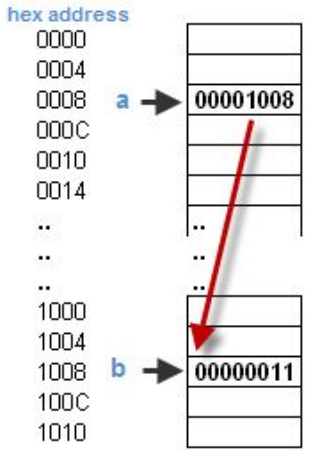
Язык C++

Указатели, массивы и строки

Указатель

- Указатель (pointer) – переменная, диапазон значений которой состоит из адресов ячеек памяти и специального значения – нулевого адреса
- Указатель «указывает» хранящимся внутреннего адресом на ячейку памяти, к которой с его помощью можно обратиться
- Значение нулевого адреса используется только для обозначения того, что указатель в данный момент не указывает ни на какую ячейку памяти



Операторы & и *

Унарный оператор **&** выдает адрес объекта

Унарный оператор * есть оператор *косвенного доступа*

Указатели. Операторы & и *

```
int x = 1;
int y = 2;
int z[10];
int* ip;        /* ip - указатель на int */

ip  = &x;       /* теперь ip указывает на x */
y   = *ip;      /* y теперь равен 1 */
*ip = 0;        /* x теперь равен 0 */
ip  = &z[0];    /* ip теперь указывает на z[0] */
```

