

Семинар #9: Память. Классные задания.

Часть 1: Системы счисления

Мы привыкли пользоваться десятичной системой счисления и не задумываемся, что под числом в десятичной записи подразумевается следующее:

$$123.45_{10} = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 + 4 \cdot 10^{-1} + 5 \cdot 10^{-2}$$

Конечно, в числе 10 нет ничего сильно особенного с математической точки зрения. Оно было выбрано исторически, скорее всего по той причине, что у человека 10 пальцев. Компьютеры же работают с двоичными числами, потому что оказалось, что процессоры на основе двоичной логики сделать проще. В двоичной системе счисления есть всего 2 цифры: 0 и 1. Под записью числа в двоичной системе подразумевается примерно то же самое, что и в десятичной:

$$101.01_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = 5.25_{10}$$

При работе с компьютером на низком уровне имеет смысл использовать двоичную систему за место десятичной. Но человеку очень сложно воспринимать числа в двоичной записи, так как они получаются слишком длинными. Поэтому популярность приобрели восьмеричная и шестнадцатеричная системы счисления. В шестнадцатеричной системе счисления есть 16 цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f.

$$1a.8_{16} = 1 \cdot 16^1 + 10 \cdot 16^0 + 8 \cdot 16^{-1} = 26.5_{10}$$

Задача. Переводите следующие числа в десятичную систему:

$$- 11011_2$$

$$- 2b_{16}$$

$$- 40_8$$

$$- 1.1_2$$

$$- a.c_{16}$$

$$- 10_{123}$$

Шестнадцатеричная и восьмеричная системы в языке C:

Язык C поддерживает шестнадцатеричные и восьмеричные числа. Чтобы получить восьмеричное число нужно написать 0 перед числом. Чтобы получить шестнадцатеричное число нужно написать 0x перед числом.

```
#include <stdio.h>
int main() {
    int a = 123;    // Десятичная система
    int b = 0123;   // Восьмеричная система
    int c = 0x123;  // Шестнадцатеричная система
    printf("%i %i %i\n", a, b, c);
}
```

Также, можно печатать и считывать числа в этих системах счисления с помощью спецификаторов %o (для восьмеричной системы – octal) и %x (для шестнадцатеричной – hexadecimal). Спецификатор %d можно использовать для десятичной системы – decimal. Пример программы, которая считывает число в шестнадцатеричной системе и печатает в десятичной:

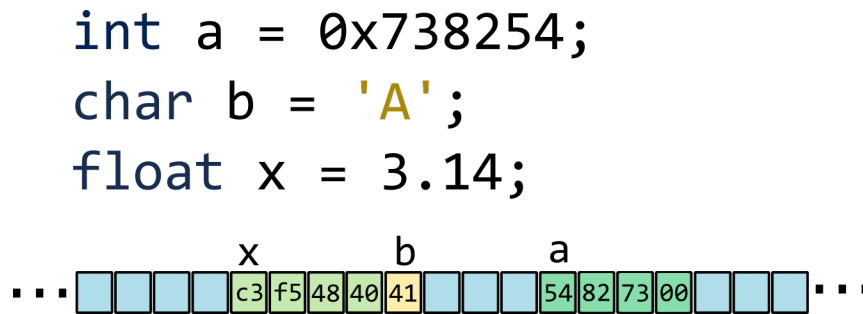
```
#include <stdio.h>
int main() {
    int a;
    scanf("%x", &a);
    printf("%d\n", a);
}
```

Задача:

Напишите программу, которая будет считывать число в десятичной системе и печатать в шестнадцатеричной. Переведите с помощью этой программы в шестнадцатеричную систему числа: 14, 255, 256, 65535, 14598366.

