Увод в програмирането

Лекция 10:

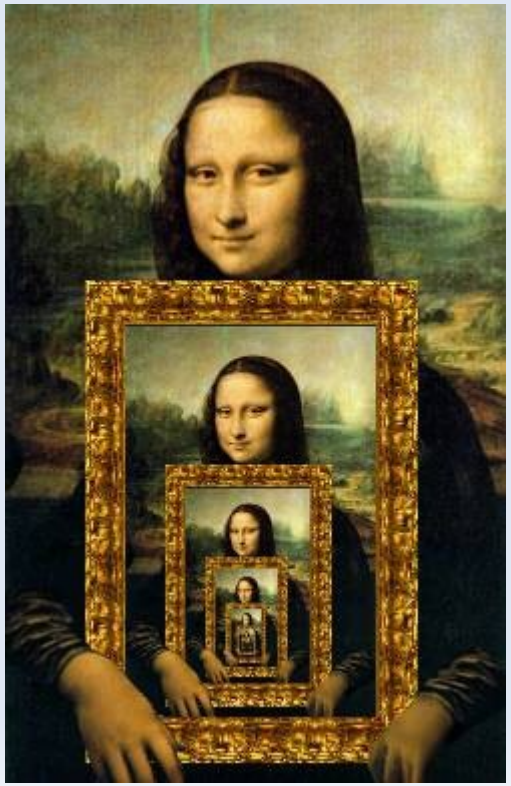
Рекурсия

Преговор

- Стрингове

Какво е рекурсия?

- Дефиниция на нещо, която включва позоваване на самото нещо
- Рекурсивни примери:
  - Директориите съдържат файлове и директории
  - PHP = PHP Hypertext Preprocessor
  - <израз> ::= <константа> | <променлива> | <унарна_опрация> <израз> | <израз> <бинарна_оп.> <израз> | (<израз>)
  - Картинки, съдържащи себе си:



Рекурсия в математиката

- В дефинициите на следните функции се използват самите функции, които описваме:

$$n! = \begin{cases} 1, & n = 0 \\ n(n-1)!, & n > 0 \end{cases}$$

$$x^n = \begin{cases} 1, & n = 0 \\ x.x^{n-1}, & n > 0 \\ \dfrac{1}{x^{-n}} & n < 0 \end{cases}$$

Рекурсия в математиката - 2

- Още един пример:
  - Най-голям общ делител (НОД):

$$gcd(a,b)=\begin{cases} a, & a=b \\ gcd(a-b,b), & a>b \\ gcd(a,b-a), & a<b \end{cases}$$

Какво забелязваме във всичките примери?

- Показва се решението на най-простите случаи (дъно на рекурсията)
  - Например при n=0 имаме n!=1
- Показва се как се свежда сложна задача към една или няколко по-прости задачи (стъпка)

Рекурсия в програмирането

- Рекурсивна функция: функция, която извиква себе си
  - Пример: факториел – същата идея

```
int fact(int n) {
    if (n == 0) return 1; // дъно
    return n * fact(n - 1);
}
```

Рекурсия в програмирането (2)

- По-точна дефиниция: функция, която извиква себе си пряко или косвено
  - Следващият пример е илюстративен:

```
bool is_odd(unsigned int); // декларация
bool is_even(unsigned int n)
{
    if (n == 0) return true;
    else return is_odd(n – 1);
}
bool is_odd(unsigned int n)
{
    if (n == 0) return false;
    else return is_even(n – 1);
}
```

- Какво ще отпечата следната функция, ако я извикаме със стойност *n* = 2?

```
void test(int n)
{
    cout << "Before: " << n << endl;
    if (n > 0)
        test(n – 1);
    cout << "After: " << n << endl;
}
```

