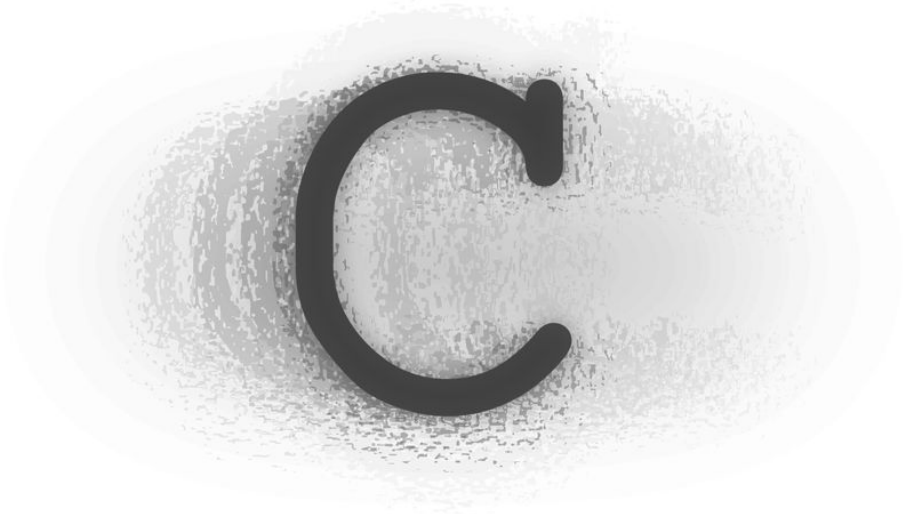
Занятие 3.
Функции.

- Программные модули в С.
- Функции математической библиотеки. Другие библиотеки языка С.
- Определения функций, аргументы и параметры, возвращение значения функцией.
- Генерация случайных чисел. Функции rand и srand.
- Рекурсия.
- Классы памяти.
    - Перегрузка функции.
    - Шаблон функции.



Проверка домашнего задания.

1. *Что выведет на экран данный кусок кода?*

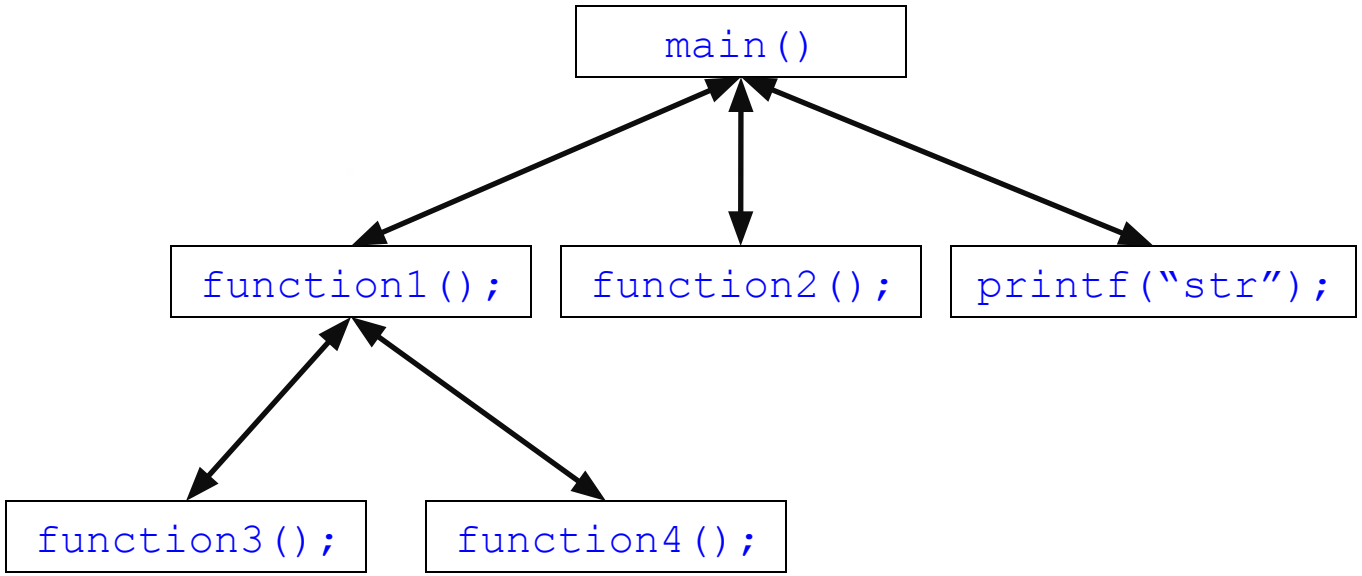
```
int a = 5, b =2 ;
char c1 = '+', c2 = '\n';
double d1 = 12.3, d2 = .2;
printf( "a=%3d, b=%d%c%.2lf %c %.2lf = %2.2lf \n", a, b, c2, d1, c1, d2, d1 + d2);
```