Часть 2: Переменные в памяти

Положение любой переменной в памяти характеризуется двумя числами: её адресом(номером первого байта этой переменной) и её размером. Рассмотрим ситуацию, когда были созданы 3 переменные типов `int` (размер 4 байта), `char` (размер 1 байт) и `float` (размер 4 байта). На рисунке представлено схематическое расположение этих переменных в памяти (одному квадратику соответствует 1 байт):



Какие выводы можно сделать из этого изображения:

- Каждая переменная заняла столько байт, чему равен её размер.
- Переменные в памяти могут храниться не в том порядке, в котором вы их объявляете.
- Переменные в памяти хранятся не обязательно вплотную друг к другу.
- Каждый байт памяти представляется двузначным шестнадцатиричным числом (это удобно).
- Байты переменной `a` хранятся в обратном порядке. Такой порядок байт называется **Little Endian**. Обратите внимание, что обращается только порядок байт, а не бит. Большинство компьютеров применяют именно такой порядок байт. Но в некоторых системах может использоваться обычный порядок байт – **Big Endian**. Обратный порядок байт применяется не только к типу `int`, но и ко всем базовым типам.
- Переменная `b` хранит ASCII-код символа `A`. Он который равен $65 = 41_{16}$.

Каждый байт памяти занумерован и номер первого байта переменной называется её адресом. В 64-х битных системах адрес – это 64-битное число и может принимать значения от 0 и до `0xffffffffffffffff`. Это адресное пространство предоставляется операционной системой и не соответствует адресному пространству физической памяти. Поэтому, например, адрес переменной может быть больше чем общее количество оперативной памяти.

Операторы получения адреса и размера переменных

Чтобы найти адрес переменной, нужно перед ней поставить знак амперсанда `&`. Для печати адреса используется спецификатор `%p`, хотя можно использовать и `%llx` или `%llu`. Чтобы найти размер переменной нужно использовать оператор `sizeof`. Размер это обычно тоже 8-ми байтовое беззнаковое число (потому что размер какой-нибудь структуры может быть очень большим), поэтому нужно использовать `%llu`.

```
#include <stdio.h>
int main() {
    int a = 123;
    printf("Address = %p\n", &a);
    printf("Size = %llu\n", sizeof(a));
}
```

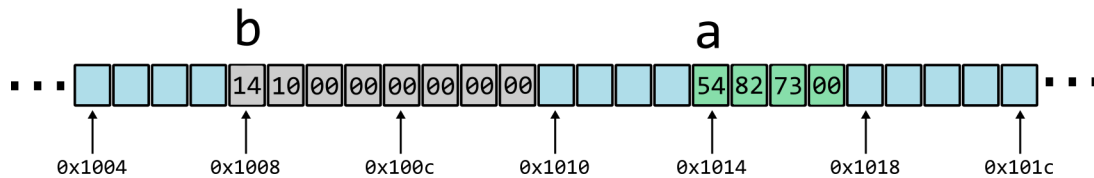
Задачи:

- Найдите чему равны размеры следующих типов на вашей системе: `char`, `short`, `long`, `long long`, `float`, `double`, массив чисел `int` размером 100 элементов.
- Объявите 2 переменные `int` и напечатайте их адреса в шестнадцатеричном и десятичном виде. Чему равна разница между этими числами? Убедитесь, что адреса переменных разные при каждом запуске программы.

Часть 3: Указатели

Для хранения адресов в языке C введены специальные переменные, которые называются указатели. Тип переменной указателя = тип той переменной, чей адрес он хранит + звёздочка на конце. Например, указатель, который будет хранить адреса переменных типа `int` должен иметь тип `int*`.

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



Пояснения по рисунку:

- В данном примере для простоты выбраны очень маленькие адреса. В действительности же адрес скорее всего будет очень большим числом.
- Указатель тоже является переменной и хранится в памяти.
- Указатель хранит номер одной из ячеек памяти (в данном случае – первый байт `a`).
- Для указателя применяется тот же порядок байт, что и для других переменных базовых типов. В данном случае – обратный.

Задачи:

- Для примера выше напечатайте следующие величины:
 - Значение, адрес и размер переменной `a`
 - Значение, адрес и размер переменной `b` (указатель, хранящий адрес `a`)
- Напечатайте размеры следующих типов: `char*`, `short*`, `int*`, `long long*`, `float*`, `double*`, `int**`.

Операция разыменования:

Разыменования – это получение самой переменной по указателю на неё. Чтобы разыменовать указатель нужно перед ним поставить звёздочку. Не следует путать эту звёздочку со звёздочкой, используемой при объявлении указателя. То есть, если `b` это указатель, хранящий адрес `a`, то `*b` означает следующее:

Пройди по адресу, хранящемуся в `b`. Возьми соответствующее количество байт, начиная с этого адреса. Воспринимай эти байты как переменную соответствующего типа.

```
#include <stdio.h>  
int main() {  
    int a = 123;  
    int* b = &a;  
    *b += 1;  
    printf("%d\n", a);  
}
```

Задачи:

- Умножьте значение переменной `a` на 10 используя только указатель на неё.
- Возведите значение переменной `a` в квадрат используя только указатель на неё.

