Семинар #4: Типы данных.

Часть 1: Целочисленные типы данных.

Различные целочисленные типы языка C представлены в следующей таблице:

тип	размер (байт)	диапазон значений ($2^{\#bits}$)	спецификатор
char	1	от -128 до 127	`%hhi`
short	2	от -32768 до 32767	`%hi`
int	4	примерно от -2-х миллиардов до 2-х миллиардов	`%i`
long	4 или 8	такой же как у int или long long в зависимости от системы	`%li`
long long	8	примерно от -10^{19} до 10^{19}	`%lli`
unsigned char	1	от 0 до 255	`%hhu`
unsigned short	2	от 0 до 65535	`%hu`
unsigned int	4	примерно от 0 до 4-х миллиардов	`%u`
unsigned long	4 или 8	такой же как у unsigned int или unsigned long long	`%lu`
unsigned long long	8	от 0 до $2^{64} \approx 2 * 10^{19}$	`%llu`
size_t	4	примерно от 0 до 4-х миллиардов	`%zu`

Это наиболее распространённые значения размеров типов для 64-х битных систем, но на некоторых системах эти значения могут быть другими. Чтобы узнать эти значения можно использовать оператор `sizeof`.

Часть 2: Новое имя типа

В языке C существует возможность вводить новое имя для уже существующего типа с помощью ключевого слова `typedef`. Чтобы ввести новое имя для типа нужно написать следующее:

```
typedef староеимя новоеимя;
```

После этого для типа можно будет использовать и новое и старое имя.

Тип `size_t`

Тип `size_t` – это беззнаковый тип целых чисел, который выбирается таким образом, чтобы он вмещал размер любого массива. Внутри стандартной библиотеки может быть написано примерно следующее:

```
typedef unsigned long long size_t;
```

Но размер типа `size_t` может различаться в зависимости от вычислительно системы. Выясните чему он равен на вашей системе.

Часть 3: Числа с плавающей точкой. Библиотека `math.h`

тип	размер (байт)	значимые цифры	диапазон экспоненты	спецификатор
float	4	6	от -38 до 38	`%f`
double	8	15	от -308 до 308	`%lf`
long double	от 8 до 16	≥ 15	не хуже чем у double	`%Lf`
печать только 3-х чисел после запятой	-	-	-	`%.3f`
печать без нулей на конце	-	-	-	`%g`
печать в научной записи	-	-	-	`%e`

Библиотека `math.h`

В библиотеке `math.h` содержатся множество полезных математических функций.

функция	что делает
`sqrt`	Вычисляет корень числа
`abs`	Вычисляет модуль целого числа
`fabs`	Вычисляет модуль числа с плавающей точкой
`exp`	Экспонента e^x
`log`	Натуральный логарифм $ln(x)$
`sin`, `cos`, `tan`	Синус, косинус и тангенс (радианы)
`asin`, `acos`, `atan`	Арксинус, арккосинус и арктангенс
`floor`	Округление до ближайшего меньшего целого числа
`ceil`	Округление до ближайшего большего целого числа
`pow(x, y)`	Возведение числа в `x` степень `y`

Точность чисел с плавающей точкой

Количество вещественных чисел на любом отрезке бесконечно, а количество возможных значений чисел с плавающей точкой ограничено, поэтому не каждое вещественное число можно закодировать числом `float` или `double`. Это означает, что числа с плавающей точкой всегда вычисляются с погрешностью. Поэтому сравнивать 2 таких числа оператором сравнения `==` очень опасно. Например, следующая программа напечатает `No`.

```c
#include <stdio.h>
int main()
{
    float a = 3 * 0.1;
    float b = 0.3;
    if (a == b)
        printf("Yes\n");
    else
        printf("No\n");
}
```

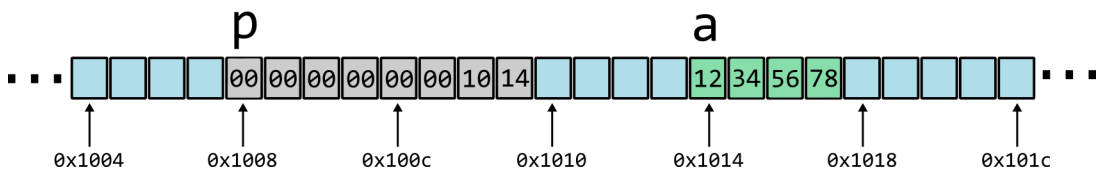
Такие числа всегда нужно сравнивать с некоторой точностью ϵ по формуле $|a - b| < \epsilon$. Вот так:

```c
#include <stdio.h>
#include <math.h>
int main()
{
    float eps = 1e-5;
    float a = 3 * 0.1;
    float b = 0.3;
    if (fabs(a - b) < eps)
        printf("Yes\n");
    else
        printf("No\n");
}
```

Часть 4: 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 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)`

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 6: Схематическое изображение указателей в памяти

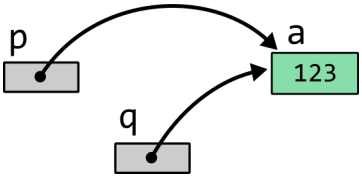
Так как постоянно рисовать переменные в памяти слишком громоздко и затруднительно, будем изображать из схематически. Стрелочкой будем указывать на переменную, адрес которой хранит указатель. Размеры прямоугольников не соответствуют размерам переменных. Пример выше тогда будет выглядеть так:

Задачи:

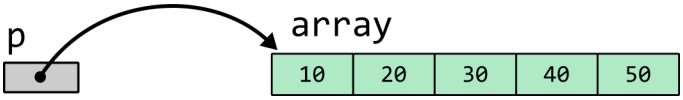
Напишите код, который будет соответствовать следующим рисункам. В каждой задаче разыменуйте указатели и напечатайте то, на что они указывают.

- Указатель на переменную типа `char`.

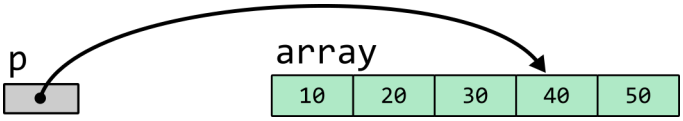
- Два указателя, которые указывают на одну переменную типа `int`



- 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`.