Семинар #2: 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
10-тичная система	-	-	%d
8-ричная система	-	-	%o
16-ричная система	-	-	%x

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`.

Задача 1: Размеры типов

Оператор `sizeof` возвращает размер типа в байтах. Например, следующая программа печатает размер переменной типа `int`:

```c
#include <stdio.h>
int main() {
    int a;
    printf("%i\n", sizeof(a));
}
```

Проверьте чему равен размер остальных целочисленных типов.

Задача 2: `unsigned char`

Тип `unsigned char` - это целочисленный тип, который принимает значения от `0` до `255` включительно. Чему будет равно значение `a` после выполнения следующих операций:

- ```
  unsigned char a = 200;
  a += 100;
  ```
- ```
  unsigned char a = 255;
  a *= 2;
  ```
- ```
  unsigned char a = 10;
  a *= -1;
  ```
- ```
  unsigned char a = -1;
  ```
- ```
  unsigned char a = 2;
  a *= 150;
  ```
  ```

Задача 3: Сложение чисел

Была написана простейшая программа, которая считывает 2 числа и печатает их сумму.

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

Однако, оказалось, что такая программа не работает на следующих тестах:

вход		выход
2000000000	2000000000	4000000000
2147483647	1	2147483648
1234567890	9876543210	11111111100
1	-5	-4

Почему так происходит? Исправьте программу так, чтобы она работала на всех этих тестах.

Задача 4: Печать в разных системах счисления

На вход подаётся число. Напечатать его в 10-тичной, 8-ричной и 16-ричной системах счисления. Используйте спецификаторы функции `printf`.

Часть 2: 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

Задачи на основы чисел с плавающей точкой

1. Напишите программу, которая считывает 2 числа `float` и печатает их произведение

2. Напишите программу, которая считывает 2 числа `double` и печатает результат деления первого на второе

3. Деление для целых чисел и чисел с плавающей точкой работает по-разному. Из-за этого можно сделать ошибку, которую потом будет трудно отыска в программе. Например, что напечатает следующая программа? Как её исправить?

```c
#include <stdio.h>
int main() {
    float x = 2 / 3;
    printf("%f\n", x);
}
```

4. Пример программы, которая считывает вещественное число – радиус круга и печатает его площадь. Слово `const` означает, что переменную `pi` нельзя будет изменить (константа).

```c
#include <stdio.h>
int main() {
    const float pi = 3.14159;
    float radius;

    scanf("%f", &radius);
    printf("%f\n", pi * radius * radius);
}
```

   Напишите программу, которая считывает радиус и печатает площадь сферы такого радиуса. Формула площади сферы: $S = 4\pi r^2$.

вход	выход
1	12.566360
0.5	3.141590

5. Напишите программу, которая считывает радиус и печатает объём шара такого радиуса. Формула для объёма шара:

$$V = \frac{4}{3}\pi r^3$$

   При этом мы хотим посчитать объём очень точно, как минимум с 10 знаками после запятой.

вход	выход
1	4.1887902048
3	113.0973355292
0.5	0.5235987756

   Вам понадобится более точное значение числа $\pi = 3.14159265358979323846$.

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`

Пример программы, которая считывает число x и вычисляет значени функции $f(x) = ln(|cos(x)|)$.

```
#include <stdio.h>
int main() {
    float x;
    scanf("%f", &x);
    printf("%f\n", log(fabs(cos(x))));
}
```

Задачи на математические функции

1. На вход подаётся 2 числа - стороны прямоугольника. Нужно напечатать значение длины диагонали этого прямоугольника с 3-мя знаками после запятой.

вход	выход
1 1	1.414
2 1	2.236
7 3	7.616

2. На вход подаётся 2 стороны треугольника и угол между ними в градусах. Найти третью сторону треугольника по формуле косинусов:

$$c^2 = a^2 + b^2 - 2 \cdot a \cdot b \cdot cos(\alpha)$$

Тригонометрические функции работают с радианами. Чтобы радианы перевести в градусы нужно домножить на $\frac{180}{\pi}$.

вход	выход
1 1 90	1.41421
1 1 60	1
5 3 15	2.24103

3. На вход подаётся 2 числа - стороны прямоугольника. Найдите значения угла между диагональю и наибольшей стороной в градусах.

вход	выход
1 1	45
2 1	26.565
1 10	5.710

4. Известно, что число π можно вычислить с помощью следующего ряда:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - ... = \sum_{i=1}^{\infty} \frac{(-1)^{i+1}}{2i-1}$$

Используйте эту формулу, чтобы вычислить приблизительно число π. На вход должно подаваться целое число `n` - число членов суммируемой последовательности, а вам нужно вычислить приближённое значение:

$$\pi \approx 4 \cdot \sum_{i=1}^{n} \frac{(-1)^{i+1}}{2i-1}$$

Точность чисел с плавающей точкой

Так как количество вещественных чисел на любом отрезке бесконечно, а количество возможных значений чисел с плавающей точкой ограничено, то далеко не каждое вещественное число можно закодировать числом `float` или `double`. Это может привести к некоторым ошибкам.

- **Неточность вычислений:** Числа с плавающей точкой всегда вычисляются с некоторой погрешностью. Поэтому сравнивать 2 таких числа с помощью оператора сравнения == очень опасно. Проверьте, что напечатает следующая программа.

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

  Такие числа всегда нужно сравнивать с некоторой точностью ϵ по формуле $|a - b| < \epsilon$.
  Вот как это выглядит в программе:

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

- **Нехватает значащих цифр:** Числа с плавающей точкой могут хранить ограниченное число значащих цифр. Например, `float` может точно хранить только 6 цифр. Проверьте, что напечатает следующая программа:

```
#include <stdio.h>
int main() {
    float a = 1234.567;
    printf("%f\n", a);
    a += 0.001;
    printf("%f\n", a);
    a += 0.00001;
    printf("%f\n", a);
}
```

  Видно, что если прибавлять к большому числу маленькое, то оно может вообще не измениться.

- **Нехватает значений экспоненты:** Числа с плавающей точкой не могут хранить слишком большие или слишком маленькие числа. Например, максимум для `float` это примерно 10^{38}. Все числа превышающие это значение становятся равны специальному значению `inf`.

