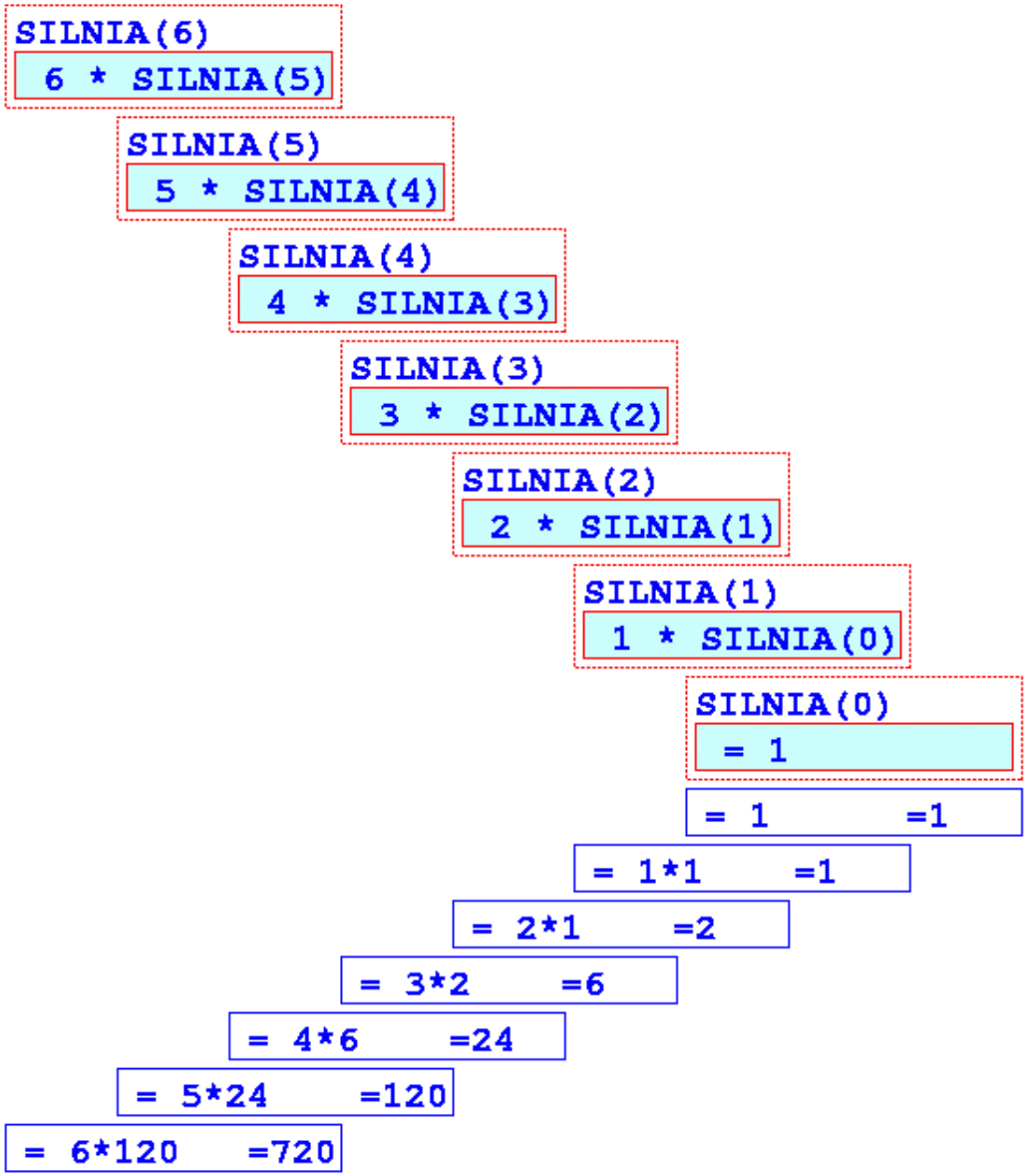
Rekurencja, zwana także rekursją (ang. recursion, z łac. recurrere, przybiec z powrotem) to w logice, programowaniu i w matematyce odwoływanie się np. funkcji lub definicji do samej siebie.

W logice wnioskowanie rekurencyjne opiera się na założeniu istnienia pewnego stanu początkowego oraz zdania (lub zdań) stanowiącego podstawę wnioskowania (przy czym aby cały dowód był poprawny zarówno reguła jak i stan początkowy muszą być prawdziwe). Istotą rekurencji jest tożsamość dziedziny i przeciwdziedziny reguły wnioskowania, wskutek czego wynik wnioskowania może podlegać tej samej regule zastosowanej ponownie.

