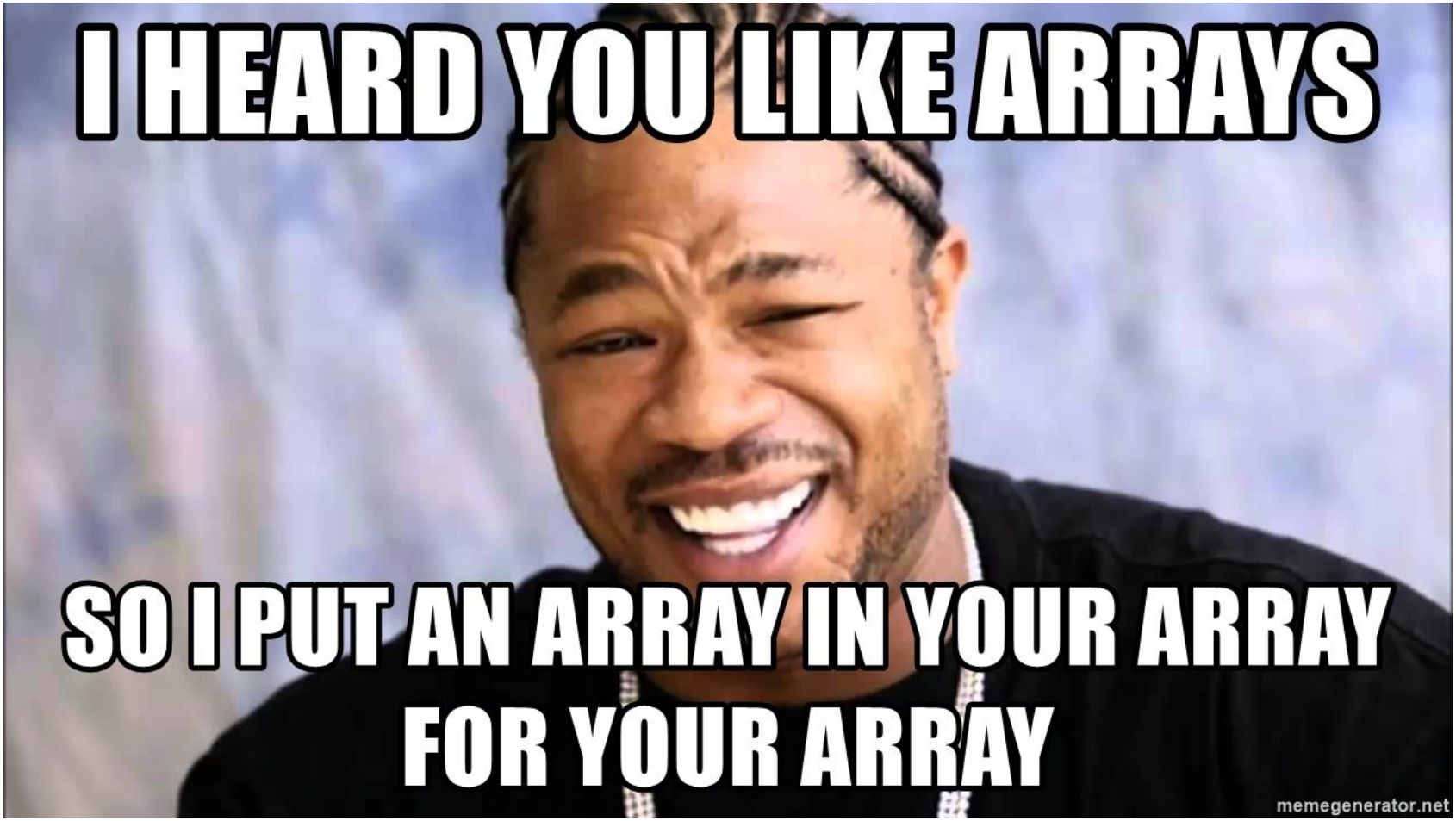
Основы алгоритмизации и программирования

