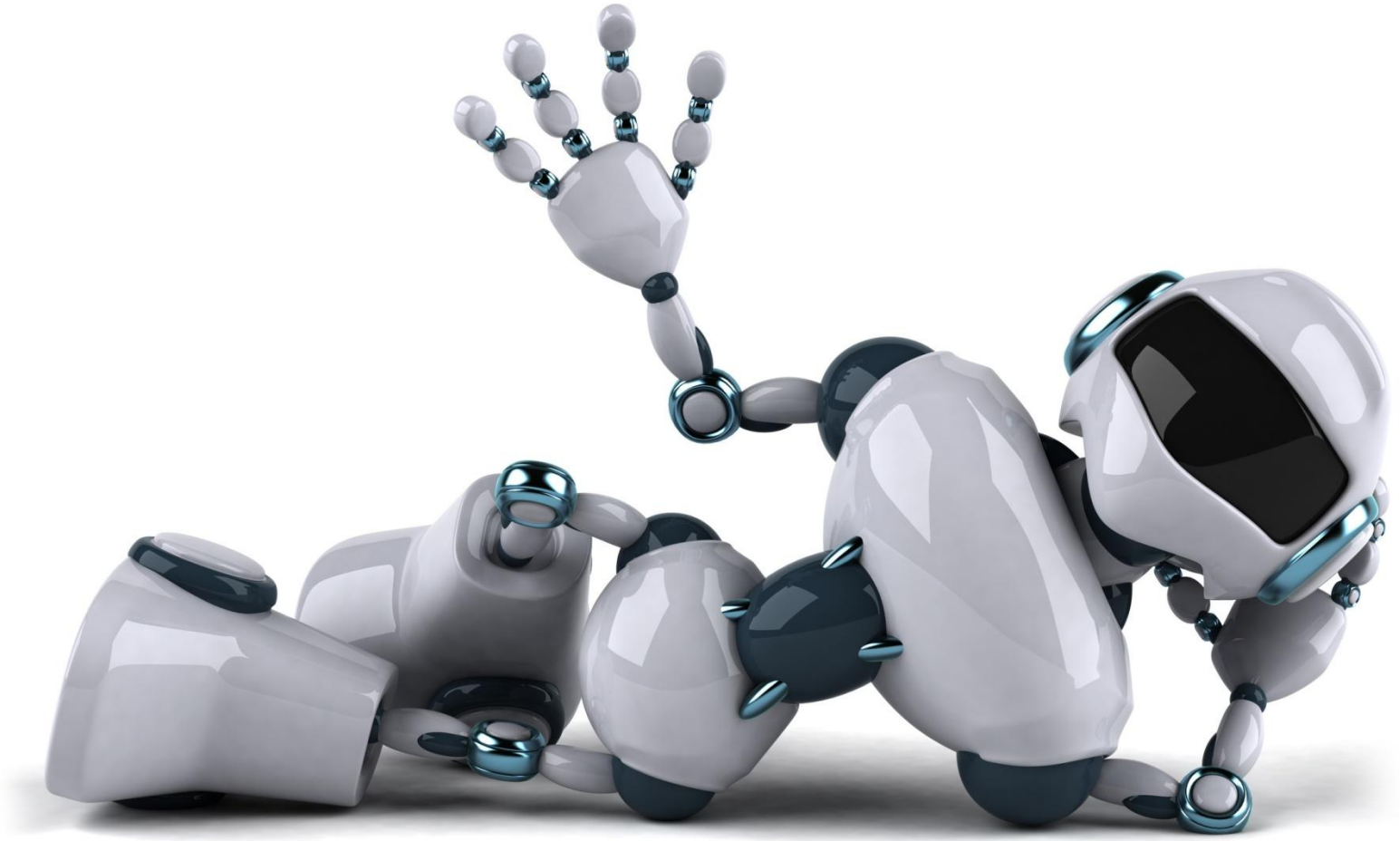


# Основные понятия языка Си

Тема 11

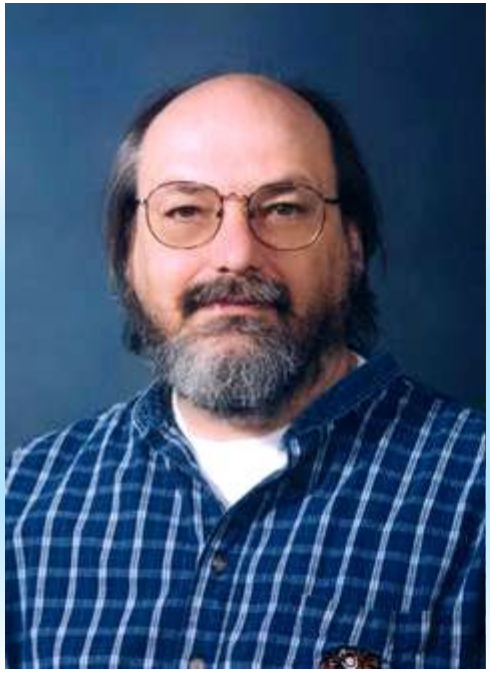
# Основные понятия

- Programming language C;
- Procedural programming
- Preprocessor;
- Function
- Heap, dynamic memory
- Globals
- Local variable
- lexeme , token
- identifier, ID
- keyword

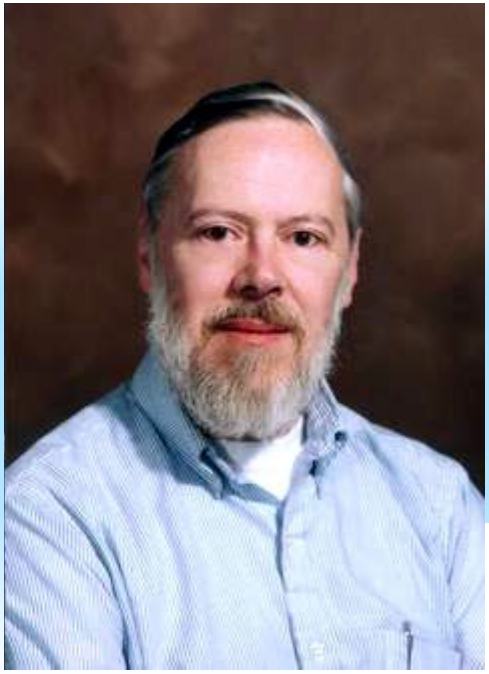
