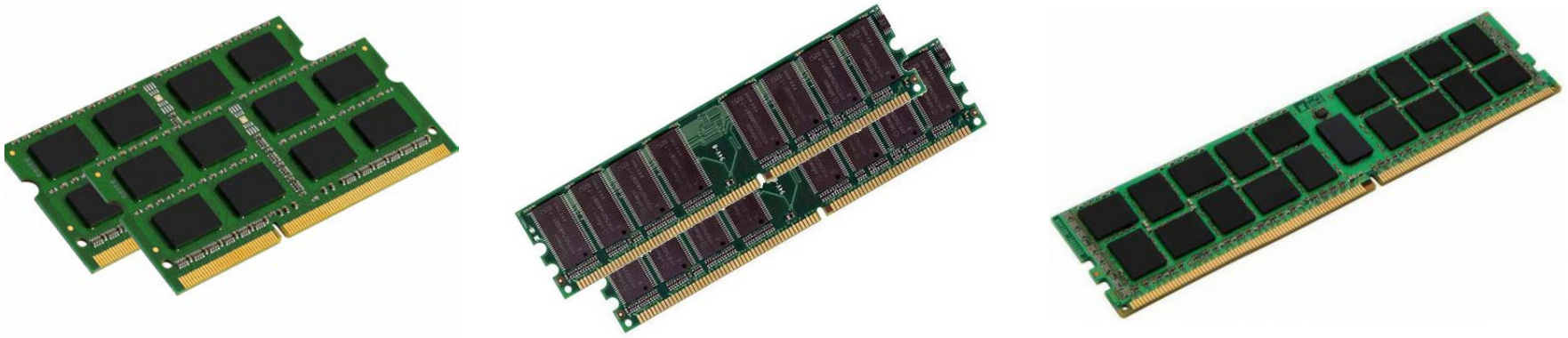
Лекция 8

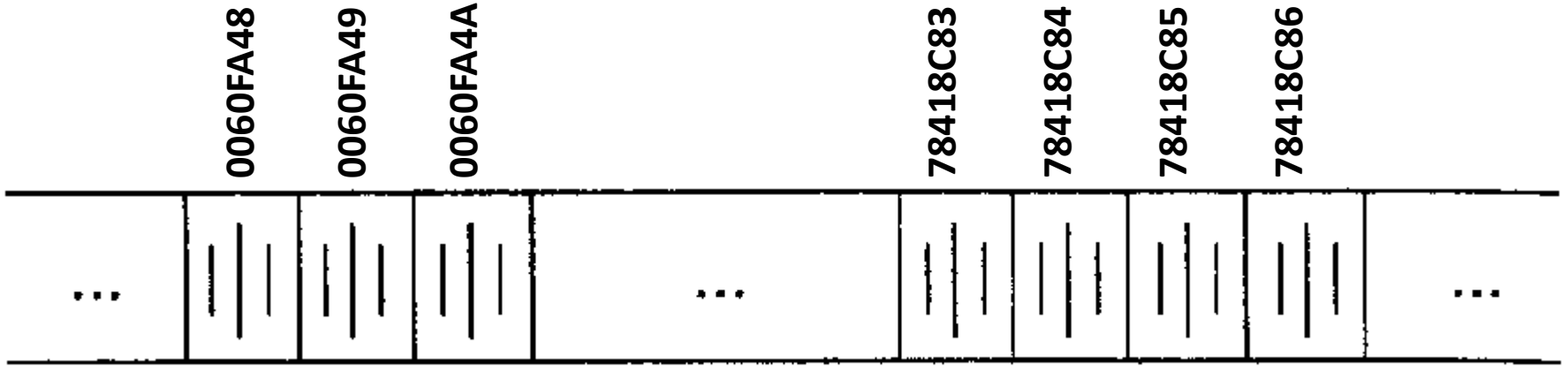
Указатели

Преподаватель Палехова Ольга Александровна,
кафедра О7 БГТУ «Военмех»

Организация оперативной памяти



Логически оперативная память представляет собой непрерывную последовательность байт, каждый из которых имеет свой порядковый номер, называемый **адресом**.



Указатели и объекты

Указатель (pointer) – это переменная, значением которой является адрес некоторого объекта в оперативной памяти.

