

# Рекурсия

---

Один из самых мощных  
методов решения задач

# Определения

---

- Рекурсивным называется объект, частично содержащий себя, или определенный с помощью самого себя.

# Рекурсивные определения

---

- ***Натуральные числа:***
  - 0 – натуральное число
  - Если  $N$  – натуральное число, то число  $N+1$  также натуральное

# Рекурсивные определения

---

- ***Факториал числа:***
  - $0! = 1$
  - $N! = (N-1)! * N$ , для любого  $N > 0$ ;

# ДОСТОИНСТВО

---

- ***Рекурсивное определение*** позволяет определить бесконечное множество объектов с помощью конечного высказывания

# Рекурсия в программировании

---

- С помощью конечной рекурсивной программы можно описать бесконечное вычисление, причем программа не будет содержать явных повторений

# Рекурсивные функции

---

- Если некоторая подпрограмма (функция) содержит явную ссылку на саму себя, то ее называют *прямо-рекурсивной*
- Если подпрограмма  $P$  ссылается на другую подпрограмму  $Q$ , которая содержит ссылку на  $P$ , то такую подпрограмму называют *косвенно-рекурсивной*

# Рекурсия

---

- Рекурсия сводит решение задачи к решению более мелких ***идентичных задач***
- Сложные задачи могут иметь простые рекурсивные решения
- Не всегда рекурсивный метод решения лучше итеративного (основанного на использовании циклов)



# Факториал числа

---

Рекурсивное определение:

```
Fact(int n)
{ if(n==0) return 1;
  else return (Fact(n-1));
}
```

# Факториал числа: итеративное определение

---

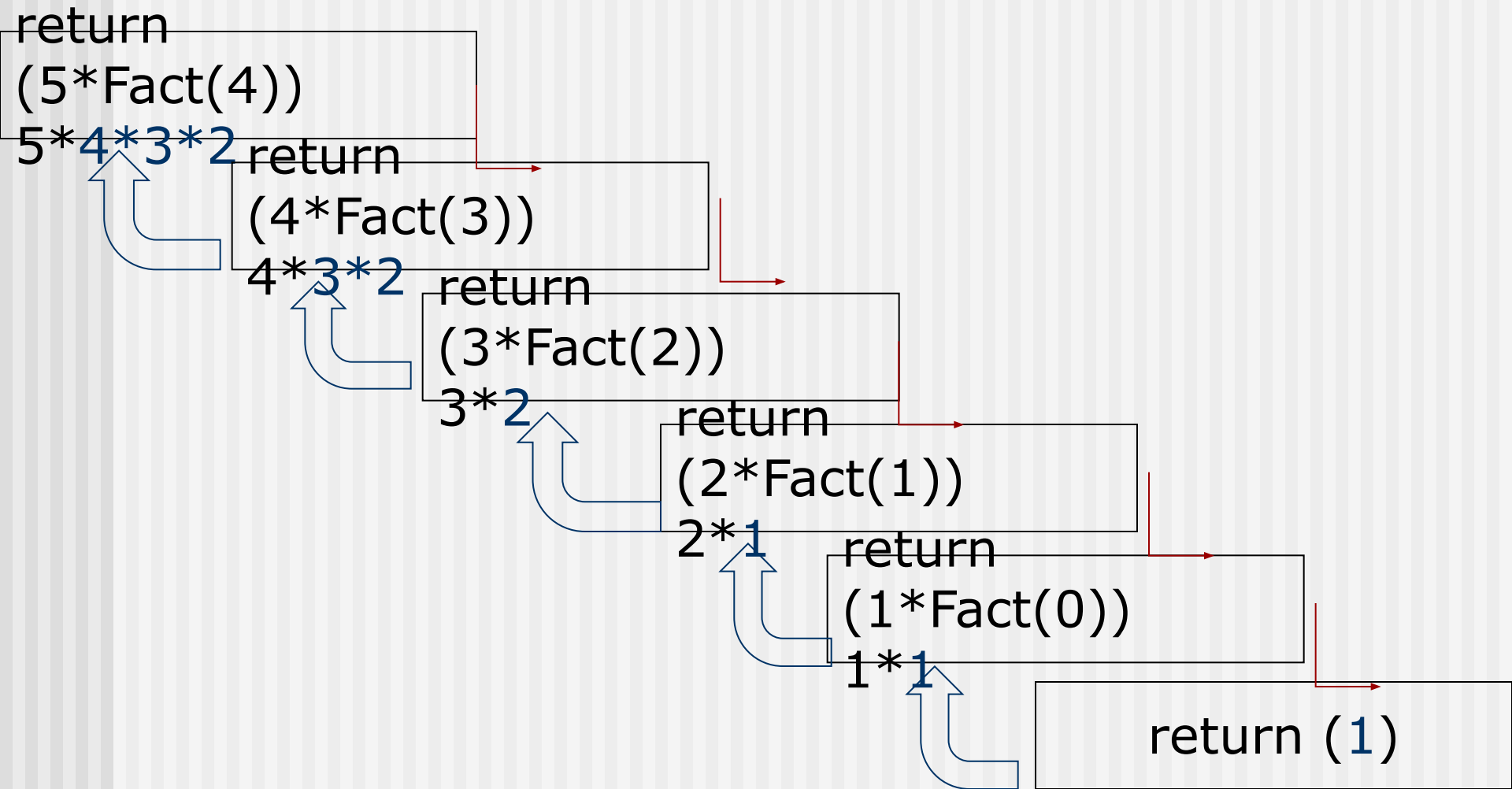
```
long Factorial (int n)
{
    int i;
    long f;
    if (n==0) return 1;
    else
        for(i=1;i<=n;i++) f=f*I;
    return (f);
}
```