Пашук Александр Владимирович

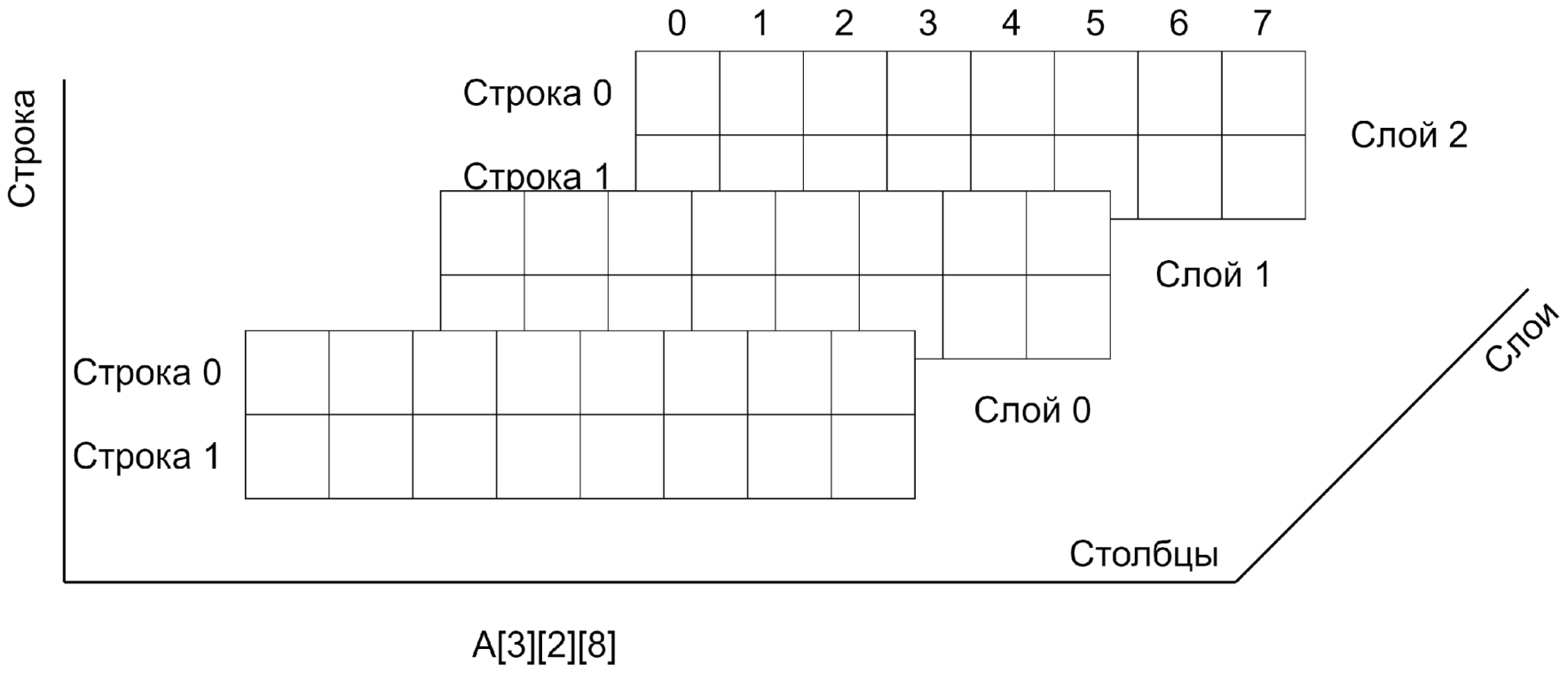
pashuk@bsuir.by

Содержание лекции

1. Многомерные массивы
2. Адреса и указатели
3. Операция получения адреса &
4. Указатели и массивы
5. Выделение памяти
6. Вопросы из теста


I HEARD YOU LIKE ARRAYS
SO I PUT AN ARRAY IN YOUR ARRAY FOR YOUR ARRAY
memegenerator.net

Что такое массивы?



Объявление массивов

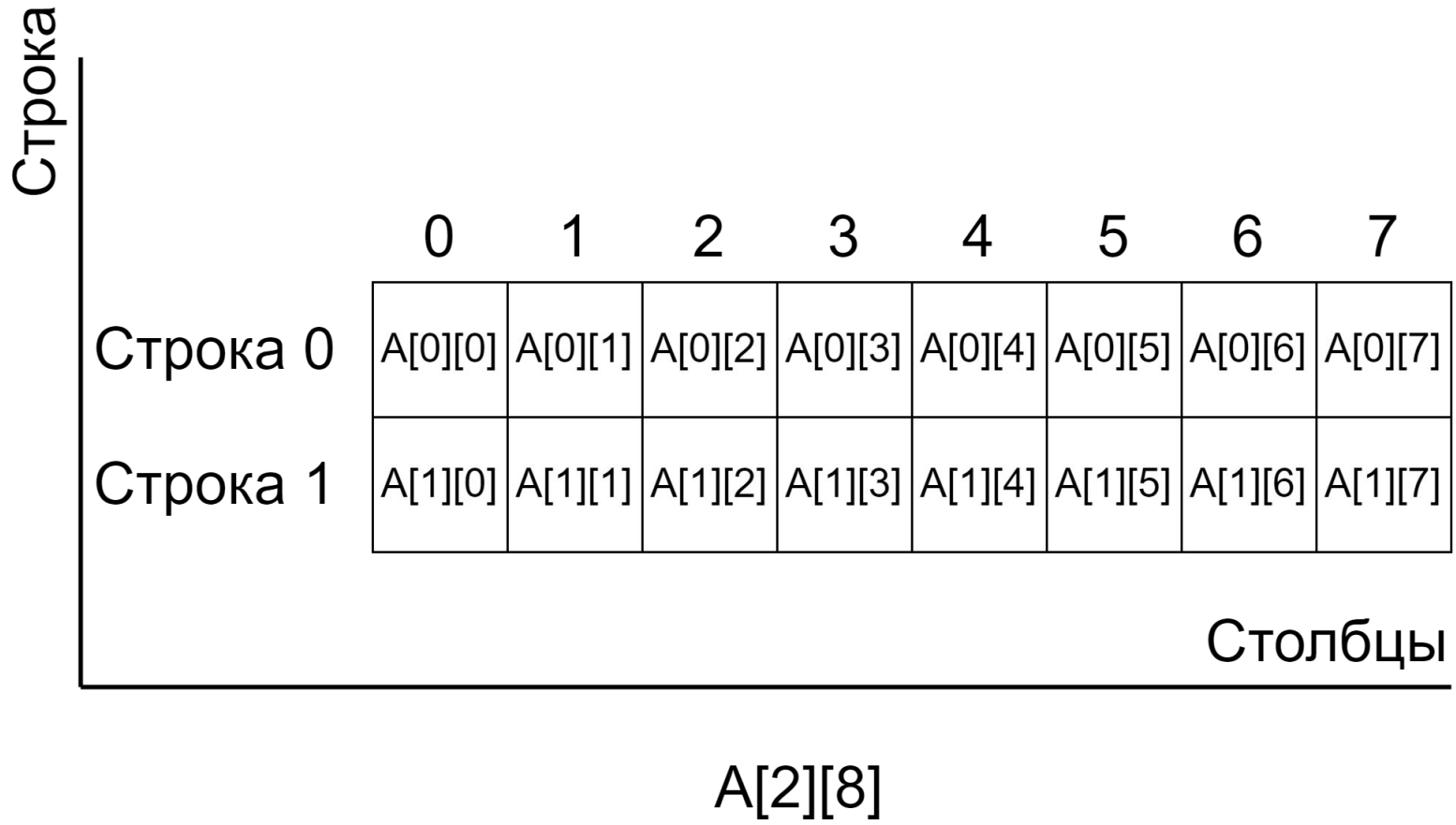
Синтаксис определения массива без дополнительных спецификаторов и модификаторов имеет два формата:

```
<type> <array_name>[<size_1>][<size_2>]…;
```

Например:

```
double stat[3][2][8];
```

Доступ к элементам



A[2][8]

Инициализация массивов

```
// int a[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};

int a[3][3] = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};

for (int i = 0; i < 3; i++)
    for (int j = 0; j < 3; j++)
        cout << a[i][j] << " ";
    cout << endl;
```

Пример

```
int main() {
  int numbers[2][3];
  cout << "Enter 6 numbers: " << endl;
  for (int i = 0; i < 2; ++i)
    for (int j = 0; j < 3; ++j)
      cin >> numbers[i][j];

  cout << "The numbers are: " << endl;
  for (int i = 0; i < 2; ++i)
    for (int j = 0; j < 3; ++j)
      cout << "numbers[" << i << "][" << j << "]: «
           << numbers[i][j] << endl;
   return 0;
}
```

Указатели

Для чего нужны указатели?

