Информатика (основы программирования)

Т.И. Комаров

НИЯУ МИФИ

2023

Почему мы изучаем язык C (субъективное мнение)? I

Может быть следует изучать язык Pascal?

- Преимущества языка Pascal:
  - Изучается в рамках школьной программы
  - Простой и элегантный синтаксис
  - Строгая типизация
  - Возможность прямой работы с памятью
  - Высокая производительность
- Недостатки языка Pascal:
  - Низкая популярность, которая продолжает снижаться
  - Крайне медленное развитие

Вывод

Pascal — отличный язык, который проиграл в битве за место под солнцем

Почему мы изучаем язык C (субъективное мнение)? II

Может быть следует изучать язык Python?

- Преимущества языка Python:
  - Простой, элегантный и очень гибкий синтаксис
  - Крайне высокая популярность и востребованность
  - Огромное количество библиотек
- Недостатки языка Python:
  - Динамическая типизация
  - Низкая производительность
  - Не так прост, как кажется (что там «под капотом»?)
  - Не даёт представления о работе компьютера на низком уровне

Вывод

Python — отличный язык, который подходит для разработки прикладного ПО и может изучаться параллельно в качестве второго языка

Почему мы изучаем язык C (субъективное мнение)? III

Может быть следует изучать язык C?

- Преимущества языка C:
  - Компактный и понятный синтаксис
  - Высокая популярность и востребованность
  - Высокая производительность
  - Полный контроль над памятью
- Недостатки языка C:
  - Сложность разработки (множество возможностей «выстрелить себе в ногу»)
  - Недостаточная стандартизация

Вывод

C — отличный язык, который подходит для разработки системного ПО и может быть использован в качестве основного языка

Исторические факты о языке C I

- Годы разработки первой версии языка: 1969 — 1973 (Деннис Ритчи, Кен Томпсон)
- Официальный год появления языка: 1972
- Место разработки: компания Bell Labs
- Наследник интерпретируемых языков BCPL (Мартин Ричардс) и B (Кен Томпсон, Денис Ритчи)
- Разработан в ОС UNIX и для неё
- К 1973 году большая часть ядра UNIX была переписана на C (одно из первых ядер ОС, написанных не на ассемблере)
- В 1978 году была опубликована первая версия K&R — книги Брайана Кернигана и Денниса Ритчи «Язык программирования C»

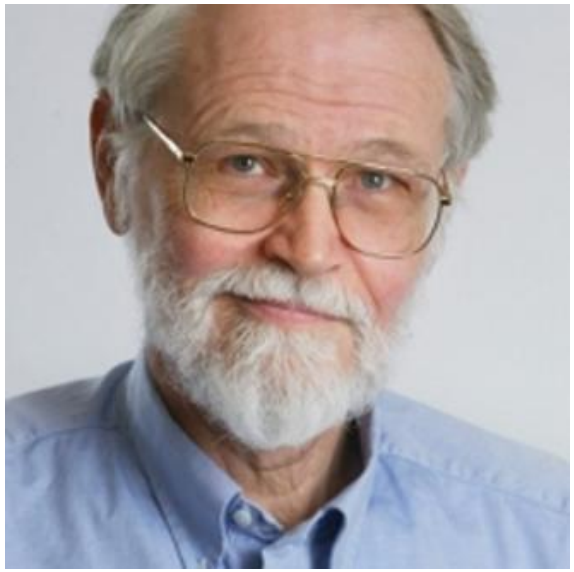
Исторические факты о языке C II

- Существует легенда о том, что язык C появился, т.к. его будущие авторы любили компьютерную игру, аналог Asteroids, и захотели портировать её с компьютера PDP-11 (главного сервера компании) на свободный компьютер PDP-7, стоящий в офисе. Этот компьютер не имел ОС, поэтому её пришлось разработать, для чего и понадобился новый язык программирования