2. Есть 4 числа :

```
int a = 15;
double c = 14.223;
unsigned d = 340304;
double e = -23.140;
```

Напишите оператора printf(), который бы выводил на экран следующий текст:

Программные модули в С.



Функции математической библиотеки.

Для использования математических функций необходимо включить в программу заголовочный файл математики с помощью директивы препроцессора #include<math.h>

Функция	Назначение
sqrt(x)	Квадратный корень из x
exp(x)	Экспоненциальная функция
log(x)	Натуральный логарифм
log10(x)	Десятичный логарифм
fabs(x)	Модуль числа
ceil(x)	Округляет до ближайшего целого, не меньшего x
floor(x)	Округляет до ближайшего целого, не превосходящего x
pow(x,y)	Возводит x в степень y
fmos(x,y)	Остаток от деления x/y как число с плавающей точкой
sin(x)	Тригонометрический синус x (x задается в радианах)
cos(x)	Тригонометрический косинус x (x задается в радианах)
tan(x)	Тригонометрический тангенс x (x задается в радианах)

Другие библиотеки языка С.

Заголовочный файл	Содержимое библиотеки
<assert.h>	Содержит диагностические макросы и информацию, которая помогает при отладке программы.
<ctype.h>	Содержит прототипы для функций, которые проверяют определенные характеристики символов и прототипы для функций, которые могут использоваться для преобразования символов нижнего регистра в символы верхнего регистра и наоборот.
<errno.h>	Определяет макросы которые полезны для сообщения об ошибке.
<float.h>	Содержит пределы значений для чисел с плавающей точкой.
<limits.h>	Содержит пределы значений для целых чисел.
<locale.h>	Содержит прототипы функций и другую информацию, которая позволяет изменять работу программы согласно правилам текущей местности, в которой она выполняется.
<math.h>	Содержит прототипы функций математической библиотеки.
<setjmy.h>	Содержит прототипы для функций, которые позволяют обойти обычный вызов функции и последовательность возврата.
<signal.h>	Содержит прототипы функций и макросы для обработки различных условий, которые могут возникнуть во время выполнения программы.
<stdrag.h>	Определяет макросы для обработки списка параметров, для которых неизвестно число параметров и их тип.
<stddef.h>	Содержит общие определения типов, используемых в С для выполнения некоторых вычислений.
<stdio.h>	Содержит прототипы функций ввода/вывода и информацию, используемую ими.
<stdlib.h>	Содержит прототипы функций для преобразования чисел в текст и текста в числа, прототипы функций размещения памяти генерации случайных чисел и других сервисных функций.
<string.h>	Содержит прототипы функций для работы со строками.
<time.h>	Содержит прототипы функций и типы для функций управления временем и датой.

