of the letters 'i' and 'f'. The 'i' has a solid green circle above it, and the 'f' has a curved top that extends to the right. The entire logo is contained within a thin black rectangular border.

if

ГЛАВА 5. ФУНКЦИИ

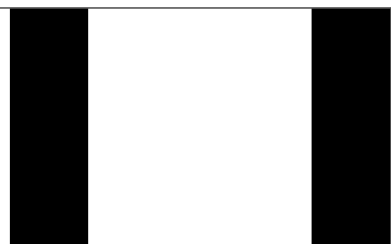
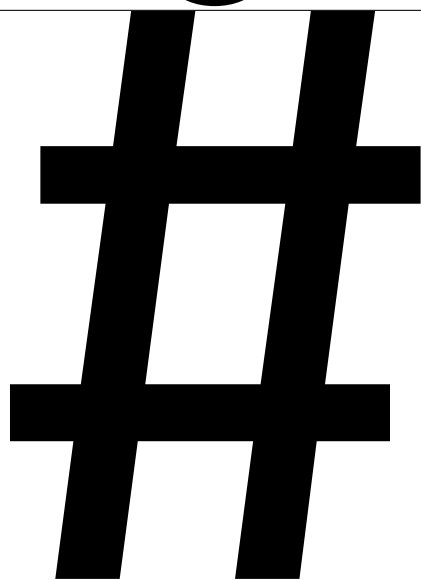
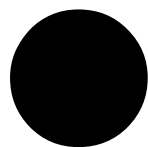
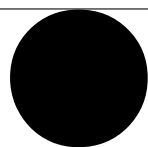
5.3.5 Приоритет операторов

В выражении, состоящем из подвыражений, операторы будут применяться согласно определённому порядку, который называется-

ГЛАВА 5. ФУНКЦИИ **ся приоритетом операторов:**

ГЛАВА 5. ФУНКЦИИ

Приоритет
операто-
ров (от
высшего к
низшему)

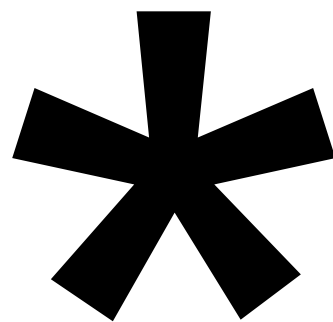
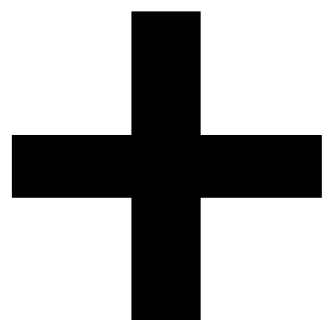


ГЛАВА 5. ФУНКЦИИ
Оператор с наивысшим приоритетом вычисляется первым. Операторы с одинаковым приоритетом вычисляются согласно их ассоциативности. Левоассоциативные операторы вычисляются слева направо, правоассоциативные —

ГЛАВА 5. ФУНКЦИИ

наоборот, справа налево:

6
5



ГЛАВА 5. ФУНКЦИИ

4

—

3

/

ГЛАВА 5. ФУНКЦИИ

26

+

ГЛАВА 5. ФУНКЦИИ

20.5

ГЛАВА 5. ФУНКЦИИ

20

1

⇒



ГЛАВА 5. ФУНКЦИИ

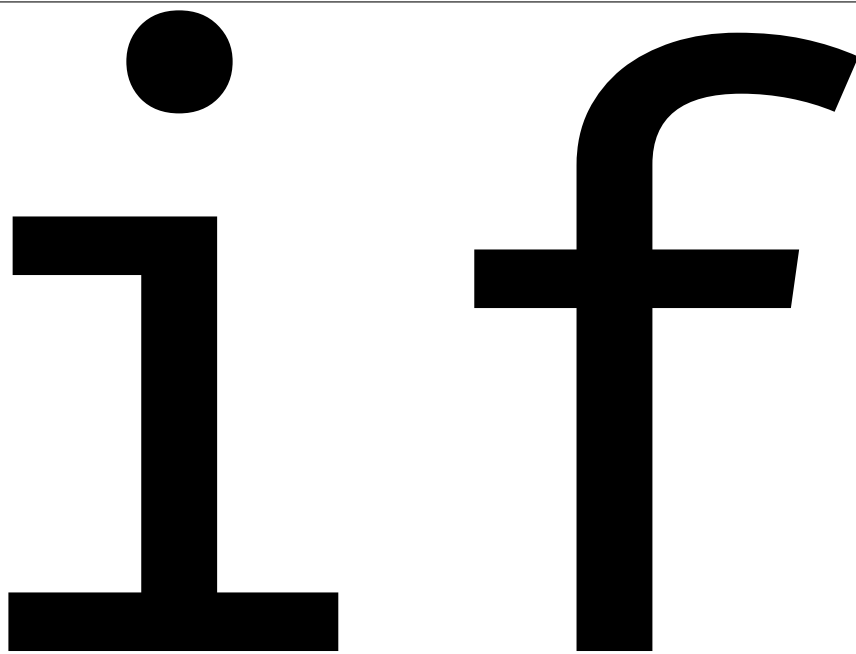
2



ГЛАВА 5 ФУНКЦИИ

5.4 Составные выражения

5.4.1 If



i f

ГЛАВА 5. ФУНКЦИИ

Ветки выражения

if ска-
нируются после-
довательно свер-
ху вниз, пока не

ГЛАВА 5. ФУНКЦИИ

встретится

результатом ко-

торого будет

Соответствующее

ГЛАВА 5. ФУНКЦИИ

Тел

(последовательность выражений, разделённых запятыми) будет вычислено. Возвра-

ГЛАВА 5. ФУНКЦИИ

щаемое

значение будет ре-
зультатом всего

●

выражения

1

ГЛАВА 5. ФУНКЦИИ

Если ни одно из
охранных выра-
жений не вычис-

ляется в

то возникнет ошиб-
ка времени выпол-

ГЛАВА 5. ФУНКЦИИ

нения

1

f

При необходимости можно использовать
охранное выраже-

ГЛАВА 5. ФУНКЦИИ

ние

В последней вет-

ке

поскольку такое
охранное выраже-

ГЛАВА 5. ФУНКЦИИ
ние всегда сраба-
тывает, если дру-
гие не сработали
и называется «сра-
батывающим для
всех значений» (cat
all).

ГЛАВА 5. ФУНКЦИИ

is



ГЛАВА 5. ФУНКЦИИ



ГЛАВА 5. ФУНКЦИИ



ГЛАВА 5. ФУНКЦИИ

Следует заметить, что сопоставление с образцом в уравнениях функций может почти всегда использоваться для заме-

ГЛАВА 5. ФУНКЦИИ

Чрезмерное использование

внутри функций
считается плохой

ГЛАВА 5. ФУНКЦИИ практикой.

5.4.2 Case

С
Выражения
используются для
сопоставления с
образцом прямо
в коде, подобно

ГЛАВА 5. ФУНКЦИИ

тому, как сопоставляются аргументы в заголовках функций.

ГЛАВА 5. ФУНКЦИИ

cas

ГЛАВА 5. ФУНКЦИИ

Выр
о

вычисляется и

последовательно

ГЛАВА 5 ФУНКЦИИ
сопоставляются с
результатом. Если
сопоставление
проходит успешно,
и необязатель-

ное **ОХ**

ГЛАВА 5. ФУНКЦИИ

Ное

Выр

ГЛАВА 5. ФУНКЦИИ

равно

Тогда вычисляется соответствующее

Возвращаемое значение

ГЛАВА 5 ФУНКЦИИ

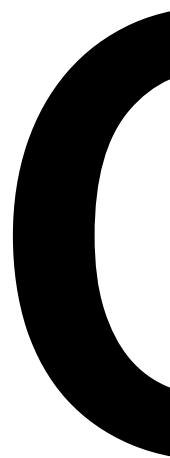
Течение
является резуль-
татом всего вы-

Сса
ражения

В случае, если ни
один из образцов
и их охранных вы-

ГЛАВА 5. ФУНКЦИИ
ражений не под-
ходит, возникает
ошибка време-

НИ ВЫПОЛНЕНИЯ



ГЛАВА 5. ФУНКЦИИ

iS



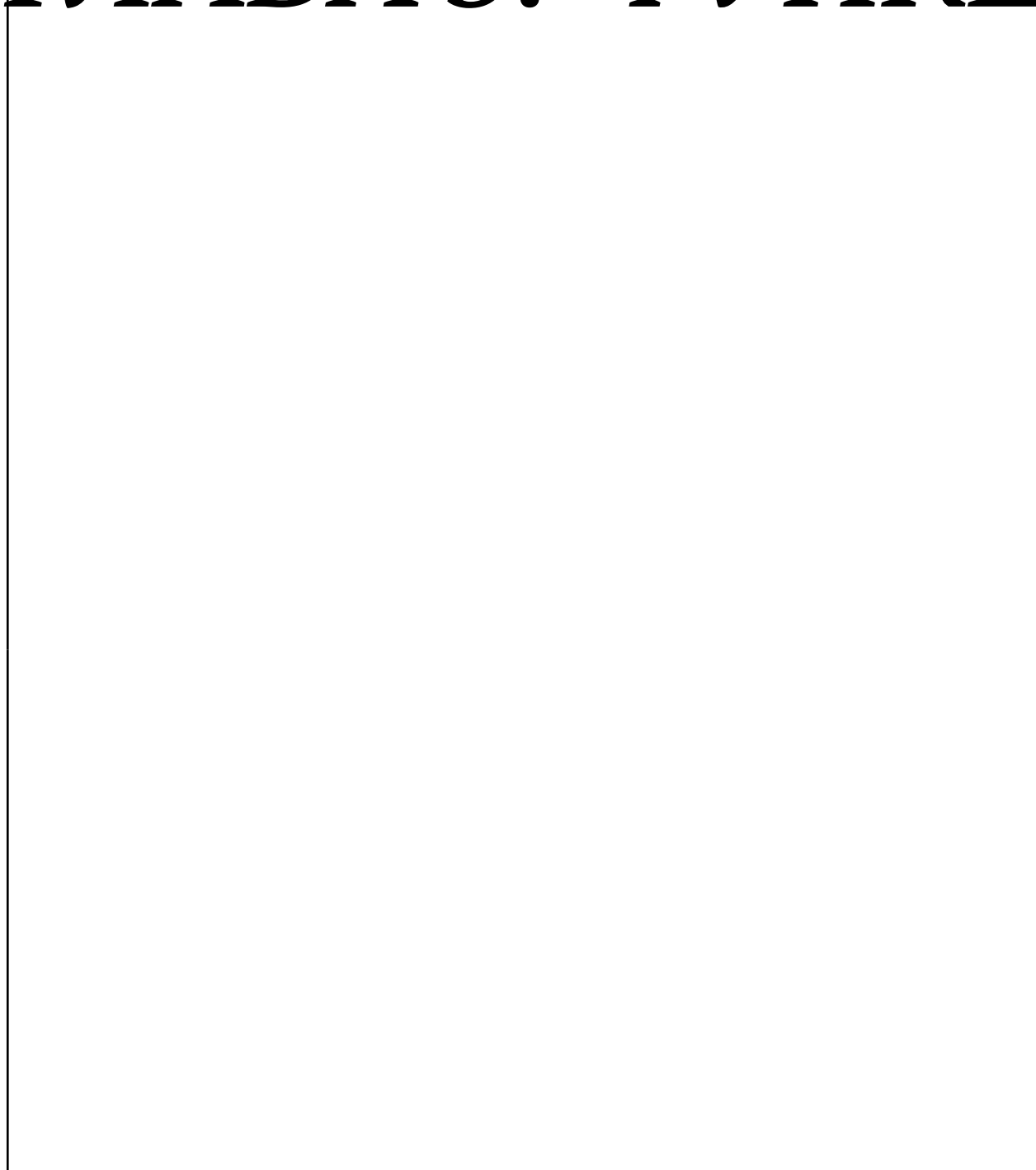
ГЛАВА 5. ФУНКЦИИ



ГЛАВА 5. ФУНКЦИИ



ГЛАВА 5. ФУНКЦИИ



ГЛАВА 5. ФУНКЦИИ



5.4.3 Генераторы СПИСКОВ

Генераторы спис-
ков аналогичны

ГЛАВА 5. ФУНКЦИИ

предикатам **S**
f i n
и
в языке Prolog.

ГЛАВА 5. ФУНКЦИИ

ВЫ

ГЛАВА 5. ФУНКЦИИ

Выір

— произвольное
выражение, и каж-

ГЛАВА 5. ФУНКЦИИ

КВ дый

— это либо **генератор** (источник данных) либо **фили**
Генератор записывается, как:

ГЛАВА 5. ФУНКЦИИ

Обр

ГЛАВА 5. ФУНКЦИИ

ВЫК

где

должно быть выражением, результатом которого является список термов. Фильтр — это выражение, результатом которого яв-

ГЛАВА 5. ФУНКЦИИ

ляется

или

Переменные внутри генератора списков *затеняют* переменные с таки-

ГЛАВА 5. ФУНКЦИИ
ми же именами,
принадлежащие
функции, окружа-
ющей генератор.

Квалификаторы въ
числяются слева
направо, генера-
торы создают зна-
чения и
фильтры отбира-
ют нужные из них.
Генератор спис-
ков возвращает спи

ГЛАВА 5. ФУНКЦИИ
СОК, В КОТОРОМ ЭЛЕ-
МЕНТЫ ЯВЛЯЮТСЯ
РЕЗУЛЬТАТОМ ВЫ-

числения **В** **Ы**

для каждой ком-
бинации резуль-
тирующих значе-
ний.