# Стандарты языка Си

# Язык Си (1972 г)

**Кен То́мпсон**

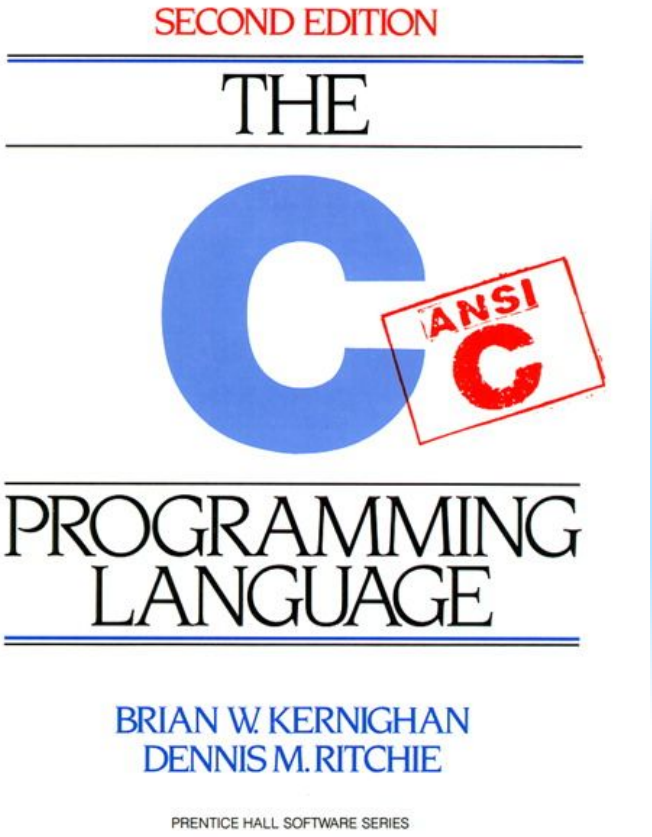


**Де́ннис Ри́тчи**



# Стандарты Си

**K&R C** «Язык программирования Си»
(1978г.):
Работа с памятью;
Препроцессор;
Типы и структуры данных;
Функции;
UNIX



# Стандарты Си

**C89 «Язык программирования Си» ANSI X3.159-1989.**

Многоплатформенность;

