

Семинар №8

ФАКТ 2022

Бирюков В. А.

October 19, 2022

Хранение переменных и массивов в памяти

Система счисления по целочисленному основанию 16.

В качестве цифр этой системы обычно используются цифры от 0 до 9 и латинские буквы от A до F.

Примеры:

$$6 = 0x6$$

$$12 = 0xc$$

$$20 = 0x14$$

$$200 = 0xc8$$

$$255 = 0xff$$

$$256 = 0x100$$

$$1000 = 0x3E8$$

$$1024 = 0x400$$

Пример работы шестнадцатеричной системой в языке C:
Вместо %d (decimal) используем %x (hexadecimal)

```
#include <stdio.h>
int main()
{
    int a = 1000;
    printf("%d\n", a);
    printf("%x\n", a);
}
```

Для создания шестнадцатеричного числа нужно дописать к числу приставку 0x

```
#include <stdio.h>
int main()
{
    int a = 0x14;
    printf("%d\n", a);
    printf("%x\n", a);
}
```

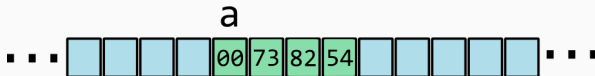
Хранение переменных типа `int` в памяти

Предположим, что мы создали переменную типа `int` в памяти и присвоили ей число 7570004.

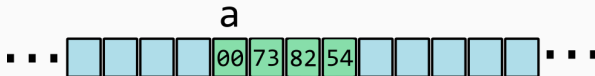
Как это число будет выглядеть в памяти?

Если это число перевести в 16-ричную систему, то получится 0x738254.

```
int a = 7570004;
```



```
int a = 0x738254;
```

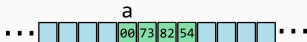


Порядок байт:

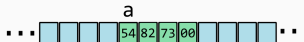
- ① Big Endian – от старшего байта к младшему.
- ② Little Endian – от младшего байта к старшему. Используется в большинстве вычислительных системах.

```
int a = 0x738254;
```

Big Endian

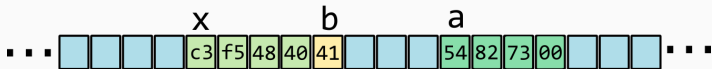


Little Endian



В дальнейшем будем предполагать, что используется порядок Little Endian


```
int a = 0x738254;  
char b = 'A';  
float x = 3.14;
```



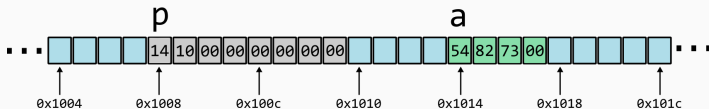
Указатель на переменную типа `int`

Каждая ячейка памяти имеет номер (адрес).

Указатель – переменная, которая хранит адреса.

```
int a = 0x738254;
```

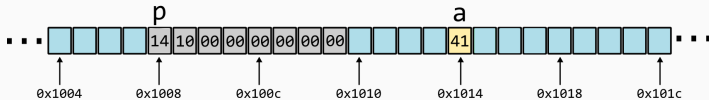
```
int* p = &a
```



В 64-х битных система размер указателя равен 64 бит (т.е. 8 байт).

Обратите внимание, что для указателя тоже используется Little Endian.

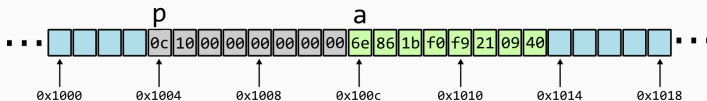
```
char a = 'A';  
char* p = &a
```



Код ASCII символа A равен 65 или 0x41.

Независимо от размера самой переменной, размер указателя равен 8 байтам.

```
double a = 3.14159;  
double* p = &a
```

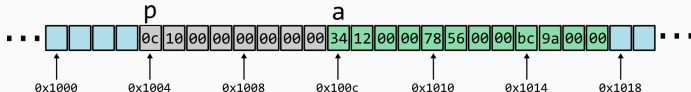


Представление чисел с плавающей точкой в памяти задаётся стандартом IEEE 754.

Шестнадцатиричное представление 3.14159 по этому стандарту:
0x400921f9f01b866e.

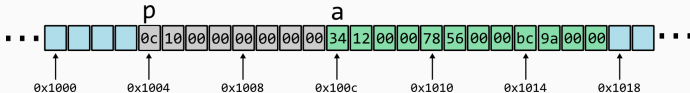
Элементы массива лежат в памяти последовательно, без зазоров.
Порядок байт – обратный, но только в рамках одного базового типа.

```
int a[3] = {0x1234, 0x5678, 0x9abc};  
int* p = &a[0];
```



При присваивание указателю или при передаче в функцию название массива ведёт себя как указатель на первый элемент (т. е. `a == &a[0]`)

```
int a[3] = {0x1234, 0x5678, 0x9abc};  
int* p = a;
```



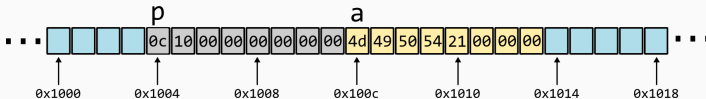
Указатель на элемент массива char-ов (строку)

Символ М имеет код ASCII равный 77 = 0x4d.

