//Кирилл Юрьевич Богачев 1 занятие

Мы сейчас не помещаемся, но после первого семестра поместимся.

// Организационные моменты:

Занятия в аудитории, материал практический, лекции есть, но их трудно вести, как трудно научиться любому языку по лекциям.
К примеру, по английскому лекций нет

Изучаем Си, раньше говорили купить, сейчас говорят скачать книжку
Вышлет в качестве справочника

Основной материал рассказывается здесь, никаких знаний не предполагается
Некоторые знания - лучше бы их не было

Есть разница в том, что я считаю "рабочей программой" и вы
Цель - научиться писать хорошие программы

2-3 недели до кода
IDE - грамотность обезьяны

Условия сдачи задач на зачете:
3 попытки по 3 часа
Оценка сверху - 6 задач

/* От себя (опыт 1 семестра в 2022/2023 учебном году, правила игры относятся к заявленным на то время):

Процесс сдачи зачета искусственно усложнен: Кирилл Юрьевич действительно будет выдавать только по одной задаче раз в очередь (нельзя попросить "а дайте мне сразу 4 задачи, я за 3 часа все успею накодить", вернее, попросить можно, но он специально скажет "нет, я вас предупреждал, у меня такая политика сдачи зачета")

Таким образом, если задачу на зачете вы можете сделать за 40 минут, то потратить придется не меньше, чем полную ротацию очереди, в нашем случае на 23 человека это около двух часов, то есть, не больше 2 задач на один зачет (3-4 часа)

Цена каждого косяка в течение семестра дороже, чем может показаться, процесс отладки важнее, чем решение задачи и написание кода, ключ к успеху - тратить как можно больше времени на внимательное тестирование каждой задачи на как можно большем разнообразии входных данных

Задачи хоть и даются темами по 8-12, также организованы в блоки по 3-4 задачи - именно каждый такой блок является зачетной единицей, провал которой в течение семестра эквивалентен +1 задаче на зачет (которых, из оценки сверху, ни в коем случае не должно быть больше 5)

Из условия >50% следует, что на практике зачтенными должны быть 3 задачи из 4, косячить едва ли можно в каждой четвертой задаче; Все не так просто: 50% превращаются в 75%

Ошибка компиляции, утечка памяти, сегфолт, что-угодно, что не понравится компилятору Кирилла Юрьевича - блок незасчитывается полностью, независимо от результатов в других задачах

Задача считается зачтенной только после прохождения всех-всех тестов (особенное внимание уделяется "пограничным" случаям: вводу пустых файлов, последовательностей, минимальных по объему наборов данных, максимальных, в зависимости от формата задачи - следует догадываться, какой такой хитрый набор входных данных все-таки может уронить вашу программу, ведь именно он обязательно будет лежать в тестовом наборе у Кирилла Юрьевича, словом, программы очень сильно проверяются на отказоустойчивость)

2 пропуска занятия == +1 задача на зачет

*/

Можно переприсылать ошибку до дедлайна

/*
Это значит, что если вы нашли ошибку в своей программе после того, как отправили её в письме, то можно еще раз отправить исправленную версию с каким-нибудь комментарием вроде "нашел ошибку, финальная версия" (всех задач, а не только той, где была ошибка) - проверяться будет последняя присланная версия
*/

Не запускать программу 1 раз на идеальных данных
70% времени необходимо тратить на отладку

Вы поколение ЕГЭ: если скомбинировать 2 задачи в одну, то вы не справитесь
Работаем над восстановлением мыслительного процесса

// Начало занятия:

Кирилл Юрьевич Богачев
+ корпоративная почта с мехматовской сети

1 курс - C - появился в 1977 - пользуемся форматом C1990, после он устаканился
2 курс - C++

Специфика языка:

FORTRAN - 40 год появился, 77 завершенный формат, 1990 совсем последняя версия; научный язык(матпакетики), один из самых ранних
Авторы C знали фортран, другого языка не было

Алфавит - набор букв и цифр, пробельные символы( ), разделители(;)
Неудачная практика, когда пробельные это разделители(; ==)
Константы - зарезервированные наименования неизменяемых выражений
Идентификаторы - зарезервированная последовательность букв и цифр, начинающаяся с буквы и не содержащая пробельных символов
Ключевые слова - зарезервированные идентификаторы

Операции - +-, не все, есть те, которые буквочками; имеют типы ввода; типы вывода; значение; аргументы
Операторы - законченное утверждение языка (нет ни типа, ни значения)
Комбинации - то, чего нет на клавиатуре, к примеру спецсимволы, escape-последовательности: \n \\ \t

Простейшая программа:

```
int main(void)
{
      return 0;
}
```

Это функция, у нее есть тип возвращаемого значения(int), имя(main), список типов аргументов(void)
; - конец оператора

{} - блочный оператор (он умеет хранить в себе другие операторы)

Командная строка:
$ gcc f.c
Курсор($), вызов компилятора (gcc), название файла(f), расширение файла(.c) - запускается компоновка (линковка) - linker - получается исполняемый файл a.out

./a.out
./ - текущий каталог
Программы завершаются молча.

