# Условные операторы
# Логические операторы

Основы языка C, лекция 4

# Задача

- Через реку можно переплыть на лодке, которая вмещает **k** пассажиров.
- На берегу стоит **n** человек и хочет переправиться на другой берег.
- Сколько нужно сделать рейсов, чтобы переправить на другой берег всех людей? Формула?
-

| # теста | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| На берегу n | 10 | 11 | 22 | 6 | 15 |
| В лодке k | 5 | 3 | 7 | 6 | 10 |
| Рейсов res | 2 | 4 | 4 | 1 | 2 |

# Пишем, как думаем

- сколько рейсов полной лодки
- **ЕСЛИ** на берегу остались люди
  +1 рейс
- 
```
int n, k;
scanf("%d%d", &n, &k);
int res = n / k;     // рейсов полной лодки
int ostalos = n %k;  // осталось на берегу
if ( ostalos > 0 )
    res ++;
printf("%d рейсов лодки\n", res);
```

# Истина и ложь

- **if** ( *условие* )
  оператор;
- Если условие истинно,
  то выполняется оператор

  Если ложно, то оператор не выполняется
- **0** — ложь (False)
  **все остальное** — истина (True)
- условие пишем в круглых скобках
- табуляция — уровень вложенности для оператора

# Операторы сравнения

| Математика | Язык С | Русский |
|---|---|---|
| > | > | больше |
| ≥ | >= | больше или равно |
| < | < | меньше |
| ≤ | <= | меньше или равно |
| = | == | равно |
| ≠ | != | не равно |

Не путайте x = 5 и x == 5
Чаще присваивают, поэтому 1 равенство

# Блочный оператор

- ```
  if ( n % k > 0 )
      res ++;
      printf("Дополнительный рейс\n");
  ```
- Отступы нужны, но компилятор на них не смотрит (это не python)
- Блочный оператор (считается одним оператором)

```
if ( n % k > 0 ) {
    res ++;
    printf("Дополнительный рейс\n");
}
```
```

# if — альтернативы нет

- Вычисление модуля числа
- if (x < 0)

  x = -x;

# if .. else — альтернатива

- Проверим число четное или нечетное:

```
if ( x % 2 == 0 ) {
    printf("четное (even)\n");
} else {
    printf("НЕ четное (odd)\n");
}
```

```
if ( условие ) {
    операторДА;
} else {
    операторНЕТ;
}
```

| x | x / 2 | x % 2 |
|---|---|---|
| 8 | 4 | 0 |
| -8 | -4 | 0 |
| 7 | 3 | 1 |
| -7 | -3 | **-1** |

# Не надо так:

- Проверим число четное или нечетное:

```
if ( x % 2 == 0 ) {
    printf("четное (even)\n");
}

if ( x % 2 == 1 ) {
    printf("НЕчетное (odd)\n");
}
```

- Кто найдет
  больше ошибок в этом коде?

| x | x / 2 | x % 2 |
|---|---|---|
| 8 | 4 | 0 |
| -8 | -4 | 0 |
| 7 | 3 | 1 |
| -7 | -3 | **-1** |

# Множественный выбор

```
if ( x == 0 ) {
    printf ("zero\n");
} else {
    if ( x < 0 ) {
        printf ("negative\n");
    } else {
        printf ("positive\n");
    }
}
```

```
if ( x == 0 ) {
    printf ("zero\n");
} else if ( x < 0 ) {
    printf ("negative\n");
} else {
    printf ("positive\n");
}
```

- Для компилятора ничего не изменилось
- Для человека – по-другому поставили отступы

# Разные признаки

- четное — нечетное и знак — разные признаки:

```
if ( x % 2 == 0 )
    printf ("even\n");
else
    printf ("odd\n");

if ( x == 0 )
    printf ("zero\n");
else if ( x < 0 )
    printf ("negative\n");
else
    printf ("positive\n");
```

# После return жизни нет

- Год високосный, если он делится на 4, но не делится на 100. Если он делится на 400, то все же вискосный.

