# Информатика

Кочанов Марк

29 сентября 2018 г.

МИФИ

# Функции

# Объявление функций

```
1 return_type function_title(type1 arg1, type2 arg) {
2     ...
3
4     return ...;
5 }
```

- Для объявления функции, не возвращающей значения, необходимо использовать void в качестве возвращаемого значения и опустить оператор возврата return
- Если в функции опущен оператор возврата, а тип возвращаемого значения отличен от void, то согласно стандарту результат не определен
- Значения в функцию передаются по значению. Модификация передаваемой переменной внутри функции не приведет к её изменению в вызывающей функции.
- Для завершения исполнения функции, не возвращающей значения, можно использовать return;.

# Прототипы функций

```
1  int foo(int n); // prototype
2
3  int main() {
4      int i = foo(5);
5      return 0;
6  }
7
8  int foo(int n) { // definition
9      return n + 1;
10 }
```

- Имена аргументов в прототипе опциональны, но лучше писать в целях документации кода
- Все стандартные заголовочные файлы (<stdio.h>, <math.h> и т.д.) содержат прототипы функций

# Прототипы функций

```
1  int foo(int n); // prototype
2
3  int main() {
4      int i = foo(5);
5      return 0;
6  }
7
8  int foo(int n) { // definition
9      return n + 1;
10 }
```

- Прототипы позволяют отделить определение интерфейса функций от их объявления и поместить в заголовочный файл, что позволяет делать раздельную компиляцию
- В языке C отсутствует перегрузка функций (объявление двух функций с одинаковым именем, но разными типами аргументов), в отличие от языка C++

# Указатели (pointers)

# Указатели

```
1  #include <stddef.h>
2  double *p1 = NULL;
3  int n, *p2;
4
5  n = 10;
6  p2 = &n;
7  *p1 = 11; // n = 11
8
9  int **p3 = &p2;
10 **p3 = 12; // n = 12
11
12 int **p4 = &&n // ERROR
```

Указатель — тип переменной, содержащий адрес ячейки памяти. Операция разыменования указателя (*p1, например) позволяет получить доступ (на чтение и запись) к переменной, на которую указывает указатель. Указатель в своем объявлении должен содержать тип.

# Указатели

```
1  #include <stddef.h>
2  double *p1 = NULL;
3  int n, *p2;
4
5  n = 10;
6  p2 = &n;
7  *p1 = 11; // n = 11
8
9  int **p3 = &p2;
10 **p3 = 12; // n = 12
11
12 int **p4 = &&n // ERROR
```

- Указатель после разыменования ДОЛЖЕН давать возвращать объявленную переменную, то есть запрещается делать композицию операции взятия адреса (строка 11 кода)
- Указатели используются для передачи в функции переменных (передача по указателю), чтобы функция могла изменить значение переданной переменной

# Указатели

```
1  #include <stddef.h>
2  double *p1 = NULL;
3  int n, *p2;
4
5  n = 10;
6  p2 = &n;
7  *p1 = 11; // n = 11
8
9  int **p3 = &p2;
10 **p3 = 12; // n = 12
11
12 int **p4 = &&n // ERROR
```

- Указатели по умолчанию не имеют значения
- Указатель может быть инициализирован нулём
- В заголовочном файле stddef.h определена константа NULL, которую следует использовать вместо нуля для инициализации указателя, либо его проверки на пустоту (указатель ни на что не указывает)

# Передача по значению и по указателю

```
1  void cube_by_value(int n) {
2      n = n * n * n;
3  }
4
5  void cube_by_pointer(int *n) {
6      *n = *n * *n * *n;
7  }
8
9  int n1 = 10;
10 cube_by_value(n1); // n1 = 10
11
12 int n2 = 10;
13 cube_by_pointer(&n2); // n2 = 1000
```

При передаче переменной по значению (pass by value) вызываемая функция не может изменить исходное значение аргумента, так как функция работает с копией исходной переменной.

