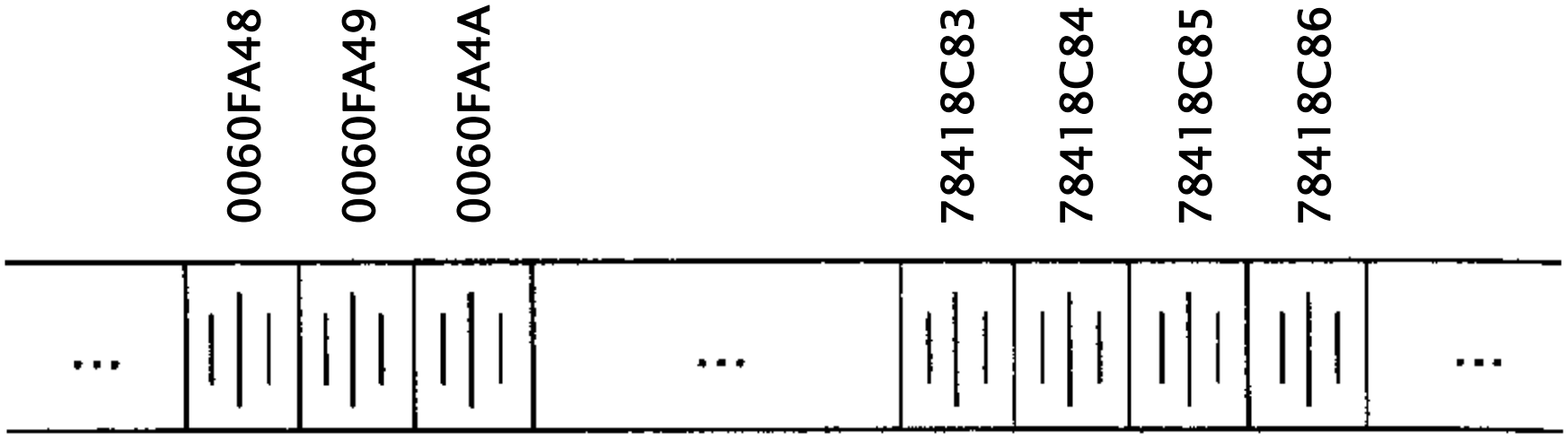
Информатика:
Основы программирования

Язык Си
Указатели

Ракова Ирина Константиновна
каф.О7

Организация оперативной памяти

Логически оперативная память представляет собой непрерывную последовательность байт.
Каждый байт имеет свой порядковый номер – адрес.
Адреса представляют собой целочисленные значения и могут быть выведены на экран, обычно в 16-ричной системе счисления.
Размер адреса зависит от платформы.



Имя и адрес переменной

Переменная в программе – это объект, имеющий имя, тип, значение и адрес. При объявлении переменной в памяти для ее хранения выделяется определенное количество байт, зависящее от типа переменной.

По имени можно обратиться к переменной и получить ее значение.

С точки зрения машинной реализации имя переменной соответствует адресу того участка памяти, который для нее выделен, а значение переменной – содержимому этого участка памяти.

В Си к переменной можно обратиться и традиционно **по имени**, и **по адресу** - через переменную-указатель.

Указатели используют:

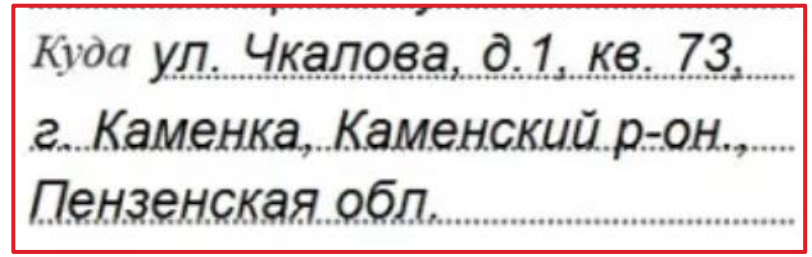
- при работе с массивами
- для передачи параметров в функции
- для работы с файлами
- для работы с динамической памятью

Указатели и объекты

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

Адреса **есть** у:
- переменных
- функций
- динамически выделенных блоков памяти
- отдельных байтов