Адреса **есть** у:

- переменных классов *auto* и *static*
- функций
- динамически выделенных блоков памяти
- отдельных байтов

Адресов **нет** у:

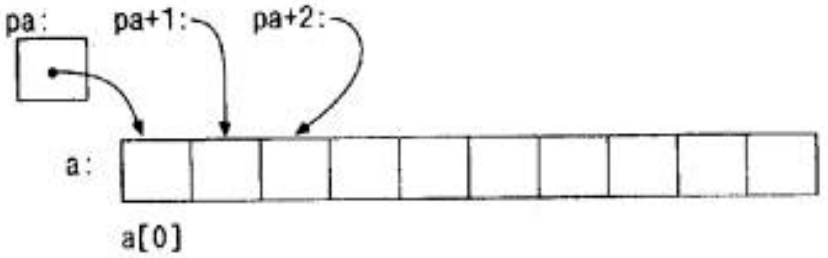
- констант
- выражений
- переменных класса *register*

Синонимы

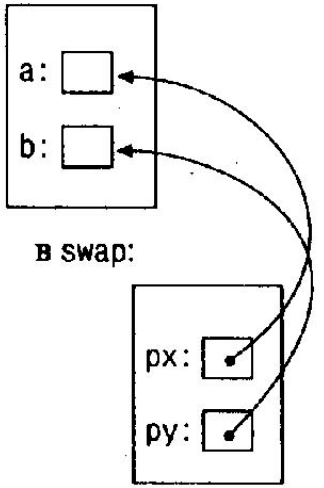
- Значением указателя является адрес объекта
- Указатель хранит адрес объекта
- Указатель указывает на объект

Когда используют указатели

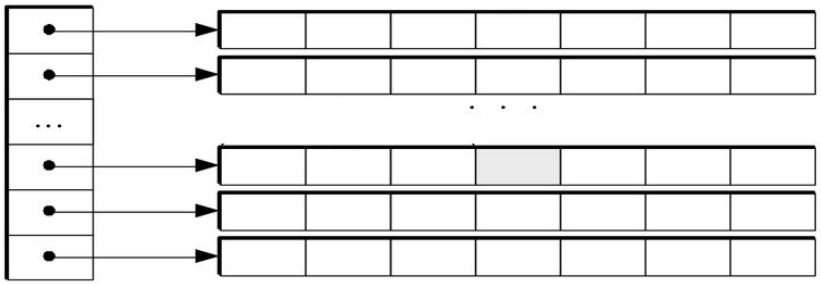
Указатели используют:



- при работе с **массивами**
- для передачи параметров в **функции**



- для работы с **файлами**
- для работы с **динамической памятью**



Объявление указателей

Общий вид объявления указателя такой же, как и у обычной переменной. Отличие в одном символе:

```
базовый_тип * имя_указателя;
```

Тип указателя

Признак указателя

```
char * pC;     /* указатель на char */
int * pI;      /* указатель на int */
double * pD;   /* указатель на double */
void * p;      /* указатель на нечто */
int * pI_1, * pI_2;  /* указатели на int */
```

Звездочку повторяем!

Неинициализированный указатель использовать нельзя!

Виды указателей

В зависимости от базового типа:

- **типизированный** указатель «знает» тип объекта, на который он указывает (размер, способ кодирования, операции)
- **обобщенный** указатель (**указатель на void**, **нетипизированный** указатель, указатель на «нечто») «знает» **только** адрес

Чем инициализировать указатель?

1. Константа **NULL** – нулевой адрес, не соответствующий никакому реальному адресу. По нулевому указателю не может быть ничего ни записано, ни прочитано.