ГЛАВА 5. ФУНКЦИИ

1

>

[

}

5

ГЛАВА 5 ФУНКЦИИ

5.5 Охранная последовательность

Охранная последовательность — это набор охраняемых выражений, разделённых точ-

ГЛАВА 5. ФУНКЦИИ

ками с запятой (

Охранная после-
довательность рав-

tr

няется

если как минимум
одно из составля-

ГЛАВА 5. ФУНКЦИИ ЮЩИХ её охр- НЫХ выражений

равно

t r

Охр

ГЛАВА 5. ФУНКЦИИ
Охрана — это мно-
жество **охранных**
выражений, раз-
делённых запяты-
ми (**,**). Резуль-

t
татом будет
если все охранные
выражения рав-

ГЛАВА 5. ФУНКЦИИ

няются

Охр

**В охранных вы-
ражениях, кото-**

ГЛАВА 5. ФУНКЦИИ
рые иногда еще
называются охран-
ными тестами, раз-
решены не любые
выражения Erlang,
а ограниченный
набор, выбранный
авторами Erlang,
поскольку важно,
чтобы ход вычис-
ления охранного
выражения не имел

ГЛАВА 5. ФУНКЦИИ ПОБОЧНЫХ ЭФФЕК- ТОВ.

ГЛАВА 5.. ФУНКЦИИ

Разрешённые
охранные
выражения:

АТОМ

т r u

Другие кон-
станты (тер-
мы, связан-
ные пере-
менные),
все считают-
ся равными

ГЛАВА 5. ФУНКЦИИ

Небольшой пример

ГЛАВА 5. ФУНКЦИИ

fas

ГЛАВА 5. ФУНКЦИИ

fas

ГЛАВА 5. ФУНКЦИИ

5.6 Хвостовая рекурсия

Если последнее выражение тела функции является вызовом функции, то выполняется **хвостовой рекурсивный** вызов так,

ГЛАВА 5. ФУНКЦИИ
Что ресурсы системы (например, стек вызовов) не расходуются. Это означает, что можно создать бесконечный цикл, такой, как, например, сервер, если использовать хвостовые рекурсивные вызовы.

ГЛАВА 5. ФУНКЦИИ

Функция

f

выше может быть
переписана для ис-
пользования хво-
стовой рекурсии
следующим обра-
зом:

ГЛАВА 5. ФУНКЦИИ

f a c

f a c

ГЛАВА 5. ФУНКЦИИ

fac

ГЛАВА 5. ФУНКЦИИ

fas

ГЛАВА 5. ФУНКЦИИ

5.7 АНОНИМНЫЕ ФУНКЦИИ

Ключевое слово **fun** определяет функциональный объект. Анонимные функции делают возможным передавать целую функ

ГЛАВА 5. ФУНКЦИИ
цию, а не только
её имя, в качестве
аргумента. Выра-
жение, определя-
ющее анонимную
функцию, начи-
нается с ключе-

вого слова
и заканчивается
ключевым словом

ГЛАВА 5. ФУНКЦИИ

end

вместо точки (●)
Между ними долж-
но находиться обы-
чное объявление фу-
нкции, за тем исклю-
чением, что имя
её не пишется.

ГЛАВА 5. ФУНКЦИИ

fun

ГЛАВА 5. ФУНКЦИИ
Переменные в за-
головке аноним-
ной функции за-
тенеяют перемен-
ные в уравнении
функции, которое

окужает
но переменные,
связанные в те-

f u

ГЛАВА 5. ФУНКЦИИ

func

де
являются для него
локальными. Воз-
вращаемое выра-

func

жением

ГЛАВА 5. ФУНКЦИИ

ИМЯ

значение является функцией. Вы-

ражение

f

ГЛАВА 5. ФУНКЦИИ

ИМЯ

**ЭКВИВАЛЕНТНО СЛЕ-
ДУЮЩЕЙ ЗАПИСИ:**

ГЛАВА 5. ФУНКЦИИ

fun

ГЛАВА 5. ФУНКЦИИ

f

Выражение

МОД

также разрешено,

ГЛАВА 5. ФУНКЦИИ

НО ТОЛЬКО ЕСЛИ

ЭКСПОРТИРОВАНА ИЗ

МОД

ГЛАВА 5. ФУНКЦИИ

Fun

Fun

ГЛАВА 5. ФУНКЦИИ

Fun

ГЛАВА 5. ФУНКЦИИ

Fun

Поскольку функ-

ция, созданная

f

ГЛАВА 5. ФУНКЦИИ
анонимна, то есть
не имеет имени
в определении функ-
ции, то для опре-
деления рекурсив-
ной функции сле-
дует сделать два
шага. Пример ни-
же показывает, как
определить рекур-
сивную функцию

ГЛАВА 5. ФУНКЦИИ

sum

(смотрите раздел
??) как аноним-
ную с помощью

ГЛАВА 5. ФУНКЦИИ

f u n

Такой подход ещё
называется *Y-комб*

ГЛАВА 5. ФУНКЦІИ

Sum

Sum

ГЛАВА 5. ФУНКЦИИ

Sum

Определение
сделано так, что
оно принимает *sa-*

ГЛАВА 5. ФУНКЦИИ

мо себя в качестве
аргумента, сопо-

ставляется с

(пустым списком)

или

foo

ГЛАВА 5. ФУНКЦИИ
которые затем ре-
курсивно вызы-
ваются. Опреде-

ление

вызывает

также передавая

ГЛАВА 5. ФУНКЦИИ

Sum

в качестве аргумента.

Примечание: В Erlang версии R17 эта проблема устранена и анонимные функции могут ссылаться са-

ГЛАВА 5. ФУНКЦИИ МИ НА СЕБЯ.

5.8 Встроенные функ- ции (BIF)

**Встроенные функ-
ции, или BIF (built-
in functions) — это**

ГЛАВА 5. ФУНКЦИИ
функции, реализованные на языке C, на котором также написана система Erlang, и делают вещи, которые трудно или невозможно реализовать на языке Erlang. Большинство встроенных функций принад-

ГЛАВА 5. ФУНКЦИИ

лежат модулю

но есть некоторые,
принадлежащие
и другим модулям,

ГЛАВА 5. ФУНКЦИИ

таким как

ets

и

Используемые чаще всего встроенные функции,

ГЛАВА 5 ФУНКЦИИ принадлежащие

модулю **е** **т**

импортируются во
все ваши модули
автоматически, то
есть перед ними
не требуется пи-
сать имя модуля.

ГЛАВА 5. ФУНКЦИИ

Некоторые
полезные
встро-
енные
функции

date

Возвращает
сего-
дняш-
нюю дату
в
формате

ГЛАВА 5. ФУНКЦИИ

Словарь — это список кортежей в формате

материала

{ K. ... }

ГЛАВА 5. ФУНКЦИИ

Зна

(см. также раздел
??).

ГЛАВА 5. ФУНКЦИИ

•
siz
ato

ГЛАВА 5. ФУНКЦИИ

data

time

Про- цессы

Процесс соответствует одному *потоку управления*.
Erlang разрешает создавать очень

ГЛАВА 6. ПРОЦЕСС
большое количе-
ство параллель-
но работающих про-
цессов, каждый из
которых исполня-
ется, как будто он
имеет свой соб-
ственный вирту-
альный процес-
сор. Когда процесс,
исполняющийся в ну

ГЛАВА 6. ПРОЦЕСС

ри функции
вызывает другую

функцию
он будет ждать,

ГЛАВА 6. ПРОЦЕСС

f u

пока
не завершится и
затѐм извлечѐт или
получит резуль-
тат. Если вместо
этого он *породит*
новый процесс, ис-
полняющий ту же

ГЛАВА 6. ПРОЦЕСС

fun

то оба процесса
продолжат испол-
няться одновре-
менно (конкурент-

ГЛАВА 6. ПРОЦЕСС

но).

не будет ждать за-

вершения

и единственный

ГЛАВА 6. ПРОЦЕСС
способ передать
результат — это
передача сообще-
ний.

Процессы Erlang
— очень лёгкие с
малым расходом
памяти, легко стар-
туют и легко за-
вершают работу,
и расходы на их
планировку во вре-
мя выполнения оче

ГЛАВА 6. ПРОЦЕСС небольшие. Иден- тификатор про-

Р
цесса, или
идентифицирует
существующий или
недавно существо-
вавший процесс.
Встроенная функ-

ГЛАВА 6. ПРОЦЕСС

ция Se

Р
возвращает
вызвавшего её про
цесса.

ГЛАВА 6 ПРОЦЕСС

6.1 Создани процес- сов

Процесс создаёт-
ся с помощью встро

ГЛАВА 6. ПРОЦЕСС

енной функции

ГЛАВА 6. ПРОЦЕСС

spa

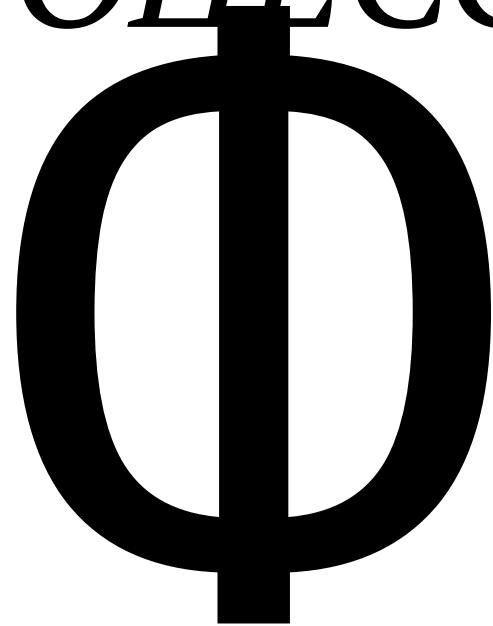
ГЛАВА 6. ПРОЦЕСС

Аргумент

МО

должен быть равен имени модуля, который содержит нужную

ГЛАВА 6. ПРОЦЕСС



функцию, и

— имени экспор-
тированной функ-
ции в этом моду-

ГЛАВА 6. ПРОЦЕСС

ле. Список

— параметры, которые будут переданы запущенной в новом про-

ГЛАВА 6. ПРОЦЕСС

цессе функции.

создаёт новый процесс и возвращает его идентифи-

катор,

Pi

ГЛАВА 6. ПРОЦЕСС

Новый процесс начинается с выполнения такого кода:

Модд

ГЛАВА 6. ПРОЦЕСС

ФУН

должна быть экспортирована, даже если процесс с ней порождает другую функцией в том же мо-

ГЛАВА 6. ПРОЦЕСС
дуле. Есть и дру-
гие встроенные фу-
нкции для порожд-
дения процессов,

например **S**

порождает про-
цесс на другом уз-

ГЛАВА 6. ПРОЦЕСС ле Erlang.

6.2 Зарегис процес- СЫ

Процесс может быть
связан с некото-
рым именем. Имя
процесса должно
быть атомом и оно

ГЛАВА 6. ПРОЦЕСС
автоматически осв
бождается, если
процесс заверша-
ет свою работу. Сле
дует регистриро-
вать только ста-
тические (посто-
янно живущие) про
цессы.

ГЛАВА 6. ПРОЦЕСС

Встроенные
функции
для реги-
страции
имён

reg
Начинает
атом
DIA

ГЛАВА 6 ПРОЦЕССЫ

6.3 Сообщение между процессами

Процессы сообщаются друг с другом посредством отправки и получения сообщений

Сообщения отправ

ГЛАВА 6. ПРОЦЕСС
ляются используя
оператор отпра-



ки (●) и при-
нимаются с помо-
щью конструкции

рес

ГЛАВА 6. ПРОЦЕСС
Передача сообще-
ний *асинхронная*
(не блокирует от-
правителя до до-
ставки сообщения)
и *надёжная* (сооб-
щение гаранти-
рованно достига-
ет получателя, ес-
ли он существу-
ет).

ГЛАВА 6. ПРОЦЕСС

6.3.1 Отправк

Pid

ГЛАВА 6. ПРОЦЕСС

Оператор отпра-



КИ (●) ПОСЫ-

лает значение

в форме сообще-

В

ГЛАВА 6. ПРОЦЕСС ния процессу, указ- занному иденти-

Р
фикатором
где сообщение бу-
дет помещено в
конец его очере-
ди сообщений.

ГЛАВА 6. ПРОЦЕСС

Значение

ВЫ

также будет значением, возвращённым операцией

ГЛАВА 6. ПРОЦЕСС



тором (●).
должен быть иден-
тификатором про-
цесса, зарегистри-
рованным в СИ-
стеме именем или
кортежем в виде

ГЛАВА 6. ПРОЦЕСС

{ ИМ

ИМЯ

где

— это зарегистри-
рованное имя про-

ГЛАВА 6. ПРОЦЕСС
цесса на удален-

УЗ

НОМ

(см. главу ??). Опе-
ратор отправки со-

!