```
int is_leap_year(int year) {
    if ( year % 400 == 0 )
        return 1;
    else if ( year % 100 == 0)
        return 0;
    else if ( year % 4 == 0 )
        return 1;
    else
        return 0;
}
```

```
int leap_year (int x) {
    if ( x % 400 == 0 )
        return 1;
    if ( x % 100 == 0 )
        return 0;
    if ( x % 4 == 0 )
        return 1;
    return 0;
}
```

можно поменять порядок строк?

# Логические операторы

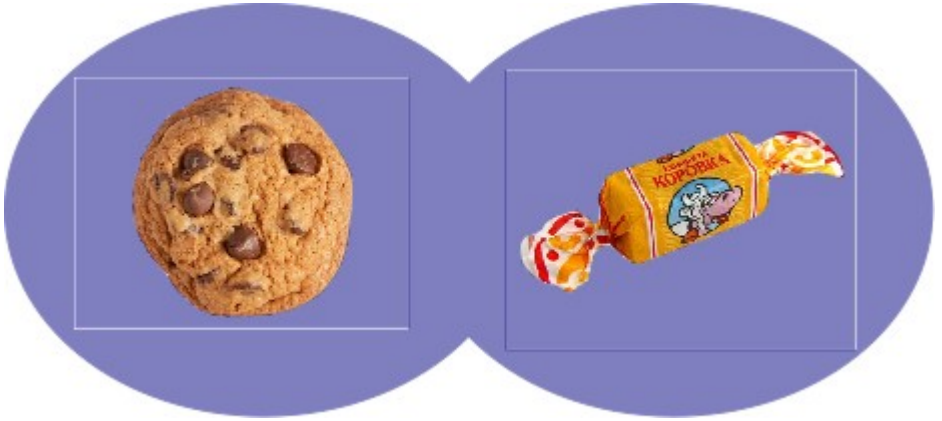
- && логическое И
- || логическое ИЛИ
- ! отрицание

cup && teabag => cup of tea

cookie || chocolate => dessert

| && | false | true |
|---|---|---|
| false | false | false |
| true | false | **true** |

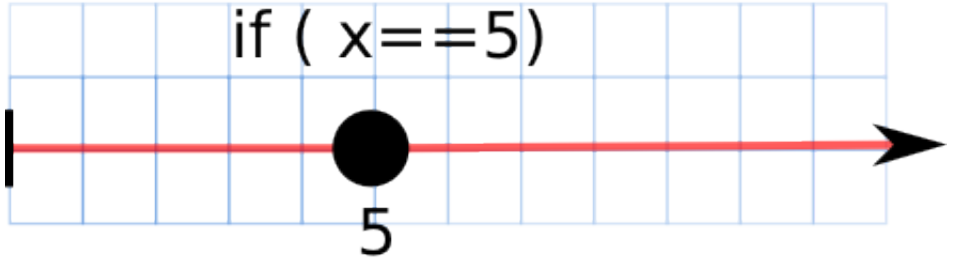
| \|\| | false | true |
|---|---|---|
