Указатели

Лекция 5

Указатель

- это переменная, которая содержит в качестве своего значения адрес памяти
- указатель может хранить адрес:
  - переменной
  - функции
  - массива
  - объекта
  - другого указателя

Объявление указателя

Объявление указателя на целое и целого числа:

```
int *countPtr = nullptr, count = 0;
```

Объявление двух указателей типа `float`:

```
float *xPtr = nullptr, *yPtr = 0;
```

Хорошо!
`nullptr` - специальная константа для инициализации и обнуления указателей.
Можно также воспользоваться числом 0.

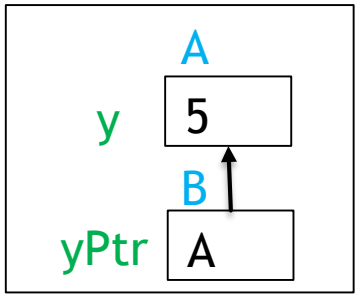
```
float *xPtr, *yPtr;
int *countPtr;
```

Плохо!
Объявлять неинициализированный указатель

Операция адресации

- Взятия адреса или адресации **&** – унарная операция, которая возвращает адрес своего операнда

```
int y = 5;
int *yPtr = nullptr;
yPtr = &y;
```



- Операнд операции адресации должен быть L-величиной (т.е. чем-то таким, чему можно присвоить значение так же, как переменной)
- Операция адресации не может быть применена к константам, к выражениям, не дающим результат, на который можно сослаться

Операция разыменования

- Разыменования или косвенной адресации * - возвращает значение объекта, на который указывает ее операнд (т.е. указатель)

```
cout << *yPtr << endl; //5
```

- Применяется только к переменным, хранящим адрес (либо к выражениям, результатом которых будет адрес)

Операции с указателями

```
int a = 7;
int *aPtr = &a;
std::cout << "Address a is: " << &a << std::endl;
std::cout << "Value aPtr is: " << aPtr << std::endl;
std::cout << "Value a is: " << a << std::endl;
std::cout << "Value *aPtr is:" << *aPtr << std::endl;
std::cout << "&*aPtr is " << &*aPtr << std::endl;
std::cout << "*&aPtr ia " << *&aPtr << std::endl;
```

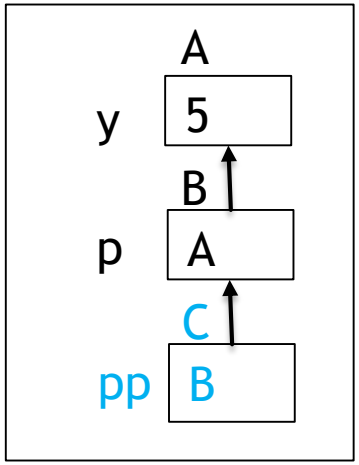
Результат:

```
0012FF7C
0012FF7C
7
7
0012FF7C
0012FF7C
```

Указатель на указатель

- Позволяет хранить адрес переменной, хранящей адрес
- При объявлении нужно использовать две звездочки
- Для получения значения нужно использовать операцию разыменования дважды

```
int y = 5;
int *p = nullptr;
p = &y;
int **pp = nullptr;
pp = &p;
```



Указатель на указатель

```
int a = 5;
int * p = &a;
int ** pp = &p;
std::cout << a << std::endl;
std::cout << *p << std::endl;
std::cout << **pp << std::endl;
std::cout << &a << std::endl;
std::cout << p << std::endl;
std::cout << *pp << std::endl;
std::cout << &p << std::endl;
std::cout << pp << std::endl;
std::cout << &pp << std::endl;
```

Результат:

```
5
5
5
0012FBA0
0012FBA0
0012FBA0
0012FB94
0012FB94
0012FB88
```

Взаимосвязь указателей и массивов

- Имя массива – это адрес первого элемента массива
- Имя массива – это постоянный указатель
- Можно объявить указатель на первый элемент массива и использовать его вместо имени массива
- Указатель на первый элемент массива и имя массива могут использоваться практически эквивалентно

Взаимосвязь указателей и массивов

```
const int SIZE = 5;
int b[SIZE] = {1, 2, 3, 4, 5};
int* bPtr = nullptr;

bPtr = b;

bPtr = &b[0];
```