Przykład programu wykorzystującego rekurencje:

```
PROGRAM       REK;
VAR ZMIENNA:BYTE;

FUNCTION SILNIA(N: BYTE):BYTE;
BEGIN
        IF(N < 1) THEN
                SILNIA := 1
        ELSE
                SILNIA := N * SILNIA(N– 1);
END;
BEGIN
        WRITELN('PODAJ WARTOSC');
        READLN(ZMIENNA);
        WRITELN('SILNIA WYNOSI:', SILNIA(ZMIENNA));
        READLN;
END.
```



Uwaga na niebezpieczeństwo rekursji:

```
FUNCTION F(A:BYTE):BYTE;
BEGIN
        F := 0;
        IF(A > -1) THEN F := A * F(A+1);
END;
```

Niebezpieczeństwa rekurencji

Programując programy lub funkcje rekurencyjne należy uważać na pewne dodatkowe efekty uboczne, które można niechcący wywołać. Kilka z nich opisano poniżej; praktyka programistyczna dostarczyć może więcej takich przykładów.

Ciąg Fibonacciego

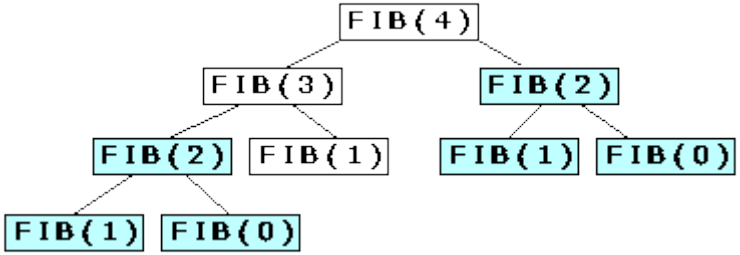
Obliczanie parametrów ciągu tego i innych, podobnych typów, wykonane bezpośrednio na podstawie wzoru matematycznego, może powodować problemy w postaci zbyt wielu dublujących się obliczeń. Ciąg Fibonacciego jest zdefiniowany rekurencyjnie, w sposób analogiczny do definicji funkcji silnia:

```
F(0) = 1 F(1) = 1 F(n) = F(n-1) + F(n-2)
```

Zadanie obliczania elementów takiego ciągu można w języku C zapisać niemal dokładnie tak samo, jak wyraża to wzór definicyjny:

```
unsigned long int FIB( int n ){ if (x<2) return 1; else return FIB(n-1) + FIB(n-2); }
```

Schemat wywołań rekurencyjnych, jak wykaże prosta analiza kodu, doprowadzi do następującego drzewa wywołań:



Drzewo wywołań funkcji *FIB* dla parametru 4.

Widać od razu, że gałęzie zaznaczone kolorem wykonują się dwa razy. Zupełnie niepotrzebnie. W sumie, wywołanie funkcji FIB dla większych parametrów n, spowoduje że wykonane zostanie w przybliżeniu 2n obliczeń, co jest z oczywistych powodów nieefektywne. Dlatego należy pamiętać, że nie jest dobrą metodą programowanie rekurencyjne tam, gdzie wystarczą proste funkcje iteracyjne.

Nieoczywistość kodu

Istnieją przypadki w których program napisany jest poprawnie, jednak przewidywanie sposobu jego działania staje się nieco trudne. Pierwszym takim przykładem była funkcja obliczająca elementy ciągu Fibonacciego, teraz coś jeszcze bardziej dziwnego - funkcja MacCarthy'ego:

```
unsigned long int MC( int n ){ if (n>100) return n-10; else return MC( MC(n+11) ); }
```

Analiza kodu doprowadza do zawrotu głowy - funkcja jest jakaś dziwna: oto, dla wartości n większych od 100 zwraca ładnie wartość n-10 - jest to prawidłowe i mające szansę nastąpić zakończenie procesu wywołań rekurencyjnych; dla większych parametrów n zwracane jest wywołanie rekurencyjne z parametrem, którym jest wartość wywołania rekurencyjnego dla wartości n+11. Przebieg wywołań takiej funkcji jest na pierwszy rzut oka trudny do określenia, ale po wnikliwej analizie można dość do wniosku, że:

- dla wszystkich n większych od 100 funkcja wykona się tylko raz
- dla n równego 100 funkcja wywoła się dwa razy: raz z wartością 111 (n+11), drugi raz z wartością 101 (n-10).
- dla n równego 99 funkcja wywoła się: z wartością 110 i 100, z wartością 111 i 101 - cztery razy
- dla n równego 98 - sześć razy, itp..., zgodnie z nieścisłą zależnością empiryczną 1+(100-n)*2.

Ilość wywołań przedstawiona na wykresie rzeczywiście potwierdza analizę, chociaż wynik nie jest oczywisty. Stąd właśnie niebezpieczeństwo - można napisać kod, który działa doskonale, ale trudno powiedzieć, co będzie się działo dla niektórych wartości parametrów wejściowych.

Zły warunek końcowy = wywołania nieskończone

Oto kolejny przykład, w którym niedostatecznie przemyślano warunek zakończenia procesu rekurencyjnego.

```
int SDW( int n ){ if (n==1) return 1; else if( (n%2)==0 ) return n*SDW(n-2); else return n*SDW(n-1); }
```