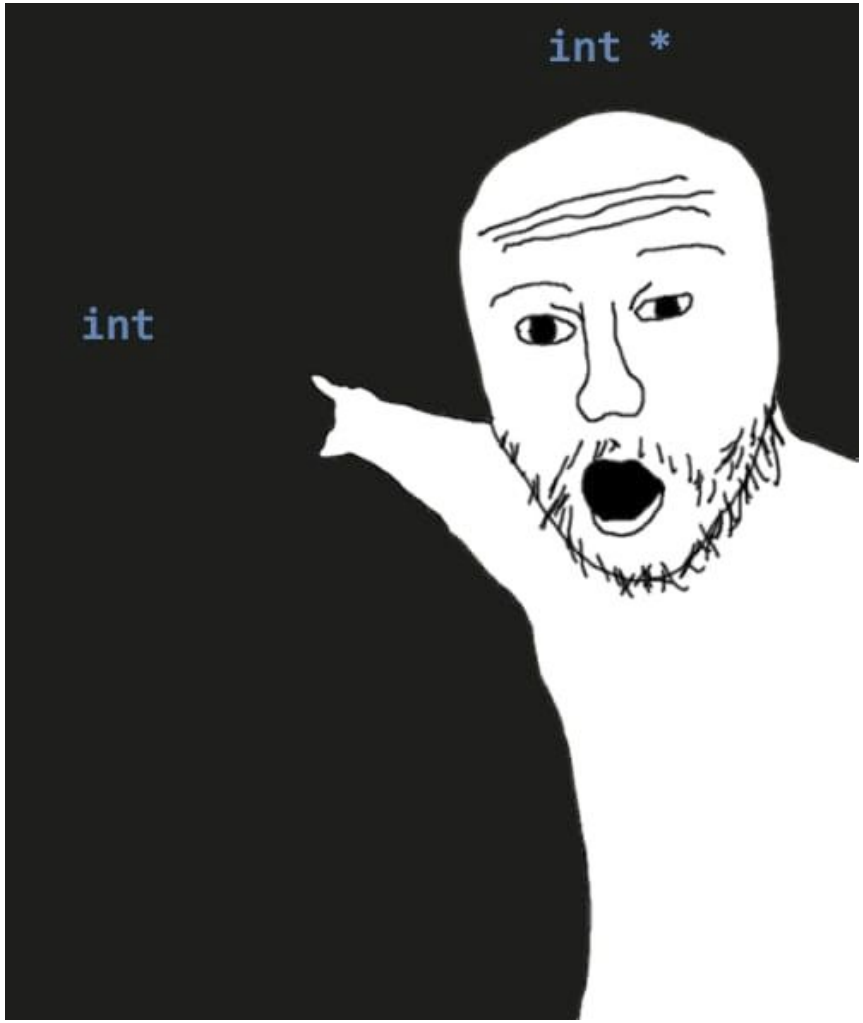
Указатели

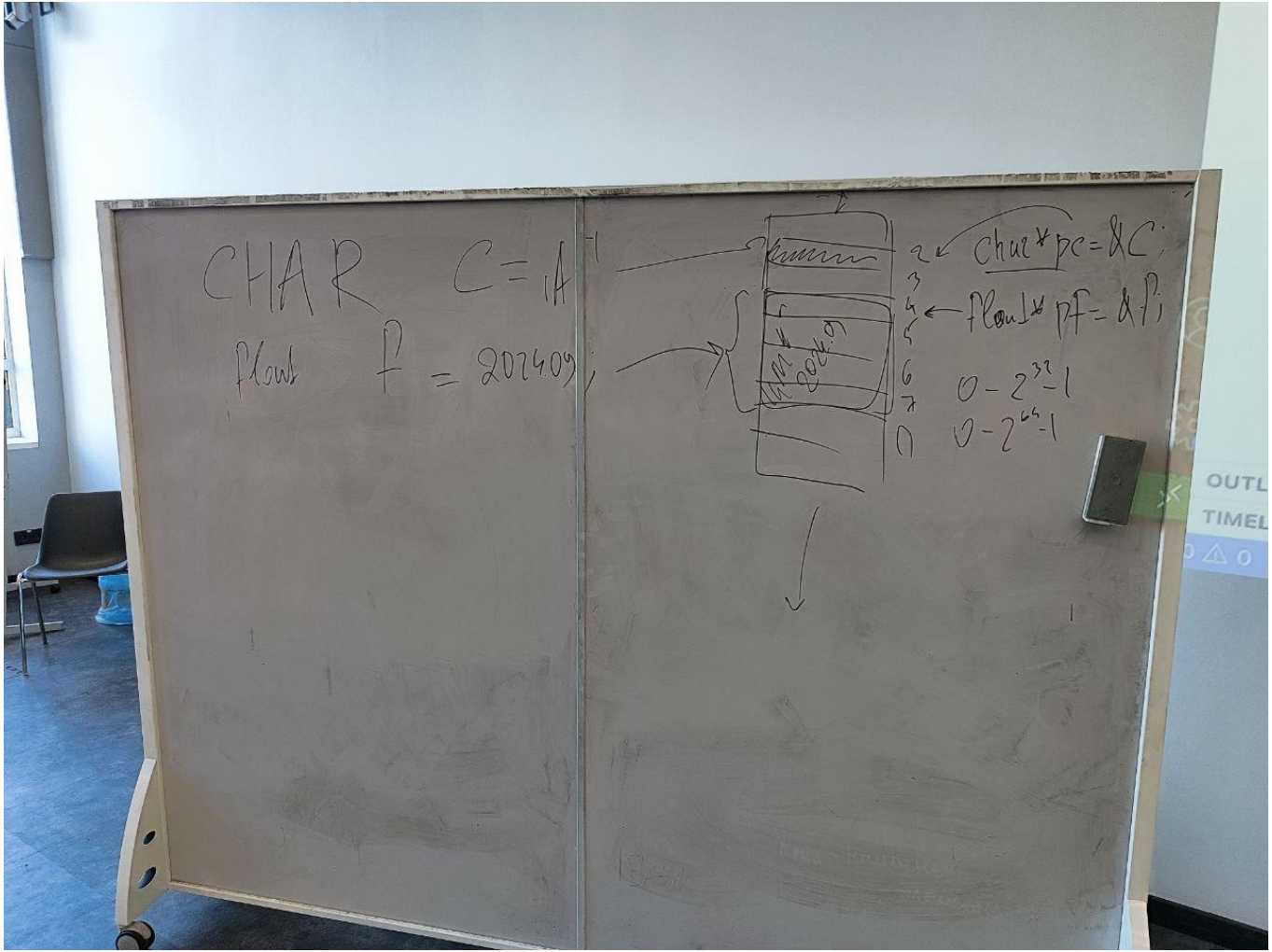
```
int main() {
    int i = 10;
    int j = 12;
    long l = 128L;
    float f = 129.1;

    std::cout << &i << std::endl;
    std::cout << &j << std::endl;
    std::cout << &l << std::endl;
    std::cout << &f << std::endl;

    return 0;
}
```