Адресов **нет** у:
- констант
- выражений



Синонимы

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

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 void */
int * pI_1, * pI_2;  /* указатели на int */
```

Звездочку повторяем!

Неинициализированный указатель использовать нельзя!

Виды указателей

В зависимости от базового типа различают:

Типизированный указатель
указатель «знает» тип объекта, на который он указывает (размер, способ кодирования, операции)

Нетипизированный указатель, указатель на void
указатель на «нечто», «знает» только адрес

Указатель хранит в себе адрес, а адрес занимает 4 или 8 байт, в зависимости от аппаратуры, от разрядности операционной системы / компилятора. Поэтому под указатель ЛЮБОГО типа выделяется 4 или 8 байт.
Но размер адресуемой области (области, на которую указывает указатель) зависит от типа указателя.

Присвоение значения указателю

Для того, чтобы указатель стал указывать на ячейку памяти (переменную), нужно присвоить ему какой-то адрес.

int *b, a = 5, *c;

a – целочисленная переменная, b, c – указатели

b = &a;

Теперь указатель b указывает на переменную a (& - операция взятия адреса).

c = NULL;

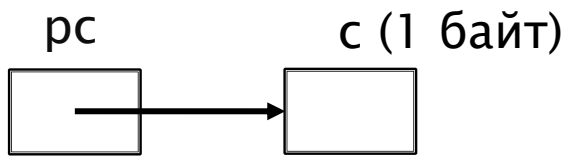
Константа NULL – нулевой адрес, не соответствующий никакому реальному адресу. По нулевому указателю не может быть ничего ни записано, ни прочитано. Нулевой указатель обычно используется для обозначения того, что указатель не указывает ни на какой объект.

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

Присвоение значения указателю

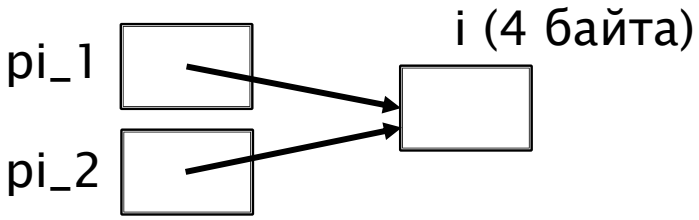
Указателю можно присвоить адрес переменной того же типа.
Указателю на void можно присвоить адрес переменной любого типа.

```
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 */
```



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

```
int i, *pi_1 = &i, *pi_2;
double d, *pd = &d;
void *p = pd;
pi_2 = pi_1;
```



Несоответствие типов указателей приводит к ОШИБКАМ

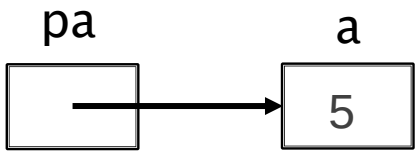
~~*pc=&d; pd=pi_1;*~~

pc=(char)(&d); pd=(double*)pi_1;*

Операция разыменования

Унарная * – операция **получения значения** по адресу или операция **разыменования**

```
int a = 5, *pa = &a;
printf ("&a=%p pa=%p \n", &a, pa);
printf ("a=%d *pa=%d \n", a,*pa);
```




```
&a=000000000022FE44 pa=000000000022FE44

a=5 *pa=5
```

Указатель *pa* содержит адрес переменной *a*, поэтому *a* и **pa* - это одно и то же значение.

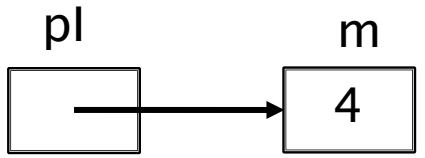
Спецификатор формата вывода адреса - %p

```
int a = 5, *pa, b;
pa = &a;
b = *pa;
```

Операция разыменования

С разыменованным значением выполняются те же операции, что и с обычно переменной того же типа.