общения (●)
не может возвра-

*ГЛАВА 6. ПРОЦЕСС
ТИТЬ ОШИБКУ, да-
же если в качестве
получателя был ука-
зан несуществу-
ющий процесс.*

ГЛАВА 6. ПРОЦЕСС

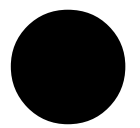
6.3.2 Получен

рес

ГЛАВА 6. ПРОЦЕСС

Это выражение принимает сообщения, отправленные процессу с помощью оператора

ра отправки (



ГЛАВА 6. ПРОЦЕСС

Обр

последовательно
сопоставляются с
первым сообще-
нием в очереди
сообщений, затем
со вторым и так
далее. Если сопо-

ГЛАВА 6 ПРОЦЕСС
ставление прохо-
дит успешно и необ
зательный список
охранных выра-

жений

ОХ

ГЛАВА 6. ПРОЦЕСС

разж

t

тоже равен
то сообщение уда-
ляется из очере-

ГЛАВА 6. ПРОЦЕСС ДИ СООБЩЕНИЙ И СООТВЕТСТВУЮЩАЯ ЦЕПОЧКА ВЫРАЖЕ-

Тел
ний
вычисляется. Именно
но порядок урав-
нений с образца-
ми решает поряд-
док получения со-

ГЛАВА 6. ПРОЦЕСС
общений, а не по-
рядок, в котором
они прибывают.
Это называется *из-*
бирательным при-
ёмом сообщений.
Значение, возвра-

щаемое

Те

ГЛАВА 6. ПРОЦЕСС
и будет значени-
ем, возвращённым
всем выражени-

ем **rec**

ГЛАВА 6. ПРОЦЕСС

рес

НИКОГДА НЕ ПРИ-
ВОДИТ К ВОЗНИК-
НОВЕНИЮ ОШИБКИ.
Процесс может быть
поставлен на па-
узу во время ожи-
дания, ВОЗМОЖНО
навсегда, до тех
пор, пока не по-

ГЛАВА 6. ПРОЦЕСС
явится сообщением,
отвечающее од-
ному из образцов
с охранной после-
довательностью,

равной

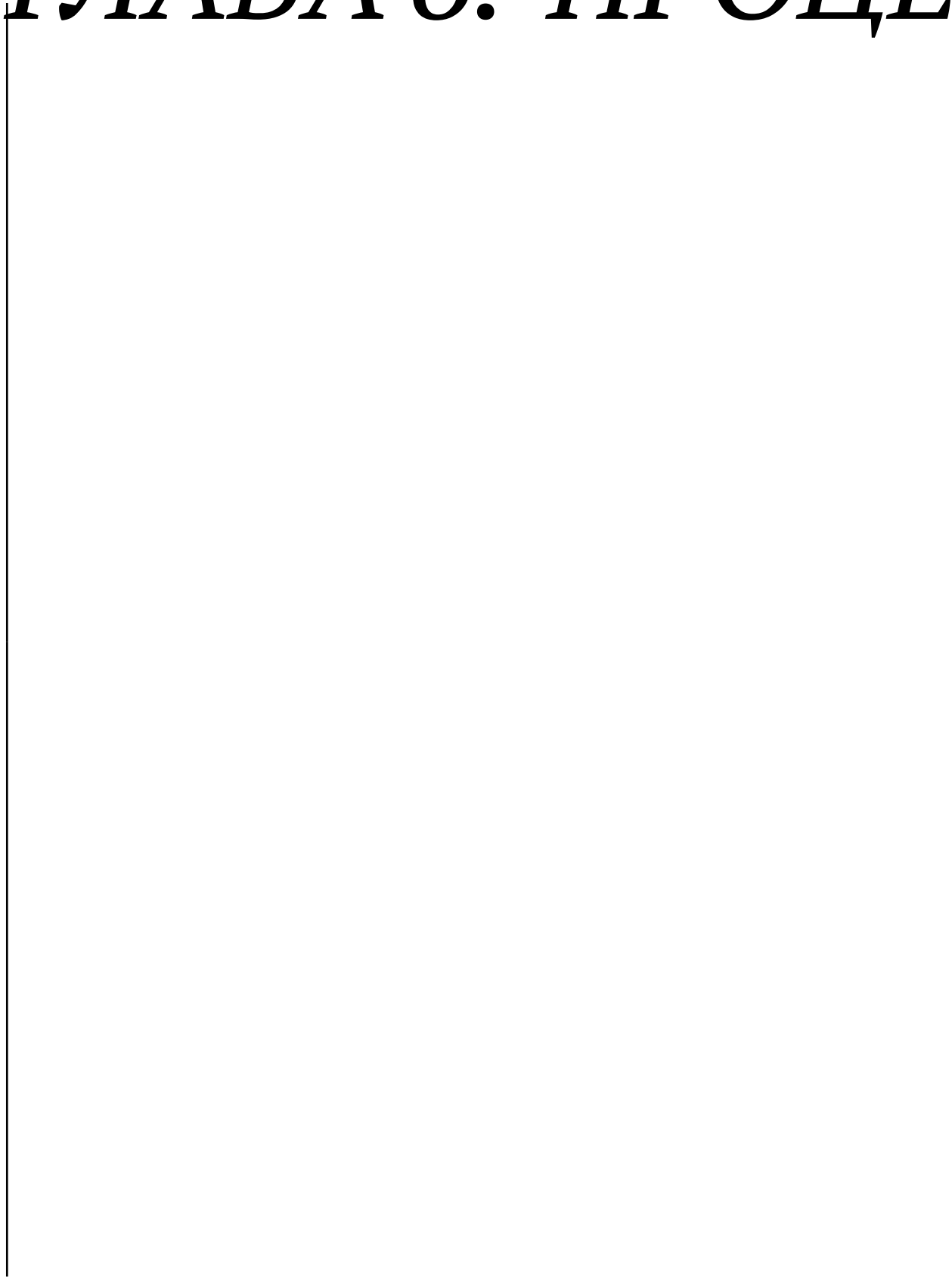
tr

ГЛАВА 6. ПРОЦЕСС

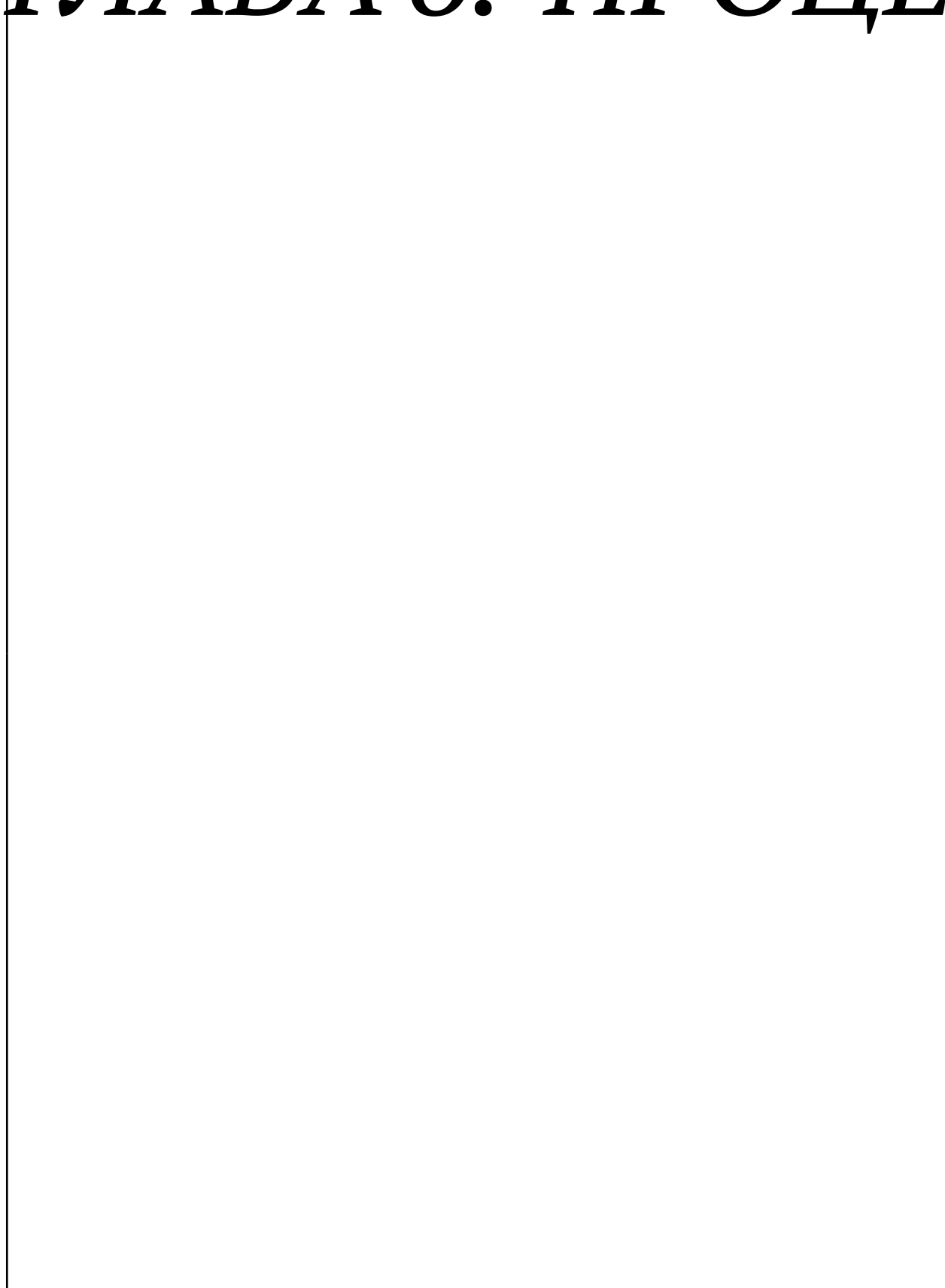
wait

ГЛАВА 6. ПРОЦЕСС

ГЛАВА 6. ПРОЦЕСС



ГЛАВА 6. ПРОЦЕСС



ГЛАВА 6. ПРОЦЕСС



ГЛАВА 6. ПРОЦЕСС

6.3.3 Получен с тай- маутом

рес

ГЛАВА 6. ПРОЦЕСС

Выр

должно вычислять-
ся в целое число

между 0 и 1

ГЛАВА 6. ПРОЦЕСС
(значение долж-
но помещаться в
32 бита). Если ни
одно подходящее
сообщение не при-

было в течение

миллисекунд, то

ГЛАВА 6. ПРОЦЕСС

выражение
вычисляется и его
возвращаемое зна-
чение становит-
ся результатом все-