Указатели

```
int main() {
    bool b = true;
    long l = 128L;
    std::cout << sizeof(b) << std::endl;
    std::cout << sizeof(l) << std::endl;

    bool* pb = &b;
    long* pl = &l;
    std::cout << sizeof(pb) << std::endl;
    std::cout << sizeof(pl) << std::endl;

   return 0;
}
```

Размер указателя не зависит от типа на который он указывает


int *
int


CHAR C = 'A';
float f = 2024.09;
char *pc = &C;
float *pf = &f;
0 - 2^32 - 1
0 - 2^64 - 1

Использование указателей в качестве аргументов функций

```
void swap(int x, int y) {
  int temp;
  temp = x;
  x = y;
  y = temp;
}
```

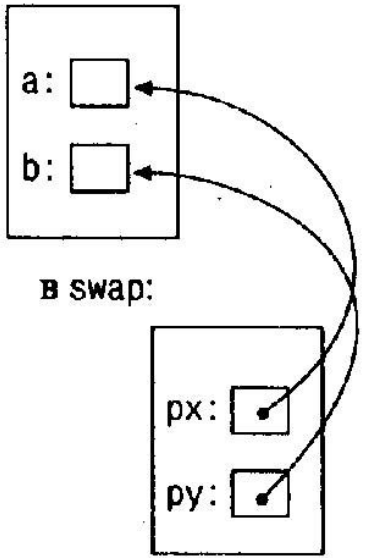
Использование указателей в качестве аргументов функций

```
int main() {
    int a = 1;
    int b = 2;

    printf("a = %d, b = %d\n", a, b);
    Swap(a, b);
    printf("a = %d, b = %d\n", a, b);
}
```