Создатели языка C

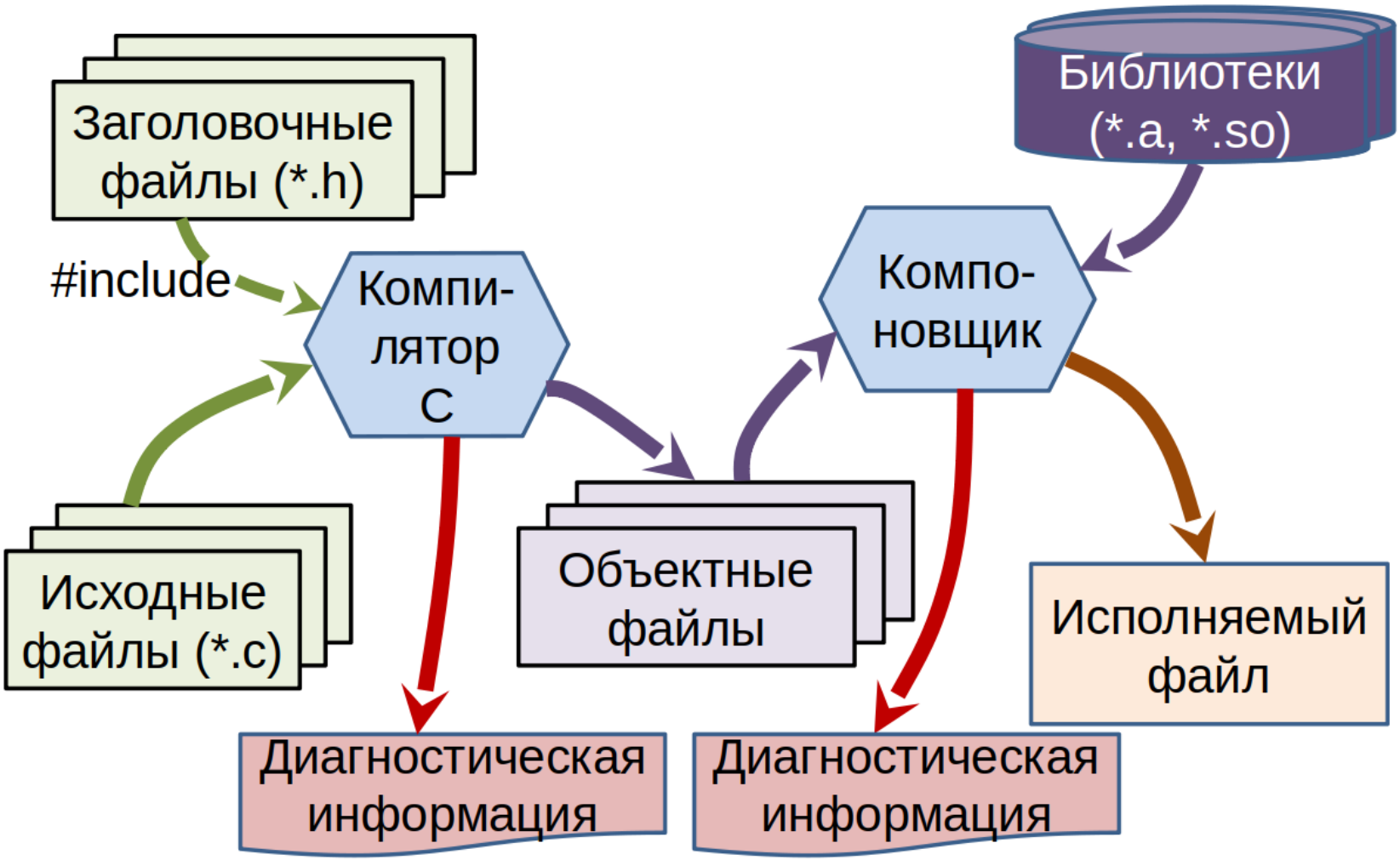
Деннис Ритчи
(1941 — 2011 гг.)

Кен Томпсон
(1943 г., 80 лет)