```
int m = 4, n, *pI;
pI = &m;
printf ("m = %d\n", *pI);  /* Значение m */
printf("&m = %p\n", pI);  /* Адрес m */
printf("&pI = %p\n", &pI);  /* Адрес pI */
n = *pI * 2;     /* n = 4 * 2 = 8 */
*pI = n - 3;     /* m = 8 – 3 = 5 */
printf("m = %d\n", m);  /* Значение m */
printf("*pI = %d\n", *pI);  /* Значение m */
```

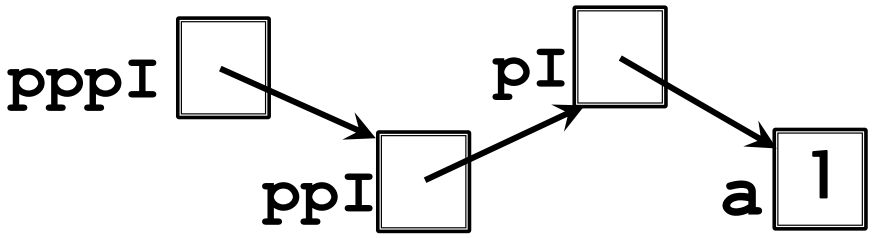



```
m = 4
&m = 000000000065FE18
&pI = 000000000065FE10
m = 5
*pI = 5
```

Операции над указателями

- **sizeof()** – размер в байтах
  Размер переменных зависит от их типа; размер указателя не зависит от типа указателя; размер области, на которую он ссылается, определяется его типом.
- **(тип)** – явное приведение типа
- ***** – получение значения по адресу (разыменование)
  Операцию разыменования нельзя применить к указателю на void без явного приведения типа (можно, например, *(float*)p).
- **&** – получение адреса
- **=** – присваивание
- **операции адресной арифметики**

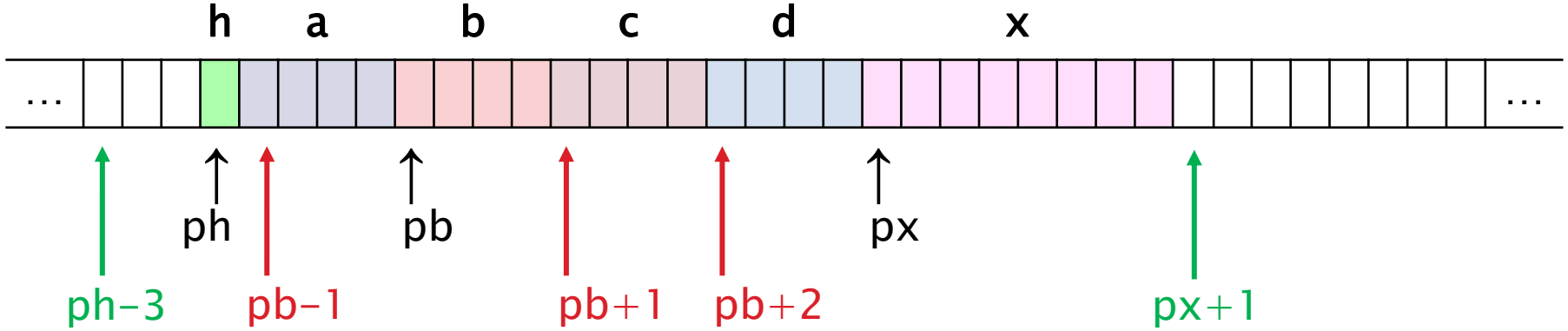
```
int a = 5, *pI = &a;
int ** ppI = &pI;
int *** pppI = &ppI;
***pppI = 1;
```



Операции адресной арифметики