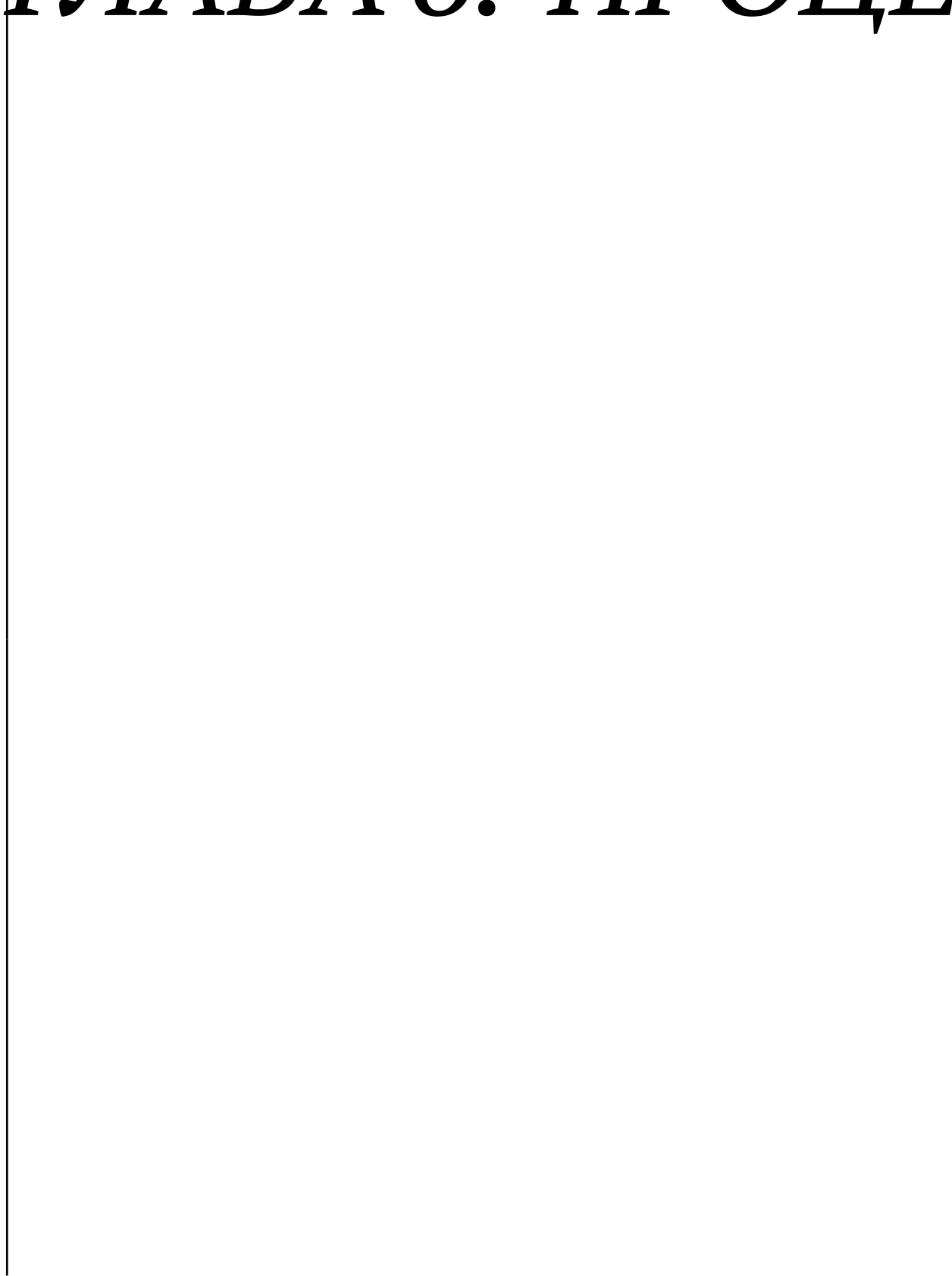
го выражения

ГЛАВА 6. ПРОЦЕСС

wait

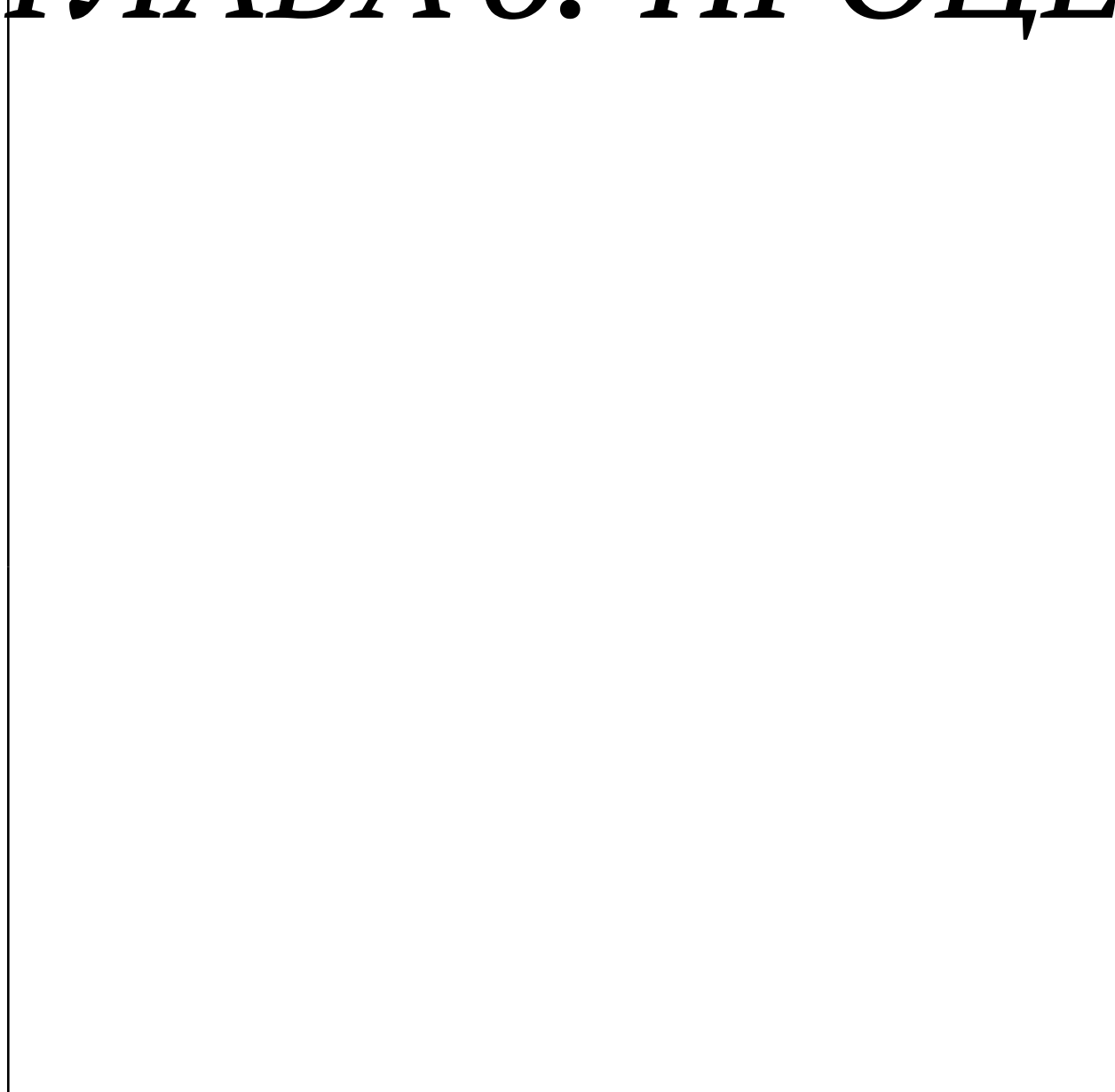
ГЛАВА 6. ПРОЦЕСС

ГЛАВА 6. ПРОЦЕСС

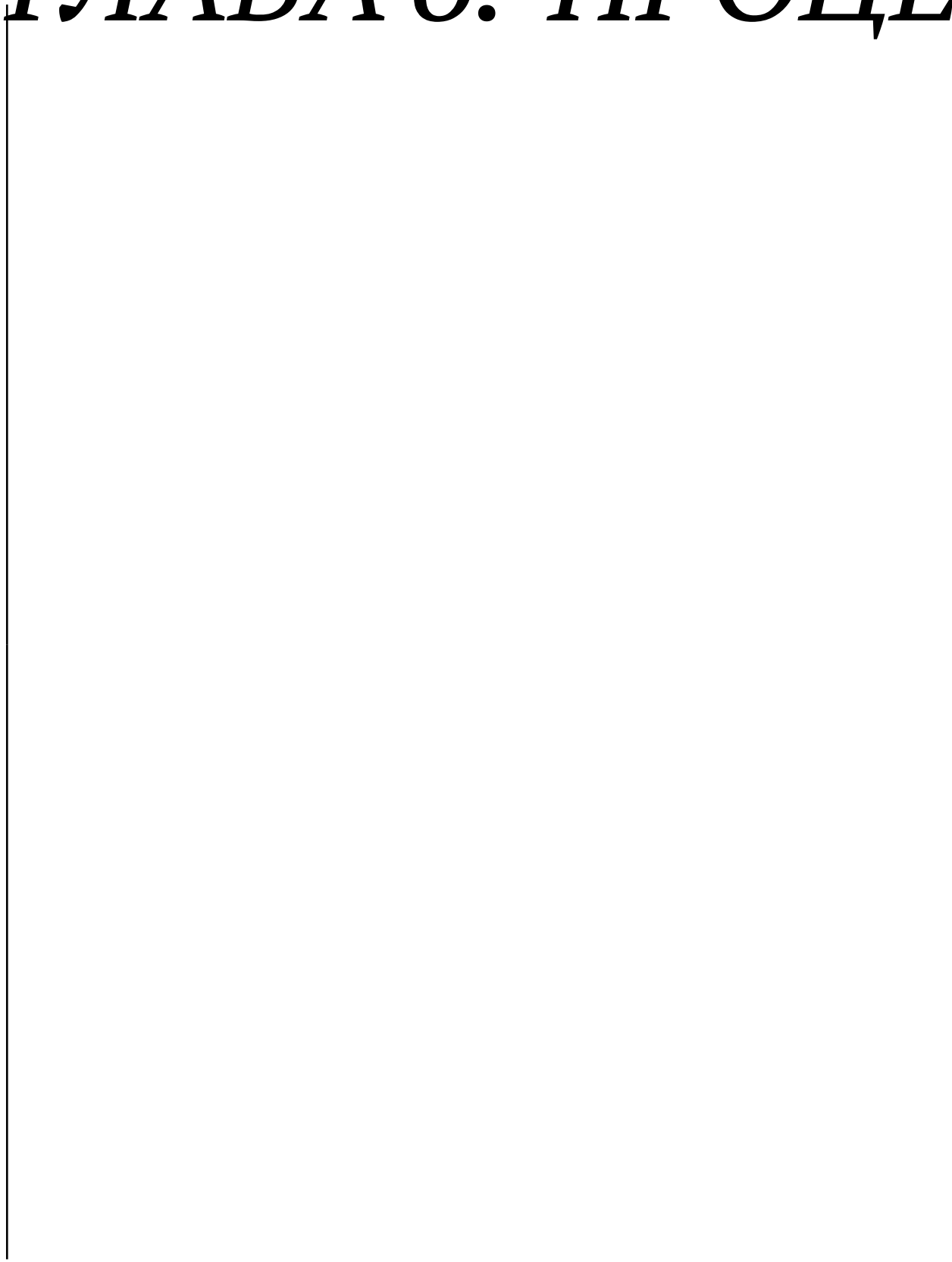


FEDERAL BUREAU OF INVESTIGATION	
Form 101 (Rev. 11-15-83)	
U.S. DEPARTMENT OF JUSTICE	
1. NAME (Last, first, middle initial)	
2. BIRTH DATE	
3. BIRTH PLACE	
4. SOCIAL SECURITY NUMBER	
5. CURRENT ADDRESS	
6. PREVIOUS ADDRESSES	
7. EMPLOYMENT HISTORY	
8. EDUCATION	
9. MARITAL STATUS	
10. PAST AND PRESENT EMPLOYERS	
11. REFERENCES	
12. COMMENTS	

ГЛАВА 6. ПРОЦЕСС



ГЛАВА 6. ПРОЦЕСС



ГЛАВА 6. ПРОЦЕСС



Выражение **I**
без образцов мо-
жет быть исполь-

ГЛАВА 6. ПРОЦЕСС зовано для ре- ализации простых таймаутов.

ГЛАВА 6. ПРОЦЕСС

reflect

ГЛАВА 6. ПРОЦЕСС

Два осо-
бых случая
для зна-
чения
таймаута

Выр

•

inf

ГЛАВА 6 ПРОЦЕСС

6.4 Заверш работы процес- са

Процесс всегда за-
вершается по неко-
торой причине вы-
хода (exit reason),
которая может быть

ГЛАВА 6. ПРОЦЕСС

любым термом Erla

Если процесс за-
вершился нормаль-
но, то есть его код
исполнился до кон-
ца, то причиной
выхода будет атом

no1

Процесс может за-

ГЛАВА 6. ПРОЦЕСС
вершить себя сам
вызывая одну из
следующих встро-
енных функций:

ГЛАВА 6. ПРОЦЕСС

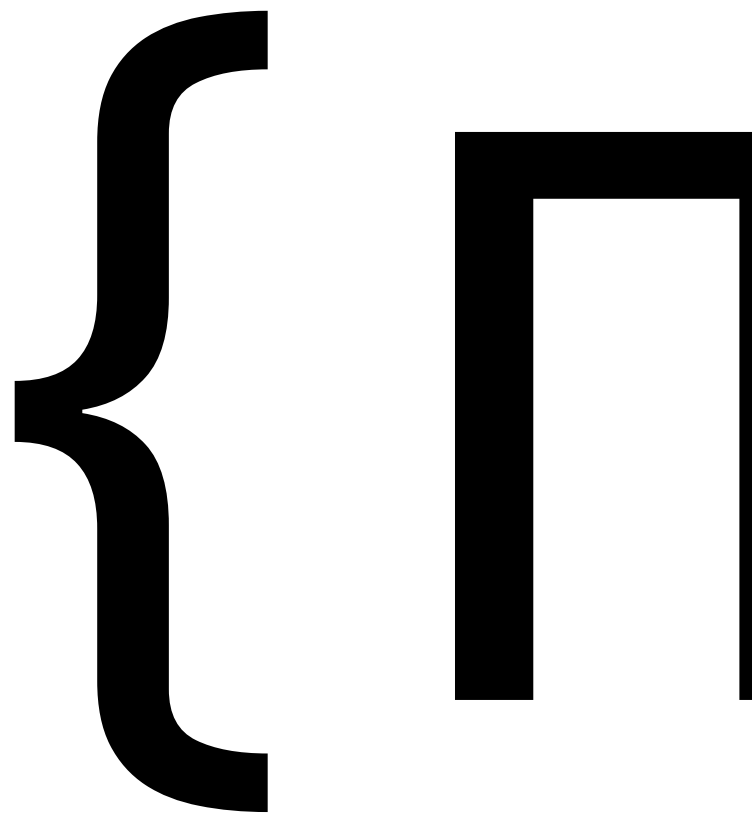
exi

erl

ГЛАВА 6. ПРОЦЕСС

Процесс заверша-
ется с причиной

выхода



когда происходит
ошибка времени
исполнения.

ГЛАВА 6. ПРОЦЕСС
Процесс также может быть завершён, если он получает сигнал выхода с любой другой причиной кро-

по
ме
(см. раздел ??).

ГЛАВА 6 ПРОЦЕСС

6.5 СВЯЗИ МЕЖДУ ПРОЦЕСС- САМИ

Два процесса могут быть **связа-
НЫ** друг с другом.
Связи двуправ-
ленны и может су-
ществовать толь-

ГЛАВА 6 ПРОЦЕССЫ
ко одна связь между любыми двумя процессами (имеются в виду уникальные идентификаторы процессов). Процесс с иде

тификатором

P

ГЛАВА 6 ПРОЦЕСС
может связаться
с процессом, име-
ющим идентифи-

Рі

катор
используя встро-
енную функцию

ГЛАВА 6. ПРОЦЕСС

Lin

Функция Sp

ГЛАВА 6. ПРОЦЕСС

ФУН

Арг

ГЛАВА 6. ПРОЦЕСС
порождает и сразу же связывает процессы одной атомарной операцией.

Связь между процессами можно упростить используя функ-

ГЛАВА 6. ПРОЦЕСС

ЦИЮ

и n

ГЛАВА 6. ПРОЦЕСС

6.5.1 Обработка ошибок между процессами

Когда процесс завершает работу, он отправляет сигналы выхода всем процессам, с которыми он свя-

ГЛАВА 6. ПРОЦЕСС
зан. Они в свою
очередь также за-
вершают работу
или обрабатыва-
ют сигнал выхо-
да каким-либо спо-
собом. Эта возмож-
ность может ис-
пользоваться для
построения иерар-
хических программ-
ных структур, где
некоторые из про-
цессов присмат-

ГЛАВА 6. ПРОЦЕССЫ
привают за другими процессами, например рестартуя их, если они завершаются аварийно.

ГЛАВА 6. ПРОЦЕСС

6.5.2 Отправка сигна- лов ВЫ- хода

Процесс всегда завершает работу с причиной выхода, которая отправляется в виде сигнала выхода всем связанным процессам. Встроен-

ГЛАВА 6. ПРОЦЕСС

ная функция е

при

ГЛАВА 6. ПРОЦЕСС
посылает сигнал
выхода другому

процессу

по указанной

ГЛАВА 6 ПРОЦЕСС

не влияя на процес
отправитель.