Остальные символы:

I (0x49), P (0x50), T(0x54) !(0x21)

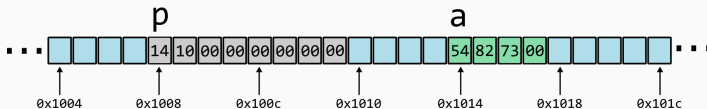
```
char a[8] = "MIPT!";  
char* p = &a[0];
```



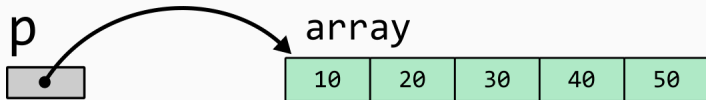
Схематическое изображение указателя

Так как постоянно рисовать память и адреса затратно, то будем использовать схематическое изображение обычных переменных (в виде прямоугольника) и указателей (в виде прямоугольничка со стрелочкой)

```
int a = 0x738254;  
int* p = &a
```



Схематическое изображение указателя на элемент массива



Так как $4660 = 0x1234$, $22136 = 0x5678$ и $39612 = 0x9abc$.

Арифметика указателей

```
int array[5] = {11, 12, 13, 14, 15};  
int* p = &array[0];  
int* q = &array[3];
```

- Прибавление/вычитание целого числа

```
p += 2;
```

- Вычитание двух указателей

```
int n = q - p;
```

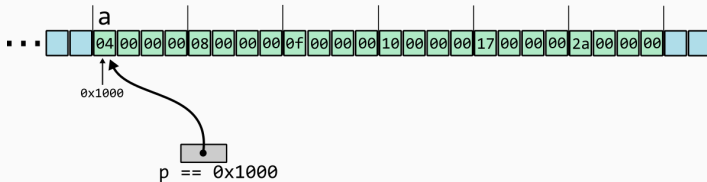
- Разыменование

```
int x = *p;
```

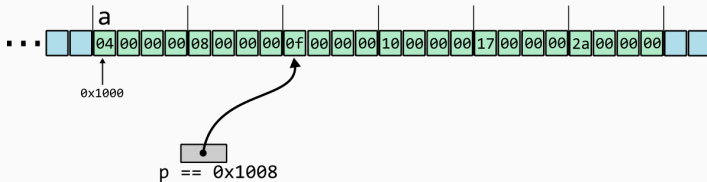
- Квадратные скобки

```
int x = p[2];
```

```
int a[6] = {4, 8, 15, 16, 23, 42};
int* p = &a[0];
```

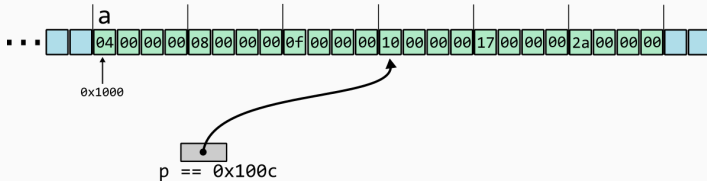


```
int a[6] = {4, 8, 15, 16, 23, 42};  
int* p = &a[2];
```



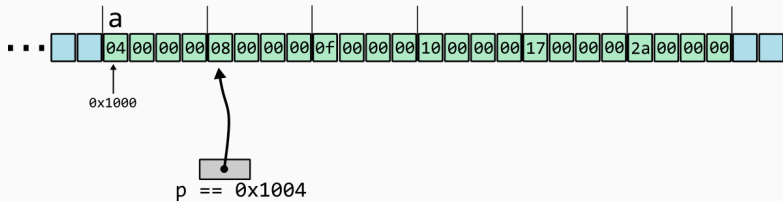
При прибавлении целого числа x к указателю типа `int*`, указатель увеличивается на $x * \text{sizeof}(\text{int})$

```
int a[6] = {4, 8, 15, 16, 23, 42};  
int* p = &a[0];  
p += 3;
```



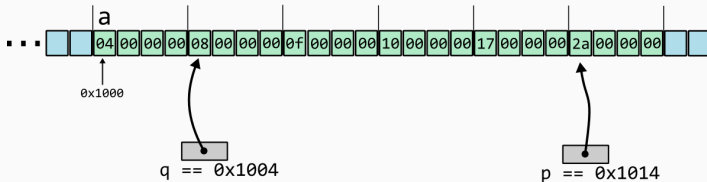
При прибавлении к `p` числа 3, указатель увеличился на 12.

```
int a[6] = {4, 8, 15, 16, 23, 42};
int* p = &a[3];
p = p - 2;
```



При вычитании указателей, результат делится на размер типа на который они указывают.