$echo - выводит свой аргумент
$ echo $?
$ - обозначение переменной окружения
? - возвращаемое значение предыдущей исполненной программы

Оператор имеет рекурсивное определение:

; -пустой оператор

return <выражение> (у выражения будет определение дальше); - оператор возвращения значения

<выражение>; - оператор(не любое выражение является оператором: a + b; или a = b;) // a=b; a==b;

```
{
      <оператор1>
      ...
      <оператор2>
```
} - рекурсия, сходится (каждую итерацию раскрытия скобочек их количество уменьшается) - блочный оператор

Операторы: условные, цикла, и т.д. - узнаем в ходе изучения языка

Си очень простой - все функции состоят ровно из одного оператора :)

Выражения:

Константы: 1 - являются выражением, у константы определен тип значения и они имеют значение
Переменная: x
Вызов функции: f(x) f(x,y)

имя(- компилятор считает, что это функция

(<выражение>) - тоже является выражением
<знак унарной операции><выражение> //<...> по-англисйки называется "плейсхолдер"

// -x унарная операция

<выражение 1> <знак бинарной операции> <выражение 2>

Описание объектов:
<тип> <идентификатор>; // int i;
<тип> <идентификатор1>, <идентификатор2>; // int i, j;
<тип> <идентификатор> = <выражение>;
<тип> <идентификатор1> = <выражение>, <идентификатор2> = <выражение>, ...; // int i = 1, j = 2, ...;

// Неплохо, чтобы все переменные в момент создания получали свои начальные значения (инициализировались)
// Инициализация переменной - присвоение переменной начального значения

У каждой переменной есть 2 параметра:
1 - Время жизни
2 - Область видимости

Время жизни - участок программы, в течение которого с переменной связана выделенная ячейка памяти
Бывает локальным(блочный оператор)
Глобальным - переменная описана вне любого оператора // Считается неприличным иметь переменные с глобальным временем жизни (могут менять свои значения неконтролируепмым образом, всегда занимая память)

Область видимости - участок кода, в пределах которого к этой переменной можно обращаться (компилятор её оттуда "узнает")
Пример:

```
int j = 1;
int main()
{
      int i = 0;
      return i;
}
// i - локальная переменная
```

// j - глобальное переменная, инициализируется компилятором, должно быть константное выражение для компилятора, чтобы он мог его посчитать на этапе компиляции

Область видимости - участок программы, на протяжении которого можно обратиться к переменной

Локальная переменная - область видимости от начала до конца блока, в котором объявлена
Глобальная - видна по всей программе // Трудно найти, где применялась глобальная переменная, табу использовать объявление вне блочных операторов
Файл - видна на весь исходный файл

Функции - полностью глобальные, объявляются вне любых функций, их создает компилятор

```
int main(void)
{
	int i = 0;
	int j = 1;
	{
		int k = 2;
	}

	return i;
}
```

Типы данных - их характеристики следуют из компьютера, архитектуры ОС, на котором работаем

Бит, побитовое представление целых чисел:
$x=(-1)^s \sum_{i=0}^{m-1}{a_i b^i}$
$a_i \in {0,...,b-1}$
s // Не расшифровал, вроде s==0 или s==1

Знаков для кодирования: 7+1 (со знаком) или 8 (без знака) //Бухгалтер: отрицательные числа - статья
Каждый знак - бит, 8 знаков - байт

Адреса есть только у байтов, у битов - нет
Есть физические адреса, виртуальные, и еще много какие

Мы не знаем, как лежат биты внутри байтов, в лучшем случае - знаем их нумерацию, прямого доступа к ним нет

Вся оперативная память заадресована (индексирована) побайтово

Биты можно просить у процессора из адресованных ячеек(байтов) по одному

Способ хранения целых чисел в байте:
abs(x)-1 + инверсия

12345678

11111111 -->
00000000+1 = 00000001 - число -1

-128..127

// Тут было объяснение двоичной системы исчисления

Процессор определяет порядок нумерации байтов
Разрядность процессора - длина машинного слова - размер для хранения адреса ячейки на оперативной памяти
// Тут я вышел, написал сверху то, что понял

Таблица характеристик типов данных:

Ключевое слово: Размер байт: Минимальное представимое: Максимальное представимое: Linux: Windows:

char 1 -128 127 // char - от слова character
unsigned char 1 0 255 // unsigned - беззнаковый

short int 2 -32768 32767 2 2
unsigned short int 0 65535 2 2
int short<=4<=long -2^31 2^32-1 4 4
long int 4
unsigned long int 8
long long int нет в стандарте

1 колонка - в стандарте не написано

Битность компьютера - максимальная длина адреса
64 бита - адрес на 8 байт - 2^64 адресов
32 бита - оперативная память не больше 32 гигабайт (2^32)

Числа с плавающей точкой - отдельная тема - (-1)^s 2^p a - знак, порядок, мантисса, 0.5<a<1

// сдвиг на единицу - ради единственности нуля

Все процессоры поддерживают одну и ту же арифметику плавающей точки по стандарту

float 4 -10^38 10^38
double 8 -10^308 10^308

min 0 max
-10^-308 10^-308 - макс и мин отрицательное и положительное соответственно
Это не имеет никакого отношения к вещественным числам - дискретизация - их конечное число, они распределены на вещественной оси