6.5.3 Получен сигна- лов вы- хода

Если процесс по-
лучает сигнал вы-
хода с причиной
выхода другой, кро

ГЛАВА 6. ПРОЦЕСС

ме

как

ног

он также завер-
шит свою рабо-
ту и отправит сиг-
налы выхода с той
же причиной всем
своим связанным
процессам. Сиг-

ГЛАВА 6 ПРОЦЕСС

нал выхода с три-

но
чиной
игнорируется и не
приводит к тако-
му поведению. Это
поведение мож-
но изменить с по-
мощью вызова вст

ГЛАВА 6. ПРОЦЕСС

енной функции

trw

ГЛАВА 6. ПРОЦЕСС
Процесс после этого способен **перехватывать сигналы выхода**. Это означает, что сигнал выхода трансформируется в обы

ГЛАВА 6. ПРОЦЕСС

ное сообщение:

Pi d

ГЛАВА 6. ПРОЦЕСС

П р и

который помеща-
ется в почтовый
ящик процесса и
может быть при-
нят и обработан,
как обычное со-

ГЛАВА 6. ПРОЦЕСС общение исполь-

зую **те**с

Однако, вызов встр

ГЛАВА 6. ПРОЦЕСС

енной функции

ki

ГЛАВА 6. ПРОЦЕСС

завершает рабо-

Рту процесса
безусловно, неза-
висимо от того,
способен ли он пе-
рехватывать сиг-
налы выхода или
нет.

ГЛАВА 6 ПРОЦЕСС

6.6 МОНИТО

Рі

Процесс
МОЖЕТ создать мо-
НИТОР для про-

ГЛАВА 6. ПРОЦЕСС

Р1

цесса
используя встро-
енную функцию:

ГЛАВА 6. ПРОЦЕСС

erl

которая возвра-
щает ссылочное

ГЛАВА 6. ПРОЦЕСС

значение (

Если после этого

Pid

завершит работу

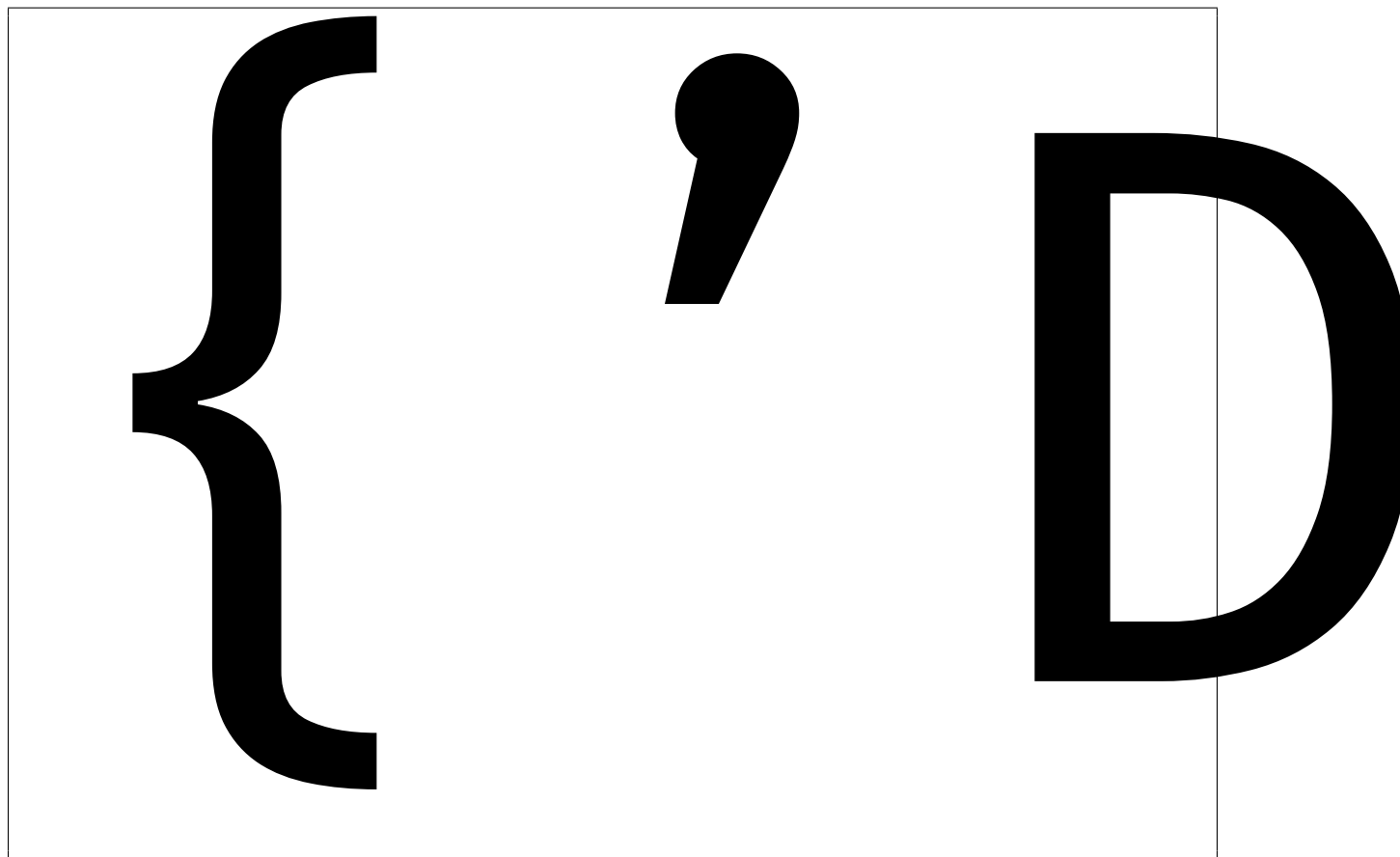
ГЛАВА 6. ПРОЦЕСС

При

с
выхода, то следу-
ющее сообщение
будет отправле-

ГЛАВА 6. ПРОЦЕСС

но процессу



ГЛАВА 6. ПРОЦЕСС

Если процесс
не существует, то

сообщение
будет отправле-
но немедленно и

ГЛАВА 6 ПРОЦЕСС

поле

пр

будет установле-

но равным

н

Мониторы одно-

ГЛАВА 6. ПРОЦЕСС

направлены, то есть

если **Pi**

R —
следит за
то он получит со-
общение о смер-

ГЛАВА 6. ПРОЦЕСС

ТИ **Ріс**

Ріс

НО
**не получит сообще-
щение о смерти**

ГЛАВА 6. ПРОЦЕСС

Рид

Повторные вызо-
вы функции

ГЛАВА 6. ПРОЦЕСС

erl

pid

ГЛАВА 6. ПРОЦЕСС
создадут несколько
ко независимых
мониторов и каж-
дый из них отпра-

вит сообщение

ГЛАВА 6. ПРОЦЕСС

Р
когда процесс
завершит работу.
Монитор можно
удалить, вызывая

ГЛАВА 6. ПРОЦЕСС

функцию **е** **1**

Возможно созда-
ние мониторов для
процессов с заре-
гистрированными
именами, а так-
же запущенных на
других узлах.

ГЛАВА 6 ПРОЦЕССЫ

6.7 Приоритетные процессы

Встроенная функ-

ция **prc**

ГЛАВА 6. ПРОЦЕСС

При

определяет при-
оритет текущего

ГЛАВА 6. ПРОЦЕСС

процесса.

может быть од-
ним из следую-

щих значений

ГЛАВА 6. ПРОЦЕСС
(по умолчанию),

**Low
high**

ГЛАВА 6. ПРОЦЕСС

или

max

Изменение приоритета процесса не рекомендуется и должно производиться только в особых случаях. Проблема, которая требует смены приорите-

ГЛАВА 6 ПРОЦЕСС
та процесса, ча-
ще всего может быть
решена и другим
подходом.

6.8 Словарь процес- са

Каждый процесс
имеет собствен-

ГЛАВА 6. ПРОЦЕСС НЫЙ СЛОВАРЬ, ЯВ- ЛЯЮЩИЙСЯ СПИС- КОМ ПАР ТЕРМОВ

в форме

{ К

ГЛАВА 6. ПРОЦЕСС

Зна

ГЛАВА 6. ПРОЦЕСС

Встроенные
функции
для ра-
боты со
словарём
процесса

решит

Содержит

в

ГЛАВА 6. ПРОЦЕСС
Словари процессов могут использоваться для того, чтобы хранить глобальные переменные в приложении, но их слишком активное использование обычно усложняет отладку и считается плохим стилем программирования.

Обра- ботка ошиб- бок

Эта глава описы-
вает обработку оши
605

ГЛАВА 7. ОБРАБОТ
бок внутри про-
цесса. Такие ошиб-
ки известны ещё
под названием **ис-**
ключения.

ГЛАВА 7 ОБРАБОТКА

7.1 Классы исключений и причины отбоя

Классы исключений

еr
ошибка
времени

ГЛАВА 7. ОБРАБОТКА
Появление исклю-
чения приводит
к аварийной оста-
новке процесса,
то есть его испол-
нение останавли-
вается и он и его
данные удаляют-
ся из системы. Так-
же это действие
называется *уни-*
чтожением (termin
После этого сиг-
налы выхода по-

ГЛАВА 7. ОБРАБОТКА
сылаются все взаимосвязанным процессам. Исключение состоит из класса, причины выхода и копии стека вызовов. Стек вызовов можно сформировать и получить в удобном виде с помощью функ

ГЛАВА 7. ОБРАБОТКА

ции **еі**

Ошибки времени
исполнения и дру-
гие исключения
могут не приво-
дить к смерти про-
цесса, если исполь-
зовать выражения

ГЛАВА 7. ОБРАБОТКА

тгу

и са т

ГЛАВА 7. ОБРАБОТКА Для исключений,

имеющих класс
например для обычных
ошибок времени
выполнения,
причиной выхода
да будет кортеж

ГЛАВА 7. ОБРАБОТКА

{ пр

сте

где **При**

— это терм, указывающий более точно на тип ошибки.

ГЛАВА 7. ОБРАБОТКА

Причины ВЫХОДА

वाद

перо
аргумент
недопу-
СТИМОГО
ТИПА.

वाद

ГЛАВА 7. ОБРАБОТКА

СТЕ

— это цепочка вызовов функций, которые исполнялись в тот момент, когда произошла ошибка, даётся в виде списка корте-

ГЛАВА 7. ОБРАБОТКА

жей

}

М

Имя

ГЛАВА 7. ОБРАБОТКА

Арн

где первым идёт
самый недавний
вызов. Иногда са-
мый последний вы

ГЛАВА 7. ОБРАБОТКА

ЗОВ ВМЕСТО

будет содержать

Арг

ГЛАВА 7. ОБРАБОТКА

{МО

ИМЯ

ГЛАВА 7. ОБРАБОТКА

Арг

ГЛАВА 7. ОБРАБОТКА

7.2 Catch и throw

cat

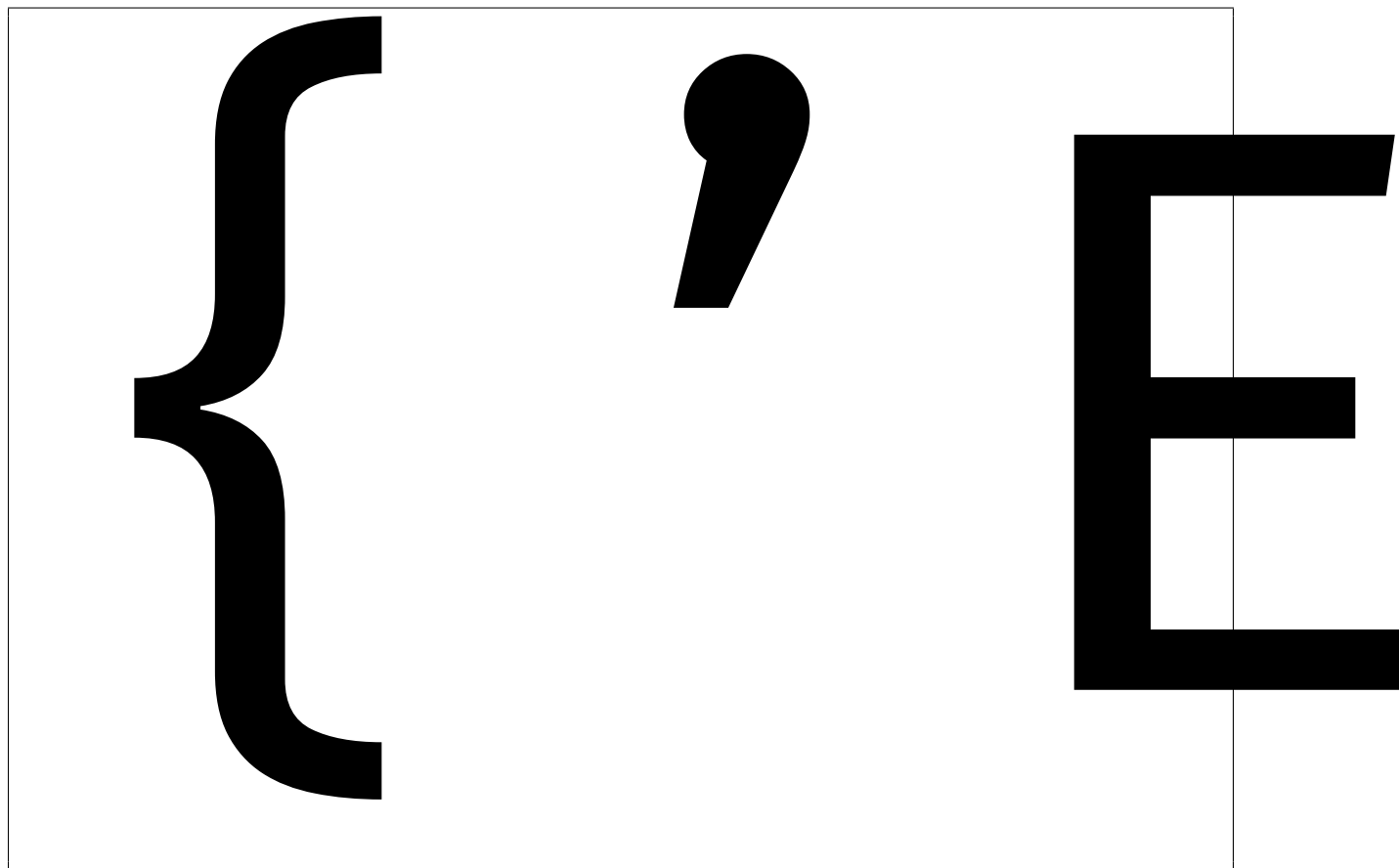
Такая запись возвращает резуль-

ГЛАВА 7. ОБРАБОТ

тат вычисления

если во время вычисления не возникло исключение. В случае исключения возвращаемое значение будет кортежем с

ГЛАВА 7. ОБРАБОТКА информацией о ней



Такое исключение
считается *пойман-*
ным. Непойман-
ное исключение
приведёт к уни-

ГЛАВА 7. ОБРАБОТКА
чтожению процес-
са. Если исклю-
чение было вы-
звано вызовом фун

ции ех:

то будет возвра-
щён кортеж с при-

ГЛАВА 7. ОБРАБОТКА

чиной в виде

Если исключение
возникло при вы-

ГЛАВА 7. ОБРАБОТКА

зове **th**.

