Массивы в языке С

Объявление, обращение.

Массив – сложный программный элемент с особенностями:

- Все элементы одного типа;
- Элементы расположены в памяти непрерывно;
- Имя массива – база(адрес участка памяти), относительно которой можно обращаться к любому элементу;
- Все элементы пронумерованы, начиная с 0;
- Доступ к элементам:
  - С помощью оператора индекса []; (2 пр.)
  - Через указатель, посредством опер. (*); (3 пр.)
- Массив может быть N-мерным;

Объявление :

- **Имя;**
- **Тип элементов;**
- **Размерность;**

T arr[N][M][L];

Num = N * M * L;
N – число слоев;
M – число строк;
L – кол-во элем. в строке;

Размерность может быть задана только **const выражением**.

Индекс – ср-во, которое позволяет компилятору исходя из размерности массива вычислить адрес требуемого элемента.

Инициализация массива.

Неявная инициализация:

- Глобальные и статические массивы инициал. по умолчанию «0»;
- Локальные и динамические массивы не инициализируются.

Явная инициализация:

- T arr[n] = {1,2,3,…,n} – полная инициализация;
- T arr1[n] = {1,1,1,…,1} – неполная инициализация;

  m < n,

Правило неполной инициализации – массив частично инициализированный, инициализируется нулями до полной инициализации, при этом число n – обязательно!

- Можно опускать старшую размерность, но тогда список инициализации должен быть полным:

  T arr[] = {1,2,3,…,n};

Массив можно проинициализировать только при определении, нельзя присвоить новые значения уже существующим элементам массива!

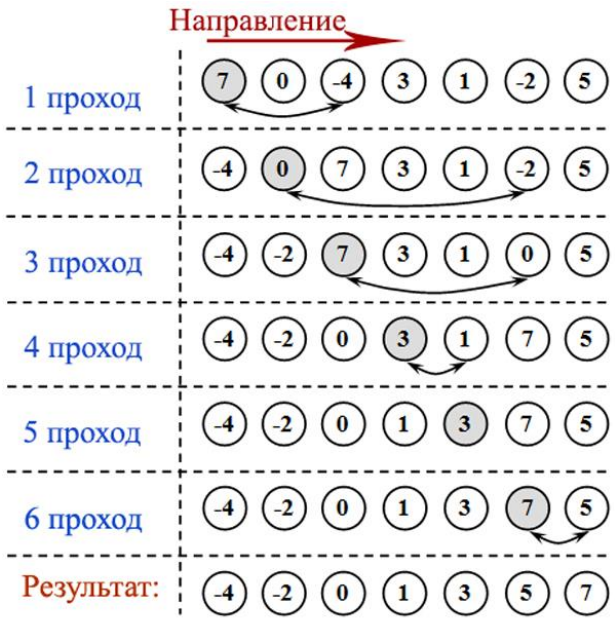
В списке инициализаторов могут встречаться переменные:

```
int n1 = 1, n2 = 2;
int arr[4] = {7, n1, n1, 8};
```

Размер массива

Вычисление количества элементов массива на этапе компиляции:

- char ar[] = “abc”;
- size_t **n**= sizeof(ar)/sizeof(char);
- **n**= sizeof(ar)/sizeof(ar[0]);



Сортировка выбором по возрастанию:

Последовательно проходим массив, сравнивая i-й элемент со всеми, находящимися после него и найдя min, переставим его на i-е место.

```
{
  int ar[]={7,2,1,6,-2,5,3,8};
  int n = sizeof(ar)/sizeof(ar[0]);

  for(int i = 0; i < n-1; ++i){
       int min = i;
       for(int j = i + 1;  j < n; ++j){
            if(ar[j] < ar[min])  min = j;
        }
        int temp = ar[min];
        ar[min] = ar[i];
        ar[i] = temp;
   }
}
```

Связь массивов и указателей

a[i] == i[a] == *(a + i);
a-имя массива, i-имя целоч. перем.