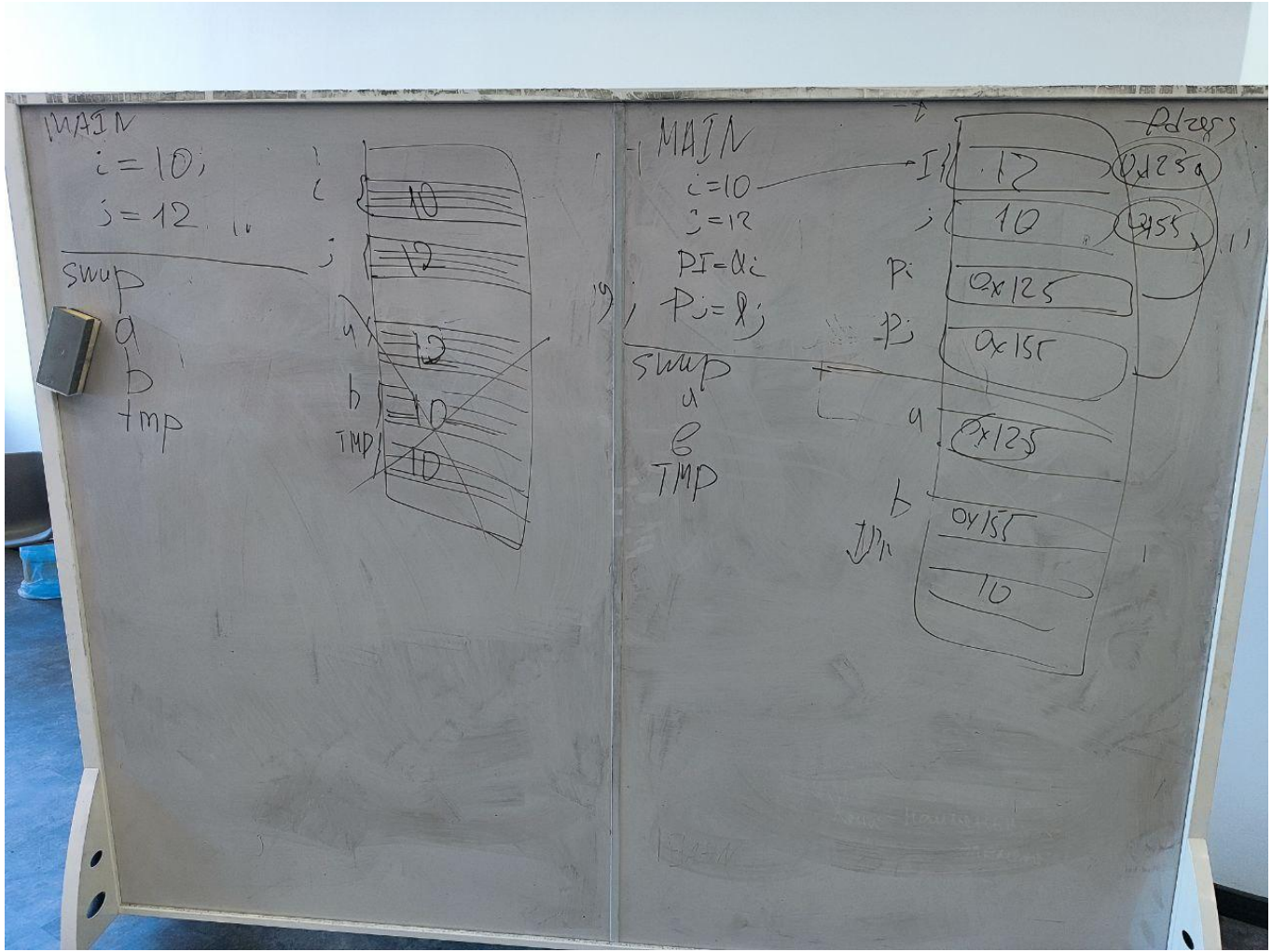
```
a = 1, b = 2
a = 1, b = 2
```

Использование указателей в качестве аргументов функций

```
void Swap(int* px, int* py) {
  int temp;
  temp = *px;
  *px = *py;
  *py = temp;
}
```



a = 1, b = 2
a = 2, b = 1


MAIN
i = 10;
j = 12
swap
a
b
tmp
i
10
j
12
a
12
b
10
TMP
10
MAIN
i=10
j=12
PI=&i
Pj=&j;
swap
a
b
TMP
I
12
j
10
Pi
0x125
Pj
0x155
a
0x125
b
0x155
10
Address
0x125
0x155

