Рекурсивні функції

Рекурсія - це спосіб організації обчислювального процесу, при якому функція в ході виконання звертається сама до себе.

Функція називається *рекурсивною*, якщо під час її виконання можливий повторний її виклик безпосередньо (прямий виклик) або шляхом виклику іншої функції, в якій міститься звертання до неї (непрямий виклик). *Прямою* (безпосередньою) рекурсією називається рекурсія, при якій всередині тіла деякої функції міститься виклик тієї ж функції.

```
void fn (int i)
 {
      ...
    fn (i);
      ...
}
```

Непрямою рекурсією називається рекурсія, що здійснює рекурсивний виклик функції шляхом ланцюга викликів інших функцій.

Якщо функція викликає сама себе, то в стеку створюється копія значень її параметрів, як і при виклику звичайної функції, після чого управління передається першому оператору функції. При повторному виклику цей процес повторюється.

Приклад - функція, що рекурсивно обчислює факторіал.

$$n! = 1*2*3*4*\ldots*(n-1)*n$$

```
#include <stdio.h>
double fact (int n)
{
    if (n<=1) return 1;
    return (fact (n-1)*n);
}
void main()
{
     int n;
    double value;
     printf ("N=");
    scanf ("%d", &n);
    value = fact(n);
    printf ("%d! = %g", n, value);
    getch();
}
```

Роботу рекурсивної функції ***fact()*** розглянемо на прикладі ***n=6*** **.** За рекурентним співвідношенням :

fact (n)= fact (n-1)*n .

Таким чином, щоб обчислити ***6!*** ми спочатку повинні обчислити ***5!***. В кроках 1-5 завершення обчислення кожний раз відкладається, а шостий крок є ключовим.

Отримане значення визначається безпосередньо, а не як факторіал іншого числа. Відповідно, можемо Повернутися від 6-ого кроку до 1-ого, послідовно використовуючи значення :

6). 1!=1	***5). 2!=2***	***4). 3!=6***
3). 4!=24	***2). 5!=120***	***1). 6!=720***

В рекурсивних функціях можна виділити дві серії кроків. Перша серія - це *кроки рекурсивного занурення* функції в саму себе до тих пір, поки вибраний параметр не досягне граничного значення. Ця вимога завжди повинна виконуватися, щоб функція не створила нескінченну послідовність викликів самої себе.

Кількість таких кроків називається *глибиною рекурсії*.

Друга серія - це *кроки рекурсивного виходу* до тих пір, поки вибраний параметр не досягне початкового значення. Вона, як правило забезпечує отримання проміжних і кінцевих результатів.

Приклад. Для обчислення степеня числа є рекурентна формула

$$a^n = a \cdot a^{n-1}, \quad a^0 = 1.$$

```
#include <stdio.h>
 double power (double x, long n)
    {   if (n == 0) return 1;
            if (n < 0) return power ( 1.0 / x, - -n);
       return  x * power (x, n - 1);
     }
void main()
  {    double x;     long n;
        while (scanf ("%lf %ld", &x, &n) == 2)
              {   printf("%lf\n", power (x, n));     }
  }
```

Розглянемо більш «розумну» рекурсивну функцію

$$a^n = \begin{cases} a \cdots a^{n-1}, & \text{if } n \text{ is odd} \\ (a^{n/2})^2, & \text{if } n \text{ is even} \end{cases}$$

Якщо позначити стрілкою «приводимо до», тоді для першої рекурсії отримаємо ланцюжок довжини 12

$$a^{12} \to a^{11} \to a^{10} \to a^9 \to a^8 \to a^7 \to a^6 \to a^5 \to a^4 \to a^3 \to a^2 \to a^1 \to a^0.$$

Для другої рекурсії – послідовність з 5 кроків

$$a^{12} \to a^6 \to a^3 \to a^2 \to a^1 \to a^0.$$

Для великих значень ***n*** маємо значну перевагу. Наприклад, першою рекурсиєю a^{10000} обчислюється за 10000 кроків, а другою - за 19 кроків.

```
double power (double x, long n)
{
     if (n == 0) return 1;
    if (n < 0) return power ( 1 / x,  - -n);
    if (n % 2) return x * power (x, n - 1);
  return power(x * x, n / 2);
}
```

Оптимальний алгоритм без рекурсії

```
double power (double x, long n)
{     double a = 1;
  while(n) {
      if (n % 2)
            {     a *= x;
                  n--;       }
     else
            {      x *= x;
                   n /= 2;      }
   }
  return a;
}
```