Определения функций.

```
#include<stdio.h>
#include<conio.h>

int square( int y );                                    //Прототип функции

int main()
{
     for( int i = 1; i <= 10; i++ )                     //Цикл от 1 до 10
          printf( "%d  ", square( i ) );                //Вызываем функцию возведения в квадрат
                                                             //и печатаем результат возведения.
     getch();
     return 0;
}

int square( int y )                                     //Заголовок функции
{
     return y * y;                                      //Возврат квадрата как int
}
```

Общий вид прототипа функции:
<Тип возвращаемого значения> <имя функции>(<список параметров>);

Общий вид определения функции:
<Тип возвращаемого значения> <имя функции>(<список параметров>)
{
*   <операторы>*
}

Аргументы и параметры. Возвращение значения функцией.

```
#include<stdio.h>
#include<conio.h>

float maximum( float a, float b, float c );            //Прототип функции

int main()
{
     float val1, val2, val3;                                    //Декларация переменных

     printf( "Enter three floats:\n" );                    //Вывод текста на экран

     scanf( "%f%f%f", &val1, &val2, &val3 );            //Чтение трех дробных чисел

     printf("Maximum is: %f",maximum(val1,val2,val3)); //Вызов функции max и вывод
                                                                  //на экран самого
большого числа
     getch();
     return 0;
}

float maximum( float a, float b, float c )        //Заголовок функции
{
     float max = a;                //Переменная для самого большого числа - изначально равна
a
     if( b > max )                 //Если b больше самого большого числа
          max = b;                 //Помещаем значение b в самое большое число
     if( c > max )                 //Если c больше самого большого числа
          max = c;                 //Помещаем значение c в самое большое число
     return max;                        //Возврат самого большого числа
}
```

Генерация случайных чисел. Функции rand и srand.

Создадим имитацию броска игрального кубика. Бросим кубик 20 раз для проверки.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
     for( int i = 0; i < 20; i++ )       //Цикл от 0 до 19 ( всего 20 бросков )
     {
          printf( "%4d", 1 + rand() % 6 );//Выводим на экран случайное число - значение
кубика
          if( i % 5 == 4 )                    //Каждыое пятое значение начинаем
               printf( "\n" );                     //с новой строки
     }
     getch();
     return 0;
}
```

Функция rand() генерирует случайные числа от 0 до RAND_MAX (обычно равно 32767).
Для того, чтобы исходное число принимало значения от 1 до 6 необходимо:

1. *Отмасштабировать случайное значение путем нахождения остатка от деления: rand() % 6. Это даст на выходе 6 чисел, расположенных в случайном порядке. Сами числа – 0, 1, 2, 3, 4, 5.*
2. *Так как числа начинаются с 0 и заканчиваются 5, необходимо сдвинуть их на 1: rand() % 6 + 1. В итоге получим нужные нам числа – 1, 2, 3, 4, 5, 6.*

Генерация случайных чисел. Функции rand и srand.

Проверим качество генерации случайных значений. Если бросить кубик 60 000 раз, каждая из граней должна выпасть примерно по 10 000 раз.

Код программы – Код программы –
list1.txt

Генерация случайных чисел. Функции rand и srand.

Если заметить, то количество выпавший раз для каждой грани будет одним и тем же. Это значит, что функция rand() генерирует не совсем случайную последовательность. Такая последовательность называется псевдослучайной. Данной свойство очень важно, так как при отладке программы программист может работать с заранее известной “случайной” последовательностью. После завершения отладки можно применить функция srand() – так называемое засевание. Параметр, передаваемый в функцию может быть от 0 до 4294967295, т.е. весь диапазон значений типа unsigned int(называется семенем). Функция srand() изменяет последовательность чисел, выдаваемую функцией rand(). Таким образом мы можем получить 4294967296 случайных последовательностей.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
     unsigned int seed;
     printf( “Enter seed : " );
     scanf(“%u”, &seed );
     srand( seed );

     for( int i = 0; i < 20; i++ )       //Цикл от 0 до 19 ( всего 20 бросков )
     {
          printf( "%4d", 1 + rand() % 6 );//Выводим на экран случайное число - значение
кубика
          if( i % 5 == 4 )                    //Каждыое пятое значение начинаем
               printf( "\n" );                     //с новой строки
     }
     getch();
     return 0;
}
```

Генерация случайных чисел. Функции rand и srand.