# Передача по указателю

```
1  void swap(int& a, int& b) {
2      int temp = *a;
3      *a = *b;
4      *b = temp;
5  }
6  int n1 = 10, n2 = 20;
7  swap(&n1, &n2); // n1 = 20, n2 = 10
```

Передача переменной по указателю требуется в случае, когда нужно изменить значение сразу нескольких переменных. Если функция должна изменить один аргумент, то новое значение можно вернуть из функции и результат вызова функции присвоить переменной.

```
1  int cube(int n) {
2      return n * n * n;
3  }
4  int n = 10;
5  n = cube(n); // n = 1000
```

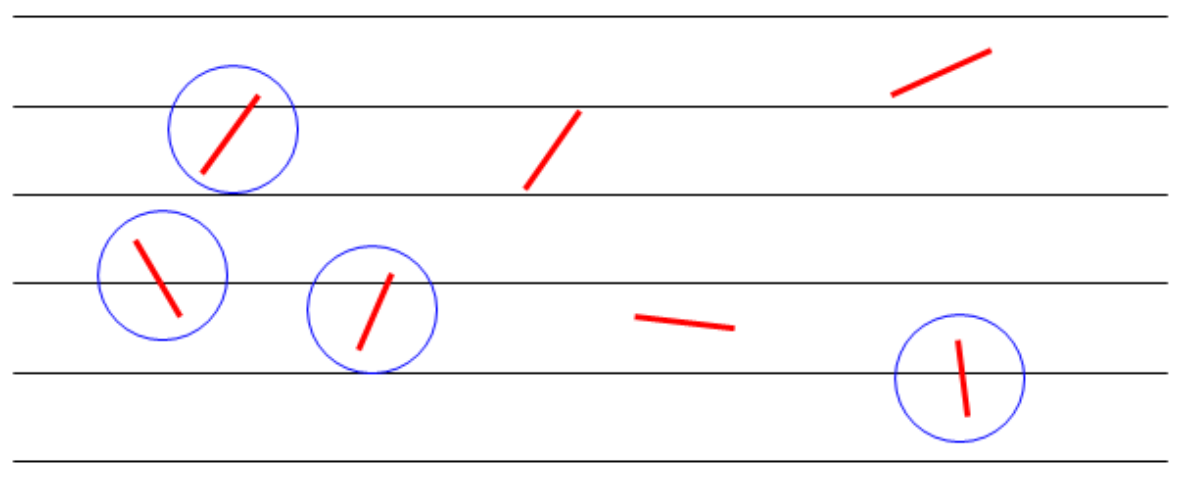
# Передача массивов в функцию

```
1  float avg(int t[], int size) {
2      // t[0], t[1], ....
3      float res = ...;
4
5      return res;
6  }
7  int max(int *t, int size) {      // int *t == int t[]
8      int res = ...;
9
10     return res;
11 }
12
13 int t_length = 5;
14 int temperatures[t_length] = {23, 24, 23, 25, 26};
15 printf("%f\n", avg(temperatures, t_length));
16 printf("%i\n", max(&temperatures[0], t_length));
```

Передача информации о длине массива лежит на программисте, так при использовании массива в качестве аргумента передается указатель на начало массива.

# Задачи

# Задача Бюффона о бросании иглы



Описание задачи
На плоскость нанесены параллельные линии на расстоянии $t$ друг от друга. На плоскость бросаются случайным образом иглы длиной $l$ ($l < t$). Тогда отношение количество игл, пересекающих прямые, к общему числу брошенных игл равно

$$p = \frac{2l}{t\pi}.$$

# Задача Бюффона о бросании иглы

Программа считывает с клавиатуры пять целых положительных чисел:

- расстояние между линиями $(t)$
- длину иглы $(l)$
- количество испытаний (бросков иглы)

На экран выводится четыре числа:

- количество иголок, пересекающих прямые
- количество иголок, оказавшихся между линиями
- отношение двух предыдущих чисел
- оценку числа $\pi$

При реализации программы код необходимо вынести в функции (генерация случайного числа в диапазоне, проверка на пересечение и т.д.).

# Задача Бюффона о бросании иглы

Генерация случайного числа в заданном диапазоне:

```c
#include <stdlib.h>      // srand, rand, RAND_MAX
#include <time.h>        // time


srand(time(NULL)); // initialization
float scale = (float) rand() / RAND_MAX; // [0, 1.0]
float rnd = min + scale * ( max - min ); // [min, max]
```

Часто-используемые математические функции:

```c
#include <math.h>

printf("%f\n", sin(M_PI / 2));
printf("%f\n", log(1.));          // natural logarithm
printf("%f\n", pow(3, 2.5));      // 3^(2.5)
printf("%f\n", sqrt(7));          // 3^(1/2)

printf("%f %f\n", round(7.3), trunc(7.7));       // 7 7
printf("%f %f\n", round(7.6), trunc(-7.7));      // 8 -7
```