# Рекурсивное решение

---

- При вызове подпрограммы всякий раз создаются новые экземпляры локальных переменных
- При этом не возникает никаких конфликтов при использовании имен

# Рекурсивное решение: факториал числа 5



# Свойства рекурсивного решения

---

1. Функция вызывает саму себя
2. При каждом вызове функции решается идентичная, но меньшая задача
3. Одна из подзадач решается иначе, чем другие : в итоге одна из подзадач является базовой
4. Проверка базисных условий позволяет остановить процесс рекурсии

# Ошибка в использовании рекурсии

---

- Отсутствие базиса:

```
void PRINT()  
{ cout << '*';  
  PRINT();  
}
```

- При вызове данной функции процесс вывода никогда не закончится, т.к. отсутствует базис

# Четыре вопроса о рекурсивном решении:

---

- Как свести задачу к идентичным задачам меньшего размера?
- Как уменьшать размер задачи при каждом рекурсивном вызове
- Какая задача является базовой
- Можно ли достичь базиса, постоянно уменьшая размер задачи?

# Числа Фибоначчи

---

- $F(n) = F(n-1) + F(n-2)$
- Необходимо 2 базиса:  $F(1) = 1, F(2) = 1;$

```
int F(int n)
{ if (n <= 2) return 1;
  else return F(n-1) + F(n-2);
}
```



# Задача о параде

---

- Необходимо на параде расставить  $k$  оркестров и  $m$  платформ, так, чтобы никакие два оркестра не стояли рядом.
- Сколькими способами можно расставить оркестры и платформы, если их всего  $N$  штук.

# Задача о параде

---

- Допустим, что у нас есть  $N$  оркестров и  $N$  платформ
- Будем считать, что варианты *оркестр-платформа* и *платформа-оркестр* – различны
- Парад может закрываться либо платформой, либо оркестром

# Задача о параде

- Введем обозначения:
  - $P(n)$  - количество вариантов организации парадов длиной  $n$
  - $F(n)$  - количество вариантов организации парадов длиной  $n$ , завершающихся платформой
  - $B(n)$  - количество вариантов организации парадов длиной  $n$ , завершающегося оркестром
  - Тогда  $P(n) = F(n) + B(n)$

# Задача о параде

- $F(n) = P(n-1)$  – парад, завершающийся платформой длины  $n$  получается из парада длиной  $n-1$ , завершающийся оркестром
- $V(n) = F(n-1)$  – если парад заканчивается оркестром, значит перед ним стоит платформа
- Таким образом получаем:  
 $V(n) = P(n-2)$
- **$P(n) = P(n-1) + P(n-2)$**

# Задача о параде

## ***Базис:***

---

- $P(1)=2$ -парад длины один может состоять либо из платформы, либо из оркестра
- $P(2)=3$ - парад длины 2 может состоять из:
  - Платформы и оркестра
  - Двух платформ
  - Двух оркестров

# Дилемма мистера Спока

---

- Сколько разных способов можно применить для исследования  $k$  планет, если солнечная система состоит из  $n$  планет ( $c(n, k)$ )

# Рассуждения мистера Спока

---

- Есть две возможности:
  - Либо мы посещаем некоторую планету  $X$  и тогда другие  $k-1$  можно выбрать из оставшихся  $n-1$  планет
  - Либо мы игнорируем планету  $X$  и тогда из остальных  $n-1$  планет можно выбрать  $k$  планет

# Получаем формулу:

---

- $c(n,k) = c(n-1,k-1) + c(n-1,k)$ , где
- $c(n-1,k-1)$  – количество групп, состоящих из  $k$  планет, включающих планету  $X$
- $c(n-1,k)$  – количество групп, состоящих из  $k$  планет, не включающих планету  $X$



# Базис:

---

- $c(k,k)=1$  – можно выбрать только одну группу, состоящую из всех планет
- $c(n,0)=1$  – есть только одна группа, не содержащая ничего
- $c(n,k)=0$ , если  $k > n$

# Разложение числа на слагаемые

---

- Сколькими способами можно разбить число  $M$  на слагаемые
- Обозначим число разбиений  $P(M)$
- Введем новую функцию  $Q(M, N)$  – количество разбиений числа  $M$  на слагаемые  $\leq N$

# Разложение числа на слагаемые

---

- **$Q(M,1)=1$**  – только одним способом можно представить число  $M$  с помощью 1:  $1+1+\dots+1$
- **$Q(1,N)=1$**  – число один можно разложить на слагаемые только одним способом, независимо от  $N$
- **$Q(M,N)=Q(M,M)$ ,  $M < N$**  – никакое разложение не может содержать число большее чем  $M$

# Разложение числа на слагаемые

---

- **$Q(M, M) = Q(M, M-1) + 1$**  – существует только одно разбиение со слагаемым  $= M$ , все остальные имеют слагаемые меньше  $M$
- **$Q(M, N) = Q(M, N-1) + Q(M-N, N)$**  – любое разбиение  $M$  с наибольшим слагаемым  $\leq N$  либо не содержит  $N$  в качестве слагаемого, или содержит  $N$  и тогда все остальные слагаемые образуют разбиение числа  $M-N$