Схематическое изображение указателей в памяти:

Так как постоянно рисовать переменные в памяти слишком громоздко и затруднительно, будем изображать их схематически. Стрелочкой будем указывать на переменную, адрес которой хранит указатель. Размеры прямоугольников не соответствуют размерам переменных. Пример выше тогда будет выглядеть так:



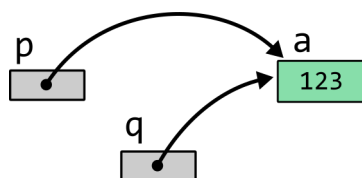
Задачи:

Напишите код, который будет соответствовать следующим рисункам. В каждой задаче размыните указатели и напечатайте то, на что они указывают.

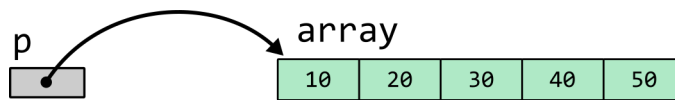
- Указатель на переменную типа `char`.



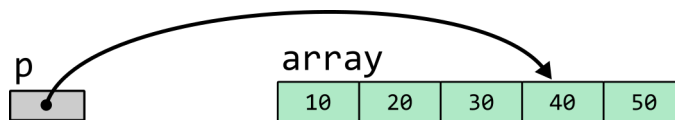
- Два указателя, которые указывают на одну переменную типа `int`



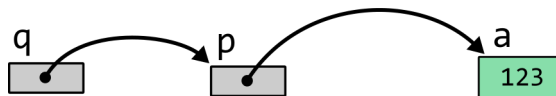
- Указатель типа `int*`, указывает на первый элемент массива `int`-ов под названием `array`



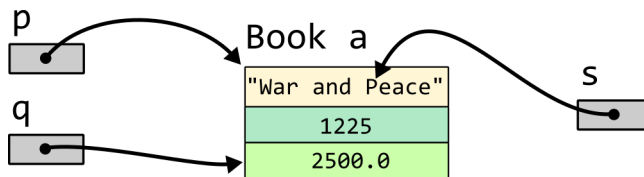
- Указатель типа `int*`, указывает на четвёртый элемент массива `int`-ов под названием `array`



- Указатель типа `int**`, указывает на указатель `int*`, который указывает на переменную типа `int`.



- Пусть есть структура `Book` из задания на структуры. Она содержит поля `title` (массив `char`), `pages` (тип `int`) и `price` (тип `float`). Создайте переменную структуры и 3 указателя `p`, `q` и `s`. Указатель `p` должен указывать на саму структуру. Указатель `q` должен указывать на поле `price`. Указатель `s` должен указывать на символ 'P' строки "War and Peace".



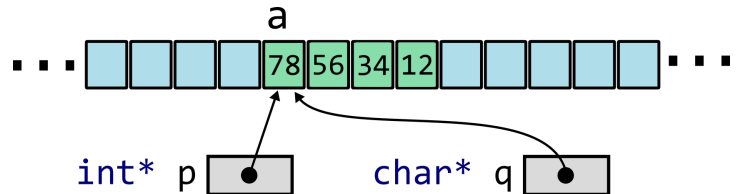
Часть 4: Указатели разных типов

Как вы могли заметить тип указателя зависит от типа элемента на который он указывает. Но все указатели, независимо от типа, по сути хранят одно и то же (адрес первого байта переменной). Чем же они различаются друг от друга? Разница проявляется как раз при их разыменовывании. Например, при разыменовывании указатель `int*` берёт 4 байта и воспринимает их как переменную типа `int`, а указатель `char*` берёт 1 байт и воспринимает его как переменную типа `char`.