Еще один вариант – использовать в качестве семени текущее время, получаемое функцией time() (находится в библиотеке <time.h>)

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
     srand( time( 0 ) );

     for( int i = 0; i < 20; i++ )       //Цикл от 0 до 19 ( всего 20 бросков )
     {
          printf( "%4d", 1 + rand() % 6 );//Выводим на экран случайное число - значение
кубика
          if( i % 5 == 4 )                    //Каждыое пятое значение начинаем
               printf( "\n" );                     //с новой строки
     }
     getch();
     return 0;
}
```

Функция time(0) возвращает значение текущего времени с точностью до секунды. Таким образом “случайная” последовательность, функции rand() будет изменятся каждую секунду.

Генерация случайных чисел. Функции rand и srand.

Создаем простую игру – “крепс”.

Игрок бросает две кости. После того как кости остановятся, вычисляют сумму точек на верхних гранях кубиков. Если выпавшая сумма на первом броске равна 7 или 11 – игрок выиграл, если 2, 3 или 12 – проиграл. Если выпадут числа 4, 5, 6, 8, 9, 10 – то это число становится числом игрока. Дальше кости бросаются до тех пор, пока снова не выпадет это число или число 7. Если выпало число игрока – игрок выиграл, если 7 – игрок проиграл.

Определим последовательность действий игры:

1.Инициализируем необходимые переменные.

2.Бросаем кости.

1. *Если сумма равна 7 или 11 – устанавливаем статус игрока как выигравший.*
2. *Если сумма равна 2, 3 или 12 – устанавливаем статус игрока как проигравший.*