- Доступ к элементам массива
- Передача аргументов в функцию, от которой требуется изменить эти аргументы
- Передача в функции массивов и строковых переменных
- Выделение памяти
- Создание сложных структур данных (связные списки и т.п.)

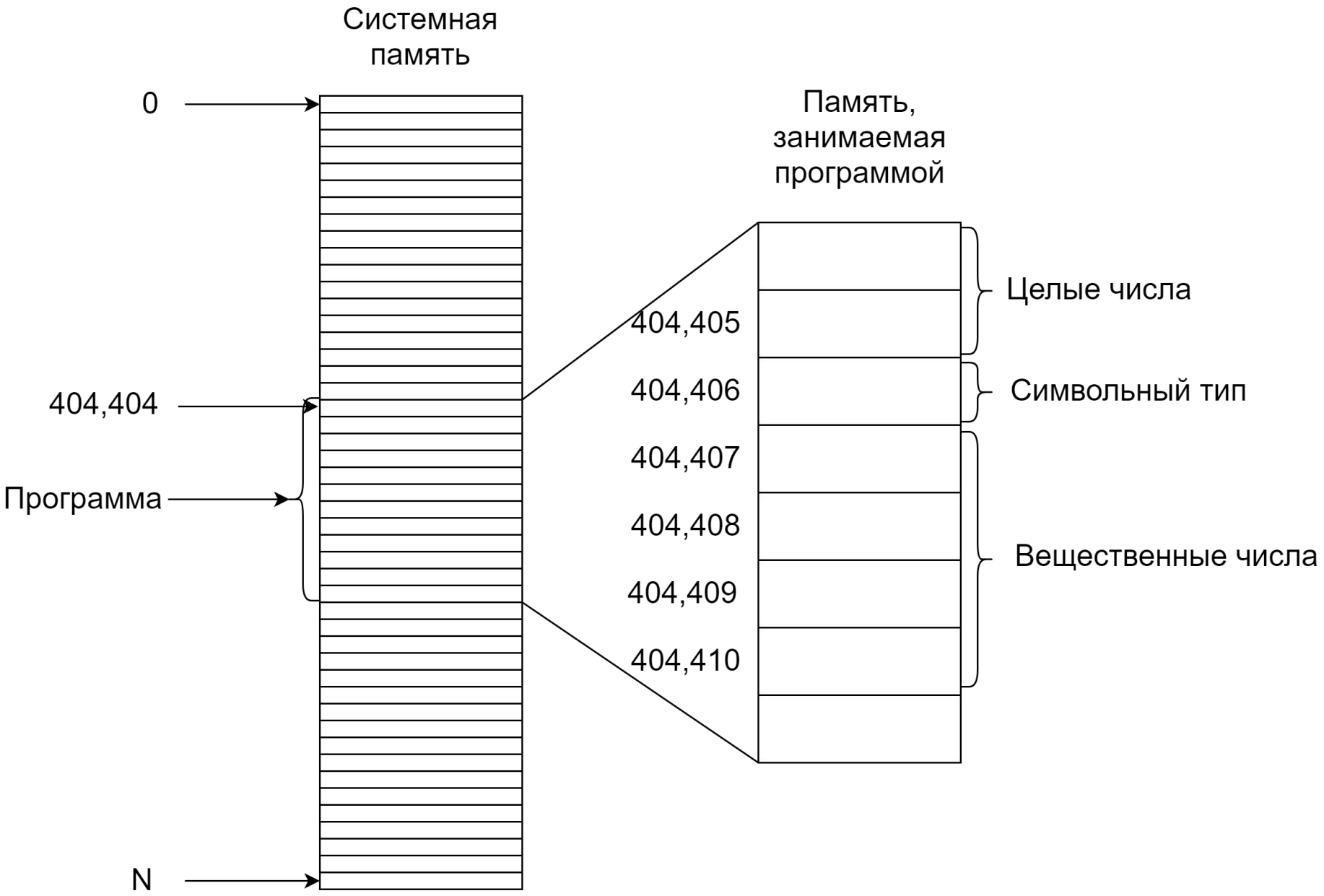
Нужны ли указатели?

- Почти все операции с указателями можно выполнить и без использования указателей (доступ к элементам массива и т.п.)
- В некоторых ситуациях указатели являются необходимым инструментом увеличения эффективности программ (создание и работа со связными списками)
- Некоторые ключевые возможности языка C++ требуют использования указателей (new, this виртуальные функции и т.п.)

Адреса и указатели

Всё довольно просто ☺️:

- Каждый байт памяти компьютера имеет адрес (например, для 1 МБ памяти: 0, 1, 2, … 1 048 575)
- Программа при запуске занимает некоторое количество адресов: **каждая переменная и функция начинаются с какого-либо конкретного адреса**.


Системная
память
0
404,404
Программа
N
Память,
занимаемая
программой
404,405
404,406
404,407
404,408
404,409
404,410
Целые числа
Символьный тип
Вещественные числа

Операция получения адреса

Адрес переменной можно получить с помощью операции получения адреса &.

Адрес != Значение

```
int main() {
    int a = 10, b;

    cout << &a << " " << &b << endl;
    // 0x61ff0c 0x61ff08
    // 6422284 6422280
    return 0;
}
```

Почему по убыванию?

- **Локальные переменные хранятся в стеке**, где адреса располагаются по убыванию.
- Для глобальных переменных адреса будут располагаться в порядке возрастания, т.к. **глобальные переменные хранятся в куче**, где адреса располагаются по возрастанию.
- Для разработчика этот нюанс в большинстве случаев не играет никакой роли.

Переменные-указатели

- Возможность узнать, где расположена в памяти переменная может быть полезна, при этом видеть адрес разработчику нет необходимости.
- Такая особенность приводит к необходимости использования переменных, **хранящих значение адреса**.
- Переменная, содержащая в себе значение адреса, называется **переменной-указателем** или просто **указателем**.