```
double x;
int d, c, b, a;
char h;
```

```
double *px = &x;
int *pb= &b;
char *ph = &h;
```



Операции умножения и деления к указателям не применимы. Складывать два указателя нельзя, но можно прибавлять к указателю целое число (и вычитать).

- **+ с целым числом N** – вычисление адреса объекта, отстоящего от текущего адреса на N*sizeof (базовый тип) байт в сторону увеличения адресов
- **- с целым числом N** – вычисление адреса объекта, отстоящего от текущего адреса на N*sizeof (базовый тип) байт в сторону уменьшения адресов

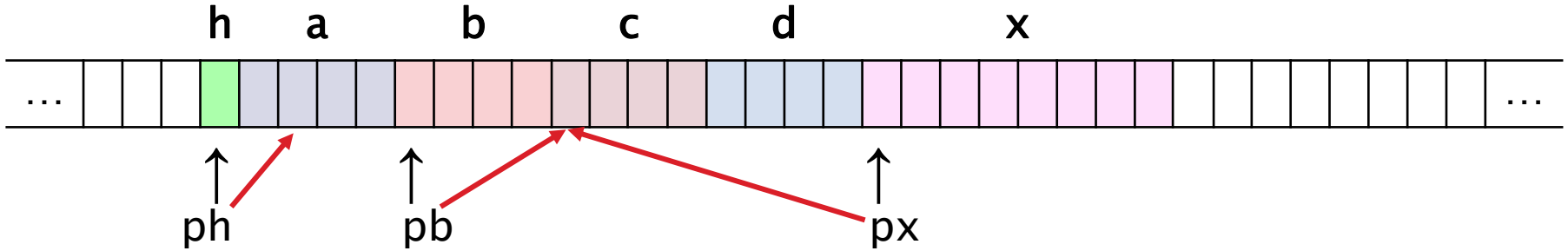
*(pb-1) *(pb+1) *(ph-3) и т.д.

Операции адресной арифметики

```
double x;
int d, c, b, a;
char h;
```

```
double *px = &x;
int *pb= &b;
char *ph = &h;
```

```
pb++;
ph += 2;
px--;
```



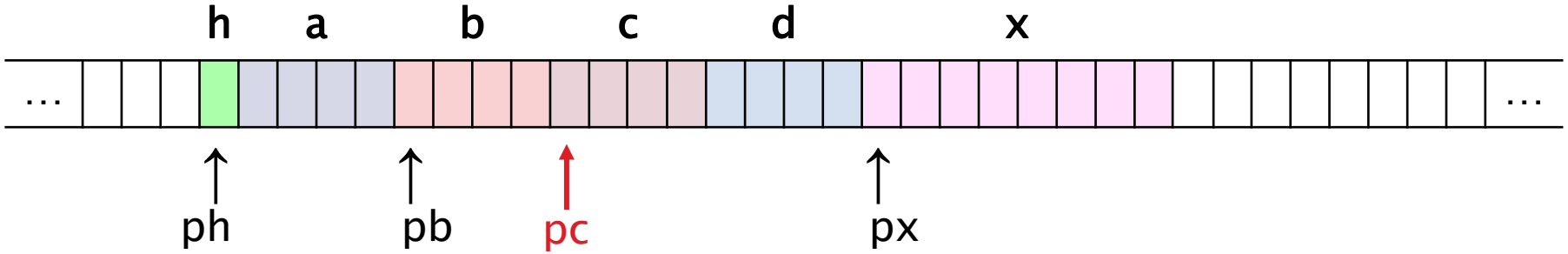
- **+= с целым N** – смещение указателя на N в сторону увеличения адресов – увеличение значения указателя на N*sizeof (базовый тип) байт, **++** - смещение на 1
- **-= с целым N** – смещение указателя на N в сторону уменьшения адресов – уменьшение значения указателя на N*sizeof (базовый тип) байт, **--** - смещение на -1

Неаккуратное смещение указателей приводит к ОШИБКАМ (например, вывод *ph, *pb, *px)

Операции адресной арифметики

```
double x;
int d, c, b, a;
char h;
```

```
double *px = &x;
int *pb = &b;
int *pc = pb + 1;
char *ph = &h;
```



- – с указателем того же типа – количество элементов базового типа, которое помещается между адресами, сохраненными в указателях *pc - pb*
- операции отношения и сравнения на равенство для указателей одного типа

pc > pb, pc == pb – сравнение адресов

**pc == *pb – сравнение значений по адресам*

Операции с указателями

```
int x=6, *p;
p = &x;
*p = *p+3;
x = ?
```

```
int x = 2, *p;
p = &x;
p++;
p = ?
```

```
int x = 2, y, *p;
p = &x;
y = *p
y = ?
```

```
int a, b, c, *p = &a;
*p = 3;
*(p+1) = 5;
*(p+2) = 7;
 p++;
*p = ?
```

```
int a =1, b = 2, c = 3;
int *pa = &a, *pb = &b, *pc = &c;
*pb = *pa + 3;
*pc = *pa;
pa = pb;
a, b, c = ?  *pa, *pb, *pc =?
```

Какая разница:
++(*p) и *(++p)

Спасибо за внимание