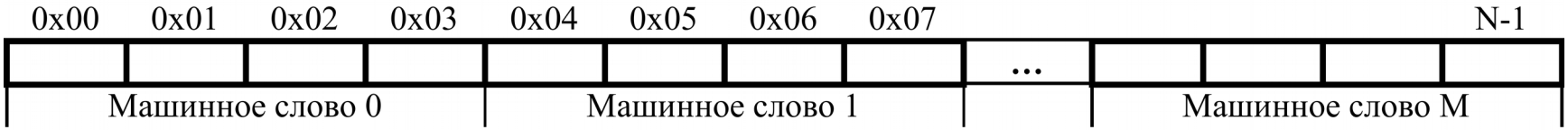
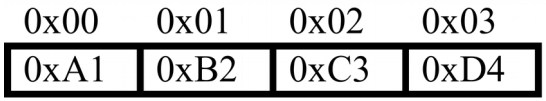
Указатели. Адресная арифметика.

Организация памяти

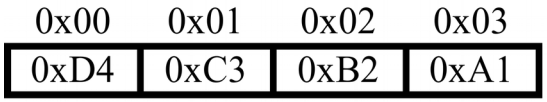


Способы представления машинного слова (значение слова 0xA1B2C3D4)

Big-endian



Little-endian



Указатели

Указатель – это объект, содержащий адрес объекта или функции, либо выражение, обозначающее адрес объекта или функции. [Жешке Р. «Толковый словарь стандарта языка Си»]

int a = 2, *p = &a; // p – указатель-объект, &a – указатель-выражение

Разновидности указателей

- типизированный указатель на данные (тип* имя);
- бестиповой указатель (void* имя);
- указатель на функцию.

Использование указателей

- Передача параметров в функцию
  - Изменяемые параметры
  - «Объемные» параметры

- Обработка областей памяти
  - Динамическое выделение памяти
  - Ссылочные структуры данных
  - ...

Указатели (базовые операции)

При определении переменной-указателя перед ее именем должен размещаться символ ‘*’.

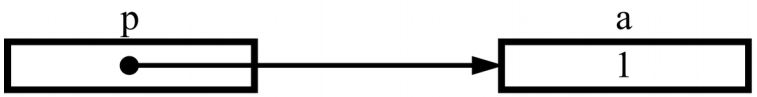
Для определения адреса данных используется операция ‘&’ (операция взятия адреса, англ. address operator, C99 6.5.3.2).

Для доступа к данным, на которые указывает указатель, используется операция ‘*’ (операция разыменования, англ. indirection operator , C99 6.5.3.2).

Операция	Название	Нотация	Класс	Приоритет	Ассоциат.
`&`	Адрес	`&X`	Префиксная	15	Справа налево
`*`	Разыменование	`*X`			

Пример использования базовых операций

```
int a = 1;
int *p = &a;                    // p теперь указывает на a
```




```
a = 3;
// операция разыменования используется для чтения
printf("%d %d\n", a, *p);      // 3 3

*p = 5;
// операция разыменования используется для записи
printf("%d %d\n", a, *p);      // 5 5

printf("%p\n", p);             // 0022ff18
```

Какие переменные являются указателями?

```
int* a, b;
```

Чему равно значение выражения?

```
int *p;
double *q;

sizeof(p) == sizeof(q)    // истина или ложь?
```

Операции ‘&’ и ‘*’ взаимно обратные.

```
int d = 5, *p = &d;

*&d = = = d    // C99, 6.5.3.2
&*p = = = p    // C99, 6.5.3.2
```

На одни и те же данные могут указывать несколько указателей

```
int a = 1;
int *p = &a;   // p теперь указывает на a
int *q = p;    // q теперь указывает туда же куда и p, т.е. на a
```

Типичная ошибка – использование неинициализированного указателя

```
int *p;

printf("%d\n", *p);    // ОШИБКА времени выполнения

*p = 1;                // ОШИБКА времени выполнения
```

Инициализация указателей

- Адресом переменной того же типа

```
int a = 1;
double d = 5.0;
int *p = &a;
// error: initialization from incompatible pointer type
int *p_err = &d;
```

- Значением другого указателя того же типа

```
int *q = p;
// error: initialization from incompatible pointer type
double *q_err = p;
```

- Значением NULL

```
int *r = NULL;
int *r_ok = 0;
// error: initialization makes pointer from integer without
// a cast
int *r_err = 12345;
```

const и указатели (1)

Указатель на константу

```
int a = 5;
int b = 7;
const int *p = &a;      // <-

printf("%d\n", *p);
*p = 4;  // error: assignment of read-only location '*p'
p = &b;
```

Константный указатель

```
int a = 5;
int b = 7;
int * const p = &a;     // <-

printf("%d\n", *p);
*p = 4;
p = &b; // error: assignment of read-only variable 'p'
```

const и указатели (2)