тогда будет воз-
вращено значение

ГЛАВА 7. ОБРАБОТКА **Теп**

ГЛАВА 7. ОБРАБОТКА

cat
1 + 2

ГЛАВА 7. ОБРАБОТКА

3



cat

ГЛАВА 7. ОБРАБОТКА

1 + a
{ , e

⇒

ГЛАВА 7. ОБРАБОТКА

С++

Оператор
имеет низкий при-
оритет и выраже-
ния, использую-
щие его часто тре-
буют заключения

ГЛАВА 7. ОБРАБОТКА

в блок **be**

или круглые скоб-
ки.

ГЛАВА 7. ОБРАБОТКА

A

=

=

(

c

a

ГЛАВА 7. ОБРАБОТКА

1 + 2

3

⇒

ГЛАВА 7. ОБРАБОТКА

Встроенная функ-

ция

th.

ИСПОЛЬЗУЕТСЯ ДЛЯ
НЕЛОКАЛЬНОГО ВЫ-
ХОДА ИЗ ФУНКЦИЙ.
ЕЁ МОЖНО ВЫЗЫ-
ВАТЬ ТОЛЬКО ПОД

ГЛАВА 7. ОБРАБОТКА

защитой
что возвратит ре-
зультат вычисле-

ния
ВЫИ

ГЛАВА 7. ОБРАБОТКА

cat

beg

ГЛАВА 7. ОБРАБОТКА

1, 2

end

ГЛАВА 7. ОБРАБОТКА

for

⇒

th

Если

вычисляется за пределами оператора-

ГЛАВА 7. ОБРАБОТКА

ра **с а т**

ТО ВОЗНИКНЕТ ОШИБ
КА ВРЕМЕНИ ВЫПОЛ-

нения **п о**

ГЛАВА 7. ОБРАБОТКА

С++

Оператор
не спасёт процесс
от завершения по
сигналу выхода из
другого связанно-
го с ним процес-
са (если только не
был включен ре-
жим перехвата сиг-

ГЛАВА 7. ОБРАБОТКА налогов выхода, trap_

7.3 Try

Выражение

t

способно разли-
чить различные

ГЛАВА 7. ОБРАБОТКА
классы исключе-
ний. Следующий
пример эмулирует
поведение опи-
санного чуть вы-

ше

cat

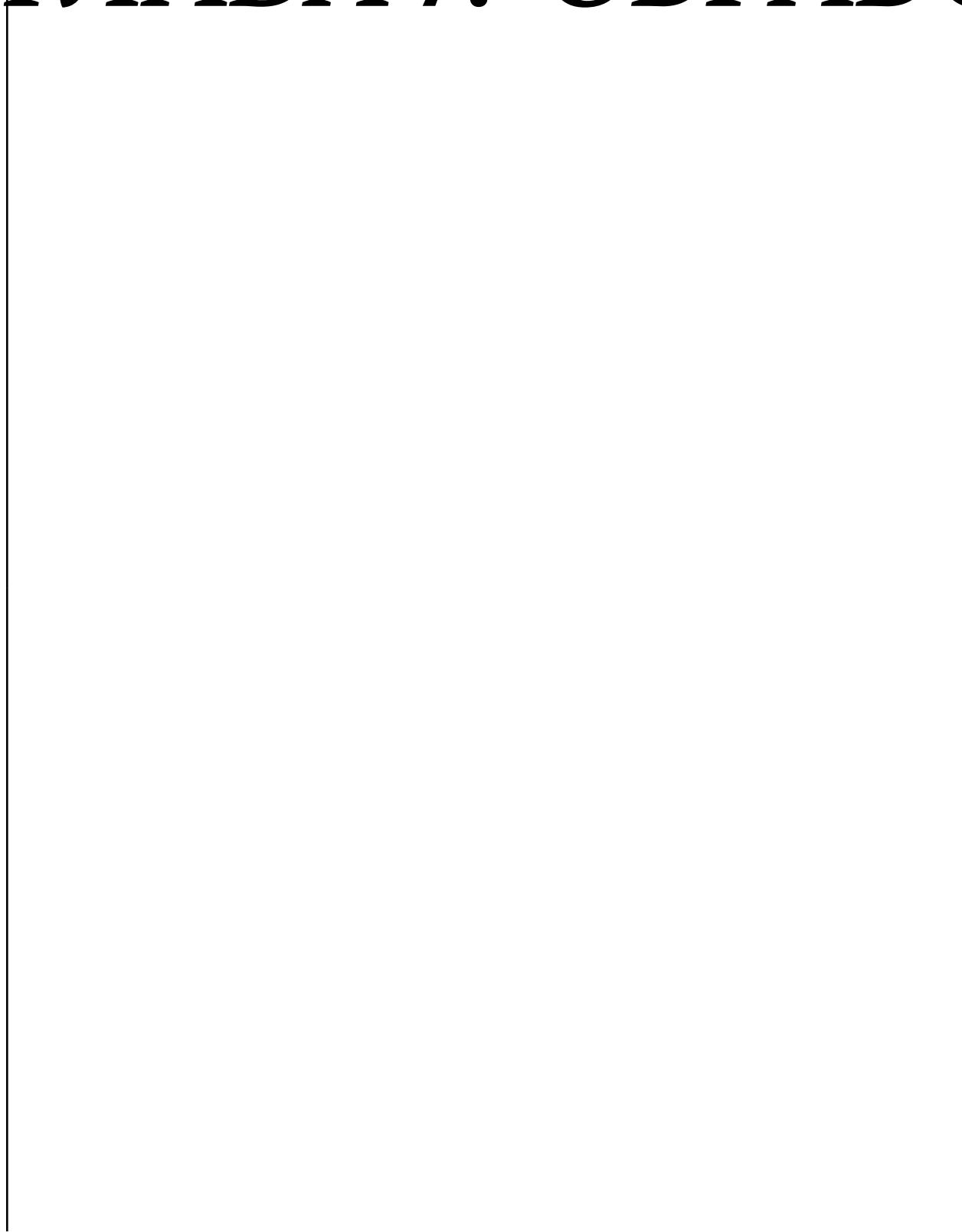
ГЛАВА 7. ОБРАБОТКА

Выр

ГЛАВА 7. ОБРАБОТКА

т 1 у
с а т

ГЛАВА 7. ОБРАБОТ



ГЛАВА 7. ОБРАБОТКА

end

ГЛАВА 7. ОБРАБОТКА

Полное описание

t ru

следующее:

ГЛАВА 7. ОБРАБОТКА

t r y

ГЛАВА 7. ОБРАБОТКА
В наличии долж-
ны обязательно бы-
как минимум од-

на строка
или одно предло-
Сс

ГЛАВА 7. ОБРАБОТ

жение

Может дополни-
тельно использо-
ваться оператор

О

f

по-

ГЛАВА 7. ОБРАБОТКА

сле **ВЫК**

который добавля-

ет вычисление

С

ГЛАВА 7. ОБРАБОТКА

к значению

В

try

возвратит значе-

ГЛАВА 7. ОБРАБОТКА

Вычисление

если только не произойдёт исключение во время его вычисления. Тогда исключение

ГЛАВА 7. ОБРАБОТКА

ловится и ряд

с подходящим
сопоставляются од
за другим с пой-

ГЛАВА 7. ОБРАБОТКА материальным исклю-

чением. Если
не указан, то под-

разумеваются
Если сопоставле-
ние удачно про-

ГЛАВА 7. ОБРАБОТКА ХОДИТ И НЕОБЯЗА- ТЕЛЬНЫЕ ВЫРАЖЕ-

ОХ
НИЯ

тоже равны
ТО ВЫЧИСЛЯЕТСЯ СО-

ГЛАВА 7. ОБРАБОТ

ответствующее
и его результат ста-
новится возвра-
щаемым значени-
ем.

ГЛАВА 7. ОБРАБОТКА

Если не найдено

подходящего

с таким же
и выражениями

ГЛАВА 7. ОБРАБОТКА

Охр т

равными
то исключение пе-
редаётся дальше,
как если бы на-

ГЛАВА 7. ОБРАБОТКА

чальное

ВВ

не было заключе-

т ru

НО В

Исключение, про-

ГЛАВА 7. ОБРАБОТКА

Исходящее во вре-

мя вычисления
не будет пойма-
но.

ГЛАВА 7. ОБРАБОТКА

Если ни один из

Обр

не совпал, то произойдёт ошибка времени исполнения

ГЛАВА 7. ОБРАБОТКА

нения

tr

Если определено,

То
всегда вычисля-

ГЛАВА 7. ОБРАБОТКА данных после всех операций В

try

независимо от возникновения ошибки. Его возвращаемое значение игнорируется и не влияет на значе-

ГЛАВА 7. ОБРАБОТКА
ние всего опера-

тора

tr

(как если бы

a

ГЛАВА 7. ОБРАБОТКА

не было).
вычисляется да-
же если исключе-
ние произошло в

Тел

ГЛАВА 7. ОБРАБОТКА

Текст

или

В ЭТОМ случае исключение передается выше по коду.

Исключение, которое происходит во время вычис-

ГЛАВА 7. ОБРАБОТКА

ления
не ловится, то есть

если
вычислилось по
причине исклю-

ГЛАВА 7. ОБРАБОТКА

чения в

Вь

Тел

ГЛАВА 7. ОБРАБОТКА Теле

или

то оригинальное

исключение будет
потеряно и вме-

сто него полёт вверх

по стеку вызовов

продолжит уже но-
вое исключение.



8

673

Рас- преде- лён- ный Erlang

Распределённая
Erlang-система

ГЛАВА 8. РАСПРЕД
состоит из неко-
торого количества
систем времени
исполнения Erlang
которые сообща-
ются друг с дру-
гом. Каждая та-
кая система на-
зывается **узлом**
(node). Узлы мо-
гут находиться на
одной физической
машине или на раз

ГЛАВА 8. РАСПРЕД
НЫХ И БЫТЬ СОЕДИ-
НЁННЫМИ ПОСРЕД-
СТВОМ СЕТИ. Стан-
дартный механизм
распределения ре-
ализован на ос-
нове TCP/IP соке-
тов, но могут быть
реализованы и дру-
гие механизмы.
Передача сообще-
ний между про-

ГЛАВА 8. РАСПРЕД
цессами на раз-
ных узлах, так же
как и связи меж-
ду процессами и
мониторы, про-
зрачно реализо-
вана используя иде
тификаторы про-
цессов (pid). Од-
нако, зарегистри-
рованные имена
локальны для каж-
дого из узлов. На

ГЛАВА 8. РАСПРЕД
зарегистрирован
ный процесс на
конкретном узле
можно ссылать-
ся с помощью кор-

тежа

{ И

Служба отображе-

ГЛАВА 8. РАСПРЕД
ния портов Erlang
(Erlang Port Mapper
Daemon, или **epmd**
автоматически ста-
тует на каждом ком-
пьютере, где име-
ется запущенный
узел Erlang. Он от-
вечает за отобра-
жение имён узлов
в сетевые адре-
са компьютеров.

ГЛАВА 8 РАСПРЕДЕЛЕНИЕ 8.1 УЗЛЫ

Узел — это исполняемая в данный момент Erlang-система, которой было назначено имя, используя параметр командной строки

ГЛАВА 8. РАСПРЕД

—
ки

(длинное имя) или

— sn

(короткое имя).

Формат имени узла — атом вида

ГЛАВА 8. РАСПРЕД

ИМЯ

(помните, а
является допустимым в атомах сим-

ГЛАВА 8. РАСПРЕД

волом), где
задаётся пользо-
вателем, запустив-

шим узел, а

— полное имя сер-

ГЛАВА 8. РАСПРЕД
вера, если были
включены длин-
ные имена, или
первая часть име-
ни сервера (если
были использова-
ны короткие име-

на). Функция

n

ГЛАВА 8. РАСПРЕД
возвращает имя
узла. Узлы, исполь-
зующие длинные
имена не могут свя-
зываться с узла-
ми, использующи-
ми короткие име-
на.