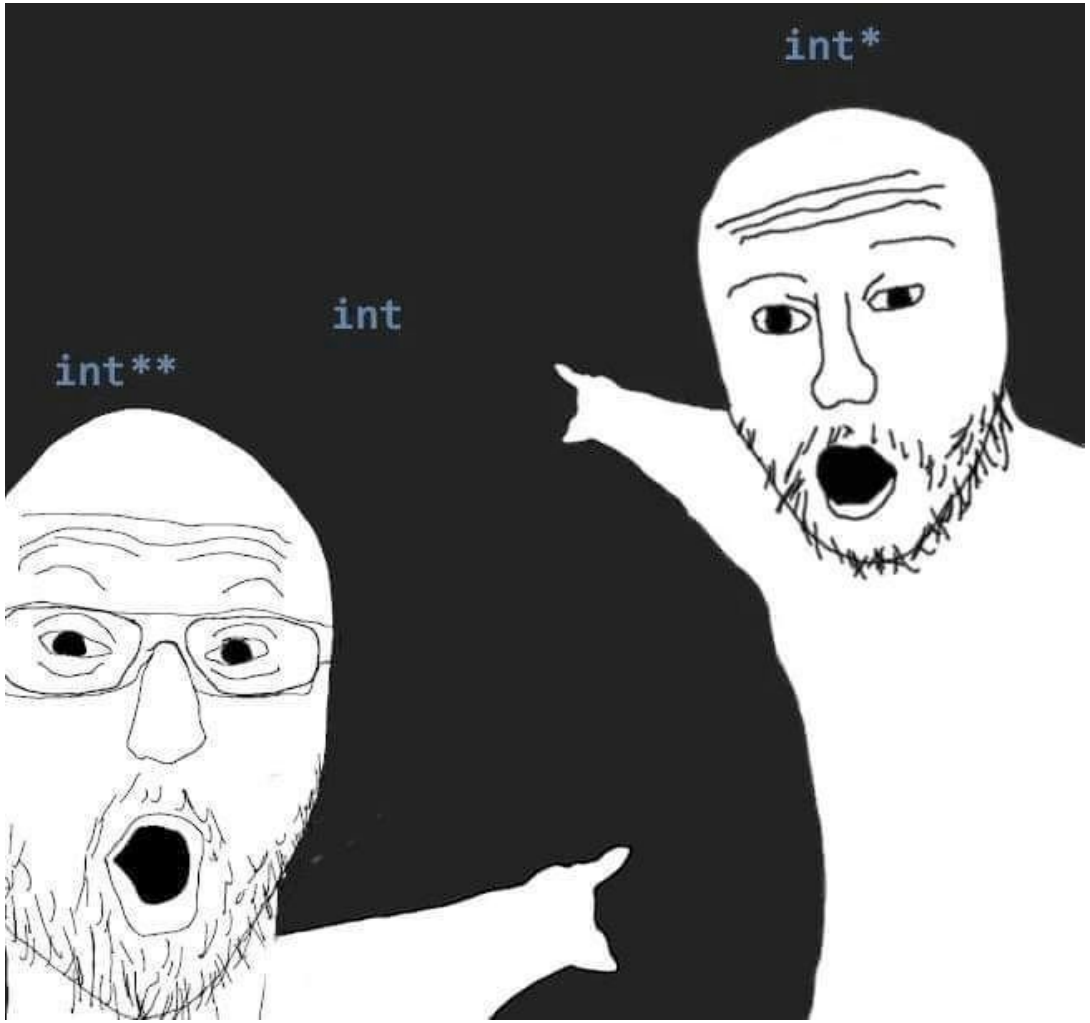
Указатели

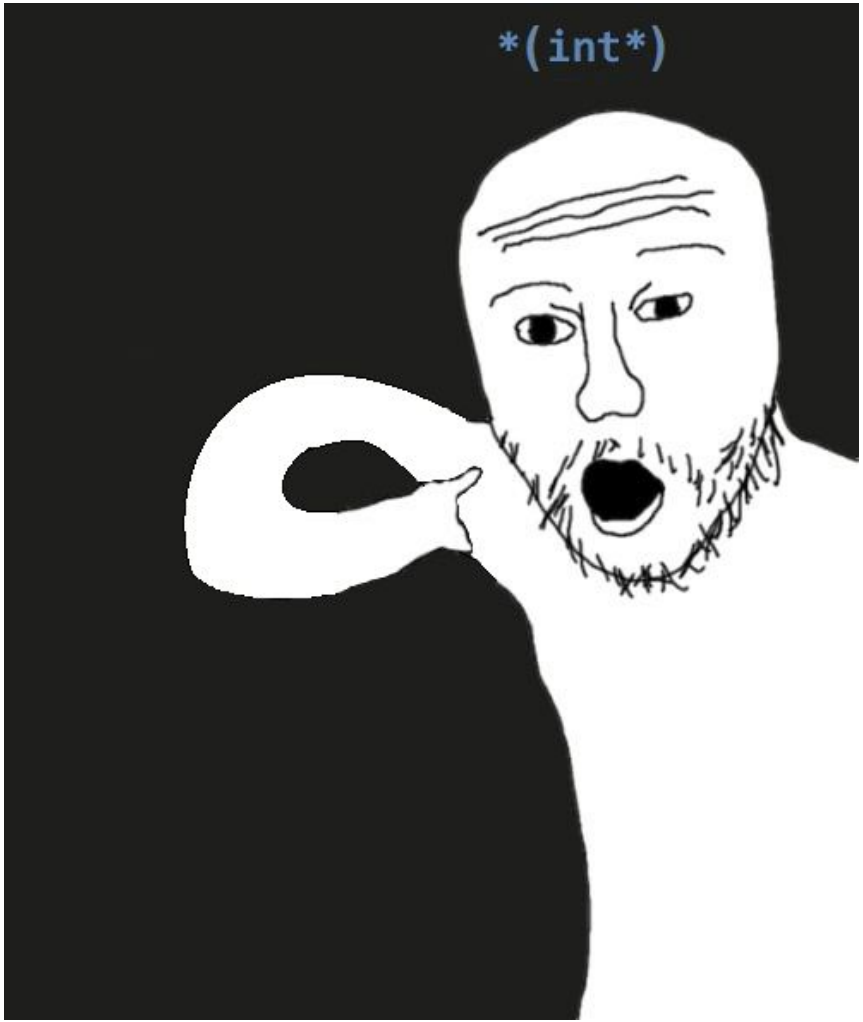
```
int i = 0;
std::cout << "Value: " <<  i << " Address: " << &i << std::endl;

int* pi = &i;
std::cout << "Value: " <<  pi << " Address: " << &pi << std::endl;

int** ppi = &pi;
std::cout << "Value: " <<  ppi << " Address: " << &ppi << std::endl;

int*** pppi = &ppi;
std::cout << "Value: " <<  pppi << " Address: " << &pppi << std::endl;
```