Константный указатель на константу

```
int a = 5;
int b = 7;
const int *const p = &a;    // <-

printf("%d\n", *p);
*p = 4; // error: assignment of read-only location '*p'
p = &b; // error: assignment of read-only variable 'p'
```

Задача, которую любят давать на собеседованиях: что есть что?

```
const int *p;
int const *q;
int * const r;
```

Указатели и массивы

Результат выражения, состоящего из имени массива, представляет собой адрес области памяти, выделенной под этот массив (англ. “array decay to pointer”, C99 6.3.2.1 #3) .

```
int a[10], *pa;
`
pa = a;                 // pa = &a[0]; но НЕ pa = &a;
```

pa[0] = = = a[0]

pa[1] = = = a[1]

…

pa[9] = = = a[9]

«Преобразование» массива в указатель

Исключение 1: операция sizeof для массива

```
int a[10];
int *pa = a;

printf("sizeof(a) = %d\n", sizeof(a));
// sizeof(a) = 40

printf("sizeof(pa) = %d\n", sizeof(pa));
// sizeof(pa) = 4
```

«Преобразование» массива в указатель

Исключение 2: массив – операнд операции получения адреса

```
int a[10];
int *pa;

printf("a = %p, &a = %p\n", a, &a);
// a = 0022fefff0, &a = 0022fefff0
```

&a возвращает указатель, НО тип этого указателя - указатель на массив (не указатель на целое)

```
pa = &a;
// warning: assignment from incompatible pointer type [enabled by default]
```

«Преобразование» массива в указатель

Исключение 3: строковый литерал-инициализатор массива char[]

```
char a[] = “abcdef”;

printf(“sizeof(a) = %d\n”, sizeof(a));
// sizeof(a) = 7
```

Отличие массивов и указателей

- При определении массива и указателя под соответствующие переменные выделяется разное количество памяти

- Выражению из имени массива нельзя присвоить другое значение

Передача массива в функцию

Любое похожее на массив объявление параметра функции рассматривается компилятором как указатель.

```
#define N 10

int f_1(int a[N])        int f_2(int a[])        int f_3(int *a)
{                        {                       {
}                        }                       }
```

Аргументом может быть любой (!) одномерный массив соответствующего типа.

```
int b[25];

f_1(b);   // f_2(b); или f_3(b);
```

Передача массива в функцию

Функция не может узнать размер массива.

```
void f_1(int a[N])
{
  printf("%d\n",
         sizeof(a));
  // 4
}
```

```
void f_2(int a[])
{
  printf("%d\n",
         sizeof(a));
  // 4
}
```

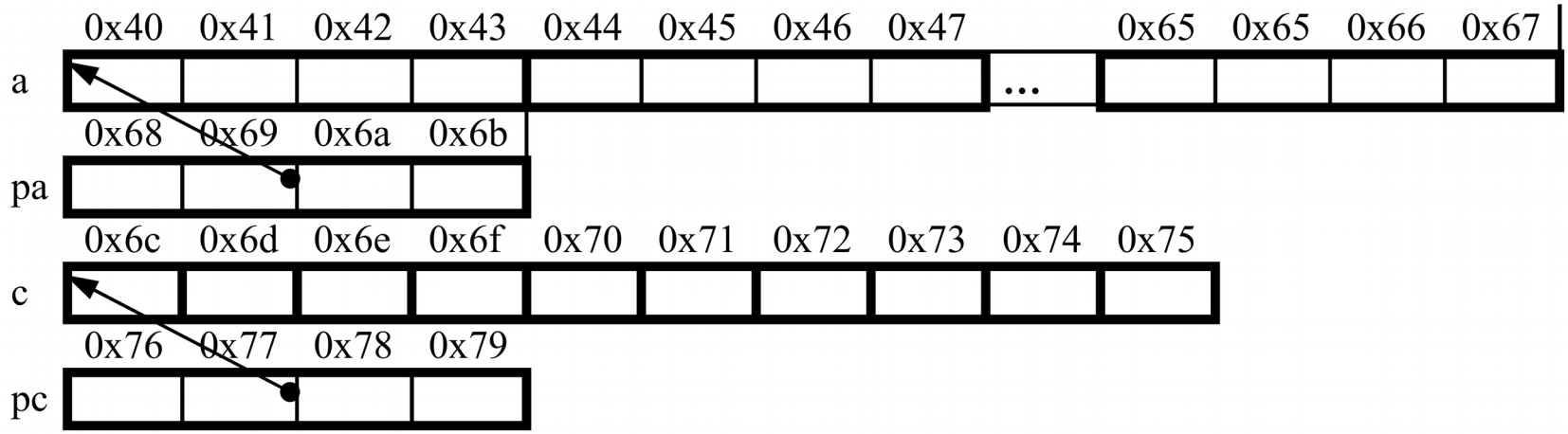
```
void f_3(int *a)
{
   printf("%d\n",
          sizeof(a));
   // 4
}
```