Эти строчки эквивалентны

```
for (int i = 0; i < SIZE; i++) {
    std::cout << b[i] << std::endl;
}
```

```
for (int i = 0; i < SIZE; i++) {
    std::cout << bPtr[i] << std::endl;
}
```

После инициализации указателя работать с массивом можно через его имя, а можно через указатель

Арифметика указателей

Возможные действия:

```
<указатель> = <указатель> + <целое число>
<указатель> = <указатель> – <целое число>
<указатель> = <указатель> ++
<указатель> = <указатель> --
<целое число> = <указатель> - <указатель>
```

Арифметические действия с указателями имеют смысл, только если указатель ссылается на массив

Арифметика указателей

```
const int ROW = 5;
double arr[ROW] = { 1.1, 2.2, 3.3, 4.4, 5.5 };
double *p = arr;
```

Код программы	Преобразует компилятор
`p + 3`	`p + 3 * sizeof (double)`
`p += 4`	`p += 4 * sizeof (double)`
`p - 3`	`p - 3 * sizeof (double)`
`p -= 2`	`p -= 2 * sizeof (double)`
`p++`	`p += sizeof (double)`
`--p`	`p -= sizeof (double)`
`p - arr`	`(p - arr) / sizeof (double)`

Арифметика указателей

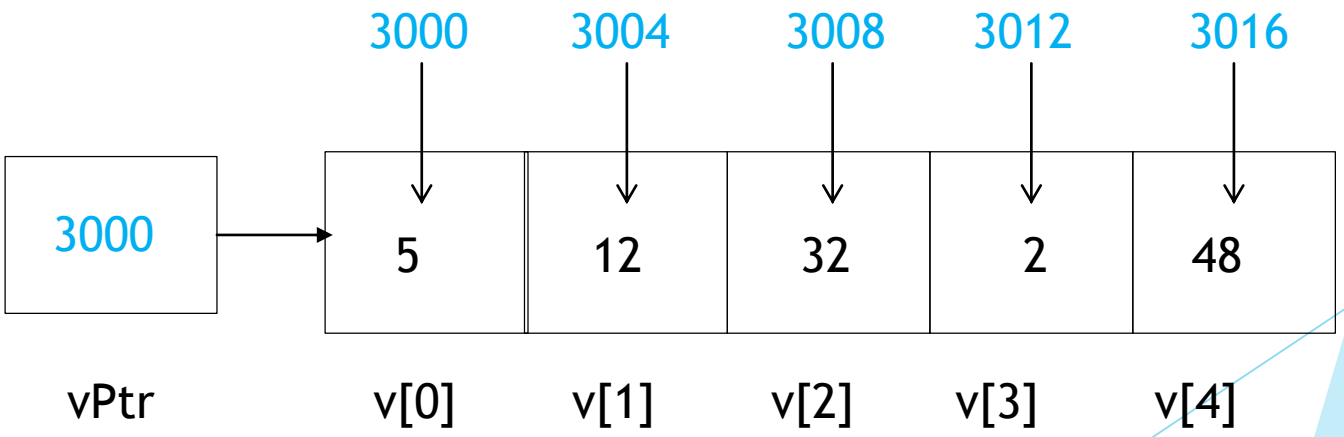
```
const int SIZE = 5;
int v[SIZE] = { 5, 12, 32, 2, 48 };
int *vPtr = &v[0];
std::cout << vPtr++ << std::endl;
vPtr +=2;
std::cout << vPtr << std::endl;
std::cout << vPtr - v << std::endl;
```

Результат:

```
3000

3012

3
```



Взаимосвязь указателей и массивов

```
int b[5] = {1, 2, 3, 4, 5};
int* bPtr = nullptr;

bPtr = b;

bPtr = &b[0];
```

Имя массива - это постоянный указатель, значит, оно не является L-величиной:

```
bPtr +=3;   //OK
b    += 3;  //error
bPtr ++;    //OK
b    ++;    //error
```

Операция индексации и запись указатель-смещение

- Для доступа к элементу массива или для сдвига указателя по массиву можно использовать два варианта обращения:
  - Через операцию индексации:

```
std::cout<<b[3];
bPtr[3] = 5;
```