Имя одномерного массива компилятор воспринимает как :

T *const p на [0] элемент массива.

T arr[n] = {1,…,1} – опре. массива,

int *const p = arr; // int[]->int*

Отличия между указателем и массивом:

- Имя массива не содержит адреса как указатель (адрес 1-го эл. ассоциируется с именем массива);
- Под имя массива не отводится память как под указатель;
- p++; // ок
- arr++; // error

К i-му элементу массива можно обратится как по имени, так и по указателю:

- int tmp = arr[i];
- int tmp = p[i];
- int tmp = *(p + i);
- int tmp = *(arr + i);

Для вычисления адреса i-го элемента 1-о мерного массива:

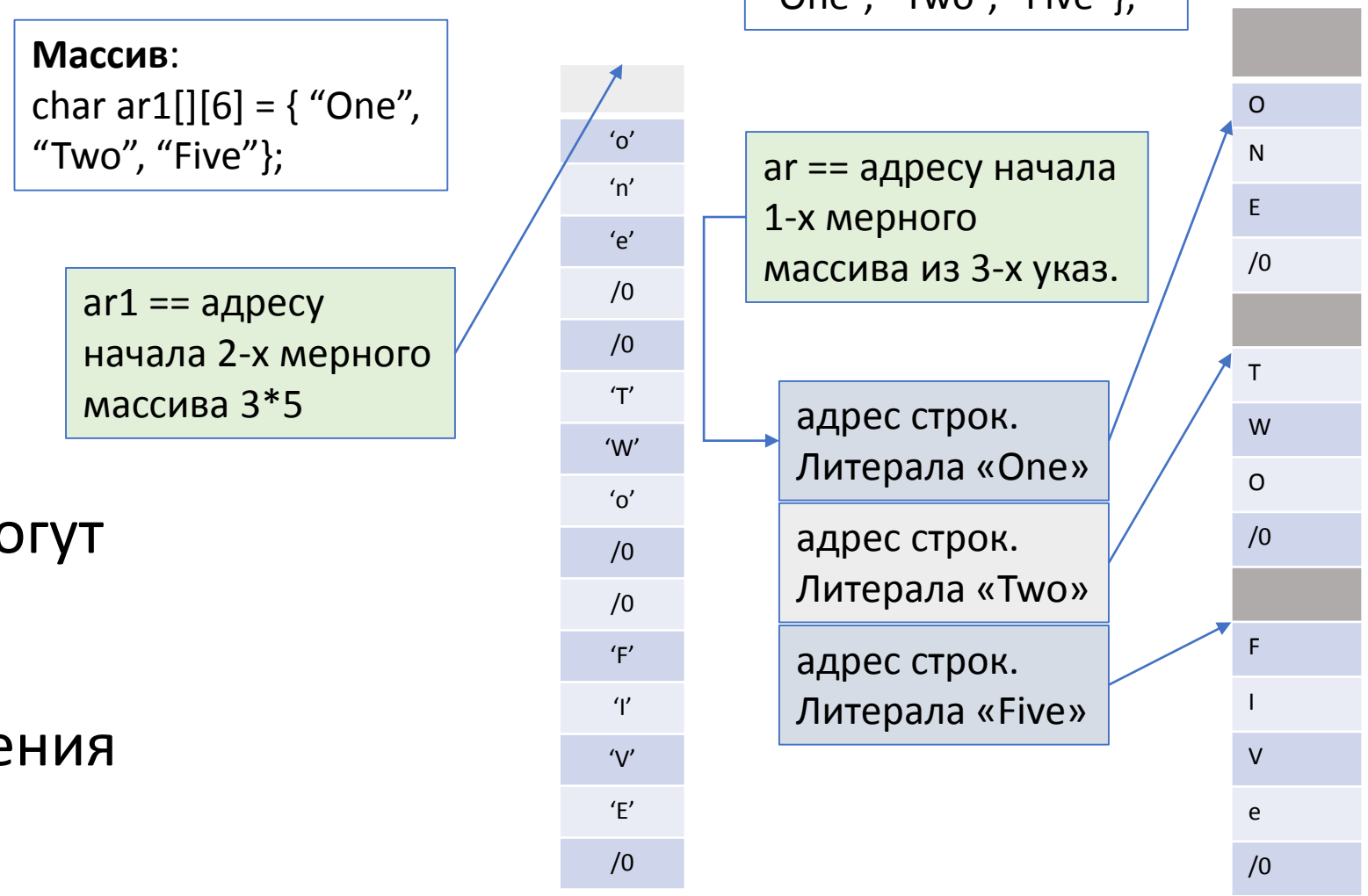
адр[i] = адр[0] + i * sizeof(T)