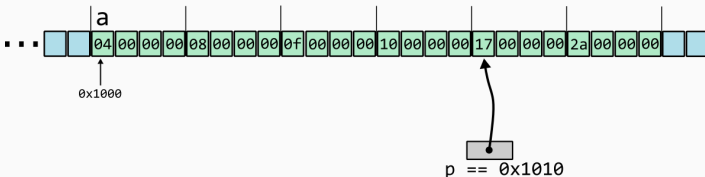
```
int a[6] = {4, 8, 15, 16, 23, 42};  
int* p = &a[5];  
int* q = &a[1];  
printf("%d\n", p - q);
```



$0x1014 - 0x1004 = 0x10 = 16$, но $p - q = 4$.

Разыменование означает – пойдти по адресу `p` и воспринимай 4 байта по этому адресу как переменную типа `int`

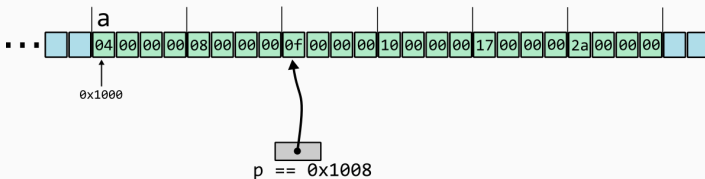
```
int a[6] = {4, 8, 15, 16, 23, 42};  
int* p = &a[4];  
printf("%d\n", *p);
```



Скобочки заменяются на разыменование:

$$p[i] = *(p + i)$$

```
int a[6] = {4, 8, 15, 16, 23, 42};
int* p = &a[2];
printf("%d\n", p[3]);
```



- Приведение типов int и float:

```
float x = 5.2;  
int y = (int)x;           // явное  
int z = x;                // неявное
```

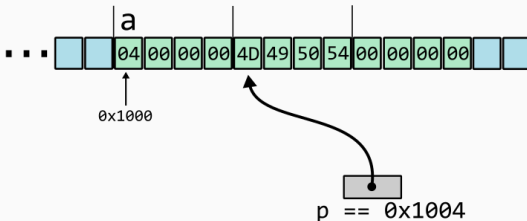
- Верно и для указателей:

```
float a;  
float* pf = &a;  
int* pi1 = (int*)pf;      // явное  
int* pi2 = pf;            // неявное, нет в C++
```

Пример приведения указателей

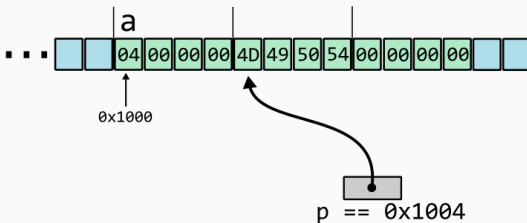
Тут использовано, что $1414547789 = 0x5450494d$

```
int a[3] = {4, 1414547789, 0};  
char* p = (char*)&a[1];  
printf("%s\n", p);
```



Что напечатает этот кусок кода?

```
int a[3] = {4, 1414547789, 0};  
char* p = (char*)&a[1];  
printf("%s\n", p + 1);
```



Что напечатает этот кусок кода?

- Указатель, который не связан ни с каким типом

```
void* p;
```

- Его нельзя разыменовывать и применять другие операции
- Но его можно привести к другому указателю:

```
int a = 10;  
void* p = (void*)&a;  
*((int*)p) += 5;
```

- Используется если нужно передать переменную, но её тип на момент написания кода неизвестен

Сегменты памяти

- Операционной системой предоставляется процессу адресное пространство размером 2^{64} байт.
- Это пространство делится на сегменты
- При попытке чтения/записи ячейки памяти недоступного для записи сегмента возникает ошибка `SegmentationFault`
- Для каждого процесса область памяти независима от других процессов. Т.е. нельзя случайно повредить память другого процесса.

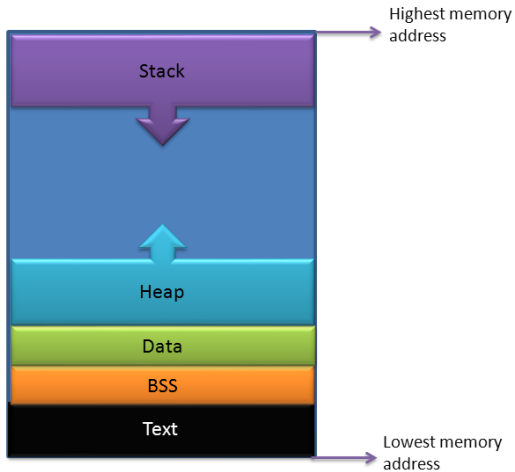


Figure : Process memory organization

- В этом сегменте создаются все переменные и статические массивы

```
int a;  
float b[10];
```

- Имеет небольшой размер, определяемый операционной системой (обычно не более 10 мегабайт)
- Быстрее выделяется память, чем в Куче
- Память выделяется в начале выполнения функции(под все локальные переменные) и освобождается при завершении функции

- В этом сегменте выделяется динамическая память

```
int* p = malloc(100); // Выделим 100 байт
```

- Имеет размер, ограниченный только свободной оперативной памятью
- Медленней выделяется память, чем в Стеке
- Память выделяется с помощью функции `malloc` или аналогов и освобождается с помощью функции `free` в любом месте программы.