  - Через запись указатель-смещение

```
std::cout<<*(bPtr+3);
*(b+3) = 5;
```

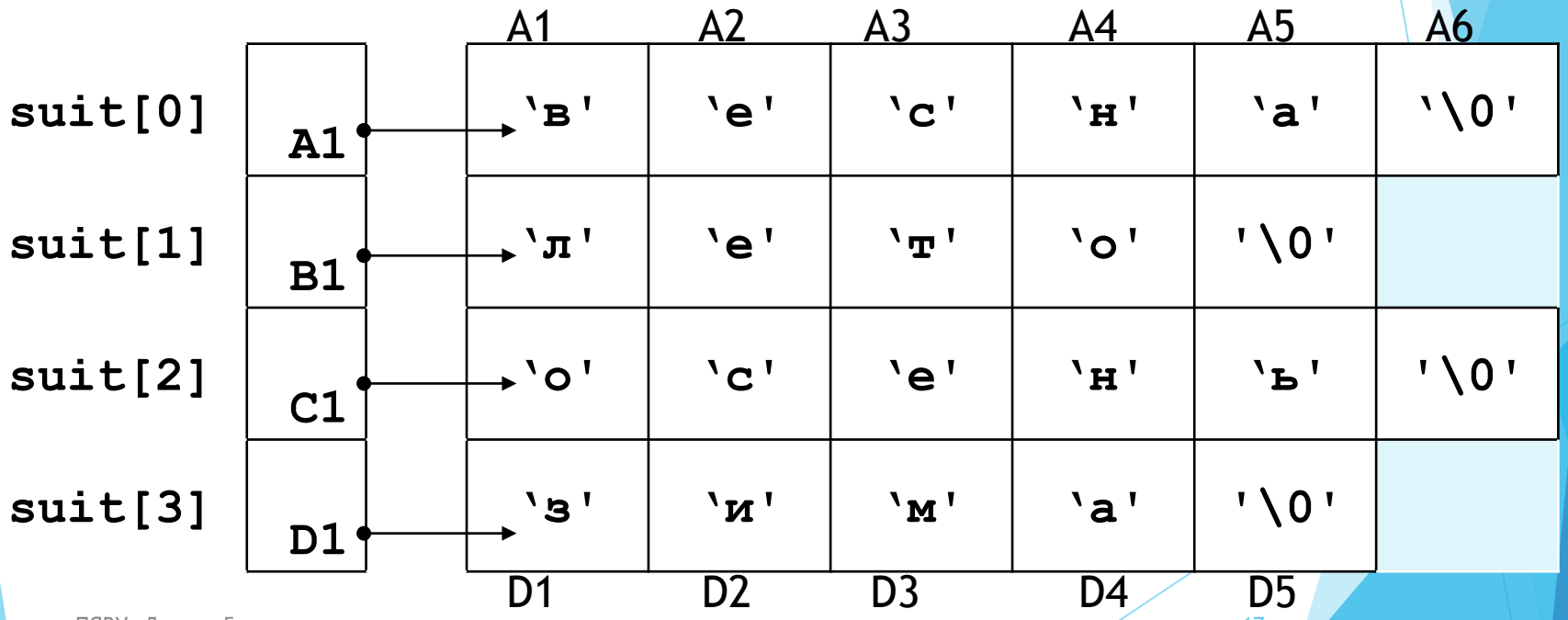
- Эти варианты эквивалентны

Массивы указателей

- Это массивы, элементами которых являются указатели
- Используются при работе с динамическими объектами
- Указатели внутри массива могут ссылаться на массивы переменной длины
- Часто используются при работы со строками формата Си
- Массивы указателей можно рассматривать как двумерные массивы. У таких массивов известно количество строк, но не известно количество столбцов (оно может быть разным в каждой строке)

Массивы указателей

```
char *suit[4] = { "весна",
                  "лето",
                  "осень",
                  "зима" };
```



Массивы указателей

```
char* c[]     = {"ENTER",
                 "NEP",
                 "POINT",
                 "FIRST"};
char **cp[]   = {c+3, c+2, c+1, c};
char *** cpp = cp;


std::cout << * * ++ cpp;
std::cout << * -- * ++ cpp + 3 << " ";
std::cout << *cpp[ -2 ] + 3;
std::cout << cpp[ -1 ][ -1 ] + 1 << std::endl;;
```