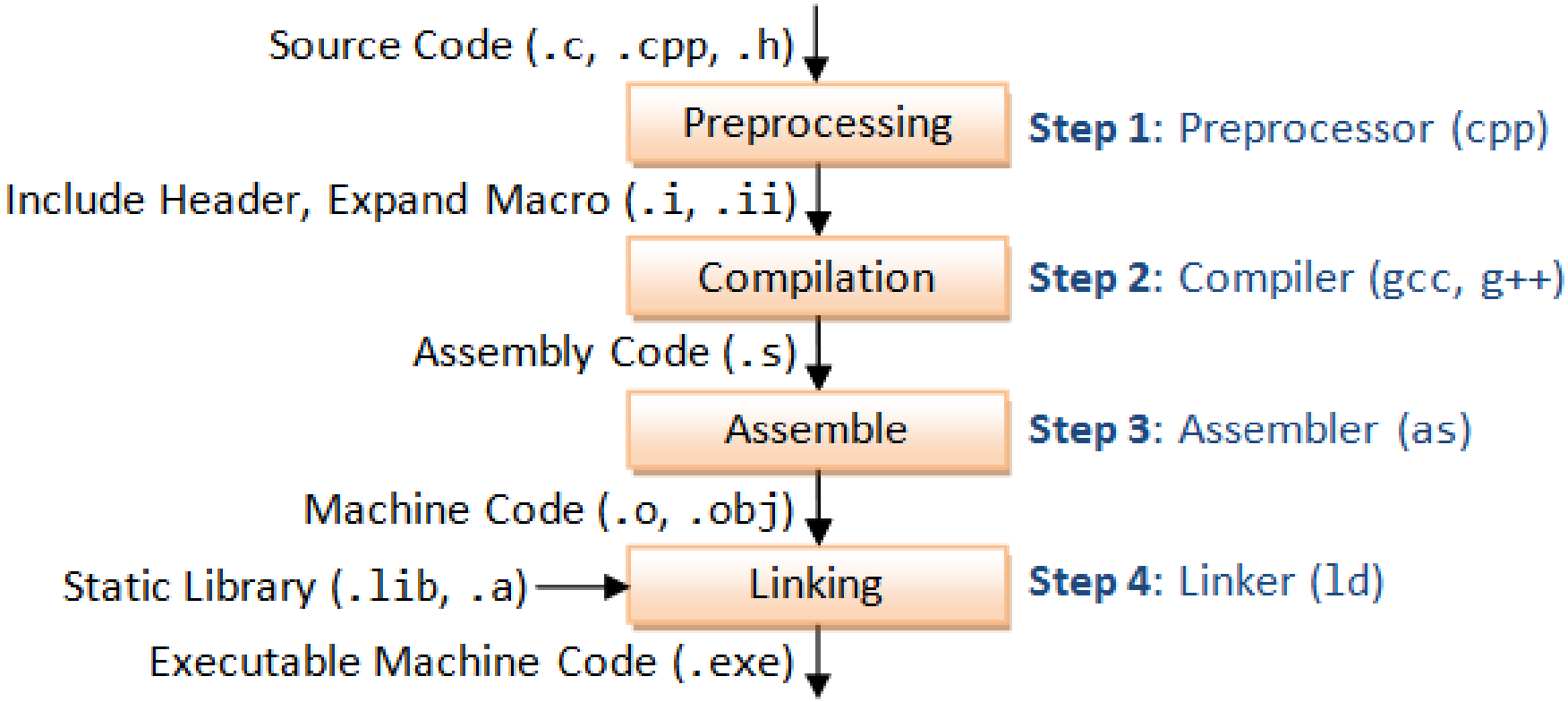


Брайан Керниган
(1942 г., 81 лет)

Порядок сборки программы на языке C I



Порядок сборки программы на языке C II



Препроцессинг

Определения

Препроцессор (preprocessor, предобработчик) — программа, подготавливающая код программы на языке C к компиляции

Основные функции препроцессора:

- Вставка содержимого заголовочных файлов (`#include`)
- Обработка макросов (`#define`)
- Условная компиляция (`#if`, `#else`, `#ifdef`, `#elif`, `#endif`)
- Удаление комментариев

Примеры

```
$ cc -E prog.c -o prog.i
```

Компиляция и ассемблирование

Определения

Компилятор (compiler) — программа, переводящая исходные коды на языке программирования высокого уровня в машинный код (либо в код на языке ассемблера)

Примеры

```
$ cc -S prog.c -o prog.s
```

Определения

Ассемблер (assembler) — программа, переводящая исходные коды на языке ассемблера в машинный код