3. *Иначе – запоминаем число как число игрока и устанавливаем статус как неопределенный.*

3.Пока статус игрока не определен – кидаем кости.

1. *Если выпало число игрока – устанавливаем статус игрока как выигравший.*
2. *Если выпало число 7 - устанавливаем статус игрока как проигравший.*

4.Проверяем статус игрока и выводим результат на экран.

Примечание. Статус игрока – это обычная переменная, принимающая некоторые значения. В данном случае возможен такой вариант : 1 – игрок выиграл, 2 – игрок проиграл, 3 – статус не определен.

Код программы – Код программы – list2.txt

Рекурсия.

Фактически, рекурсия – это использование функции самой в себе. Практически, рекурсия – это метод решения задач, построенный на том, что рекурсивная функция может решить напрямую только частную задачу. Все остальные задачи функция делит на 2 части: часть, которую можно решить напрямую и часть, которую напрямую решить нельзя. Притом вторая часть должна являться такой задачей, которую можно было бы решить вызовом этой же функции.

Пример рекурсии – нахождение факториала числа.

Предположим, что нам надо найти значение 5!. Математически выводим, что:

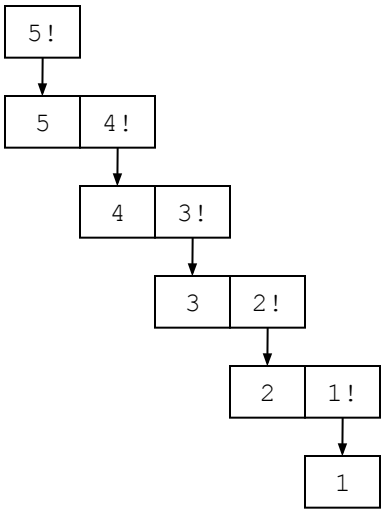
*5! = 5 * 4 * 3 * 2 * 1*

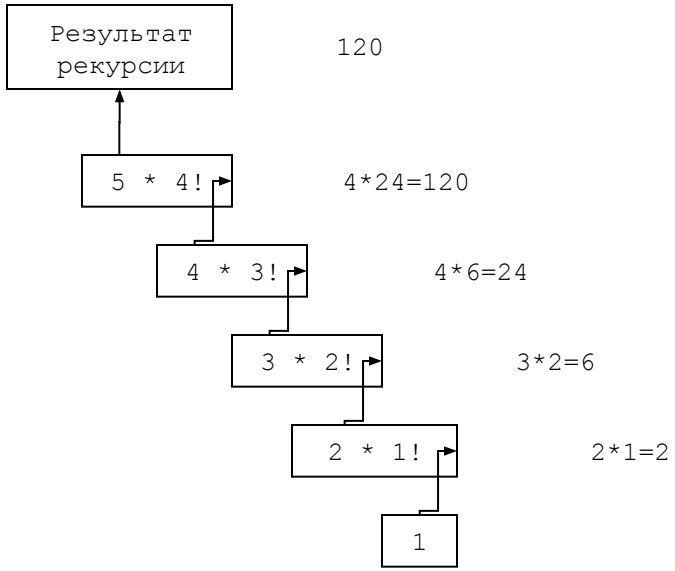
*5! = 5 * (4 * 3 * 2 * 1)*

*5! = 5 * 4!*

И т.д.

То есть функция должна постепенно “упрощать” поставленную задачу. В конце концов она дойдет до значения 1! = 1, после чего произойдет обратная подстановка.





Рекурсия.

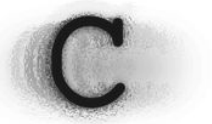
Пример рекурсии – нахождение факториала числа.