```
#include <stdio.h>
int main() {
    float a = 1e20;
    printf("%f\n", a);
    a = a * a;
    printf("%f\n", a);
}
```

Часть 3: Простейшие функции

Помимо уже известных нам функций типа `printf`, `sqrt`, `sin` и других можно писать свои. Например, в языке `C` нет функции, которая возводит число в квадрат. Но её можно написать самим:

```c
#include <stdio.h>

int sqr(int a) {
    return a * a;
}

int main() {
    int x = 4;
    printf("%i\n", sqr(x));
}
```

Функция `sqr` принимает на вход 1 число типа `int` и возвращает 1 число типа `int`. Очень важно следить за типами аргументов функции и возвращаемого значения. Например, данная функция `sqr` будет работать правильно только с целыми числами `int`. Если передать в неё число типа `float`, то эта функция даст ошибку. Проверьте, что выдаст эта функция, если передать в неё `float`.

Задачи

1. Измените функцию `sqr` так, чтобы она работала с числами типа `float`.
2. Напишите функцию `func`, которая будет принимать на вход число `x` типа `float` и возвращать следующее выражение:

$$f(x) = x^3 + 5x + 7$$

```c
#include <stdio.h>

// Тут вам нужно написать функцию func

int main() {
    float x = -1.0;
    float y = func(x);
    printf("%f\n", y);
}
```

x	func(x)
-1	1
0.5	9.625
-0.7	3.157

3. Измените функцию `func`. Теперь она должна возвращать следующее выражение:

$$f(x) = \sqrt{|\sin(x)|};$$

x	func(x)
-1	0.917
0.2	0.446
100	0.712

Функции могут принимать несколько значений, а также внутри функций можно вызывать любой код, также как и внутри функции `main`. Вот пример функции `max`, которая вычисляет максимум от двух чисел.

```
#include <stdio.h>

float max(float a, float b) {
    if (a > b)
        return a;
    else
        return b;
}

int main() {
    float x = 2, y = 8;
    printf("%f\n", max(x, y));
}
```

Задачи

1. Напишите функцию, которая вычисляет среднее геометрическое от двух чисел. Проверьте эту функцию, вызвав её из функции `main`.

2. Напишите функцию `float distance(float x1, float y1, float x2, float y2)`, которая принимает на вход 4 числа – координаты точек на плоскости (x_1, y_1) и (x_2, y_2) и возвращает расстояние между этими точками.

3. На вход программе подаются 6 чисел типа `float` – координаты вершин треугольника в следующей последовательности: $x_1, y_1, x_2, y_2, x_3, y_3$. Программа должна считывать эти координаты и печатать площадь треугольника. Можно воспользоваться формулой Герона:

   $$S = \sqrt{p(p-a)(p-b)(p-c)}$$

   где $p = (a+b+c)/2$. a, b и c – стороны треугольника.

вход	выход
0 0 0 1 1 0	0.5
0 0 1 1 1 0	0.5
0 0 1 2 2 1	1.5
0 0 1 2 2 1	1.5
10 7 -4 -2 6 -1	38
3.1 6.2 -4.1 8.3 0.5 10.7	13.47

4. Посчитать приближённое значение определённого интеграла для функции $f(x)$:

   $$S(x) = \int_a^b f(x)dx$$

   Функция $f(x)$ задаётся в коде программы. Эта функция должна принимать один `float` и возвращать `float`. Числа типа `float` a, b и целое число n поступают на вход. Вы должны поделить площадь под кривой на маленькие прямоугольники и вычислить интеграл. Отрезок `[a, b]` нужно разбить на `n` частей.

Часть 4: Указатели.

Указатель – это специальная переменная, которая хранит адреса.

```c
#include <stdio.h>
int main() {
    // Предположим у нас есть переменная:
    int x = 42;
    // Положение этой переменной в памяти характеризуется двумя числами - адресом и
    // размером переменной. Узнать их можно, используя операторы & и sizeof:
    printf("Size and address of x = %d and %llu\n",  sizeof(x), &x);
    // Обратите внимание, что для отображения адреса использовался модификатор %llu, так
    // как в 64 битных системах адрес это 64 битное число. Также можно было бы
    // использовать модификатор %p, для отображения адреса в 16 - ричном виде.

    // Для работы с адресами в языке C вводится специальный тип, который называется
    // указатель. Введём переменную для хранения адреса переменной x:
    int* address_of_x = &x;
    // Теперь в переменной address_of_x типа int* будет храниться число - адрес
    // переменной x. Если бы x был бы не int, а float, то для хранения адреса x нужно было
    // бы использовать тип float* .

    // Ну хорошо, у нас есть переменная, которая хранит адрес x. Как её дальше
    // использовать? Очень просто - поставьте звёздочку перед адресом, чтобы получить
    // переменную x.
    // *address_of_x это то же самое, что и  x
    *address_of_x += 10;
    printf("%d\n", x);
    // Запомните: &  -  по переменной получить адрес
    //            *  -  по адресу получить переменную
}
```

Задача

Пусть есть такой код:

```c
#include <stdio.h>
int main() {
    float x = 10;
}
```

Создайте переменную `p` типа `float*` и сохраните в ней адрес переменной `x`. Используйте переменную `p`, чтобы изменить переменную `x`.