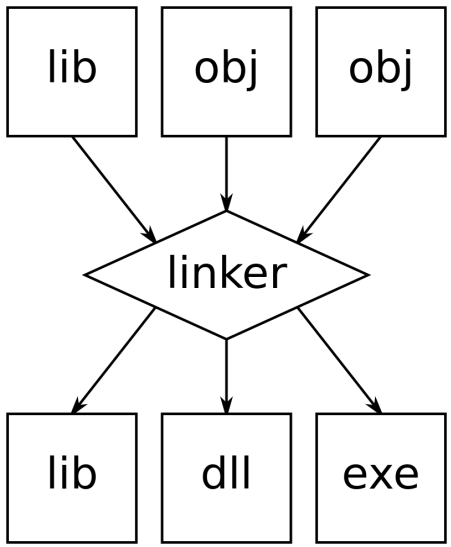
Примеры

```
$ cc -c prog.c -o prog.o
```

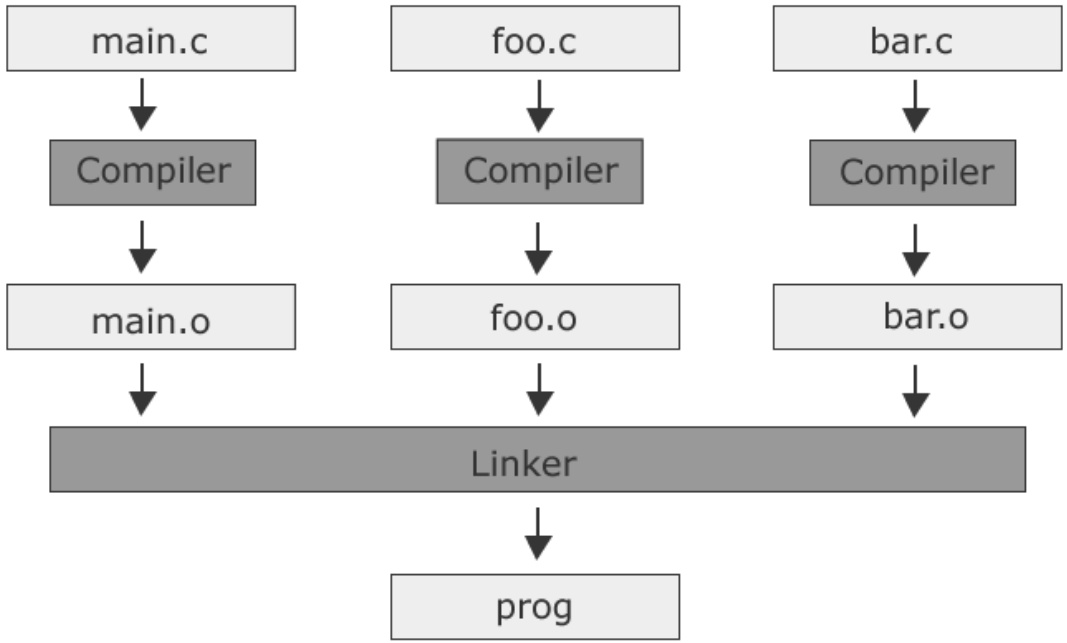
Компоновка

Определения

Компоновщик (редактор связей, linker) — программа, которая производит компоновку («линковку»): из одного или нескольких объектных модулей (в т.ч., возможно, с использованием библиотек) собирает единый исполняемый файл или библиотеку



Сборка программы, состоящей из нескольких файлов



Примеры

```
$ cc -c main.c -o main.o
$ cc -c foo.c -o foo.o
$ cc -c bar.c
$ cc main.o foo.o bar.o -o program
```

Примеры

```
$ cc main.c \
  foo.c \
  bar.c \
  -o program
```

Структура файла с исходным кодом (*.c)

- Директивы препроцессора
- Прототипы функций
- Глобальные данные
- Определение функций

Примеры

```c
#include <stdio.h>
int add(int a, int b);
int main() {
    int a = 10, b = 5;
    int r = add(a, b);
    printf("%d\n", r);
    return 0;
}
int add(int a, int b) {
    return a + b;
}
```

Определение функции

```
тип имя_функции(тип параметр_1, ...)
{
        предложение_описания_типа_1
        ...
        предложение_ языка_1
        ...
        return [выражение];
}
```

Предложения языка

- Определение данных
- Описание алгоритма:
  - Простое предложение:
    - Пустое
    - Вычисления
    - Управления
  - Составное предложение

Составное предложение

```
{
        предложение_1
        предложение_2
        ...
}
```