Переменные-указатели

```
int main() {
    int a = 10;

    cout << "Address of a: " << &a << endl;

    int* ptr;
    ptr = &a;

    cout << "Pointer value: " << ptr << endl;
    // Address of a: 0x61ff08
    // Pointer value: 0x61ff08

    return 0;
}
```

Синтаксис

Определение указателя может осуществляться как с помощью звездочки, записываемой после названия типа, так и перед именем переменной:

```
int* a;
int *b; // Not recommended

int* ptr1, * ptr2;
int *ptr1, *ptr2; // Not recommended
```

Инициализация указателей

- Указатель может хранить адрес переменной соответствующего типа
- Каждому указателю должно быть обязательно присвоено некоторое значение, иначе случайный адрес, на который он указывает, может оказаться чем угодно (включая код операционной системы).
- Неинициализированные указатели приводят к краху системы и к багам, которые тяжело выявить при отладке.

Доступ к переменной по указателю

Ситуация: мы не знаем имени переменной, но знаем ее адрес. Как получить доступ к значению переменной?

```
int main() {
    int a = 10; // Lost and unknown

    int* ptr = &a;

    cout << "Variable value: " << *ptr << endl;
}
```

Операция разыменования

- Звездочка, стоящая перед именем переменной (в примере: `*ptr`) называется **операцией разыменования**.
- Такая запись означает: **взять значение переменной, на которую указывает указатель**.
- С помощью этой операции можно использовать указатели не только для получения значения переменной, но и для выполнения действий с этими значениями.

Пример

```
int a = 10; // Lost and unknown
int* ptr = &a;

cout << "a = " << *ptr << endl;
cout << "*ptr = " << *ptr << endl;
// a = 10
// *ptr = 10

*ptr += 100;

cout << "a = " << *ptr << endl;
cout << "*ptr = " << *ptr << endl;
// a = 110
// *ptr = 110
```

Такие разные *

Звездочка, используемая в операции разыменования – это не то же самое, что звездочка, используемая при объявлении указателя.

- **Операция разыменования** предшествует имени переменной и означает **значение, находящееся в переменной, на которую указывает указатель**.
- **Звездочка в объявлении указателя** означает **указатель на**.

Доступ к значению переменной с использованием операции разыменования называется **непрямым доступом** или **разыменованием указателя**.

Повторение

```
int v;
int* ptr;
ptr = &v; // присваиваем ptr значение адреса v

v = 3; // прямой доступ
*ptr = 3; // непрямой доступ


float a = 100;
ptr = &a; // Нельзя!

void* ptrvoid;
ptrvoid = &a; // Можно!
```

Преобразование типов

Если необходимо присвоить одному типу указателя другой тип, можно использовать функцию `reinterpret_case` (Приведение типов без проверки).

```
int x = 42;

float* ptr = reinterpret_cast<float*>(&x);

cout << *ptr << endl; // ???
```

Указатели и массивы

Доступ к элементам массива можно получить как используя операции с массивами, так и используя указатели:

```
int arr[] = {5, 4, 3, 2, 1};

for (int i=0; i<5; i++) {
    // cout << arr[i] << " ";
    cout << *(arr+i) << " ";
} // 5 4 3 2 1
```

Как это работает?

Имя массива является его адресом.

- Следовательно, `arr+i` `(arr+3)` – это тоже адрес чего-то в массиве.
- **Важно:** это не сдвиг на 3 байта от начала массива, т.к. в этом нет смысла – каждая переменная типа int занимает 4 байта.
- Имея информацию о типе данных, компилятор интерпретирует выражение `arr+3` как адрес четвёртого элемента массива.
- Значение элемента массива можно получить с помощью операции разыменования.

Вопрос

```
int main() {
    int arr[] = {5, 4, 3, 2, 1};

    for (int i=0; i<5; i++) {
        // cout << arr[i] << " ";
        cout << *(arr++) << " ";
    }
    // 5 4 3 2 1

    return 0;
}
```

Указатели-константы

- Операция увеличения не может быть использована с именем массива (адресом).
- В данном случае `arr` является указателем константы. `arr++` тоже самое, что `7++`.

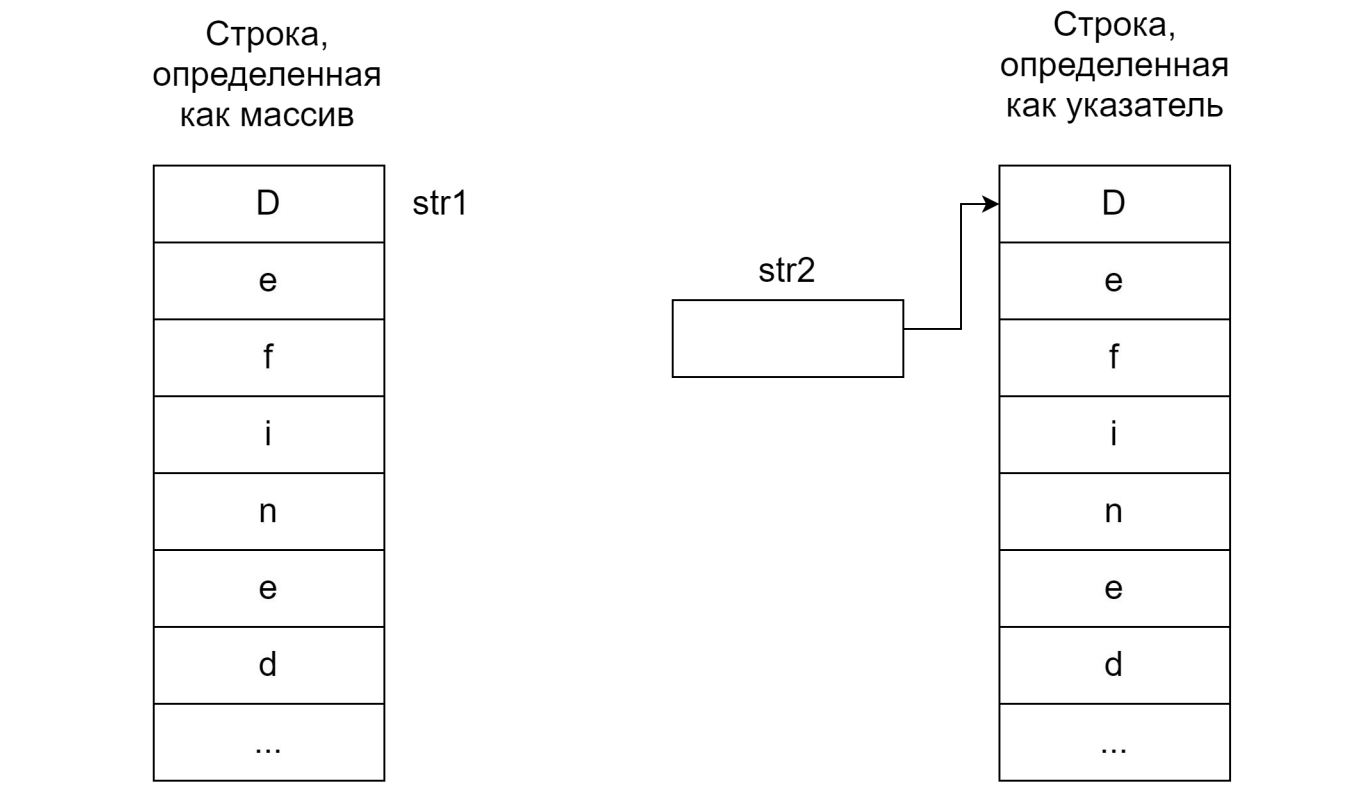
Альтернативное решение:

```
int arr[] = {5, 4, 3, 2, 1};
int* ptr = arr;

for (int i=0; i<5; i++)
    cout << *(ptr++) << " ";
// 5 4 3 2 1
```