- Отговор:

```
Before: 2
Before: 1
Before: 0
After: 0
After: 1
After: 2
```

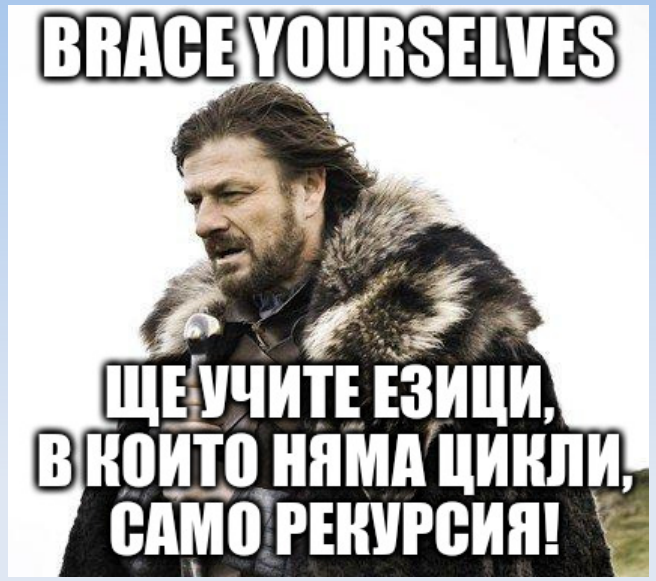
- Обяснение:
  - Извиква се test(2), която отпечатва `Before: 2`
  - После се извиква test(1), която отпечатва 4 реда
  - Накрая test(2) довършва изпълнението си – отпечатва `After: 2`
  - Разсъжденията за test(1) са аналогични – отпечатва `Before: 1`, извиква test(0), отпечатва `After: 1`

Още примери

- Повдигане на степен
  - (математическата дефиниция вече я видяхме)
- Най-голям общ делител (НОД) на две естествени числа
- Функция, която отпечатва двоичния запис на дадено естествено число

Рекурсия vs цикли

- Всяка програма с цикли може да се напише с рекурсия и обратно
  - В C++ примерните задачи е по-добре да се напишат с цикли
  - След малко ще видим примери, в които рекурсията ще ни улесни значително
- Почти всички програмни езици поддържат рекурсия
  - В някои езици дори няма цикли



← Твърдението е за Инф. и ИС

- Нека видим и примери с масиви:
  - Намиране на сума
  - Проверка за съществуване на елемент
  - Проверка за монотонно нарастване
  - Проверка за различни елементи

- Визуална демонстрация на изпълнението на рекурсивна програма за отпечатване на стойностите на масив:
- https://tinyurl.com/recursion-demo
- Това е пълният адрес към същата демонстрация:

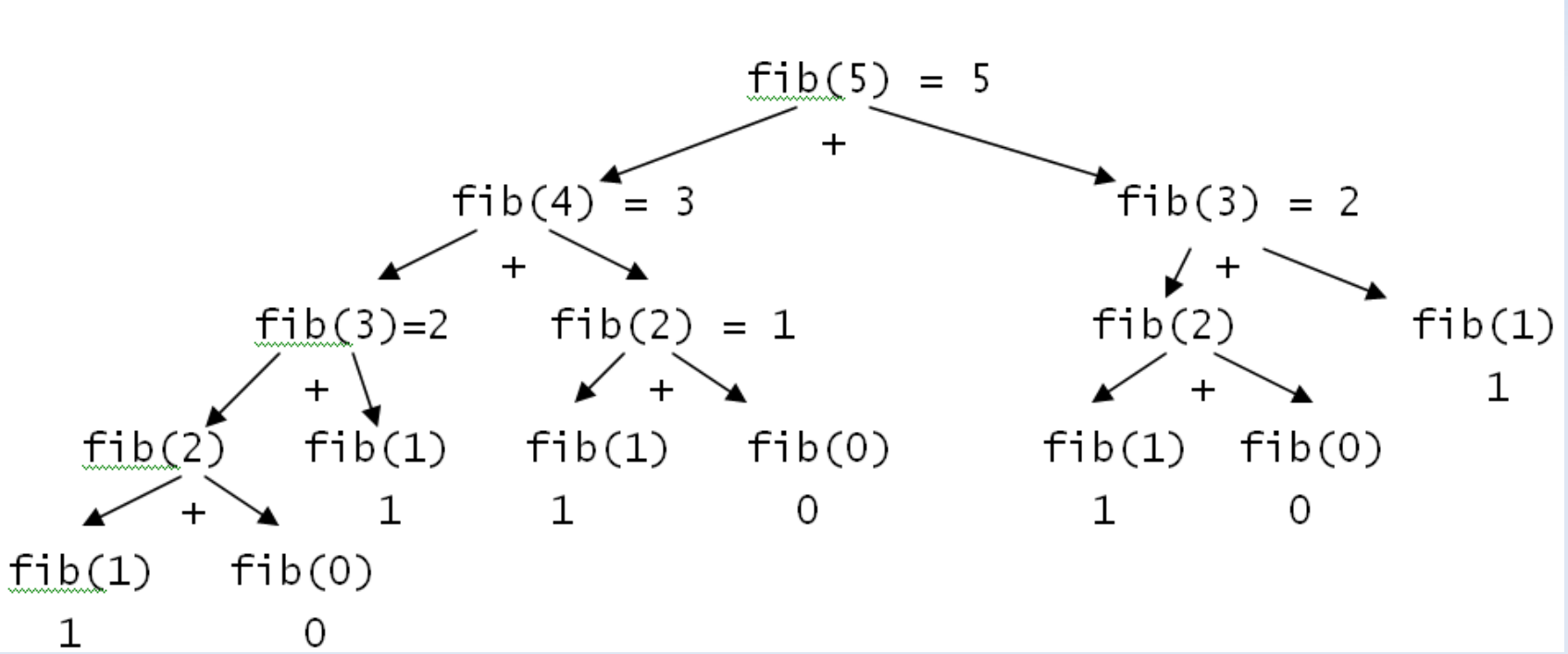
http://**pythontutor.com**/cpp.html#code=%23include%20%3Ciostream%3E%0Ausing%20namespace%20std%3B%0Avoid%20print(int%20*array,%20int%20length%29%0A%7B%0A%20%20if%20(length%20%3C%3D%200%29%0A%20%20%20%20return%3B%0A%20%20cout%20%3C%3C%20array%5B0%5D%3B%0A%20%20print(array%20%2B%201,%20length%20-%201%29%3B%0A%7D%0Aint%20main(%29%0A%7B%0A%20%20int%20a%5B3%5D%20%3D%20%7B5,%206,%207%7D%3B%0A%20%20print(a,%203%29%3B%0A%20%20return%200%3B%0A%7D&curInstr=1&mode=display&origin=opt-frontend.js&py=cpp&rawInputLstJSON=%5B%5D

Нелинейна рекурсия

- Дотук във всички примери функциите извършваха най-много едно извикване на себе си по време на тяхното изпълнение
  - Дори и в примера с НОД имахме линейна рекурсия
- Никой не ни спира да направим няколко извиквания
  - Така най-сетне ще видим примери, при които рекусивното решение е много по-лесно

Примери (1)

- Числа на Фибоначи
  - Лесно, но неефективно решение – с нелинейна рекурсия



  - Ефективно решение – с линейна рекурсия
  - Спокойно, следващият пример вече ще е хубав

Примери (2)

- Този и следващите примери са предимно за спец. Инф. и ИС
- Изчисляване на израз със скоби от следния вид:

  израз ::= <цифра> | (<израз>{+|-|*|/}<израз>)

  например: 5, (2+2), ((3+4)*(2-1))

Примери (3)

- Търсене на файл с дадено име в дадена директория и нейните поддиректории – да разгледаме само идеята
- Догодина по СДП ще се прилага същият алгоритъм за обхождане на дървовидни структури