Простое предложение

- Пустое — `;`
- Вычисления — `выражение;`
- Управления:
  - Ветвление:
    - Условное
    - Переключатель
  - Цикл:
    - Итерационный
    - Параметрический
  - Передача управления:
    - `break;`
    - `continue;`
    - `return [выражение];`

Определение данных

- Категории данных:
  - Переменные
  - Константы
- Свойства данных:
  - Тип
  - Значение
- Типы данных:
  - Числа
  - Символы
  - Адреса

Предложение определения данных

```
тип имя_объекта[=инициализация], ...;
```

Примеры

```
int a;

int b;

int c = 5, d;
```

Базовые типы данных

- char — целое число, предназначенное для хранения кода символа
- int — целое число, имеющее типовой размер для целых чисел в системе
- float — вещественное число одинарной точности с плавающей точкой
- double — вещественное число двойной точности с плавающей точкой
- указатели

Целочисленные данные I

Тип данных — int
Модификаторы:

- Размера (int):
  - long
  - long long (начиная с C99)
  - short
- Знака (char, int):
  - signed
  - unsigned

Целочисленные данные II

Примеры

```
int a;

long int b;

unsigned char c;
```

Примеры

```
#include <stdio.h>

int main() {
    printf("Size of short - %ld\n", sizeof(short));
    printf("Size of int - %ld\n", sizeof(int));
    printf("Size of long - %ld\n", sizeof(long));
    printf("Size of long long - %ld\n", sizeof(long long));
    return 0;
}
```

Целочисленные константы

Примеры

```
2525l    // long

25u      // unsinged

25ul     // unsigned long

031      // octal

0x25FE   // hex
```

Приоритет и ассоциирование операторов I

Приоритет	Оператор	Описание	Ассоциативность
1	++ -- () [] -> .	Постфиксный инкремент и декремент Вызов функции Обращение по индексу Обращение к элементу структуры по указателю Обращение к элементу структуры	→
2	++ -- + - ! ~ (type) * & sizeof	Префиксный инкремент и декремент Унарный плюс, унарный минус Логическое НЕ, побитовое НЕ Преобразование типа Разыменование указателя Вычисление адреса Размер	←
3	* / %	Умножение, деление, остаток	→
4	+ -	Сложение, вычитание	
5	<< >>	Побитовый сдвиг влево, вправо	
6	< <= > >=	Меньше, меньше или равно Больше, больше или равно	
7	== !=	Равно, не равно	
8	&	Побитовое И	
9	^	Побитовое исключающее ИЛИ	
10	|	Побитовое ИЛИ	
11	&&	Логическое И	
12	||	Логическое ИЛИ	

Приоритет и ассоциирование операторов II

13	?:	Тернарный оператор	←
14	= += -= *= /= %= <<= >>= &= ^= |=	Присваивание Сложение/вычитание и присваивание Умножение/деление/остаток и присваивание Побитовый сдвиг влево/вправо и присваивание Побитовое И/исключающее ИЛИ/ИЛИ и присваивание	
15	,	Запятая (последовательность)	→

Приоритет и ассоциирование операторов III

Инкремент и декремент I

- Инкремент — увеличение значения операнда на единицу
- Декремент — уменьшение значения операнда на единицу

Оператор	Пример	Операция
Префиксный инкремент	`++i`	«Увеличить, использовать»
Префиксный декремент	`--i`	«Уменьшить, использовать»
Постфиксный инкремент	`i++`	«Использовать, увеличить»
Постфиксный декремент	`i--`	«Использовать, уменьшить»

Инкремент и декремент II

Примеры

```
int a = 3, x;
x = ++a; // a = 4, x = 4
```

```
int a = 3, x;
x = a++; // a = 4, x = 3
```

Арифметические операторы

- Унарные
  - +
  - –
- Бинарные
  - Аддитивные
    - +
    - –
  - Мультипликативные
    - *
    - /
    - %

Примеры

```
+5  //  5
-5  //  -5
2 + 3  //  5
2.0 + 3  //  5.0
3 - 2  //  1
3.0 - 2  //  1.0
2 * 3  //  6
2.0 * 3  //  6.0
5 / 2  //  2
5.0 / 2  //  2.5
5 % 2  //  1
5.0 % 2  //  error
```