Массив указателей

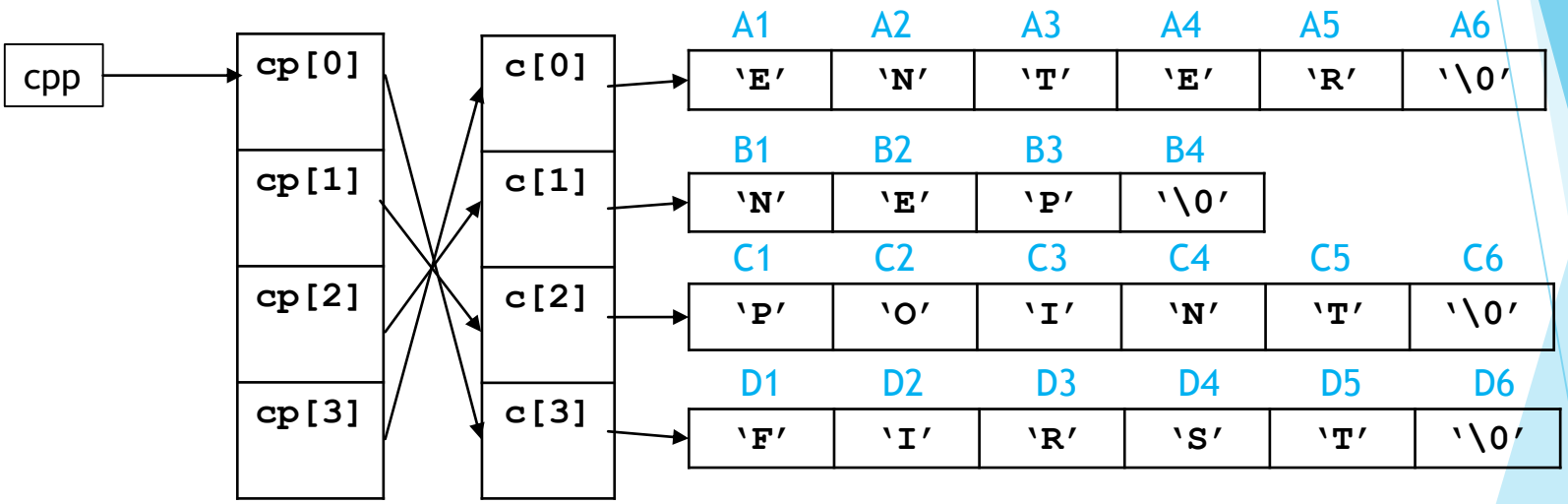
Массив указателей на указатели

Указатель на указатель на указатель

Что будет напечатано?

Массивы указателей

Распределение памяти в задаче с предыдущего слайда



Указатели на массивы

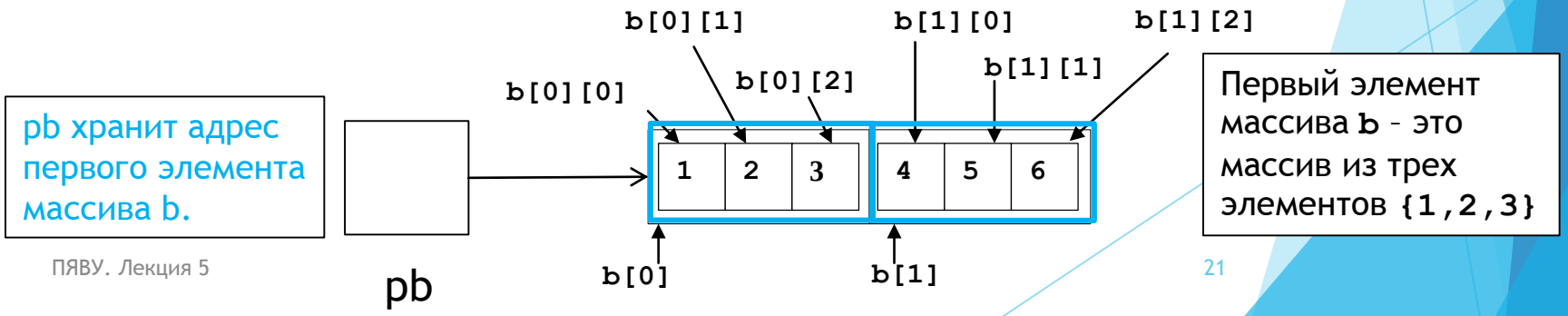
- Это указатели, которые ссылаются на целый массив, а не на отдельный элемент
- Используются при передаче многомерных массивов в функции
- При арифметике указателей смещаются на размер всего массива, на который ссылаются
- Указатели на массивы также можно рассматривать как двумерные массивы. У таких массивов может быть неизвестное число строк, но число столбцов фиксировано и не меняется