Указатели на строки

```
int main() {
    char str1[] = "Defined with arrays";
    char* str2 = "Defined with pointers";

    cout << str1 << endl;
    cout << str2 << endl;

    // str1++; // Can't do it!
    str2++;

    cout << str2 << endl;
    // efined with pointers

    return 0;
}
```

Указатели на строки




```
char str1[] = "Defined...
```

```
char* str2 = "Defined...
```

Библиотека строковых функций

Многие из использованных в предыдущей лекции функций для строк имеют строковые аргументы, которые определены с помощью указателей.

Например, функция для копирования одной строки в другую:

```
char* strcpy (char* dest, const char* src);
```

Модификатор const

Два варианта использования модификатора const при объявлении указателя:

- Указатель на константу:

```
const int* ptr1;
```

- Константный указатель:

```
int* const ptr2;
```

Указатель на константу

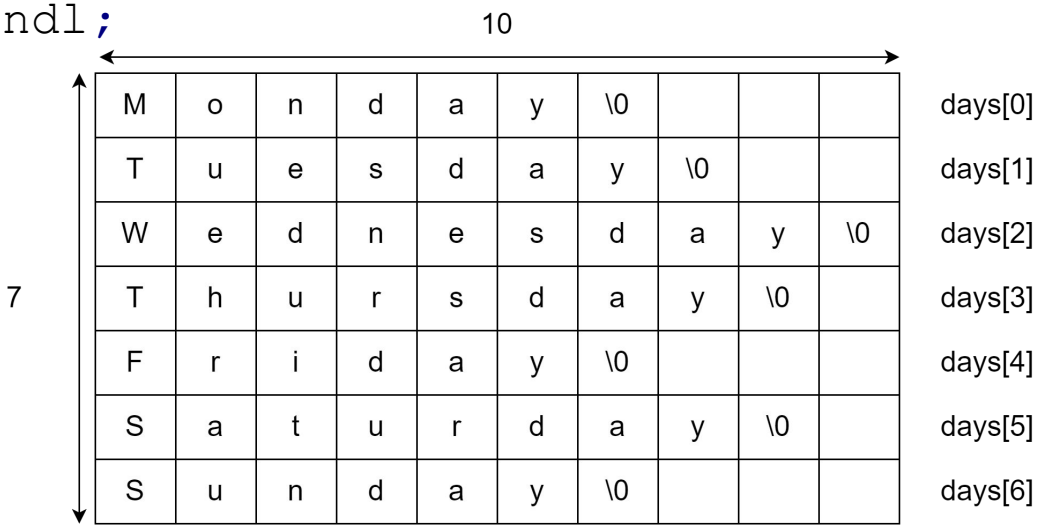
```
int main() {
    int a = 404, b = 500;
    const int* ptr;

    ptr = &a;

    // *ptr += 1; // Error!
    ptr = &b;

    return 0;
}
```

Константный указатель

```
int main() {
    int a = 404, b = 500;
    int* const ptr = &a;

    *ptr += 1;
    // ptr = &b; // Error!

    return 0;
}
```