pi - указатель на i
(хранит адрес i)

ppi - указатель на pi
(хранит адрес pi)

pppi - указатель на ppi
(хранит адрес ppi)


int*
int
int**


(int)

NULL vs nullptr

```
void func(int*) {
    std::cout << "int func(int*)\n";
}
void func(int) {
    std::cout << "int func(int)\n";
}

int main() {
    func(nullptr);
    func(0);
    func(NULL);  // Compile-time error: call to 'func' is ambiguous

    return 0;
}
```

NULL макрос. Использовался в языке C для указателя ни на что

nullptr литерал типа nullptr_t. Имеет тот же смысл,но не приводит к неоднозначностям

Массив

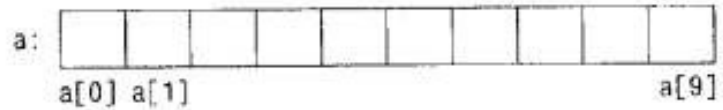
- Конечное множество однотипных элементов
- Размер множества не меняется
- Индексация с 0
- Многомерные массивы

Массив

```
int main() {
    int arr[10];
    int arr2[] = {1, 2, 3, 4, 5};
    int arr3[3] = {1, 2, 3};
    int arr4[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };

    printf("%d\n", arr2[0]);
    printf("%d\n", arr4[1][2]);
}
```

Связь массивов и указателей

- Определим массив
  int a[10];
- Определим указатель
  int *pa;
- Присвоим указатель адресу первого элемента массива
  pa = &a[0];
- Получим значение первого элемента массива через указатель
  int x = *pa;

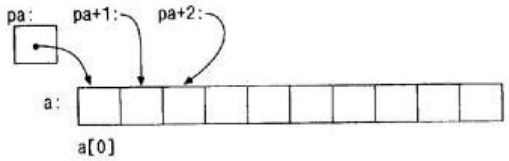


Связь массивов и указателей

- Получим указатель на следующий элемент массива
  *(pa + 1)

- Получим указатель на произвольный элемент массива
  *(pa + i) – это эквивалентно a[i]

- Компилятор преобразует ссылку на массив в указатель на начало массива, следовательно:
  - Имя массива является указательным выражением
  - Записи pa = &a[0] и pa = a эквивалентны
  - Записи a[i], *(a + i), *(pa + i) и pa[i] эквивалентны
  - Массив можно объявлять, как указатель, а потом пользоваться им, как массивом



Строки

- Массив символов
- Заключается в “”
- Escape character
- Null-terminated string

Строки и указатели

```
int main() {
   printf("здравствуй, мир\n");

   char* first_string;
   first_string = "Hello world";

   char second_string[] = "Hello world";

   char* third_string   = "Hello world";
}
```

Строки и указатели. Длина строки

```
size_t StringLenght(char* str) {
    size_t result = 0;

    while (*str != '\0') {
        str++;
        result++;
    }

    return result;
}
```

‘\0’ - символ конца строки

Строки и указатели. Сравнение

```
int StringCompare(char* first, char* second);

int main() {
   printf(
       "%d\n",
       StringCompare("hello world", "hello world")
   );
}
```