Рассмотрим следующий пример. На переменную `a` указывают две переменные разных типов: `int*` и `char*`. Оба указателя хранят одно и то же значение, но работают по разному при разыменовывании.

```
#include <stdio.h>
int main() {
    int a = 0x12345678;
    int* p = &a;
    char* q = &a;

    printf("%p %p\n", p, q);
    printf("%x\n", *p);
    printf("%x\n", *q);
}
```



Преобразование типов указателя

В предыдущем примере есть такая строка `char* p = &a;`. Необычность этой строки в том, что слева и справа от знака `=` находятся объекты разных типов. Слева – `char*`, а справа – `int*`. В этот момент происходит неявное преобразование типов один тип указателя преобразуется в другой. Это всё похоже на преобразование типов обычных переменных.

```
int a = 4.1;           // Неявное преобразование из double в int
int b = (int)4.1;      // Явное преобразование из double в int

char* p = &a;          // Неявное преобразование из int* в char* ( не работает в C++ )
char* p = (char*)&a;    // Явное преобразование из int* в char*
```

Надо отметить, что язык C++ строже относится к соблюдению типов, чем язык C, и не позволит вам неявно преобразовать указатель одного типа в указатель другого типа.

Задача: Что напечатает следующая программа и почему она это напечатает?

```
#include <stdio.h>
int main() {
    int a = 7627075;
    char* p = (char*)&a;
    printf("%s\n", p);
}
```

Указатель void*

Помимо обычных указателей в языке есть специальный указатель `void*`. Этот указатель не ассоциирован не с каким типом, а просто хранит некоторый адрес. При попытке его разыменовывания произойдёт ошибка.

Задача:

```
int a = 123;
void* p = (void*)&a;
```

Увеличьте переменную `a` в 2 раза и напечатайте её используя только указатель `p`.

Часть 5: Арифметика указателей

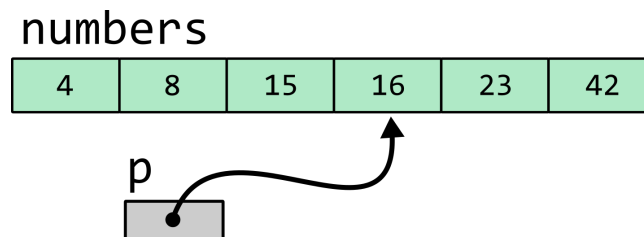
С указателями можно производить следующие операции:

- Разыменование `*p`
- Инкремент `p++`. В этом случае указатель не увеличивается на 1, как было можно подумать. Он увеличивается на размер типа, на который он указывает. Благодаря этой особенности указателей с их помощью удобно проходить по массиву.
- Декремент `p--`. Уменьшается на размер типа, на который он указывает.
- Прибавить или отнять число `p + k`. В этом случае указатель не увеличивается на `k`, как было можно подумать. Он увеличивается на `k * sizeof(*p)`. Благодаря этой особенности указателей с их помощью удобно проходить по массиву. Если `p` указывает на `i`-ый элемент массива, то `p + 1` будет указывать на `i + 1` элемент массива.
- Вычитать 2 указателя `p - q`. Вернётся разница между указателями делённая на размер типа указателя.
- Квадратные скобки (прибавить число + разыменование): `p[i] == *(p+i)`

Задачи:

- Пусть есть одномерный статический массив и указатель на 4-й элемент этого массива:

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



Чему равны следующие выражения:

- | | | |
|----------------------------|--------------------------|--|
| 1. <code>numbers[5]</code> | 5. <code>p[0]</code> | 9. <code>*(numbers+5)</code> |
| 2. <code>*p</code> | 6. <code>p[1]</code> | 10. <code>p - numbers</code> |
| 3. <code>*(p+1)</code> | 7. <code>p[-2]</code> | 11. <code>(short*)p - (short*)numbers</code> |
| 4. <code>*(p-2)</code> | 8. <code>*numbers</code> | 12. <code>(char*)p - (char*)numbers</code> |

Подсказка: имя массива во многих случаях ведёт себя как указатель на первый элемент массива.

- Обход массива с помощью указателя:

```
#include <stdio.h>  
int main() {  
    int numbers[6] = {4, 8, 15, 16, 23, 42};  
    for (int* p = &numbers[0]; p != &numbers[6]; ++p) {  
        printf("%i\n", *p);  
    }  
}
```

Используйте такой обход, но с указателем `char*`, чтобы напечатать каждый байт массива `numbers` в шестнадцатеричном виде.

Передача переменных в функции с помощью указателей

Передача массивов в функции

Стандартные функции `memcpy` и `memset`

Замечания о синтаксисе объявления указателей