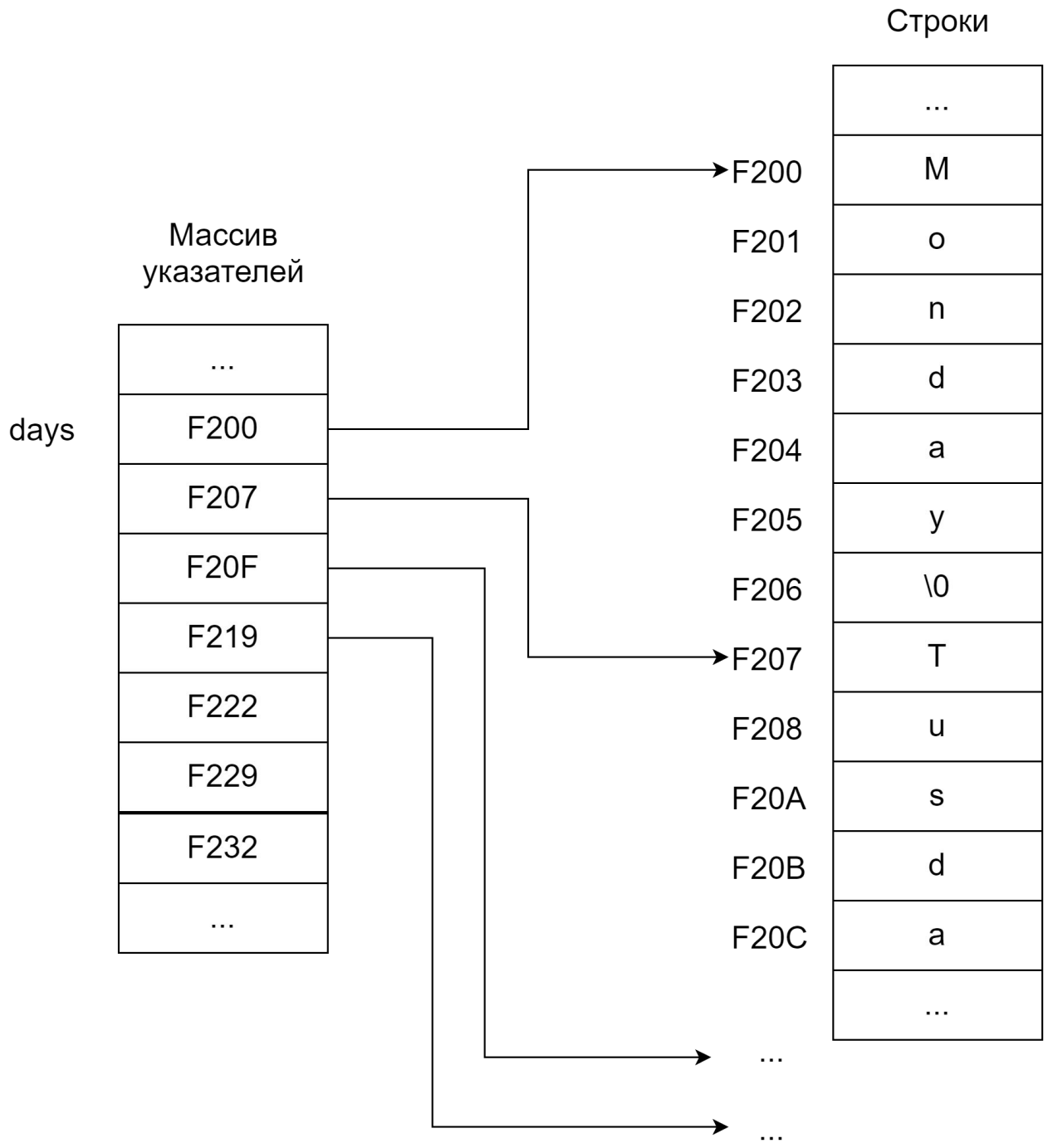
Массивы строк

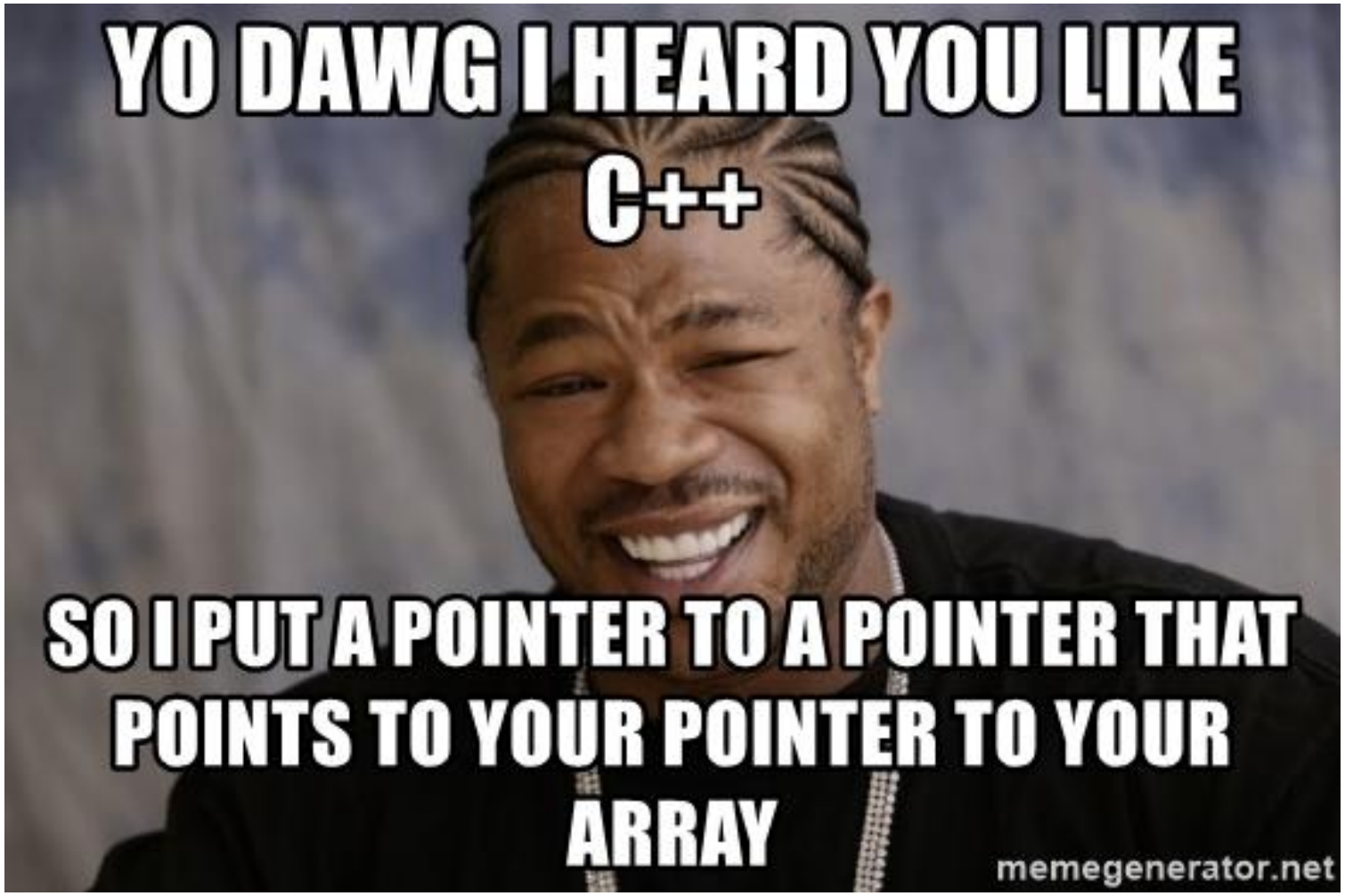
```
int main() {
  char days[7][10] = {
    "Monday", "Tuesday", "Wednesday",
    "Thursday", "Friday", "Saturday", "Sunday"
  };

  for (int i=0; i<7; i++)
    cout << days[i] << endl;

  return 0;
}
```



Массивы указателей на строки

```
int main() {
  char* days[7] = {
    "Monday", "Tuesday", "Wednesday",
    "Thursday", "Friday", "Saturday", "Sunday"
  };

  for (int i=0; i<7; i++)
    cout << days[i] << endl;

  return 0;
}
```


Строки
...
M
o
n
d
a
y
\0
T
u
s
d
a
...
F200
F201
F202
F203
F204
F205
F206
F207
F208
F20A
F20B
F20C
...
Массив
указателей
...
F200
F207
F20F
F219
F222
F229
F232
...
days
...
...