```
#include<stdio.h>
#include<conio.h>

unsigned long long int fact( unsigned a );              //Прототип функции факториала

int main()
{
     for( int i = 0; i <= 20; i++ )                          //Проходим от 0 до 20
          printf( "%d! = %llu\n", i, fact( i ) );       //Выводим значение факториала
i!
     getch();
     return 0;
}

unsigned long long int fact( unsigned a )          //Заголовок функции
{
     if( a > 1 )                                             //Если у нас не частный
случай -
          return a * fact( a - 1 );                     //Разбиваем на 2 части : a *
(a-1)!
     else                                                    //Иначе – если у нас частный
случай
          return 1;                                         //Возвращаем 1 ( т.к. 0!=1 и
1!=1)
}
```

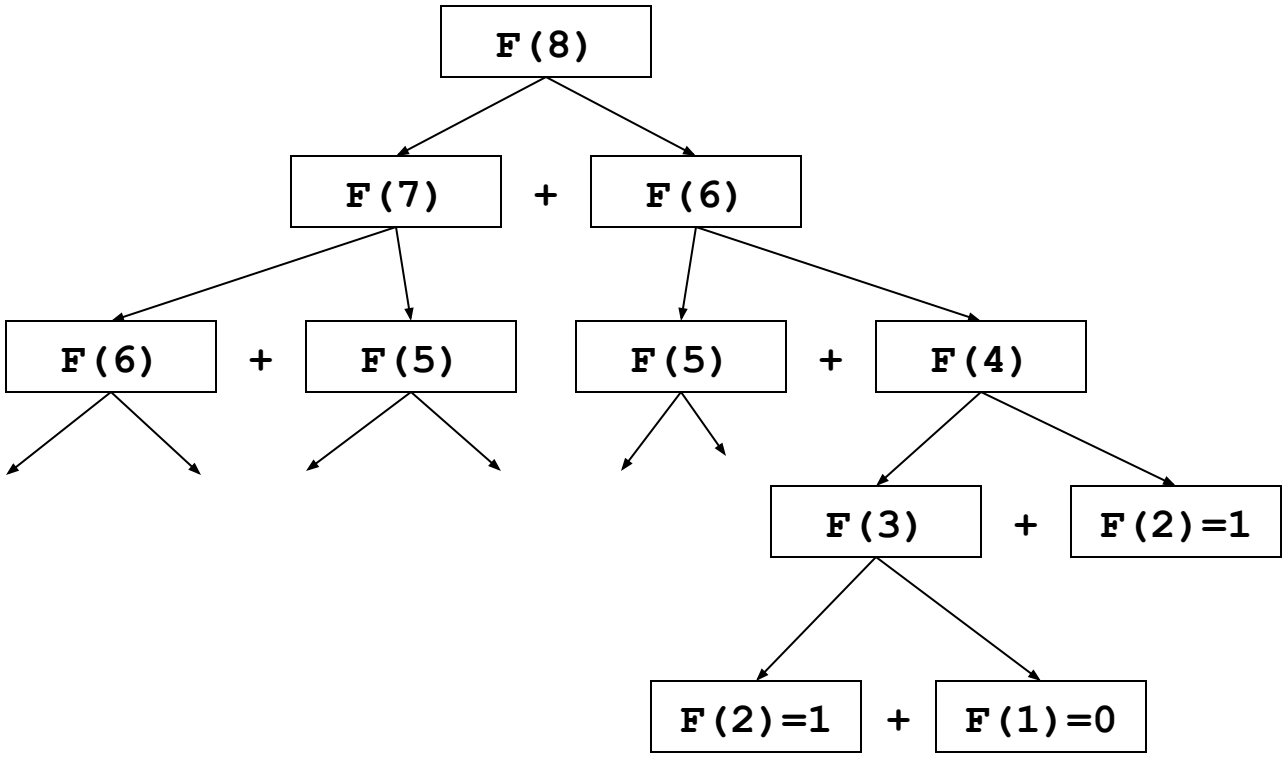


Рекурсия.

Пример рекурсии – нахождение чисел Фибоначчи.

Числа Фибоначчи – это числа вида : 0, 1, 1, 2, 3, 5, 8, 13, 21… Т.е. два первых числа последовательности – 0 и 1, а каждое последующее число получается как сумма двух предыдущих.

Скажем, нам надо найти 8-мое число из ряда Фибоначчи. Итак:
F(8) = F(7) + F(6) = (F(6)+F(5))+(F(5)+F(4)) = ((F(5)+F(4)) + (F(4)+F(3))) + ((F(4)+F(3)) + (F(3)+F(2)))



Рекурсия.

Пример рекурсии – нахождение чисел Фибоначчи.

```
#include<stdio.h>
#include<conio.h>

unsigned int fibon( unsigned a );                                    //Прототип функции

int main()
{
     for( int i = 1; i <= 20; i++ )                                  //Выводим числа Фибоначчи от 1
до 20
          printf( "fibon(%2d) = %u\n", i, fibon( i ) );
     getch();
     return 0;
}