ГЛАВА 8 РАСПРЕД

8.2 Соедин между узлами

Узлы распределённой Erlang-системы полностью соединены (каждый с каждым). Первый раз, когда используется новое имя

ГЛАВА 8 РАСПРЕД
узла, производит
ся попытка под-
ключения к это-
му узлу. Если узел

А подключа-

ется к узлу В,

ГЛАВА 8 РАСПРЕД

и узел **В** имел
открытое подклю-
чение к узлу **С**

ГЛАВА 8. РАСПРЕД

то узел **А** то-
же попытается под-
ключиться к уз-

Слу . Эта воз-
можность может
быть отключена

ГЛАВА 8. РАСПРЕД используя пара метр командной строки:

— СС

ГЛАВА 8. РАСПРЕД

fat

Если узел прекращает работу или теряет сеть, все подключения к нему удаляются. Встроенная функция:

ГЛАВА 8. РАСПРЕД

е r l

отключает задан-

у з
ный

ГЛАВА 8. РАСПРЕД

Встроенная функ-

ция **по**

вернёт список под-
ключенных в дан-
ный момент (ви-
димых) узлов.

ГЛАВА 8 РАСПРЕД

8.3 СКРЫТЬ узлы

Иногда полезно подключиться к нужному узлу, не иницилируя веер подключений ко всем остальным узлам. Для этой цели можно использовать **скры-**

ГЛАВА 8. РАСПРЕД- ТЫЙ узел. Скры- тый узел это узел, запущенный с па- раметром команд-



ной строки
Подключения меж-
ду скрытыми уз-
лами и другими

ГЛАВА 8. РАСПРЕД
узлами должны учёт
навливаться вруч-
ную и явно. Скры-
тые узлы не вид-
ны в списке уз-
лов, возвращае-

мом функцией

Вместо этого сле-

ГЛАВА 8. РАСПРЕД ДУЕТ ИСПОЛЬЗОВАТЬ

nod

или nod

ГЛАВА 8. РАСПРЕД
Скрытый узел не
будет включён в
набор узлов, за ко-
торыми следит мо-

дуль

glibc

Узел на языке C
это C-программа,

ГЛАВА 8. РАСПРЕД
написанная, что
бы действовать и
выглядеть, как скры
тый узел в распре-
делённой Erlang-
системе. Библио-

тека

er

ГЛАВА 8. РАСПРЕД
содержит необходимые для этого функции.

8.4 Секреты куки (с

Каждый узел имеет свой собственный ключ, ещё на-

**ГЛАВА 8. РАСПРЕД
зываемый маги-
ческий куки (cookies
который является
ся атомом. Сер-
вер сетевой аутен-
тикации Erlang (по**

а
названием
читает содержи-

ГЛАВА 8. РАСПРЕД мое куки из фай-

ла \$ НО

Если файл не су-
ществовал, он бу-
дет создан и в него
будет записана слу-

ГЛАВА 8. РАСПРЕД чайная строка.

Права доступа к файлу должны быть установлены в восьмеричное 0400 (только для чтения владельцем). Куки локального узла также можно установить с помощью встроенной функ-

ГЛАВА 8. РАСПРЕД

ции ег

КУК

ГЛАВА 8. РАСПРЕД

Текущему узлу поз-
воляется подклю-
чаться к другому

УЗ

узлу
если он знает зна-
чение его куки. Ес-
ли оно отличается
от куки теку-
щего узла (чей ку-

ГЛАВА 8. РАСПРЕД
ки будет исполь-
зован по умолча-
нию), то его на-
до явно устано-
вить с помощью
встроенной функ-

ции

er

ГЛАВА 8. РАСПРЕД

КУК

ГЛАВА 8 РАСПРЕД

8.5 Встроенные функции для распределения

Встроенные функции для распределения

ГЛАВА 8. РАСПРЕД

ГЛАВА 8 РАСПРЕД

8.6 Параметри команд ной стр КИ

Параметры
командной
строки для
распределён-
ного Erlang

СО

ГЛАВА 8 РАСПРЕД

8.7 Модули с под- держ- кой рас преде- лённых систем

Есть несколько до-
ступных модулей,

ГЛАВА 8 РАСПРЕД

которые приходят
ся при програм-
мировании рас-
пределённых си-
стем:

ГЛАВА 8. РАСПРЕД

Модули

с под-
держкой
распре-
делённых
систем

glo

Глобальное
сред-
ство
реги-
стра-
ции



9

714

Пор- ты и драй- веры портов

Порты предостав-
ляют байто-ориент

ГЛАВА 9. ПОРТЫ И
интерфейс к внеш-
ним программам
и связывается с
процессами Erlang
посылая и прини-
мая сообщения в
виде списков бай-
тов. Процесс Erlang
который создаёт
порт, называет-
ся владельцем пор-
та или подклю-
ченным к пор-

**ГЛАВА 9. ПОРТЫ И
ту процессом. Все
коммуникации в
и из порта долж-
ны пройти через
владельца порта.
Если владелец пор-
та завершает ра-
боту, порт тоже
закроется (а так-
же и внешняя про-
грамма, подклю-
ченная к порту,**

ГЛАВА 9. ПОРТЫ И
если она была правильно написана и реагирует на закрытие ввода/-вывода).

Внешняя программа является другим процессом операционной системы. По умолчанию, она должна считывать данные из стандартного

**ГЛАВА 9. ПОРТЫ И
ВХОДА (файловый
дескриптор 0) и
отвечать на стан-
дартный вывод (фа-
ловый дескрип-
тор 1). Внешняя
программа долж-
на завершать свою
работу когда порт
закрывается (ввод/
вывод закрывает-
ся).**

ГЛАВА 9 ПОРТЫ И

9.1 Драйвер портов

Драйверы портов обычно пишутся на языке С и динамически подключаются к системе исполнения Erlang. Встроенный драйвер ве-

ГЛАВА 9. ПОРТЫ И
дет себя как порт
и называется драй-
вером порта. Од-
нако, ошибка в дра-
вере порта может
привести к неста-
бильности во всей
системе Erlang, уте-
кам памяти, за-
висаниям и кра-
ху системы.

ГЛАВА 9 ПОРТЫ И

9.2 Встроенные функции для портов

Функция для создания порта

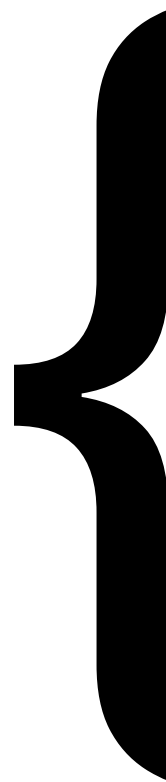
ore

ГЛАВА 9. ПОРТЫ И

ИМЯ

обычно является

кортежем вида



ГЛАВА 9. ПОРТЫ И

где строка

является именем
внешней програм-
мы. Внешняя про-
грамма выполня-
ется за предела-
ми Erlang-системы
если только не най-

ГЛАВА 9. ПОРТЫ И ден драйвер пор-

та с именем

К

Если драйвер най-
ден, он будет ак-
тивирован вместо
команды.

ГЛАВА 9. ПОРТЫ И

Нас

— это список настроек (опций) для порта. Список обычно содержит как минимум один кор

ГЛАВА 9 ПОРТЫ И

теж

указывающий, что
данные, пересы-
лаемые между пор-
том и внешней про-
граммой, предва-
ряются N-байтовыми

ГЛАВА 9. ПОРТЫ И индикатором дан- ны. Разрешённые

значения для
— 1, 2 или 4. Ес-
ли двоичные дан-
ные должны ис-
пользоваться вме-
сто списков бай-
тов, то должна быть
включена опция

ГЛАВА 9. ПОРТЫ И

bin

Владелец порта Pid

связывается с

П

**ГЛАВА 9. ПОРТЫ И
СНОМЮЩЬЮ ОТПРАВ-
КИ И ПОЛУЧЕНИЯ Erla
сообщений. (Лю-
бой процесс мо-
жет послать сооб-
щение в порт, но
ответы от порта
всегда будут от-
правлены только
владельцу порта).**

ГЛАВА 9. ПОРТЫ И СООБЩЕНИЯ, ОТСЫЛАЕМЫЕ В ПОРТ

{

Р

Посылает

•

i

{

Д

В ПОРТ.

ао

ГЛАВА 9. ПОРТЫ И
Данные должны
быть списком ввода
вывода. Список
ввода-вывода (io1
— это либо дво-
ичные данные, ли-
бо смешанный (воз-
можно вложенный
список двоичных
данных и целых
чисел в диапазоне
от 0 до 255.

ГЛАВА 9. ПОРТЫ И Сообщения, получаемые из порта

{ По
Данные
получены
от
внешней
програм-
мы } да

ГЛАВА 9. ПОРТЫ И
Вместо того, что-
бы отправлять и
получать сообще-
ния, имеется ряд
встроенных функ-
ций, которые мож-
но использовать.
Они могут быть
вызваны любым
процессом, а не
только владель-
цем порта.

ГЛАВА 9. ПОРТЫ И Встроенные функции для работы с пор- тами

рог

Отправлять

да

ГЛАВА 9. ПОРТЫ И
Есть несколько до-
полнительных встр
енных функций,
которые приме-
нимы только к драй-
верам портов: это

рог

ГЛАВА 9. ПОРТЫ И

и егип

Загрузка кода

Erlang поддерживает обновление кода во время работы без остановки системы. За-

ГЛАВА 10. ЗАГРУЗКА
мента кода выполняется на уровне модулей.

Код модуля может существовать в системе в двух версиях: **текущая** и **старая**. Когда модуль загружается в систему в первый раз, код становится *теку-*

ГЛАВА 10. ЗАГРУЗК
щим. Если загру-
жается новая вер-
сия модуля, то код
предыдущей вер-
сии, уже имеющий
ся в памяти, ста-
новится *старым*
и новая загружен-
ная версия стано-
вится *текущей*. Обр
но модуль авто-
матически загру-
жается, когда ВЫ-

ГЛАВА 10. ЗАГРУЗКА
звана одна из на-
ходящихся в нём
функций. Если мо-
дуль уже загружен,
то он должен быть
явно перезагру-
жен в новую вер-
сию.

И старый и теку-
щий код полно-
стью функциональ-
ны и могут исполь-
зоваться одновре-

ГЛАВА 10. ЗАГРУЗКА
менно. Полностью
определённые вы-
зовы функций (с
именем модуля)
всегда ссылают-
ся на текущую вер-
сию кода. Одна-
ко старый код мо-
жет продолжать
исполняться дру-
гими процессами.
Если загрузить тре-

ГЛАВА 10. ЗАГРУЗКА
тью версию загру-
женного модуля,
то сервер кода уда-
лит (операция на-
зывается *purge*) ста-
рый код и все про-
цессы, всё ещё ис-
пользующие его,
будут принудитель-
но завершены. За-
тем третья вер-
сия становится *те-*

ГЛАВА 10. ЗАГРУЗКА
кущей и предыду-
щий текущий код
становится ста-
рым.

Чтобы перейти от
старого кода к те-
кущему, процесс
должен выполнить
один полностью
определённый вы-
зов функции (с име-
нем модуля).

