Семинар #6: Указатели.

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, скорей 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 шестнадцатиричная системы счисления. В шестнадцатиричной 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}

Шестнадцатиричная и восьмеричная системы в языке `C`:

Язык `C` поддерживает шестнадцатиричные и восьмеричные числа. Чтобы получить восьмеричное число нужно написать `0` перед числом. Чтобы получить шестнадцатиричное число нужно написать `0x` перед числом.

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

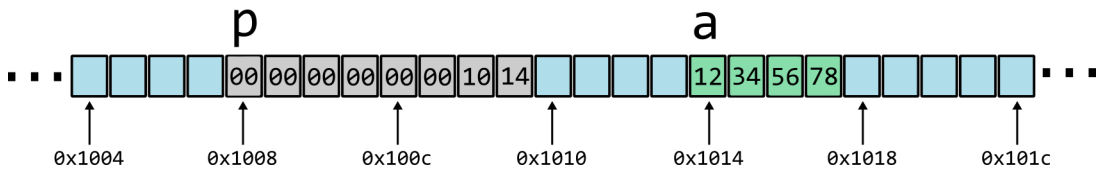
Также, можно печатать и считывать числа в этих системах счисления с помощью спецификаторов `%o` (для восьмеричной системы – **o**ctal) и `%x` (для шестнадцатеричной – he**x**adecimal). Спецификатор `%d` можно использовать для десятичной системы – **d**ecimal. Пример программы, которая считывает число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

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

Часть 2: 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 = 0x12345678;
int* p = &a
```



Пояснения по рисунку:

- Числа, начинающиеся с `0x` – это числа в шестнадцатеричной записи.
- В данном примере для простоты выбраны очень маленькие адреса. В действительности же адрес скорей 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`).

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 `p` это указатель, хранящий адрес `a`, то `*p` означает следующее:

*Пройди по адресу, хранящемуся в **`p`**. Возьми соответствующее количество байт, начиная с этого адреса (в данном случае 4, так как **`p`** указывает на **`int`**). Воспринимай эти байты как переменную соответствующего типа (в данном случа **`int`**).*

```c
#include <stdio.h>
int main()
{
    int a = 10;
    int* p = &a;
    *p += 1;
    printf("%d\n", a);
}
```

Часть 3: 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)`

Передача по указателю

Передавая в функцию не саму переменную, а указатель на эту переменную, мы можем менять саму переменную внутри, используя указатель.

```c
#include <stdio.h>

void inc(int* p)
{
    *p += 1;
}

int main()
{
    int a = 10;
    inc(&a);
    printf("%i\n", a);
}
```

При передаче в функцию массива, туда на самом деле передаётся указатель на первый элемент этого массива.

Обход массива с помощью указателя

```c
#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.

Часть 4: Схематическое изображение указателей в памяти

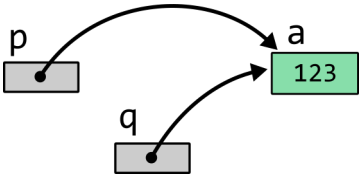
Так как постоянно рисовать переменные в памяти слишком громоздко и затруднительно, будем изображать из схематически. Стрелочкой будем указывать на переменную, адрес которой хранит указатель. Размеры прямоугольников не соответствуют размерам переменных. Пример выше тогда будет выглядеть так:

Задачи:

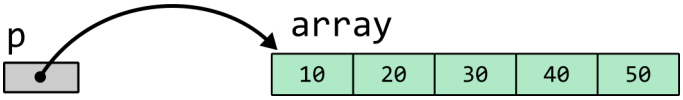
Напишите код, который будет соответствовать следующим рисункам. В каждой задаче разыменуйте указатели и напечатайте то, на что они указывают.

- Указатель на переменную типа `char`.

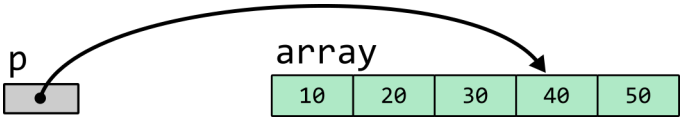
- Два указателя, которые указывают на одну переменную типа `int`



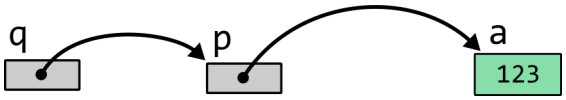
- Указатель типа `int*`, указывает на первый элемент массива `int`-ов под названием `array`



- 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`.



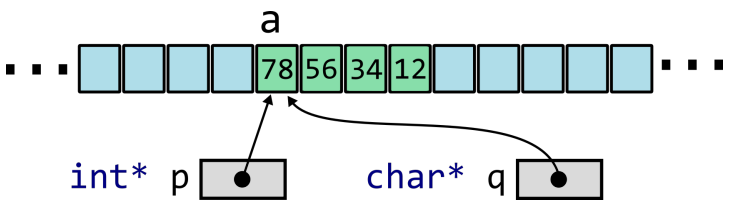
Часть 5: Указатели разных типов

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 разыменовании.

```
#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 разыменования произойдёт ошибка.