Операторы сравнения

```
<  <=  >  >=
==  !=
```

Примеры

```
3 < 5 // 1

2 > 5 // 0

2 < x < 3 // 1, doesn't depend on x value

3 == 4 // 0

3 != 4 // 1
```

Логические операторы I

! && ||

Примеры

```
3 && 5 // 1

3 || 1 // 1

!2 // 0

!-2 // 0

!0 // 1

(x > 2) && (x < 3) // depends on x value
```

Логические операторы II

Правила вычисления:

1. Вычисления операндов происходят слева направо
2. Вычисления прекращаются, как только результат становится понятным

Побитовые операторы

~ & ^ |

Примеры

```
x = 0b10110010; // not in standard (GCC-only)
y = 0b00010111;

~x // 01001101
x & y // 00010010
x ^ y // 10100101
x | y // 10110111
```

Тернарный оператор

```
expr1 ? expr2 : expr3
```

Примеры

```
a = 15;
b = 10;

x = a > b ? a : b; // x = 15

x = a < b ? a : b; // x = 10
```

Является правоассоциативным, следующие записи эквивалентны:

```
expr1 ? expr2 : expr3 ? expr4 : expr 5
  expr1 ? expr2 : (expr3 ? expr4 : expr 5)

expr1 ? expr2 ? expr3 : expr4 : expr 5
  expr1 ? (expr2 ? expr3 : expr4) : expr 5
```

Оператор «запятая»

Позволяет объединять несколько выражений, значением составного выражения являтся значение крайнего правого выражения:

`expr1, expr2, expr3, ...`

Примеры

```
a = (x = 2, y = 3, x + y) // a = 5

for (i = 0, n = 0; i < 100; ++i, n += i) {

}
```

Порядок вычисления операндов и аргументов функций I

Важно

Порядок вычисления операндов не гарантирован!!!
Порядок вычисления аргументов функции не гарантирован!!!

Порядок вычисления операндов гарантируется только для следующих операторов:

- `&&`
- `||`
- `?:`
- `,`

Порядок вычисления операндов и аргументов функций II

Примеры

```
x = f() + g(); // x value depends on call order

int x = 2, z;
z = ++x * x++; // possible: x = 4, z = 9

int x = 2, z;
z = x++ * ++x; // possible: x = 4, z = 9

printf("%d %d\n", ++n, power(2, n)); // wrong

++n;
printf("%d %d\n", n, power(2, n)); // good
```

Функции I

Идея

Функция — подпрограмма в языке C, которая решает конкретную подзадачу.

Преимущества использования функций:

- Отказ от дублирования фрагментов кода в рамках программы
- Возможность повторного использований функции в других программах
- Улучшение модульности, облегчение проектирования программы
- Облегчение чтения и программы, упрощение локализации и исправления ошибок

Функции II

Определение функции включает в себя (см. слайд 15):

- Тип возвращаемого значения
- Имя функции
- Информация об аргументах
- Тело функции

Функции III

Что нужно для вызова функции?

Для вызова функции необходимо знать её имя, перечень формальных параметров с указанием типов и тип возвращаемого значения.
Данная информация представлена в сигнатуре функции.

Примеры

```
double atof(const char *); // #1

double atof(const char *s); // #2
```

Функции IV

Вызов функции имеет следующий общий вид:

```
имя_переменной = имя_функции(список аргументов);
```

Примеры

```
double a, b;

...

a = atof("123.45");

b = a + 3;
```

Функции V

Важно

Передача параметров в функцию осуществляется только по значению!!!
Создаётся копия объекта, который затем обрабатывается.
Возвращаемое значение передаётся во внешнюю программу схожим образом.

Переменные I

Определения

Переменая — псевдоним адреса области памяти, в которой располагаются данные определенного типа.
Значенение переменной — данные, которые находятся по адресу, связанному с именем переменной.
Область видимости — часть программы, в пределах которой можно использовать имя.

Переменные, в зависимости от области видимости, могут быть:

- Локальные
- Глобальные

Переменные II

Примеры

```
int a = 10; // global


int f1() {
    int a = 20; // local
    return a;
}


int f2() {
    return a;
}


int f3(int a) { // parameter name ~ local
    return a;
}


f1();   // 20

f2();   // 10

f3(a); // 10
```