ГЛАВА 10. ЗАГРУЗКА

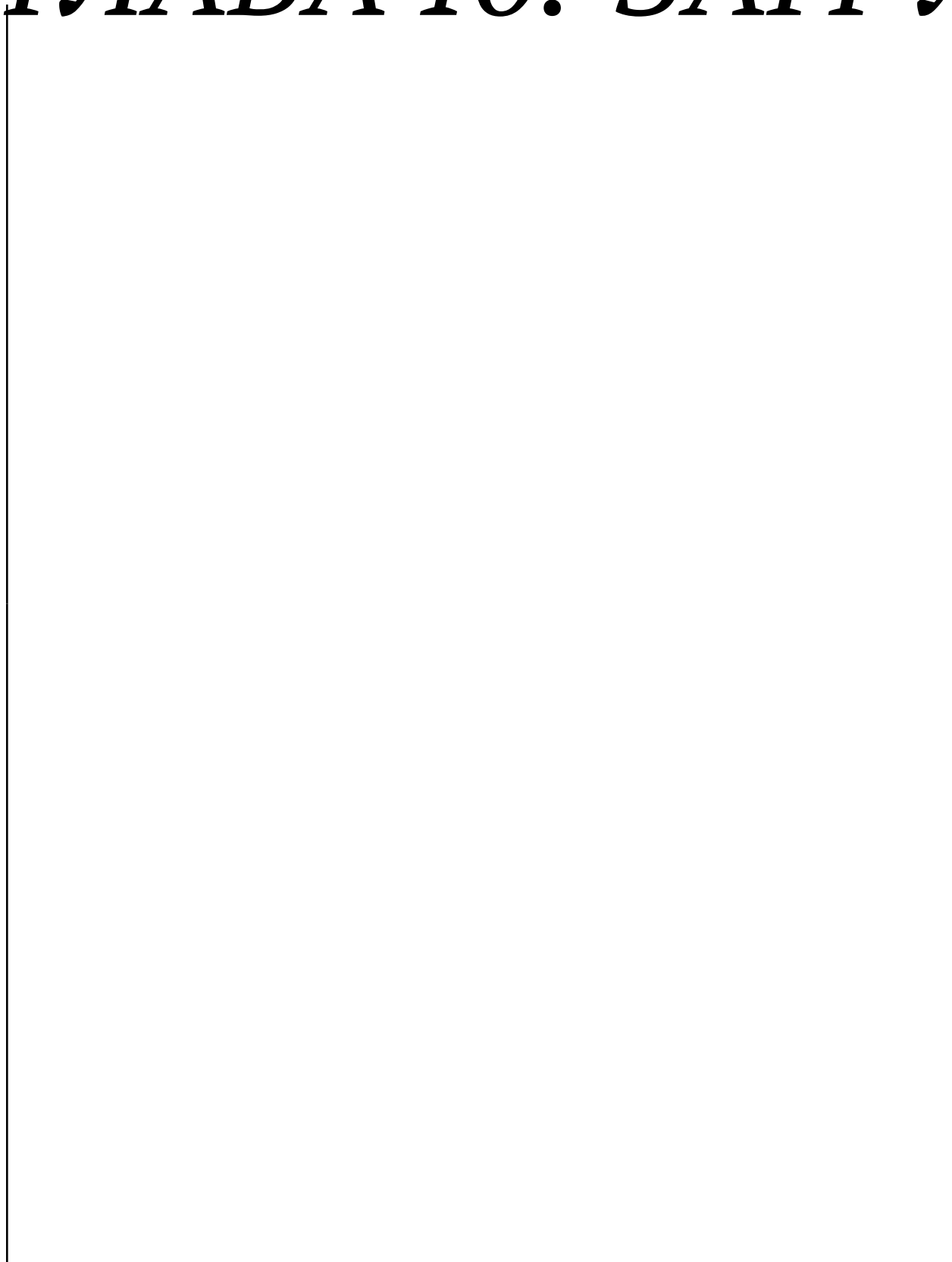
— mo

— ex

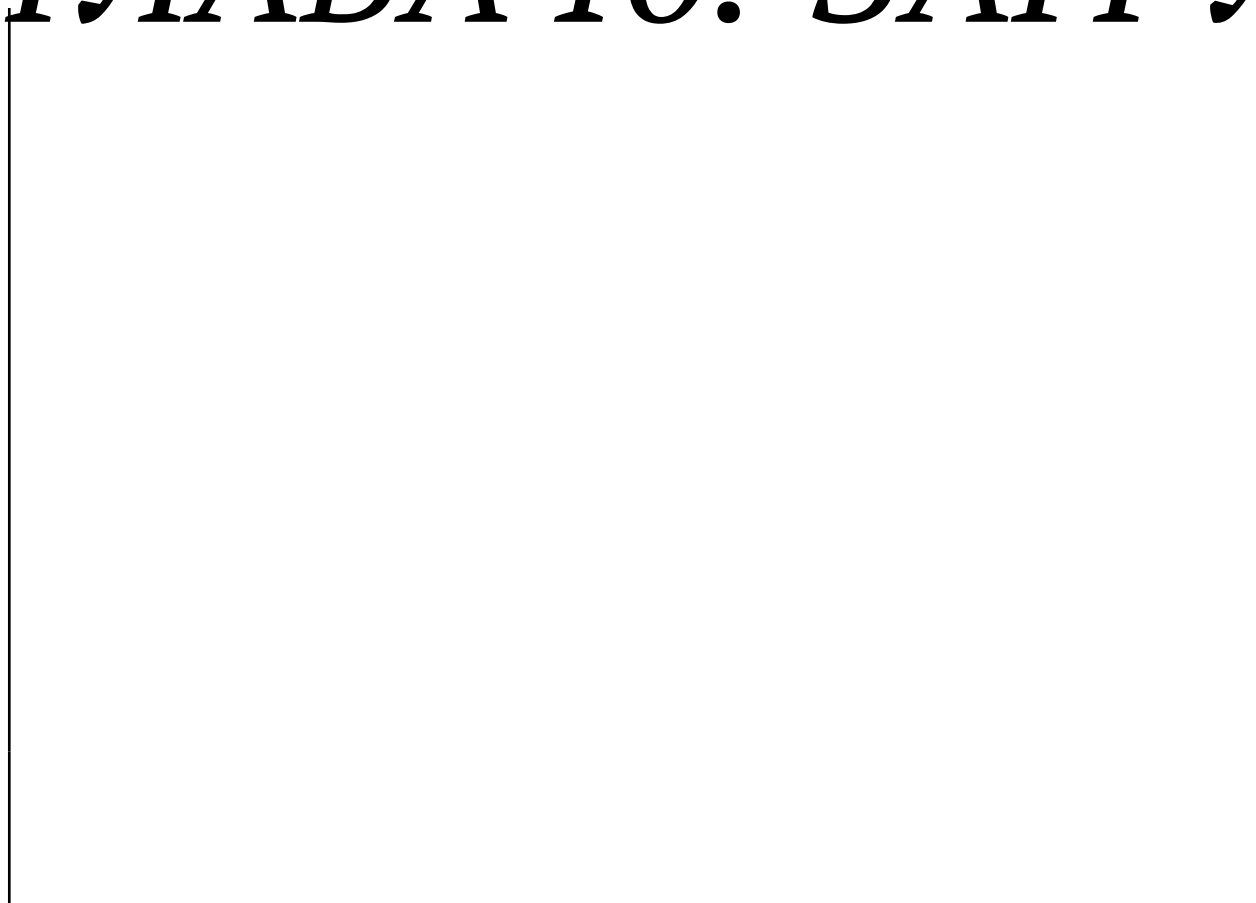
ГЛАВА 10. ЗАГРУЗКА

100

ГЛАВА 10. ЗАГРУЗКА



ГЛАВА 10. ЗАГРУЗКА



ГЛАВА 10. ЗАГРУЗКА

Чтобы заставить

ГЛАВА 10. ЗАГРУЗКА
процесс (в этом
примере) сменить
версию кода, от-
правьте ему со-

общение

СС

Процесс после это-
го выполнит пол-
ностью определён-

ГЛАВА 10. ЗАГРУЗКА

ный вызов **m**

и это переключит
его на текущую вер-
сию кода. Заметь-

ГЛАВА 10. ЗАГРУЗКА

те, что **то**

должна быть для
этого экспорти-
рована.



11

753

Макро- сы

11.1 Опред и ис- поль- зова- ние ма росов



ГЛАВА 11. МАКРОС
Макрос должен
быть определён
перед тем, как он
используется, но
определение мак-
роса можно по-
местить где угод-
но среди атрибу-
тов и определе-
ний функций в мо-
дуле. Если макрос
используется в нес

ГЛАВА 11. МАКРОС
ких модулях, ре-
комендуется по-
местить его опре-
деление во вклю-
чаемый файл. Мак-
рос используется
так:

ГЛАВА 11. МАКРОСЫ

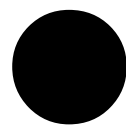


Макросы разво-

ГЛАВА 11. МАКРОС
рачиваются во вре-
мя компиляции
на самом раннем
этапе. Ссылка на

?

макрос



будет заменена на

ГЛАВА 11. МАКРОС

За

текст
так:



ГЛАВА 11. МАКРОС

cal

ГЛАВА 11. МАКРОС

разворачивается
перед компиля-
цией в:

ГЛАВА 11. МАКРОС

cal

ГЛАВА 11. МАКРОС

Ссылка на макрос

ГЛАВА 11. МАКРОС

? Φ γ

• • •

• • •

ГЛАВА 11. МАКРОС

Арг

будет заменена на

ГЛАВА 11. МАКРОС

Зам

где все вхождения

П

переменной

из определения ма

ГЛАВА 11. МАКРОС
росы будут заме-
нены на соответ-

ствующий

А

ГЛАВА 11. МАКРОС

— de

• • •

ГЛАВА 11. МАКРОС

bar

ГЛАВА 11. МАКРОС



будет развёрну-
то в:

ГЛАВА 11. МАКРОС

bar

ГЛАВА 11. МАКРОС

Для просмотра результата разворачивания макросов, модуль можно скомпилировать с парамет-

ГЛАВА 11. МАКРОС

Р

ром

таким образом:

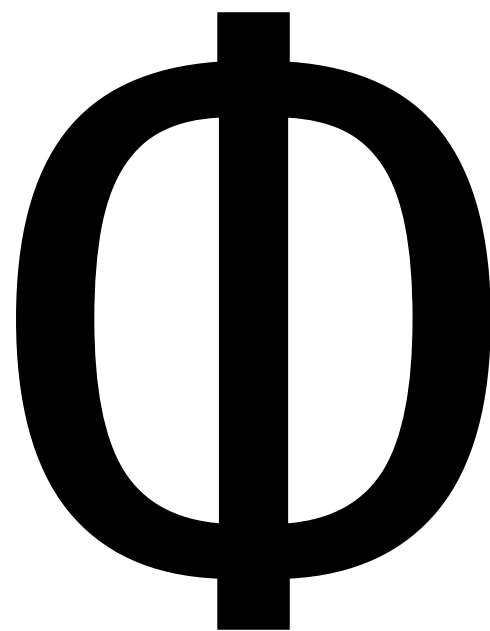
ГЛАВА 11. МАКРОС

com

Это производит
распечатку разо-
бранного кода по-
сле применения
к нему предвари-

ГЛАВА 11. МАКРОС ТЕЛЬНОЙ ОБРАБОТ- КИ И ТРАНСФОРМА- ЦИЙ РАЗБОРА (parse transform), в фай-

ле с именем



ГЛАВА 11. МАКРОСЫ

11.2. Предопределённые макросы

Предопределённые макросы

ВМ

● АТМ,
имя те-
кущего
модуля

ГЛАВА 11. МАКРОС

ГЛАВА 11 МАКРОС

11.3 управ.

испол-

нением

макр

росов

—

un

ГЛАВА 11. МАКРОС

—

i

f

ГЛАВА 11. МАКРОС

—

et

ГЛАВА 11. МАКРОС

— en

i fn

МОЖНО ИСПОЛЬЗО-

ГЛАВА 11. МАКРОС

1
вать вместо
и имеет обратный
смысл.

ГЛАВА 11. МАКРОС



ГЛАВА 11. МАКРОС

— de

— et

ГЛАВА 11. МАКРОС

—

de

—

en

ГЛАВА 11. МАКРОС

Если макрос

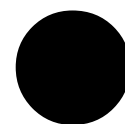
d

определён в то время,
когда идёт компиляция модуля,

ГЛАВА 11. МАКРОСЫ



то макрос



развернётся в вы-
зов печати текста

ГЛАВА 11. МАКРОС

10:

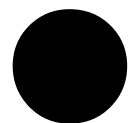
**и обеспечивает пользо-
вателя отладоч-
ным выводом в
консоль.**

ГЛАВА 11 МАКРОСЫ

11.4 Превр аргу- мен- тов ма роса в стро- ку



Запись вида



ГЛАВА 11. МАКРОС
— это параметр,
передаваемый в
макрос, развер-
нётся в представ-
ление аргумента
в строковом ви-
де.

ГЛАВА 11. МАКРОС

— de

ГЛАВА 11. МАКРОС

? T E

ГЛАВА 11. МАКРОС

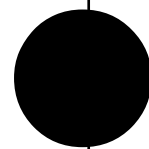
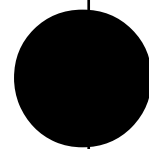
? T E

ГЛАВА 11. МАКРОС Разворачивается В:

ГЛАВА 11. МАКРОС

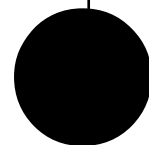
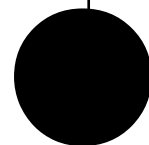


io



ГЛАВА 11. МАКРОС

10



ГЛАВА 11. МАКРОС
Таким образом, получается отладочный вывод как вызванной функции так и результата.



12

798

Даль- ней- шие мате- риа- лы для чтения

Следующие веб-
сайты предлага-

ГЛАВА 12. ДАЛЬНЕ
ют более подроб-
ное объяснение тем
и концепций, крат-
ко описанных в
данном докумен-
те:

ГЛАВА 12. ДАЛЬНЕ

12.1 Русско ресур- сы

- Русские новости

из мира Erlang



ГЛАВА 12. ДАЛЬНЕ

/ / e

и ц

ГЛАВА 12. ДАЛЬНЕ

• Группа ет

и етл

ГЛАВА 12. ДАЛЬНЕЕ на сервере Google Groups.

- Книга Ф. Чезарини «Программирование в Erlang»
русский перевод

h t t

ГЛАВА 12. ДАЛЬНЕ

/ / d

com

ГЛАВА 12. ДАЛЬНЕ

cat

com

ГЛАВА 12. ДАЛЬНЕ

pro

fun

ГЛАВА 12. ДАЛЪНЕ

978

Также обратите внимание на русско-

ГЛАВА 12. ДАЛЬНЕ

язычный канал

на сервере

h.

ГЛАВА 12. ДАЛЬНЕ

/ / j

и u

ГЛАВА 12. ДАЛЬНЕЕ

12.2. АНГЛО- ресур- сы

- Официальная документация по

Erlang:

ht

ГЛАВА 12. ДАЛЬНЕ

/

/

w

e

r

u

ГЛАВА 12. ДАЛЬНЕ

orig

doc

ГЛАВА 12. ДАЛЬНЕ
• Learn You Some Erlang for Great

Good: ht

ГЛАВА 12. ДАЛЬНЕ



com

ГЛАВА 12. ДАЛЪНЕ

- Раздел с уроками на Erlang Central

h t t

ГЛАВА 12. ДАЛЬНЕ

/ / e

o r g

ГЛАВА 12. ДАЛЬНЕ

wiki

.

ind

ГЛАВА 12. ДАЛЬНЕ

р h р
t i t

ГЛАВА 12. ДАЛЬНЕ

Cat

How

ГЛАВА 12. ДАЛЬНЕЕ

Еще вопросы? Спикер

сок рассылки

(адрес для подпис-

ГЛАВА 12. ДАЛЪНЕ

КИ

h t t

/ / e

ГЛАВА 12. ДАЛЬНЕ

o r g

m a i

ГЛАВА 12. ДАЛЬНЕ

lis

erl

является хорошим

ГЛАВА 12. ДАЛЬНЕ
местом для нести-
ных общих дис-
куссий об Erlang/O
языке, реализации
использовании и
вопросы нович-
ков. Если вы не
планируете писать
в рассылку, её мож-
но прочесть без
подписки на сер-
вере Google Groups

ГЛАВА 12. ДАЛЬНЕ

группа **e** **r**

Также обратите внимание на англо-язычный IRC ка-

ГЛАВА 12. ДАЛЬНЕ

нал **#e**

в сети Freenode.