- Хакнали сме компютъра на преподавателя и сме намерили файла със задачите за контролното
- Да, но файлът е защитен с парола
- Всички възможни пароли на света са краен брой – значи можем да ги пробваме една по една?
- Ако паролата е трудна и нямаме мощна техника, би ни отнело много, много години
- Ако обаче е нещо от рода на 1234, ще успеем

Пълно изчерпване

- Пълно изчерпване – Brute Force
- С този метод принципно може да се решат какви ли не задачи
- Генерираме всички възможни решения и за всяко от тях проверяваме дали е вярното

Примери (4)

- Генериране на всички редици от *n* естествени числа, всяко от които е по-малко от *k*
- Пример: при *n*=3 и *k*=10 получаваме 000, 001, 002, … 998, 999
- Както се вижда, дори при неголеми *n* и *k* броят на редиците става огромен
  - Затова пълното изчерпване не е ефективен начин за решаване на проблеми
  - Но пък е лесен

```
int sequence[1000], n, k;
void generate(int i)
{
    if (i == n) // край - намерено е решение -
                // отпечатваме го
    {
        for (int j = 0; j < n; j++)
            cout << sequence[j] << " ";
        cout << endl;
    }
    else
    {
        for (int j = 0; j < k; j++)
        {
            sequence[i] = j;
            generate(i + 1);
        }
    }
}
```

- Извикване: `cin >> n >> k; generate(0);`

Ами сега?

- Попадаме в лабиринт
- Искаме да излезем от него
- Какво ще направим?
  - Не, няма да звънним на приятел/мама/тати
  - Не, не можем да се катерим по стените
- Ще използваме РЕКУРСИЯ и тя ще ни спаси!

Търсене с връщане назад

- Ще използваме търсене с връщане назад (Backtracking)
- С проба и грешка се опитваме да намерим решението на задачата:
  - ако имаме няколко варианта как да продължим, избираме един от тях (стъпка напред, проба)
  - когато се окажем без никакъв избор, се връщаме и коригираме последния направен избор (стъпка назад, грешка)
  - ако получим желания резултат – намерено е решение; спираме или търсим и други решения
  - ако се върнем в началото – няма решение

Сравнение с пълното изчерпване

- Судоку – може да го решим по различни начини:
  - С функцията `generate`, но ще е много бавно
  - С backtracking

Задачи за лабиринт

- Ще ги представяме като булеви матрици
  - Нека true е проходимо квадратче, а false – непроходимо (стените ще са толкова дебели, колкото са широки коридорите)
  - Нека от дадено квадратче можем да стъпваме в друго, което има обща стена
- Пример:
- ```
  bool labyrinth[6][4] = {
      {1, 0, 1, 0},
      {1, 1, 1, 1},
      {0, 1, 0, 1},
      {1, 1, 1, 1},
      {1, 0, 1, 0},
      {1, 0, 1, 1}
  };
  ```
  ```

Задачи за лабиринт

- Да се провери дали има път (ацикличен) в лабиринт между дадени две квадратчета
- Да се намерят всички пътища в лабиринт между дадени две квадратчета
- Да се намери най-краткият път между дадени две квадратчета
- Същите задачи, но с ходове на коня (от шахмата)

Предимства на рекурсията

- Елегантен код
- Удобство при решаването на задачи, които са дефинирани рекурсивно
- Удобна за реализиране на backtracking
- Удобна за алгоритми от тип “разделяй и владей”
  - Напр. догодина по СДП ще се изучават Quicksort и Merge Sort
- Удобство при доказване на математически свойства

Недостатъци на рекурсията

- Използване на повече памет
  - При всяко извикване на функция се заделя нова памет
  - Но в някои случаи компилаторът оптимизира кода (Tail Recursion Optimization)
- При неправилно използване може да е неефективна
  - Първият пример за Фибоначи
- Понякога има нужда от помощни функции
  - Например в задачата за пресмятане на израз
- Изглежда трудна и страшна

Въпроси