Библиотеки;

Работа с АО;

Разделение с C++;

Прототипы функций;

Поддержка Microsoft и

# Стандарты Си

**C99 ISO 9899:1999**

Массивы переменной длины;

Локальные переменные в операторе;

Библиотеки;

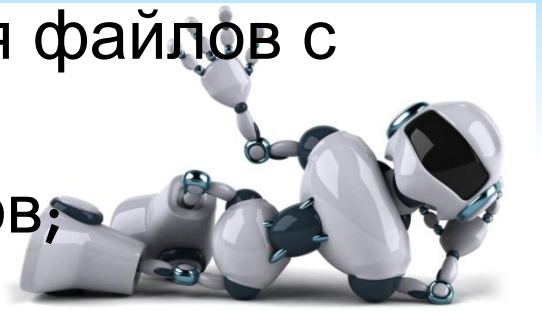
**C11 ISO/IEC 9899:2011**

Многопоточность;

Юникод;

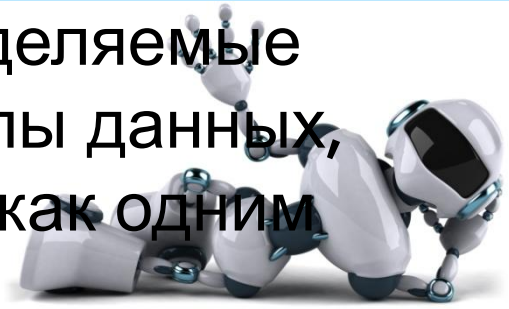
# Особенности Си

1. простую языковую базу, из которой вынесены в библиотеки многие существенные возможности;
2. ориентацию на процедурное программирование;
3. систему типов, предохраняющую от бессмысленных операций;
4. использование препроцессора для, определения макросов, включения файлов с исходным кодом;
5. минимальное число ключевых слов;



# Особенности Си

6. непосредственный доступ к памяти компьютера через использование указателей;
7. передачу параметров в функцию по значению, а не по ссылке;
8. указатели на функции и статические переменные;
9. области действия имён;
10. структуры и объединения — определяемые пользователем собирательные типы данных, которыми можно манипулировать как одним целым;



# В языке Си отсутствуют

1. автоматическое управление памятью;
2. вложенные функции;
3. поддержка объектно-ориентированного программирования;
4. полиморфизм функций и операторов;
5. поддержка многозадачности и сетевые функции;
6. функции высшего порядка;
7. сопрограммы и карринг.

# Элементы языка Си

# Алфавит языка

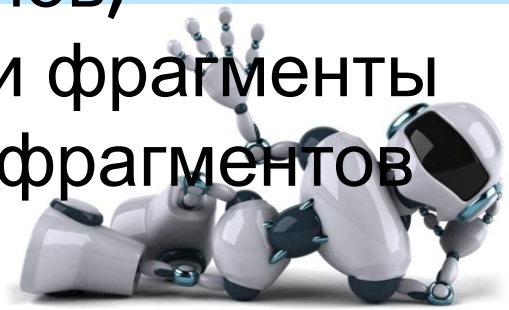
- Язык Си был создан уже после внедрения стандарта ASCII, поэтому использует почти все его графические символы (нет только $ @ ` ).
- в Си есть и круглые (), и квадратные [], и фигурные {}.
- в Си различаются заглавные и строчные буквы.
- Текст, заключённый в служебные символы /* и */ , считается комментарием.
- Компиляторы, совместимые со стандартом C99, также позволяют использовать комментарии, начинающиеся с символов // и заканчивающиеся переводом строки.