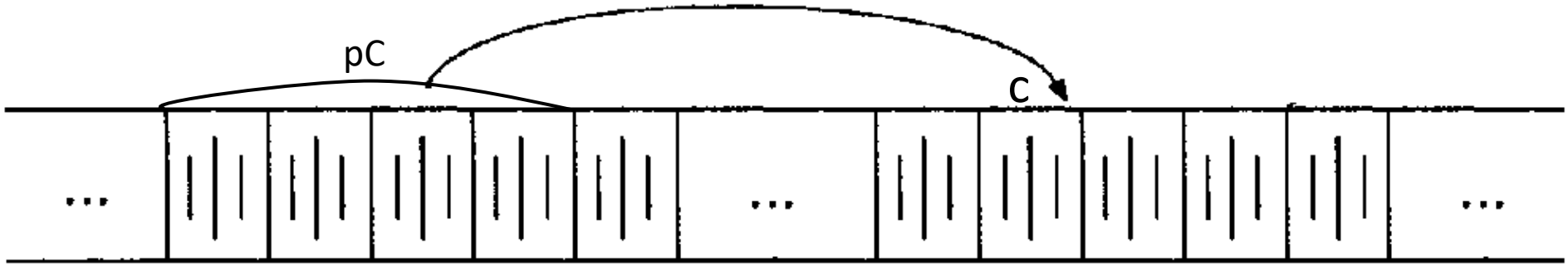
```
char * pC = NULL;
int * pI = NULL;
double * pD = NULL;
void * p = NULL;
```

Во избежание ошибок NULL рекомендуется присваивать всем временно не используемым указателям.

Чем инициализировать указатель?

2. Адрес переменной ***того же типа***. Указателю на void можно присвоить адрес переменной любого типа.

```
char c, *pC = &c;      /* адрес переменной c */
int i, *pI = &i;       /* адрес переменной i */
double d, *pD = &d;    /* адрес переменной d */
void *p = &pI;         /* адрес переменной pI */
```



~~pC = &i;~~
~~pI_1 = &d;~~
~~pD = &i;~~

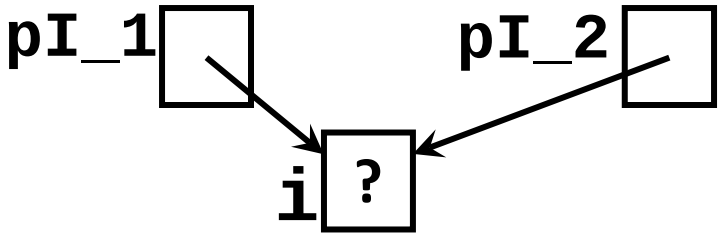
C90 – ошибка компиляции.
C99 и младше – предупреждение.
Несоответствие типов указателей приводит к ОШИБКАМ.

Чем инициализировать указатель?

3. Значение указателя того же типа. Указателю на void можно присвоить значение любого указателя.

```
int i, *pI_1 = &i, *pI_2 = pI_1;
double d, *pD = &d;
void *p = pD;
```

~~pI_1 = pD;~~
~~pD = pI_2;~~



C90 – ошибка компиляции.
C99 и младше – предупреждение.
Несоответствие типов указателей приводит к ОШИБКАМ.

Обращение к данным

Унарная * – операция получения значения по адресу или операция разыменования

```
int m = 4, n, *pI;
pI = &m;
printf ("m = %d", *pI);/* вывод значения */
n = 4*(7 - *pI);        /* n = 4*(7 - 4) = 12 */
*pI = 4*(n – m);        /* m = 4*(12 – 4) = 32 */
printf("&m = %p", pI);/* вывод адреса */
```

«вытащить» значение по адресу

спецификатор формата вывода адреса

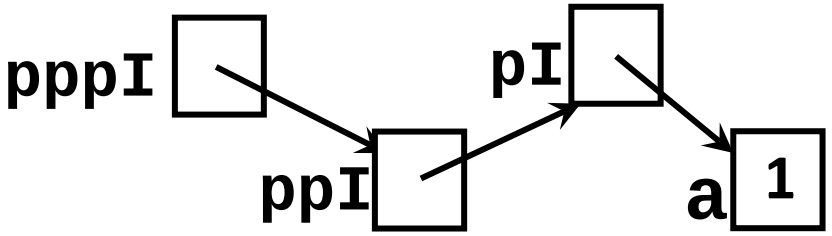
! **С разыменованным значением выполняются те же операции, что и с обычной переменной того же типа.**

Операции над указателями

- `sizeof` – размер в байтах (не зависит от типа указателя, только от платформы)
- `(тип)` – явное приведение типа
- `*` – получение значения по адресу (разыменование)

Операцию разыменования нельзя применить к указателю на void без явного приведения типа.