| false | false | **true** |
| true | **true** | **true** |



# Математика - точка

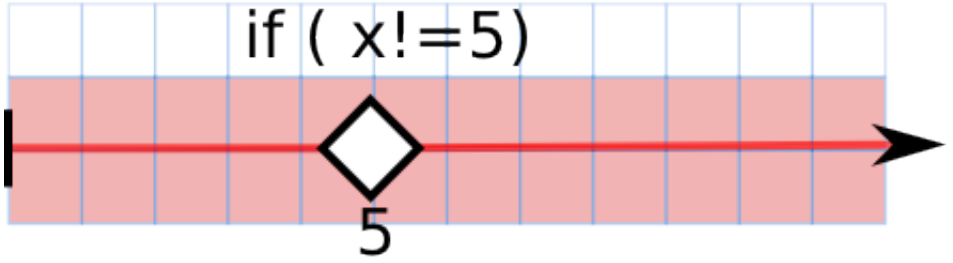
Равно (EQUAL)

x==5




Не равно (NOT EQUAL)

x!=5

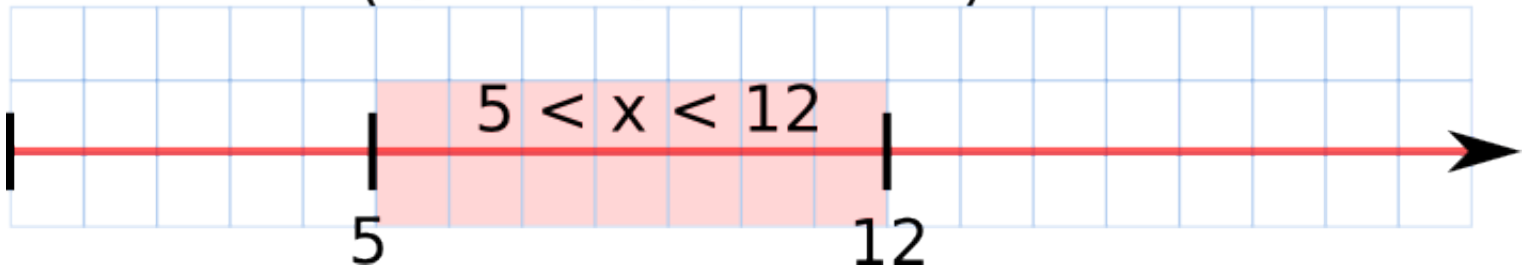


# Математика - отрезок

Между:

5 < x **И** x < 12

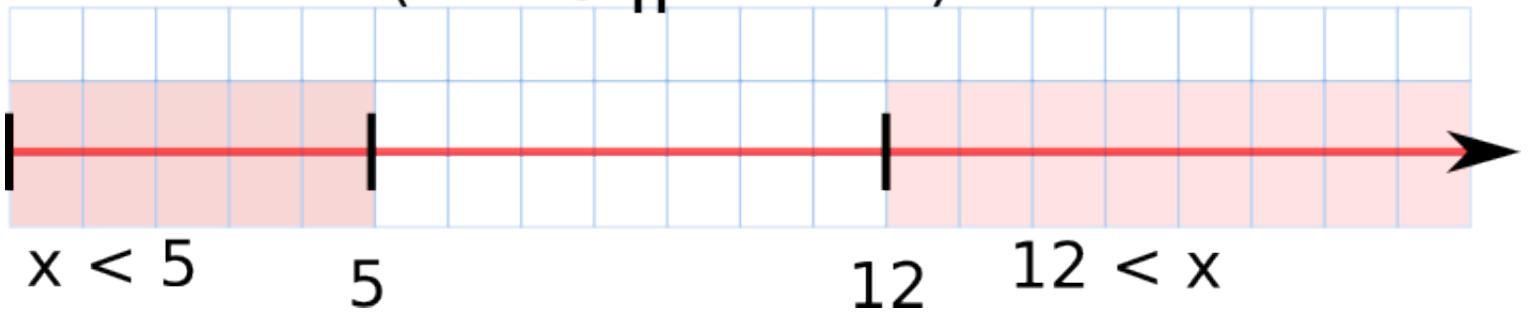
if ( 5 < x **&&** x < 12)




Вне (OUT)

x < 5 **ИЛИ** 12 < x

if ( x < 5 || 12 < x)



# Порядок вычислений

- Жестко определен
- Если можем не считать, дальше не считаем
- Не вызывайте функций, кроме функций-датчиков
  плохо:
  draw_star(10) && draw_table(3, 4)
  хорошо:
  **!** is_empty (list) && ( x = pop(list) )

# Значение выражений

- x = 5

  значению справа от =

  **Не путайте = и ==**

- x < 2

  x == 7

  будет 0 или 1

- && || !

  0 или 1