Строки и указатели. Сравнение

```
int StringCompare(char* first, char* second) {
    int i = 0;
    while(first[i] != '\0' && second[i] != '\0'){
       if(first[i] != second[i])
          return first[i] < second[i] ? -1 : 1;
       i++;
    }

    return first[i] == second[i] ? 0 : first[i] < second[i] ? -1 : 1;
}
```

Строки и указатели. Сравнение

```
int StringCompare(char* first, char* second) {
   while(*first && (*first == *second)){
       first++;
       second++;
   }

   return *first - *second;
}
```

В зависимости от удобства, мы оперируем массивам как указателями или явно обращаемся через оператор [ ] как на пред слайде

main

```
#include <iostream>

int main(int argc, char* argv[]) {
  for(int i = 0; i < argc; ++i)
      std::cout << argv[i] << " ";

  return 0;
}
```

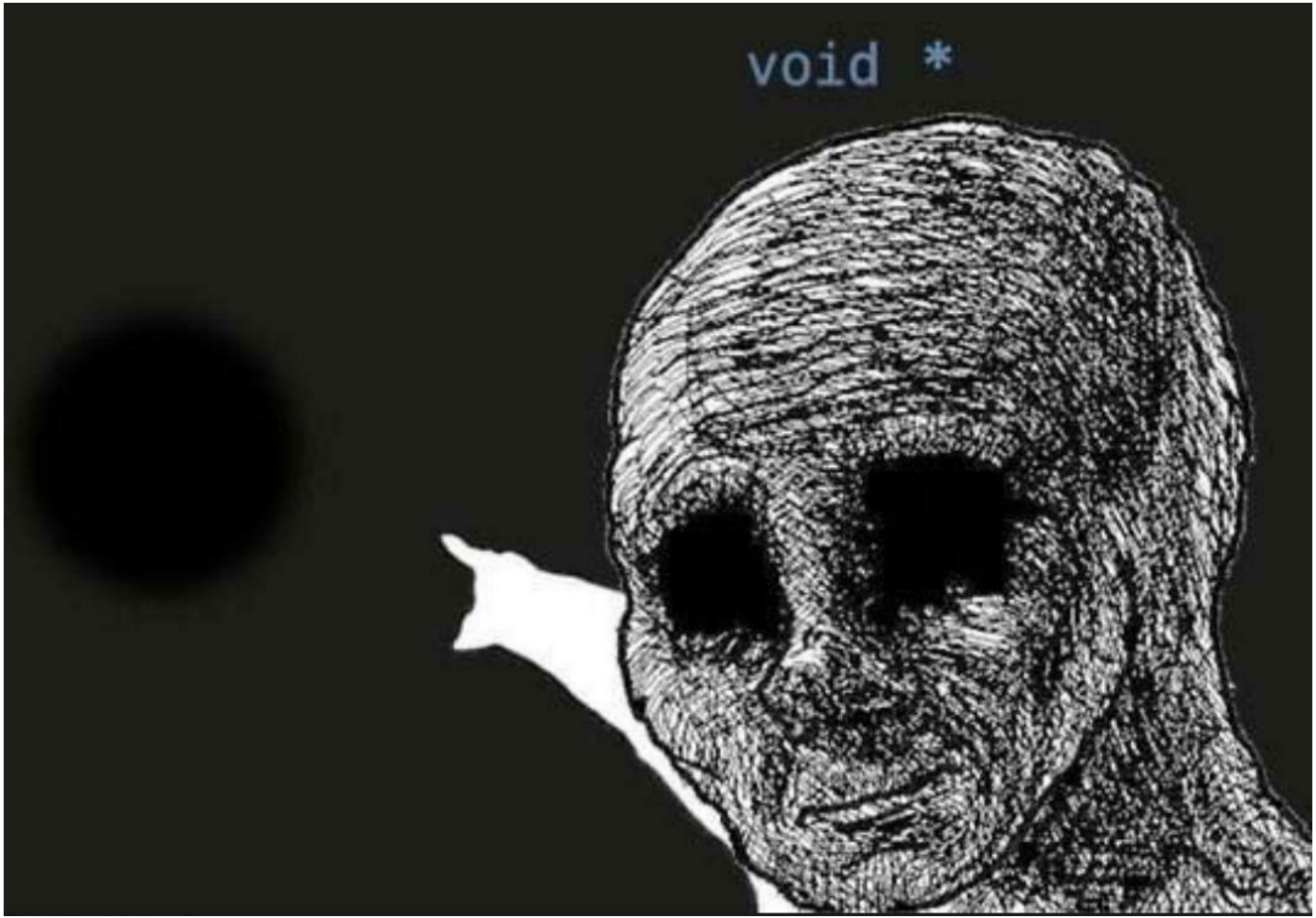
argc - размер массива
argv - массив строк

