№2

Указатели, массивы.

Массивы

↙ одномерные ↘ двумерные

int array[10]; int array[10][10]

Способы инициализации:

int array[10] = {0, 1, 2, 3};

int array[10] = {0}; - обнулить

int array[] = {0, 1, 2, 3}

компилятор сам считает

int array[10] - размер 10 · sizeof(int) =

= 10 · 4 = 40 байт

Зарезервировано

Трудноуловимая ошибка - выход за пределы массива. Компилятор это не контролирует - undefined behaviour.

Исходы: - программа корректно отработала (не задели рядные)

- программа некорректно отработала (задели рядные)

- ОС аварийно завершит программу

Указатель - это адрес соответствующего элемента в памяти.

int *p - ук-тель на ячейку, в которой хранится int.

Адреса хранятся в регистрах.

Кол-во байт для хранения указателя зависит от архитектуры компьютера (на x86-64 бита)

8 байт

Размер указателя один и тот же:

sizeof(int*) = sizeof(char*) = sizeof(double*).

int a = 42;

int *p = &a; - взять адрес a.

~~int *~~p int b = *p - взять значение по адресу p (разыменовывание)

printf("%p", p) - вывести адрес

Арифметика указателей

```
int array[5] = {1,2,3,4,5};
char str[] = "hello";
int *pi = array; - pi = &array[0] -
                взять адрес нулевого
                элемента массива
char *pc = str; - pc = &str[0];
pi += 1; - сдвиг адреса на sizeof(int
                4 байта
pc += 1; - сдвиг адреса на sizeof(cha
                1 байт
array[i] = *(array + i)
           адрес начала
```

Зачем использовать указатели? (преимущ

Если вы не всю указатель, то нужно всю вы перекопировать большой массив в доп. память.

Вместо этого можно просто передать адрес начала объекта (8 байт) и не весь огромный массив.

но! Указатели ухудшают читаемость кода

```
int strlen(char *ptr) {
    char *p = ptr;
    while (*ptr != '\0') {
        ++p;
    }
    return p - ptr;
}
```

- записывает в ук-тель адрес начала массива

p - единый адрес

ptr - адрес начала

```
void swap(double *pa, double *pb) {
    double tmp = *pa;
    *pa = *pb;
    *pb = tmp;
}
int main() {
    double c = 3; double d = 4;
    swap(&c; &d);
    return 0;
}
```

Const

- дает больше информации программисту, читающему или использующему наш код

const защищает то, что стоит перед ним

```
char s1[] = "hello"
char s2[] = "bye"
```

char const *p1 = s1; - защищается char

указателю присвоим s1. Теперь её адрес хранится в 2^x местах

p1[0] = 'a'; - compilation error нельзя менять содержимое. адрес менять можно

p1 = s2; - ok

char * const p2 = s1; - теперь нельзя менять адрес

p2[0] = 'a'; - ok

p2 = s2; - compilation error

char const * const p3 = s1; - защита и адреса, и содержимого

const подстраховывает программиста от ошибок, если он вдруг случайно хочет изменить то, что не нужно.

определение ф-ии.

Происходит ошибка - undefined behaviour - неопределенное поведение.

Решением проблемы будет помещать объявление ф-ии в заголовочный файл-эталон. Если где-то происходит расхождение с эталоном, то возникнет ошибка.

Чтобы вставить один файл в другой, используется препроцессор ~~мы используем~~ ~~мы используем~~ его команды начинаются с # (директивы).

#include "Util.h" ← "взять содержимое файла и вставить на место include."

```
#include "Util.h"
int main {
    :
    ind d=sum(c); //compilation error!
} return 0;
```

~~#include~~
//Util.h
ns sum (int a, int b); // declaration