- `&` – получение адреса



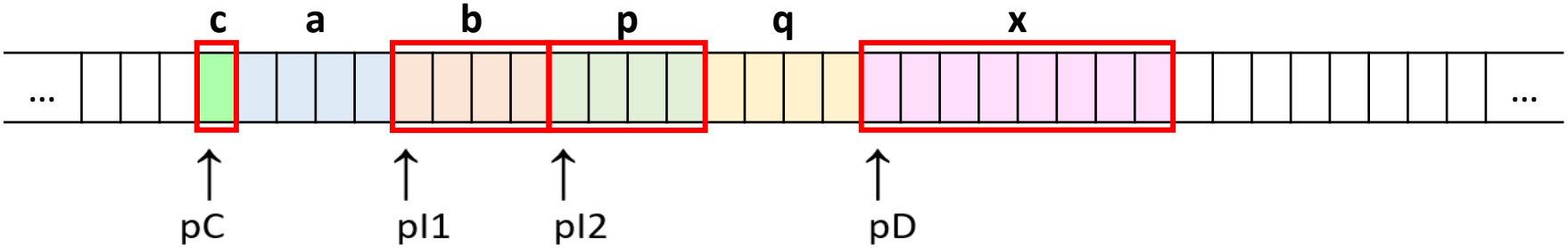

```
int a = 5, *pI = &a;
int **  ppI = &pI;
int *** pppI = &ppI;
***pppI = 1;
```

- `=` – присваивание
- Операции адресной арифметики

Операции адресной арифметики

```
double x;
int a, b, p, q;
char c;
```

```
char *pC = &c;
int *pI1 = &b;
int *pI2 = &p;
double *pD = &x;
```



- операции отношения и сравнения на равенство

```
pI1 < pI2
```

```
pI1 == pI2
```

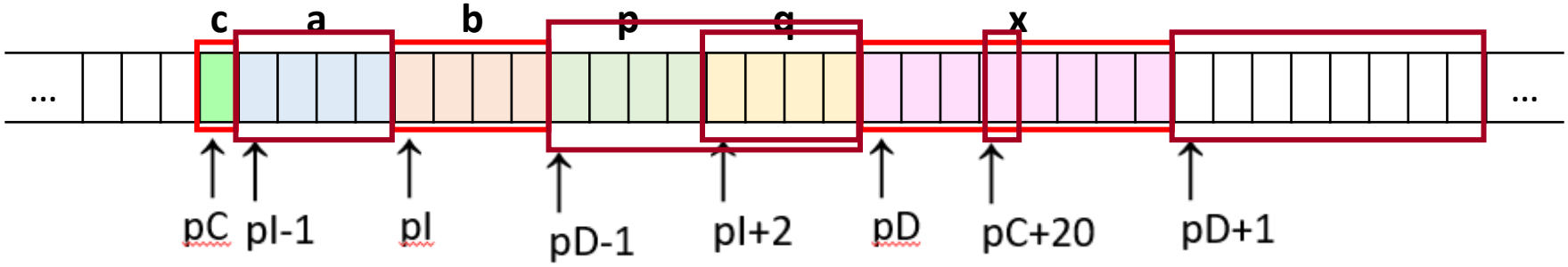
```
pI2 - pI1
```

- `-` с указателем того же типа вычисляет **смещение** – количество элементов базового типа, которое помещается между адресами, сохраненными в указателях

Операции адресной арифметики

```
double x;
int a, b, p, q;
char c;
```

```
double *pD = &x;
int *pI = &b;
char *pC = &c;
```