void*

```
int i = 239;
int* pi = &i;

void* pv = pi;

int* pj = (int*)pv;
```

void* - указатель на любой тип

void* - может быть явно приведен к указателю на другой тип


void *

void*

```
std::cout << "Addresses" << std::endl
          << pi << std::endl
          << pv << std::endl
          << pj << std::endl
          << std::endl;
```

Хранимый адрес не меняется от типа указателя

```
std::cout << "Sizes" << std::endl
          << sizeof(pi) << std::endl
          << sizeof(pv) << std::endl
          << sizeof(pj) << std::endl;
```

Размер указателя не меняется от типа указателя

Массивы и указатели

```
#include <iostream>
#include <format>

void printBytes(void* ptr, size_t size) {
    uint8_t* bytes = (uint8_t*)ptr;
    for(size_t i = 0; i < size; ++i) {
        std::cout << std::format("{:08b} ", *bytes);
        ++bytes;
    }

    std::cout << std::endl;
}
```

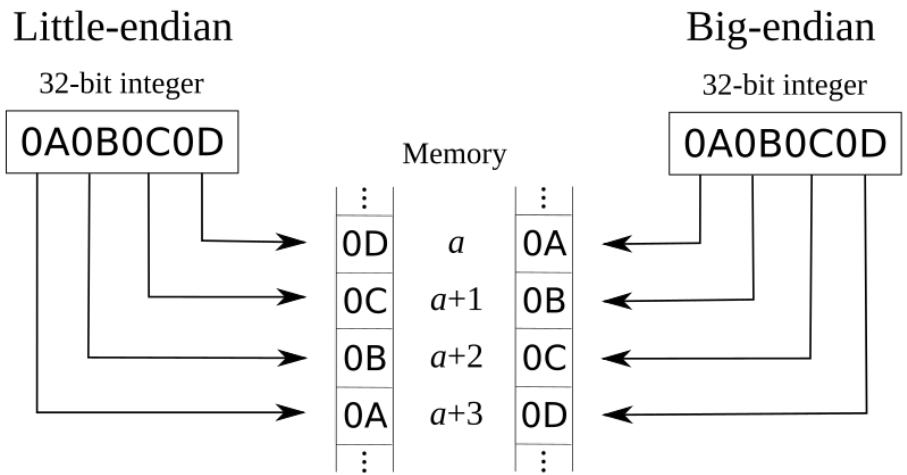
Любой указатель может быть представлен в виде массива байт

std::format - еще одни способ организовать форматированный вывод (C++20)

++bytes увеличивает адрес на 1, смещая на след байт

Представление целого числа в памяти

```
int main() {
    int i = 2 << 10;
    printBytes(&i, sizeof(4));
    i = 239;
    printBytes(&i, sizeof(4));
    return 0;
}
```



Указатели на функцию

```c
int same(int i) {
    return i;
}

int main() {
    int (*pf)(int) = same;
    int (*pf2)(int) = &same;

    pf(2);
    pf2(2);

    return 0;
}
```

Указатели на функцию

```
int* findMaxInArray(int* arr, size_t size, bool (*cmp)(int, int)) {
    int* result = arr;
    for(int i = 1; i < size; ++i) {
        if(cmp(*result, *(arr + i)))
            result = arr + i;
    }

    return result;
}
```

arr + 1 перемещает указатель на след элемент массива

cmp задает отношение порядка

Указатели на функцию

```
bool less(int a, int b) {
    return a < b;
}

bool greater(int a, int b) {
    return a > b;
}

int main() {
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8};
    std::cout << *findMaxInArray(arr, 8, less) << std::endl;
    std::cout << *findMaxInArray(arr, 8, greater) << std::endl;
    return 0;
}
```