unsigned int fibon( unsigned a )                                     //Заголовок функции
{
     if( a > 2 )                                                     //Если a - не первый или
второй элемент  -
          return fibon( a - 1 ) + fibon( a - 2 );       //Считаем его как сумму двух предыдущих
     else                          //Иначе
          return a-1;                    //Возвращаем значение. Для a=1 во звераща
ем 0, для a=2
возвращаем 1
}
```

Дополнительно – выведете числа Фибоначчи от 1 до 40. Обратите внимание на то, сколько по времени вычисляются последние числа. В чем причина?

Упражнения.

1. *Напишите функцию, которая бы по двум введенным катетам прямоугольного треугольника вычисляла гипотенузу.*
2. *Напишите функцию, которая получает 2 целых числа – X и Y, и возвращает значение* x^y
3. *Напишите 3 функции, которые бы генерировали случайным образом наборы чисел:*
   2, 4, 6, 8, 10
   3, 5, 7, 9, 11
   6, 10, 14, 18, 22
4. *Напишите функцию – аналог операции % - остаток от деления, не используя сам оператор.*
5. *Число называется простым, если оно делится только на 1 и на само себя нацело и без остатка. Напишите функцию, которая бы определяла, является ли число простым.*
6. *Число называется совершенным, если само число равно сумме всех его делителей. Для примера: число 6 – совершенное, так как 3+2+1 = 6.*
   Напишите функцию которая определяет совершенное число или нет. Выведете совершенные числа от 1 до 1000 на экран.
7. *Напишите функцию, которая инвертирует цифры числа. Например, для числа 38246 функция должна возвращать число 64283.*
8. *Измените игру в “крепс” так, чтобы компьютер сыграл сам с собой 100000 партий, после чего вывел процент выигрышных партий. Подсказка – код с партией игры необходимо оформить в отдельную функцию, которая бы сообщала о том, выиграл ли игрок или проиграл.*
9. *Напишите обучающую программу, которая проверяет учеников начальной школы на знание таблицы умножения. Программа случайно генерирует 2 числа(a и b) от 1 до 10 и спрашивает – “Сколько будет a * b ?”, после чего ученик должен ввести правильный ответ. В случае неправильного ответа – предложить ответить ещё раз. В случае правильного – продолжить опрос дальше.*
   Примечание. Во время опроса необходимо вести статистику правильных и неправильных ответов. Предусмотреть пользовательский выход из программы, при котором ученику покажут количество правильных и неправильных ответов.

Упражнения.

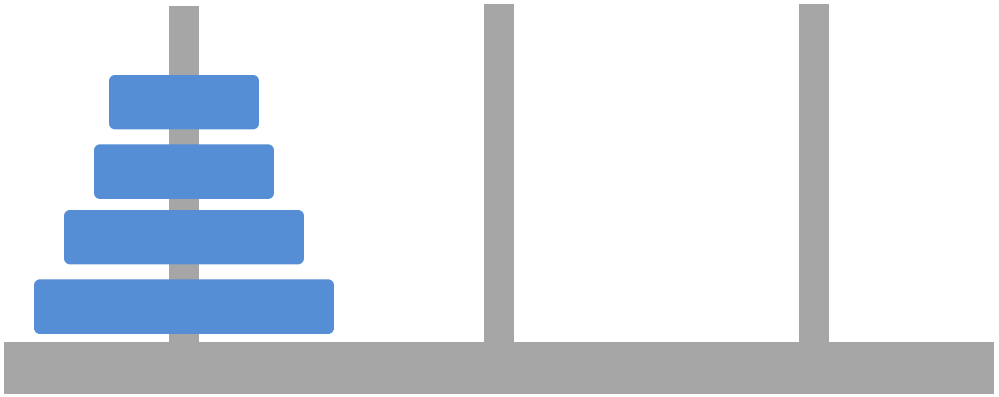
10. *Напишите программу, которая выводит один под другим следующие рисунки. Для вывода использовать функцию, которая бы выводила одну строку с переданным в неё количеством пробелов и звездочек.*

```
*          ******    ******          *
**         *****      *****         **
***        ****        ****        ***
****       ***          ***       ****
*****      **            **      *****
******     *              *     ******
```

11. *Измените рекурсивную функцию вычисления факториала так, чтобы она печатала значение своей рекурсивной переменной и параметры рекурсивного вызова. Сделай все возможное, чтобы результат вывода выглядел ясно, интересно и осмысленно.*

12. *Переписать программу игры в “Крепс” так, чтобы у игрока появилась возможность делать ставки.*

Задача!
Ханойская пирамида



Задача!
Написать игру в 21.

Игра должна быть 100% карточным аналогом. Вначале игры тасуется колода, из которой в процессе игры игрок и оппонент снимает карты. За оппонента должен играть компьютер. Логику игры компьютера выбрать и написать самому.