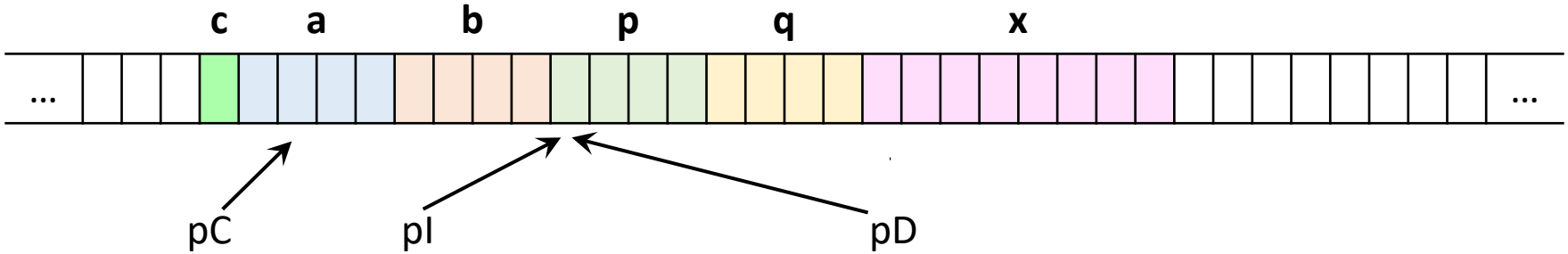
- **+** с целым числом ***k*** – вычисление адреса объекта, отстоящего от текущего адреса на ***k*sizeof*** **(базовый тип)** байт в сторону увеличения адресов. Число ***k*** - **смещение**
- **-** с целым числом ***k*** – вычисление адреса объекта, отстоящего от текущего адреса на ***k*sizeof*** **(базовый тип)** байт в сторону уменьшения адресов
- **[]** – операция индексации

```
pI[k]   ⇔   *(pI+k)
```

Операции адресной арифметики

```
double x;
int a, b, p, q;
char c;
```

```
double *pD = &x;
int *pI = &b;
char *pC = &c;
```



- **+=** с целым, **++** - **смещение указателя** на *N* в сторону увеличения адресов – увеличение значения указателя на ***N*sizeof (базовый тип)*** байт

```
pI++;
pC += 2;
pD--;
```

Неаккуратное смещение указателей приводит к ОШИБКАМ

- **-=** с целым, **--** – смещение указателя на *N* в сторону уменьшения адресов – уменьшение значения указателя на ***N*sizeof* (базовый тип)** байт

Указатели на неизменяемые данные и константные указатели

- Указатель на неизменяемое данное:

```
const базовый_тип * имя_указателя = адрес;
```

значение по адресу изменять нельзя, значение указателя изменять можно.

- Константный указатель:

```
базовый_тип * const имя_указателя = адрес;
```

значение по адресу изменять можно, значение указателя изменять нельзя.

- Константный указатель на неизменяемое данное:

```
const базовый_тип * const имя = адрес;
```

ни указатель, ни значение по адресу изменять нельзя.

Указатель на неизменяемое данное

```
const базовый_тип * имя_указателя = адрес;
```

```
const int a = 10;
int b = 1;
const int * pI = &a;
```

изменять значение переменной нельзя

изменять значение переменной можно

через указатель изменять значение переменной нельзя, а значение указателя изменять можно

```
*pI = 5;
pI = &b;
*pI = 5;
```

ОШИБКА

изменять значение указателя можно

ОШИБКА

Константный указатель

```
базовый_тип * const имя_указателя = адрес;
```

```
const int a = 10;
int b = 1;
int * const pI = &a;
```

изменять значение переменной нельзя

изменять значение переменной можно

через указатель изменять значение переменной можно, а значение указателя изменять нельзя

```
*pI = 5;
pI = &b;
```

изменять значение по адресу можно

ОШИБКА

Константный указатель на неизменяемое данное

```
const базовый_тип * const имя = адрес;
```

```
const int a = 10;
int b = 1;
const int * const pa = &a, * const pb = &b;
```

изменять значение переменной нельзя

изменять значение переменной можно

ни значение указателя, ни значение по адресу изменять нельзя

```
*pa = 5;
*pb = 5;
pa = &b;
```

ОШИБКА

ОШИБКА

ОШИБКА

Подводим итоги

- указатель – это переменная, в которой можно хранить адрес другой переменной;
- при объявлении указателя надо указать тип объектов, на которые он будет указывать, а перед именем поставить знак ***;
- знак ***&*** перед именем переменной обозначает ее адрес;
- знак ***** перед указателем в рабочей части программы (не в объявлении) обозначает значение ячейки, на которую указывает указатель;
- для обозначения недействительного указателя используется константа ***NULL*** (нулевой указатель);
- при изменении значения указателя на ***k*** он сдвигается к *k*-му следующему числу данного типа, то есть для указателей на целые числа на ***k*sizeof(int)** байт;
- указатели печатаются по формату ***%p***.

Нельзя использовать указатель, который указывает неизвестно куда!