- ```
  int x = 5;
  if (x == 5)
      printf("AAA");
  if (x = 3)
      printf("BBB");
  if (x = 0)
      printf("ZZZ");
  printf("%d\n", x);
  ```

Больше так никогда не пишем:

```
doz = !!(dig / 10) * ('A' + dig % 10)
        + ! (dig / 10) * (dig + '0');
```
```

# switch..case

- **switch ( *выражение* ) {**
  **case константа1 : операторы_1;**
  **break;**
  **case константа2 : операторы_2;**
  **break;**
  **default: операторы;**
  **}** // сюда передает управление **break**
- ***выражение*** – целочисленное
- **константаi** – целочисленные константы
- если ***выражение*** равно **константаi,** управление переходит к операторы_i
- если ничего не подошло, то default (может не быть)

# склоняем коров n

```
switch ( n % 10 ) {
    case 1 :
        printf("%d коровa", n);
        break;
    case 2 :
    case 4 :     // константы в любом порядке
    case 3 :
        printf("%d коровы", n);
        break;
    default:     // в конце, может не быть
        printf("%d коров", n);
}
// сюда передает управление break
```

- Не работает для 11, 12, 13 коров

# Оператор ? :

- ```
  if ( x > 3 )
          z = x;
  else
          z = 0;
  ```
- `y = ( x > 3 ) ? x : 0 ;`

- `printf ( x ? "YES" : "NO");`
- `printf ("%s\n", x ? "YES" : "NO");`

- Плохо `x = c1 ? c2 ? 1 : 2 : c3 ? 3 : 4;`
  ```

# Рекурсивный вызов функции

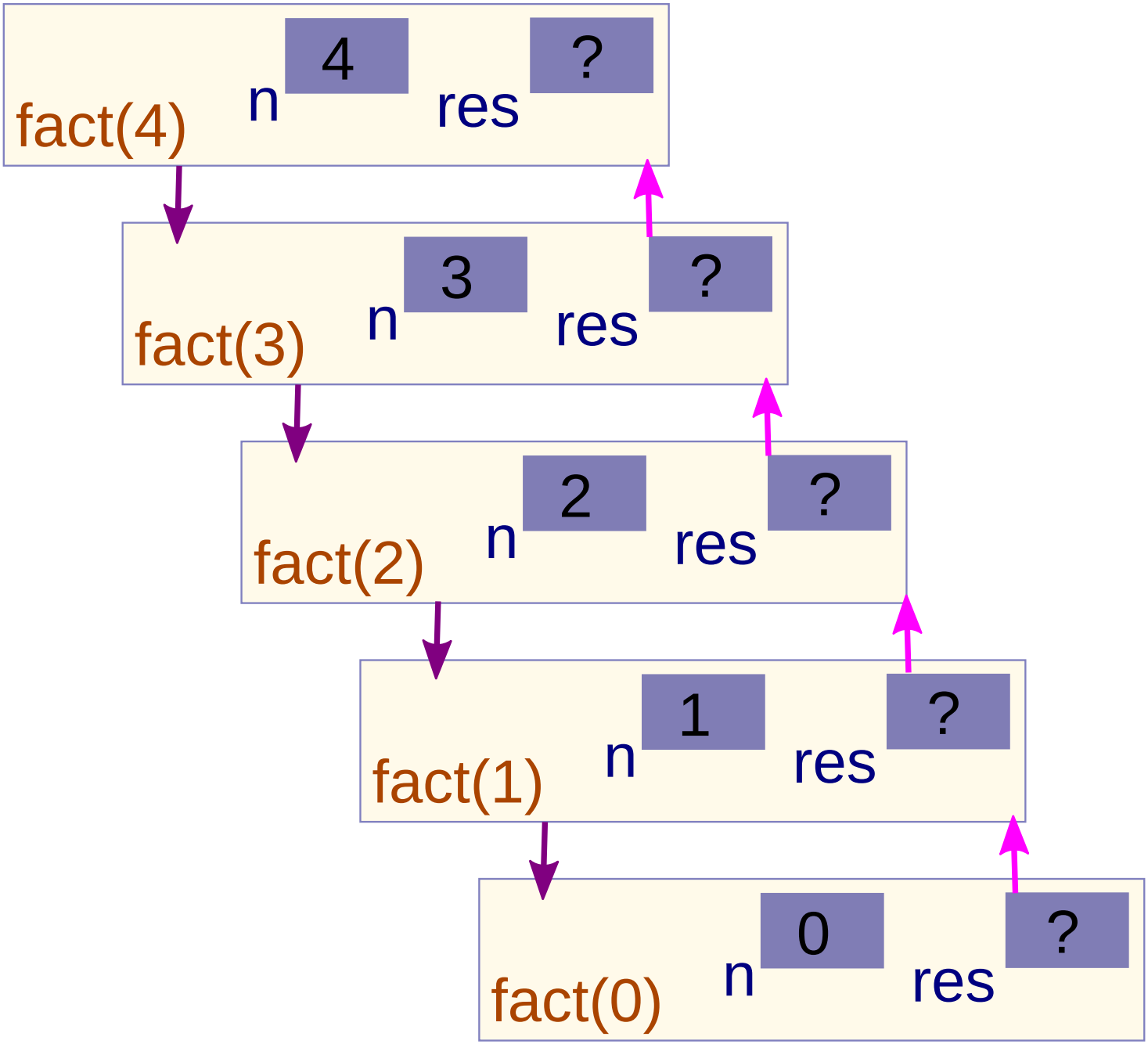
- Определение факториала n! (математика)
  F(n) = n * F(n-1)
  F(0) = 1
- Попробуем реализовать функцию fact:

```
int fact (int n) {
    int res = n * fact (n-1);
    return res;
}
int main ( ) {
    printf("5! = %d\n", fact(5));
    return 0;
}
```

Segmentation fault (core dumped)

Не работает. Почему?

# Экземпляры n и res



# Отладочная печать

```
#include <stdio.h>
int fact (int n) {
    printf("call fact(%d)\n", n);
    int res = n * fact (n-1);
    printf("%d! = %d\n", n, res);
    return res;
}
int main ( ) {
    printf("4! = %d\n", fact(4));
    return 0;
}
```

- Главное — вовремя остановиться

```
call fact(4)
call fact(3)
call fact(2)
call fact(1)
call fact(0)
call fact(-1)
call fact(-2)
call fact(-3)
call fact(-4)
call fact(-5)
...
```

# Определение: F(0) = 1

```
int fact (int n) {
    printf("call fact(%d)\n", n);
    if (n == 0) {
        printf("0! = 1\n");
        return 1;
    }
    int res = n * fact (n-1);
    printf("%d! = %d\n", n, res);
    return res;
}
int main ( ) {
    printf("4! = %d\n", fact(4));
    return 0;
```

- call fact(4)
- call fact(3)
- call fact(2)
- call fact(1)
- call fact(0)
- 0! = 1
- 1! = 1
- 2! = 2
- 3! = 6
- 4! = 24
- main: 4! = 24

# Передадим глубину вызова

- fact (4, 1)

```
int fact (int n, int depth) {
    printf("%-*c call fact(%d)\n", depth, '>', n);
    if (n == 0) {
        printf("%-*c 0! = 1\n", depth, '>');
        return 1;
    }
    int res = n * fact (n-1, depth+1);
    printf("%-*c %d! = %d\n", depth, '>', n, res);
    return res;
}
```

# Вызовы

```
int fact (int n) {
    if (n == 0)
        return 1;
    return n * fact (n-1);
}
int main() {
    printf("main: 4! = %d\n", fact(4));
    return 0;
}
```

- fact (4, 1)

- > call fact(4)
- > call fact(3)
- > call fact(2)
- > call fact(1)
- > call fact(0)
- > 0! = 1
- > 1! = 1
- > 2! = 2
- > 3! = 6
- > 4! = 24
- main: 4! = 24

# Числа Фибоначчи

- 0 1 2 3 4 5 6 7
  1 1 2 3 5 8 13 21 ...
- F(n) = F(n-1) + F(n-2), n > 2
  F(0) = F(1) = 1
- int **fib** (int n) {
  if (n == 0 || n == 1)
  return 1;
  return **fib**(n-1) + **fib**(n-2) ;
  }
  int main () {
  printf("fib(5) = %d\n", fib(5));
  }

# Эффективность

- Сколько раз будет вызвано
  fib(2) при подсчете
  20 числа Фибоначчи?