Указатели на массивы

```
const int ROW = 2, COLUMN = 3;
int b[ROW][COLUMN] = { 1,2,3,4,5,6 };
int(*pb)[COLUMN] = nullptr;
pb = b;
for (int i = 0; i < ROW; i++) {
    for (int j = 0; j < COLUMN; j++) {
        std::cout << pb[i][j] << "\t";
    }
    std::cout << std::endl;
}
```

pb ссылается на первый элемент массива **b**.
Теперь через pb можно работать с массивом b

Работа как с обычным двумерным массивом



pb хранит адрес первого элемента массива b.

Первый элемент массива **b** - это массив из трех элементов **{1,2,3}**

Указатели на массивы

```
const int ROW = 2, COLUMN = 3;
int b[ROW][COLUMN] = { 1,2,3,4,5,6 };
int(*pb)[COLUMN] = nullptr;
pb = b;
std::cout << pb    << std::endl;
std::cout << b     << std::endl;
std::cout << b[0] << std::endl;
```

Результат:

```
0028F914
0028F914
0028F914
```

pb имеет тип **int(*)[3]**
b имеет тип **int[2][3]**
b[0] имеет тип **int * const**

При этом адрес, на который они ссылаются - одинаковый

Указатели на массивы

```
const int ROW = 2, COLUMN = 3;
int b[ROW][COLUMN] = { 1,2,3,4,5,6 };
int(*pb)[COLUMN] = nullptr;
pb = b;

std::cout << pb    + 1   << std::endl;
std::cout << b     + 1   << std::endl;
std::cout << b[0] + 1 << std::endl;
```

Результат:

```
0028F920
0028F920
0028F918
```

Смещение указателей дает разные результаты:
Указатели **pb** и **b** хранят адрес массива и сдвигаются на **sizeof(int[COLUMN])**
Указатель **b[0]** хранит адрес одного целого числа и сдвигается на **sizeof (int)**

Указатели на массивы

```
const int ROW = 2, COLUMN = 3;
int b[ROW][COLUMN] = { 1,2,3,4,5,6 };
int(*pb)[COLUMN] = nullptr;
pb = b;
std::cout << pb   ++   << std::endl;
std::cout << b    ++  << std::endl;
std::cout << b[0]++ << std::endl;
```

Результат:

```
0028F914
ошибка
ошибка
```

Указатели **pb** и **b[0]** являются постоянными указателями на первый элемент массива, поэтому к ним нельзя применять операцию инкремента / декремента

Динамические массивы

- Их размер может меняться в процессе работы программы
- Память под них выделяется и освобождается только по запросу пользователя (программиста)
- Выделение памяти происходит с помощью операции **`new`**, освобождение – с помощью операции **`delete`**
- Место выделяется в специальной памяти – динамической
- Динамические массивы работают медленнее обычных (статических)

Выделение памяти под динамические массивы

```
char * MyArr;
int n = 0;
std::cout << "Enter a number" << std::endl;
std::cin >> n;
//выделение памяти под массив
//символьного типа
MyArr = new char[n];
…
//освобождение памяти из-под массивА
delete [] MyArr;
```

Выделение памяти под двумерный массив

```
int ** MyArr, n, m;
std::cout << "Enter two numbers" << std::endl;
std::cin >> n >> m;
//выделение памяти под двумерный массив
//сначала под массив указателей
MyArr = new int*[n];
//потом под каждый из подмассивов
for (int i=0;i<n;i++)
{
   MyArr[i] = new int[m];
}
```

Освобождение памяти из-под двумерного массива

```
//сначала из-под каждого подмассива
for (int i=0; i<n; i++)
   delete [] MyArr[i];
//потом из-под массива указателей
delete [] MyArr;
```

Конец