# Препроцессор

Перед компиляцией исходный текст программы на Си обрабатывается **препроцессором**.

Он разыскивает в тексте программы свои директивы (инструкции, команды), которые начинаются с символа # и выполняет их.

Директивы препроцессора позволяют вставить в программу тексты из других файлов, исключить из процесса компиляции фрагменты кода или выполнить замену одних фрагментов другими.



# Лексемы языка

- имена (идентификаторы);
- ключевые слова;
- знаки операций;
- разделители;
- литералы (константы).

# Ключевые слова C89

| | | | |
|---|---|---|---|
| auto | double | int | struct |
| break | else | long | switch |
| case | enum | register | typedef |
| char | extern | return | union |
| const | float | short | unsigned |
| continue | for | signed | void |
| default | goto | sizeof | volatile |
| do | if | static | while |

# Константы

| Константа | Формат | Примеры |
|---|---|---|
| Логическая | Обозначается ключевым словом true или false | true, false |
| Целая | Десятичный: последовательность десятичных цифр, начинающаяся не с нуля, если это не число нуль | 8, 0, 199226 |
| | Восьмеричный: нуль, за которым следуют восьмеричные цифры (0, 1, 2, 3, 4, 5, 6, 7) | 01, 020, 07155 |
| | Шестнадцатеричный: 0x или 0X, за которым следуют шестнадцатеричные цифры (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F) | 0xA, 0x1B8, 0X00FF, 0X00ff |
| Вещественная | Десятичный: [цифры].[цифры] | 5.7, 0.001, 35 |
| | Экспоненциальный: [цифры][.][цифры]{E\|e}[+\|-][цифры] | 0.2E6, .11e-3, 5E10, 1.22E-10 |
| Символьная | Один или более символов, заключенных в апострофы | 'A', 'ю', '*', 'db', 'A', '\n', '\012', '\x07\x07' |
| Строковая | Последовательность символов, заключенная в кавычки | "Здесь был Vasia", "\tСумма =\xF5\n" |

# Базовые типы C89

- **int** – целочисленный тип, целое число;
- **float** – вещественное число одинарной точности с плавающей точкой;
- **double** – вещественное число двойной точности с плавающей точкой;
- **char** – символьный тип для определения одного символа.

# Базовые типы C89

- **void** – тип без значения. служит для объявления функции, не возвращающей значения, или для создания универсального указателя (pointer);

**Модификаторы** базовых типов данных:

- Signed;
- Unsigned;
- Long;
- Short.

# Типы данных языка Си

| Тип данных | Типичный размер в битах | Минимально допустимый диапазон значений |
|---|---|---|
| char | 8 (или 1 байт) | от –128 до 127 |
| unsigned char | 8 | от 0 до 255 |
| signed char | 8 | от –127 до 127 |
| int | 16 или 32 | от –32767 до 32767 |
| unsigned int | 16 или 32 | от 0 до 65535 |
| signed int | 16 или 32 | от –32767 до 32767 |
| short int | 16 | от –32767 до 32767 |
| unsigned short int | 16 | от 0 до 65535 |
| signed short int | 16 | от –32767 до 32767 |
| long int | 32 | от –2147483647 до 2147483647 |
| long long int | 64 | от $-(2^{63}-1)$ до $(2^{63}-1)$ для C99 |
| signed long int | 32 | от –2147483647 до 2147483647 |
| unsigned long int | 32 | от 0 до 4294967295 |
| unsigned long long int | 64 | от 0 до $(2^{64}-1)$ для C99 |
| float | 32 | от 1E–37 до 1E+37 (с точностью не менее 6 значащих десятичных цифр) |
| double | 64 | от 1E–37 до 1E+37 (с точностью не менее 10 значащих десятичных цифр) |
| long double | 80 | от 1E–37 до 1E+37 (с точностью не менее 10 значащих десятичных цифр) |

# Преобразование типов

**Неявное приведение типов**

Если в выражении смешаны различные типы литералов и переменных, то компилятор преобразует их в один наиболее расширенный тип.