YO DAWG I HEARD YOU LIKE C++
SO I PUT A POINTER TO A POINTER THAT POINTS TO YOUR POINTER TO YOUR ARRAY
memegenerator.net

Пример

```
int main() {
    int var = 789;
    int* ptr1;
    int** ptr2;
    int*** ptr3;

    ptr1 = &var;
    ptr2 = &ptr1;
    ptr3 = &ptr2;

    cout << var << endl;
    cout << *ptr1 << endl;
    cout << **ptr2 << endl;
    cout << ***ptr3 << endl;

    return 0;
}
```

Пример

```
int arr[] = {1, 2, 3, 4, 5};
int* ptr = arr; // Pointer to array

int** dbl_ptr = &ptr; // Pointer to pointer to array

cout << "&arr = \t\t"
     << &arr << endl;            // &arr = 0x61fef8
cout << "ptr = \t\t"
     << ptr << endl;             // ptr = 0x61fef8
cout << "dbl_ptr = \t"
     << dbl_ptr << endl;         // dbl_ptr = 0x61fef4
cout << "*dbl_ptr = \t"
     << *dbl_ptr << endl;        // *dbl_ptr = 0x61fef8
cout << "**dbl_ptr = \t"
     << **dbl_ptr+3 << endl;         // **dbl_ptr = 4
```

Управление памятью

Все массивы до этого момента использовались без учета размера памяти.

Например:

```
int arr[100];
```

Недостаток: при написании программы необходимо знать насколько большой массив будет нужен.

Управление памятью

```
int main() {
    int size;

    cout << "Enter the number of numbers: ";
    cin >> size;

    int numbers[size]; // Error!

    return 0;
}
```

Операция new

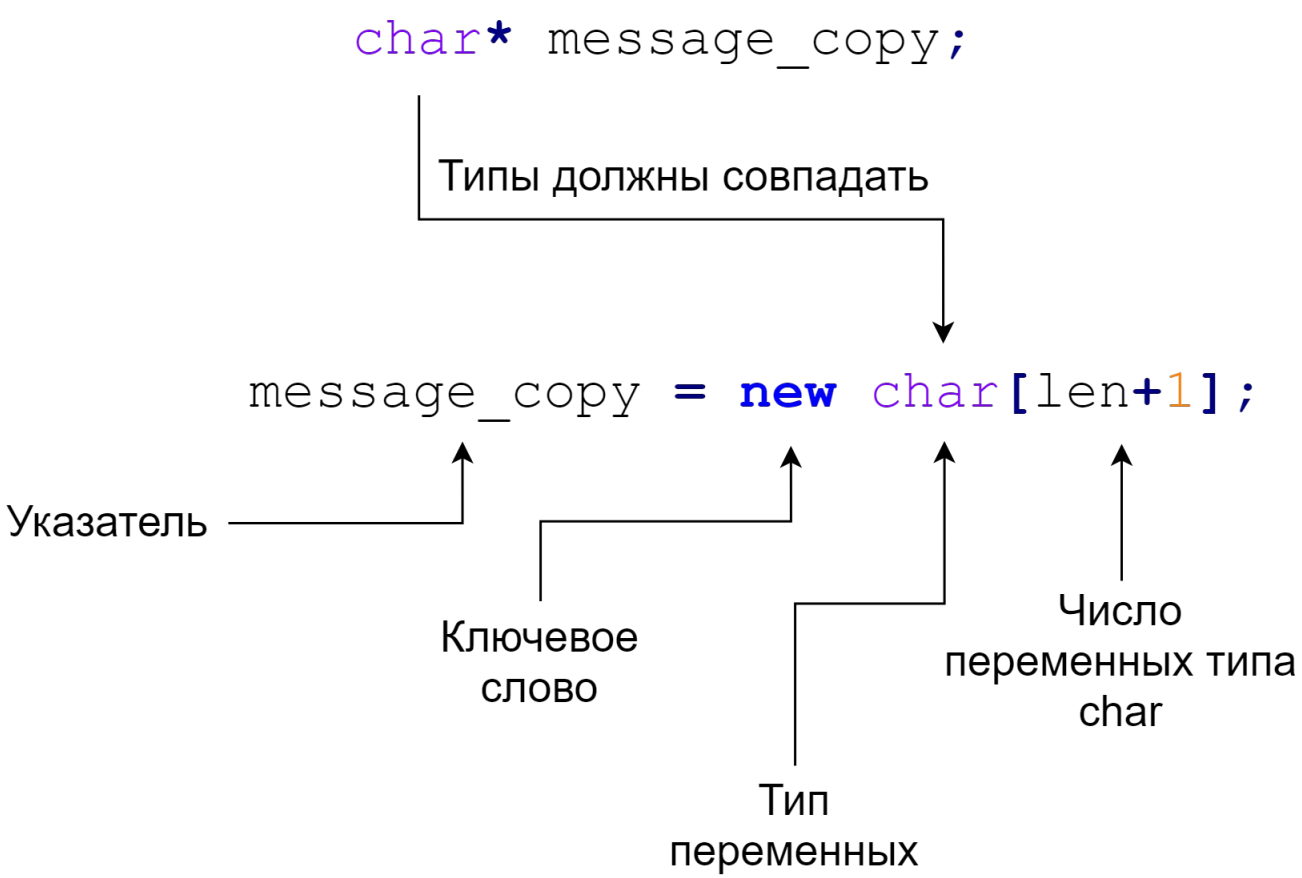
C++ предлагает другой подход к выделению памяти: операцию `new`.

Это универсальная операция, получающая память у операционной системы и возвращающая указатель на начало выделенного блока.

Пример

```
int main() {
    char* message = "Hello world";
    int len = strlen(message);

    char* message_copy;
    message_copy = new char[len+1];
    strcpy(message_copy, message);

    cout << message_copy << endl;

    delete[] message_copy;

    return 0;
}
```

Синтаксис операции new



Операция new

- В C вместо оператора new используется функция `malloc()` из библиотеки функций.
- Такой подход может быть при желании использован с в C++, однако не рекомендуется.
- Преимущество оператора new в том, что он сразу возвращает указатель на соответствующий тип данных, в то время как указатель функции `malloc()` должен быть явно преобразован к соответствующему типу.