Передается параметр, который содержит количество элементов в массиве.

```
void f_2(int a[], int n);

void f_3(int *a, int n);
```

Сложение указателя с числом

```
int a[10];
int *pa = a;
char c[10];
char *pc = c;

printf("%p %p\n", pa, pa + 1);     // ? ?

printf("%p %p\n", pc, pc + 1);     // ? ?
```



Сложение указателя с числом

```
тип *p = …;

p += n;
```

новый адрес в p = старый адрес из p + n * sizeof(тип)

```
p -= m;
```

новый адрес в p = старый адрес из p - m * sizeof(тип)

```
int a[10];
```

&a[i] = = = a + i
Выражение a[i] компилятор заменяет на *(a + i)

Сравнение указателей

Указатели допускается использовать в операциях сравнения.

При сравнении указателей сравниваются адреса.

При этом можно

- сравнивать указатель с NULL;
- сравнивать два однотипных указателя.

Пример использования

```
#include <stdio.h>

#define N 10

float sum_1(const float* a, int n)
{
    float sum = 0.0;

    for (int i = 0; i < n; i++)
        sum += a[i];

    return sum;
}

float sum_2(const float* pb, const float* pe)
{
    float sum = 0.0;

    for (const float* pcur = pb; pcur < pe; pcur++)
        sum += *pcur;

    return sum;
}
```

Пример использования

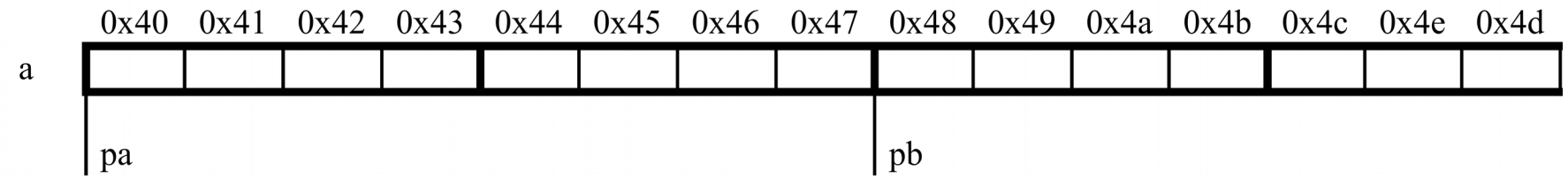
```
float sum_3(const float* pb, const float* pe)
{
    float sum = 0.0;
    const float* pcur = pb;

    while (pcur < pe)
        sum += *pcur++;

    return sum;
}

float sum_4(const float* pb, const float* pe)
{
    float sum = 0.0;

    while (pb < pe)
    {
        sum += *pb;
        pb++;
    }

    return sum;
}
```

Пример использования

```
int main(void)
{
    float a[] = {1.0, 2.0, 3.0, 4.0, 5.0};
    size_t n = sizeof(a) / sizeof(a[0]);

    printf("1. %4.2f\n", sum_1(a, n));
    printf("2. %4.2f\n", sum_2(a, a + n));
    printf("3. %4.2f\n", sum_3(a, a + n));
    printf("4. %4.2f\n", sum_4(a, a + n));

    return 0;
}
```

Вычитание указателей

```
int a[10];
int *pa = a, *pb = &a[2]; // int *pa = a, *pb = a + 2;

printf("%?", pb - pa);    // какой формат?, какое значение?
```



Вычитание указателей

```
тип a[10];
тип *pa = a, *pb = &a[2];
```

pb – pa = (адрес из pb – адрес из pa) / sizeof(тип)

Пример использования

```
int str_len(const char *str)
{
    const char *pbeg = str, *pend = str;

    while (*pend)
        pend++;

    return pend - pbeg;
}
```

Операции с указателями

- Получение адреса переменной
- Разыменование
- Присваивание однотипных указателей
- Сложение/вычитание с целым числом
- Сравнение однотипных указателей
- Вычитание двух однотипных указателей

void*

Тип указателя void* используется, если тип объекта неизвестен.

- позволяет передавать в функцию указатель на объект любого типа;
- полезен для ссылки на произвольный участок памяти, независимо от размещенных там объектов.

Особенности использования void*

- В языке С допускается присваивание указателя типа void* указателю любого другого типа (и наоборот) без явного преобразования типа указателя.

```
double d = 5.0;
double *pd = &d;
void *pv = pd;

pd = pv;
```

- Указателя типа void* нельзя разыменовывать.
- К указателям типа void* не применима адресная арифметика.