**Явное приведение типов.**

Общая форма оператора явного приведения типа:

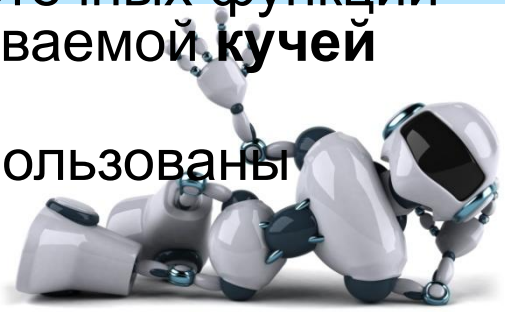
(тип) выражение.

# Работа с памятью

# Классы памяти

- **STATIC** - статическое выделение памяти: пространство для объектов создаётся ***в сегменте данных*** программы в момент компиляции; время жизни таких объектов совпадает со временем жизни этого кода.
- **AUTO** - автоматическое выделение памяти: объекты можно хранить ***в стеке***; эта память затем автоматически освобождается и может быть использована снова, после того, как программа выходит из блока, использующего его.
- **EXTERN** - динамическое выделение памяти: блоки памяти нужного размера могут запрашиваться во время выполнения программы с помощью библиотечных функций malloc, realloc, calloc из области памяти, называемой **кучей (heap)**.

Эти блоки освобождаются и могут быть использованы снова после вызова для них функции free.



# Область видимости

Каждый идентификатор имеет область действия (**potential scope**) и область видимости (**scope**), которые, как правило, совпадают (кроме случая описания такого же имени во вложенном блоке).

Область видимости начинается в точке описания.

```
const int i = 2;
```

Имя, описанное внутри блока, локально по отношению к этому блоку. Имя, описанное вне любого блока, имеет глобальную область видимости.

Область действия и класс памяти зависят не только от собственно описания, но и от места его размещения в тексте программы.

# Область видимости

| | **Глобальная** | **Локальная** | **Статическая** |
|---|---|---|---|
| Размещение | сегмент данных | сегмент стека | сегмент данных |
| Время жизни | вся программа | блок | вся программа |
| Область видимости | файл | блок | блок |
| Обнуление | да | нет | да |

# Область видимости

```
int a;              // глобальная переменная
int main(){
    int b;          // локальная переменная
    static int c = 1;  // локальная статическая
переменная
}
```

# Пространство имен

В каждой области действия различают пространства имен, в пределах которых идентификатор должен быть уникальным. В разных категориях имена могут совпадать, например:

```
struct Node{
    int Node;
    int i;
}Node;
```

# Пространство имен

В Си определено четыре раздельных класса идентификаторов, в пределах которых имя должно быть уникальным:

1. имена переменных, функций, типов typedef и констант перечислений;
2. имена типов перечислений, структур, классов и объединений;
3. элементы каждой структуры, класса и объединения;
4. метки.

# Операторы и операции

# Операторы

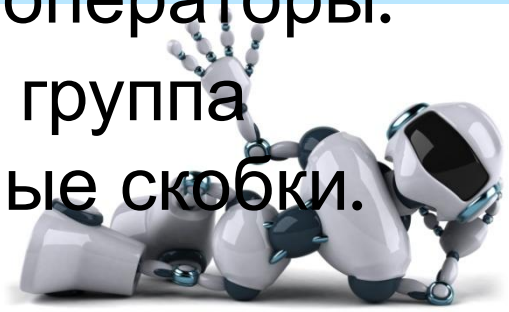
**Оператор** задает законченное описание некоторого действия.

Объединенная единым алгоритмом совокупность описаний и операторов образует **программу**.

Различают **простые** и **составные** операторы.

Составной оператор или блок - это группа операторов, заключенная в фигурные скобки.

Блоки могут быть вложенными.



# Операторы

## Неисполняемые

Неисполняемые операторы служат для описания данных, поэтому их часто называют операторами описания или просто описаниями.