Справка: c++ - In what cases do I use malloc and/or new? - Stack Overflow

new vs malloc()

```
// C-style
ptr = (int*) malloc(100 * sizeof(int));
// ...
free(ptr);


// C++-style
ptr = new int[100];
// ...
delete[] ptr;
```

Операция delete

- Операция `delete` предназначена для освобождения выделенных участков памяти, возвращая их операционной системе.
- Допустимо не использовать в некоторых случаях, например, при завершении программы (т.к. память автоматически освобождается при завершении работы программы).

Хороший тон: освобождать память, когда она больше требуется

Операция delete

- Освобождение памяти не подразумевает удаление указателя, связанного с этим блоком памяти. **Поэтому нельзя использовать указатели на освобожденную память.**
- Квадратные скобки означают, что освобождается память, выделенная под массив.

```
char* message_copy = new char[len+1];
delete[] message_copy;


ptr = new SomeClass;
delete ptr;
```

Пример вопроса на экзамене

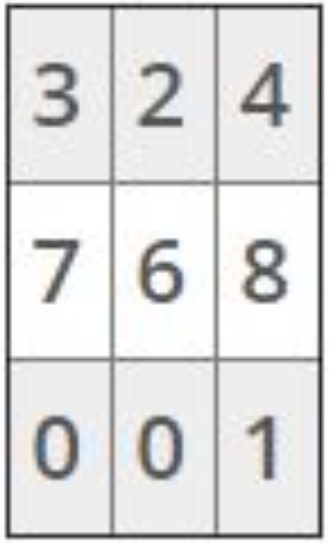
При доступе к многомерному массиву его индексы:

- Разделены запятыми
- Заключены в квадратные скобки и разделены запятыми
- Разделены запятыми и заключены в квадратные скобки
- Заключены в квадратные скобки

Пример вопроса на экзамене

Для заданного на рисунке массива, на какой элемент ссылается `q_array[0][2]`?

- 7
- 2
- 0
- 4

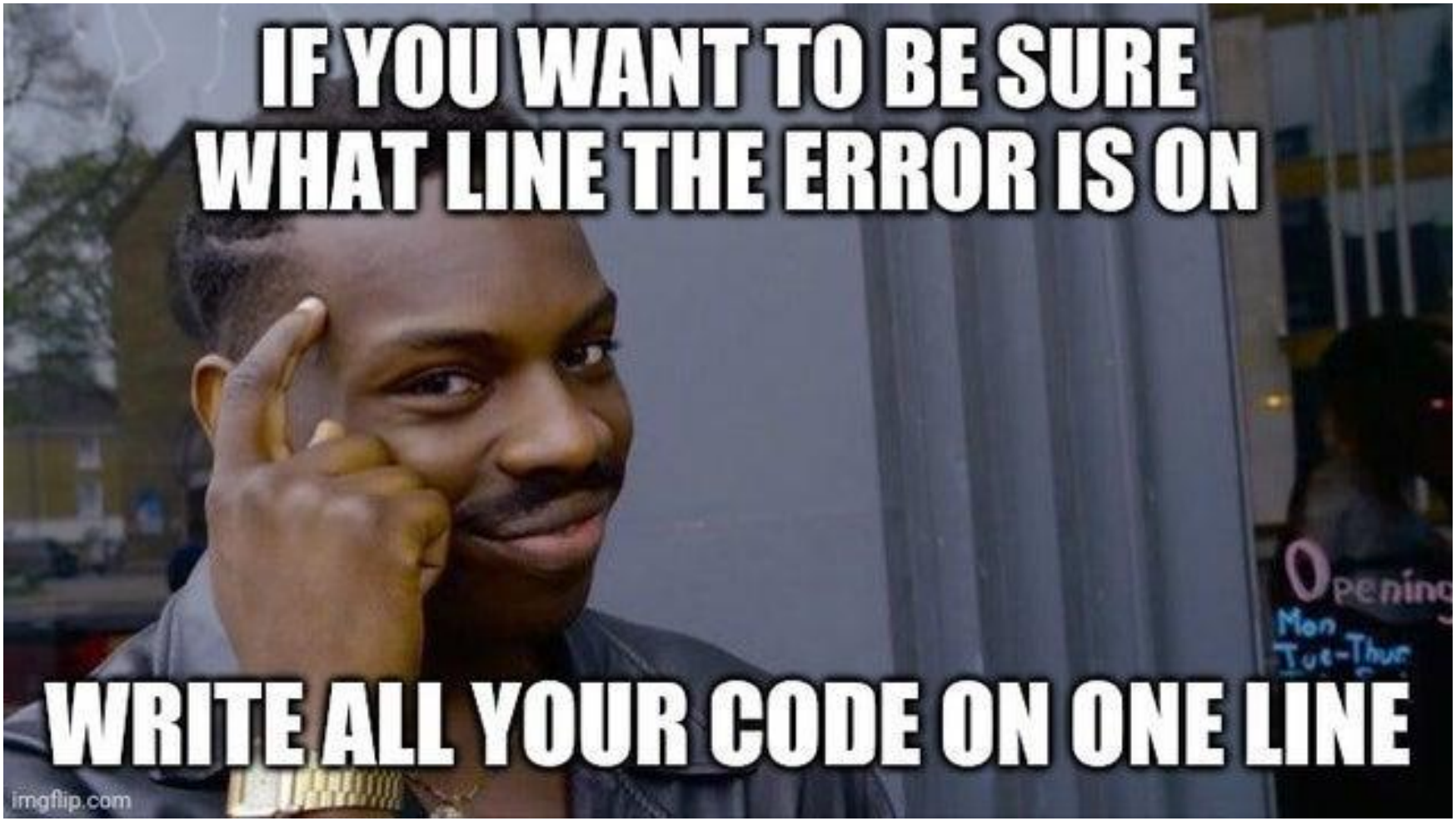


Пример задачи на экзамене

Реализовать простой вариант игры в крестики/нолики с использованием двумерного массива.

Вывод сетки игры на экран в удобном виде (отступы, выравнивания и т.п.).

Размер сетки для игры задается пользователем. Символ пользователя (x или o) определяется случайным образом перед каждой партией игры.


IF YOU WANT TO BE SURE
WHAT LINE THE ERROR IS ON
WRITE ALL YOUR CODE ON ONE LINE
imgflip.com