Размер самого массива не важен !

Массивы указателей

Массив:
char ar1[][6] = { “One”, “Two”, “Five”};

Массив указателей:
const char * ar[] = { “One”, “Two”, “Five”};



Элементами массива могут быть указатели. Часто массивы указателей используются для хранения адресов строк текста.

Динамические массивы в С

Если на этапе компиляции неизвестно какого размера массив понадобится? , используем механизм **динамического управления памятью.**

Основные функции стандартной библиотеки по работе с динамической памятью:

- **void * malloc(size_t)** – выделяет указанное количество байтов в дин. памяти.
- **void * calloc(size_t)** – выделяет динамическую память и инициирует ее нулями.
- **void * realloc(void *, size_t)** - пере выделяет динамическую память если потребовалось больше и освобождает прежнюю.
- **free(*void)** – освобождает память.

```
unsigned int NUM_ARR; // размер массива,
                      // запрашиваем у пользователя
{
  size_t n = NUM_ARR * sizeof(int); // байт для Arr
  int *p = static_cast<int*>(malloc(n));
  if(p){          // память выделили ☺, но хотим >
       p = static_cast<int*>(realloc(p, 2 * NUM_ARR));
        if(p){
              for(int i = 0; i < 2 * NUM_ARR; ++i){
                    p[i] = 1;
              }
        }
        else  {// придется просить еще раз ☹
        }
  }
  else { // придется просить еще раз ☹
   }
  ……
  free(p);   // когда работа с памятью завершена
}
```

Управление памятью в C++

Для динамического выделения памяти одной переменной используется оператор

new

int *ptr = new int(7); // динамически выделяем целочисленную переменную и присваиваем её адрес ptr, чтобы затем иметь доступ к ней

***ptr = 8;** // присваиваем значение 8 только что выделенной памяти

Освободить память после переменной можно с помощью оператора

delete

// Предположим, что ptr ранее уже был выделен с //помощью оператора new
delete ptr; // возвращаем память, на которую указывал ptr, обратно в ОС
ptr = 0; // (используйте **nullptr** вместо 0 в C++11)

Динамическое выделение памяти — это способ запроса памяти из ОС запущенными программами по мере необходимости.

Без указателя с адресом на только что выделенную память у нас не было бы способа получить доступ к ней.

Указатель, указывающий на освобожденную память, называется **висячим указателем**.
Разыменование или удаление висячего указателя приведет к неожиданным результатам.

Динамические массивы в C++

Для выделения динамического массива и работы с ним используются отдельные формы операторов **new** и **delete**:

new[] и delete[].

```
#include <iostream>

int main()
{
    std::cout << "Enter a positive integer: ";
    int length;
    std::cin >> length;
    int *array = new int[length];
   std::cout << allocated an array of integers of length " << length << '\n';
    array[0] = 7;     // присваиваем элементу под индексом 0 значение 7
    delete[] array; // освобождения памяти под массив
    array = 0;        // используйте nullptr вместо 0 в C++11
    return 0;
}
```

На сегодня всё!