Например,

```
int a ;
```

- это оператор описания целочисленной переменной a.

## Исполняемые

Исполняемые операторы задают действия над данными.

Например, присваивание, цикл, ввод и т.д.

# Описания идентификаторов

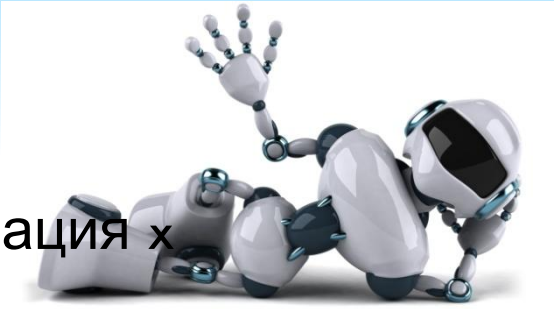
[класс памяти] [const] тип имя [инициализатор];

инициализатор: = значение

```
short int a = 1;
const char C = 'C';
char s, sf = 'f';
char t (54);
float c = 0.22, x(3), sum;
```

# Пример описаний

```
int a;              // 1 глобальная переменная a
int main()          // 2
{   int b;          // 3 локальная переменная b
    static int c; // 4 локальная статическая переменная c
    a = 1;          // 5 присваивание глобальной переменной
    int a;          // 6 локальная переменная a
    a = 2;          // 7 присваивание локальной переменной
    ::a = 3;        // 8 присваивание глобальной переменной
    extern int x; // 9 переменная x объявлена; определение
дальше
    ...
    return 0;       // 10
}                   // 11
int x = 4;          // 12 определение и инициализация x
```



# Пример 1

```
#include <stdio.h>
int main(){
    int i;
    printf("Введите целое число\n");
    scanf("%d", &i);
    i = i*i;
    printf(“Квадрат числа равен%d", i);
}
```

# Операции

Знак операции - это один или более символов, определяющих действие над операндами. Внутри знака операции пробелы не допускаются. Символы, составляющие знак операции, могут быть как специальными, например, &&, | и <, так и буквенными, такими как reinterpret_cast или new.

Операции делятся на унарные, бинарные и тернарную по количеству участвующих в них операндов.

# Приоритет операций

| Лексемы | Операция | Приоритет |
|---|---|---|
| имена, литералы | простые лексемы | 16 |
| a[k] | индексы | 16 |
| f(…) | вызов функции | 16 |
| ++ -- | положительное и отрицательное приращение | 15 |
| sizeof | размер | 15 |
| ~ | побитовое НЕ | 15 |
| ! | логическое НЕ | 15 |
| - + | изменение знака, плюс | 15 |
| & * | Адрес (разыменование) | 15 |
| (имя типа) | приведение типа | 14 |
| * / % | мультипликативные операции | 13 |
| + - | аддитивные операции | 12 |
| << >> | сдвиг влево и вправо | 11 |
| < > <= >= | отношения | 10 |
| == != | равенство/неравенство | 9 |
| & | побитовое И | 8 |
| ^ | побитовое исключающее ИЛИ | 7 |
| \| | побитовое ИЛИ | 6 |
| && | логическое И | 5 |
| \|\| | логическое ИЛИ | 4 |
| ? : | условие | 3 |
| = += -= *= /= %= <<= >>= &= ^= \|= | присваивание | 2 |

# Операции инкремента

```
#include <stdio.h>
int main(){
int x = 3, y = 3;
printf("Значение префиксного выражения: %d\n",
++x);
printf("Значение постфиксного выражения: %d\n",
y++);
}
```

Результат работы программы:
Значение префиксного выражения: 4

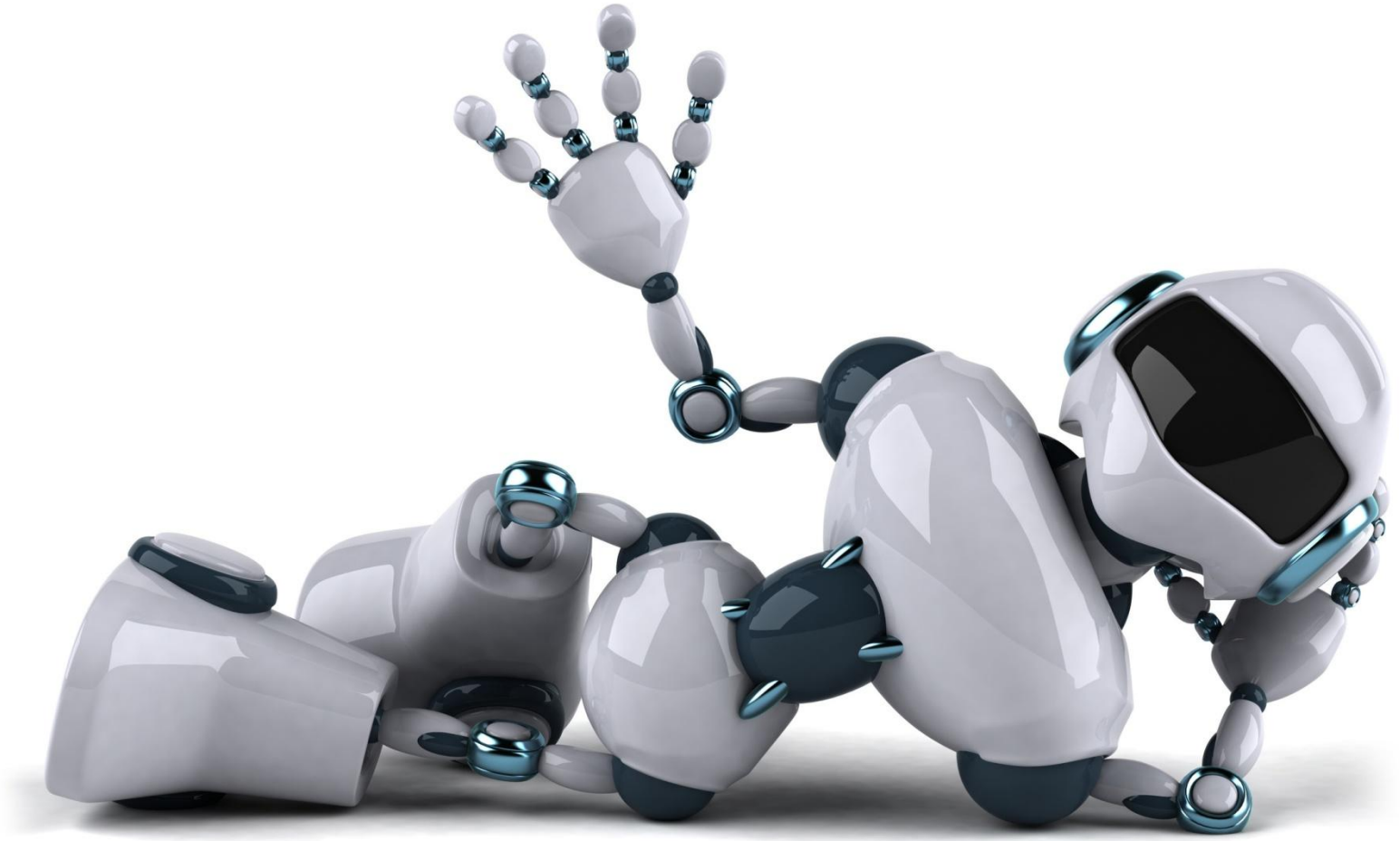
# Операции деления и остатка

```
#include <stdio.h>
int main(){
    int x = 11, y = 4;
    float z = 4;
    printf(" %d  %f\n", x/y, x/z);
    printf("Остаток: %d\n", x%y);
}
```

Результат работы программы:
2  2.750000
Остаток: 3

# Основные понятия

- Subprogram, subroutine;
- Procedure
- Function
- Heap, dynamic memory
- Globals
- Local variable
- Procedure invocation
- Parameters
- Argument
- Recursive function
- Unit
- Interface
- Implementation



# Основные понятия языка Си

Тема 11