Δ.Π.Θ Δομημένος Προγραμματισμός

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΑΡΑΓΩΓΗΣ & ΔΙΟΙΚΗΣΗΣ

Αναδρομικές Συναρτήσεις

Dr. Αθανάσιος Μπαλαφούτης
Εργαστηριακό Διδακτικό Προσωπικό
Τομέας Συστημάτων Παραγωγής
Εργαστήριο Ρομποτικής και Αυτοματισμών
abalafou@pme.duth.gr
Γραφείο 304, τηλ.: 25410 – 79892

Αναδρομή

Είναι η διαδικασία με την οποία μια συνάρτηση καλεί τον εαυτό της, είτε άμεσα είτε έμμεσα.

Παράδειγμα:

Υπολογισμός παραγοντικού

```
n! = n x (n-1) x (n-2) x ... x 1
```

Αναδρομή

Είναι η διαδικασία με την οποία μια συνάρτηση καλεί τον εαυτό της, είτε άμεσα είτε έμμεσα.

Παράδειγμα:

Υπολογισμός παραγοντικού

$$n! = n \times \underbrace{(n-1) \times (n-2) \times \ldots \times 1}_{(n-1)!}$$

Αναδρομή

Είναι η διαδικασία με την οποία μια συνάρτηση καλεί τον εαυτό της, είτε άμεσα είτε έμμεσα.

Παράδειγμα:

Υπολογισμός παραγοντικού

$$n! = n \times \underbrace{(n-1) \times (n-2) \times \ldots \times 1}_{(n-1)!}$$

Αναδρομική Συνάρτηση

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

Αναδρομή

Αναδρομική Συνάρτηση

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

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

Αναδρομή

Αναδρομική Συνάρτηση

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

Έστω ότι θέλω να υπολογίσω αναδρομικά το 4!

Συνάρτηση	Επιστροφή	Κατάσταση
factorial(4)	4 * factorial(3)	ανοικτή

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

Αναδρομή

Αναδρομική Συνάρτηση

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

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

Έστω ότι θέλω να υπολογίσω αναδρομικά το 4!

Συνάρτηση	Επιστροφή	Κατάσταση
factorial(4)	4 * factorial(3)	ανοικτή
factorial(3)	3 * factorial(2)	ανοικτή

Αναδρομή

Αναδρομική Συνάρτηση

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

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

Έστω ότι θέλω να υπολογίσω αναδρομικά το 4!

Συνάρτηση	Επιστροφή	Κατάσταση
factorial(4)	4 * factorial(3)	ανοικτή
factorial(3)	3 * factorial(2)	ανοικτή
factorial(2)	2 * factorial(1)	ανοικτή

Αναδρομή

Αναδρομική Συνάρτηση

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

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

Έστω ότι θέλω να υπολογίσω αναδρομικά το 4!

Συνάρτηση	Επιστροφή	Κατάσταση
factorial(4)	4 * factorial(3)	ανοικτή
factorial(3)	3 * factorial(2)	ανοικτή
factorial(2)	2 * factorial(1)	ανοικτή
factorial(1)	1 * factorial(0)	ανοικτή

Αναδρομή

Αναδρομική Συνάρτηση

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

Έστω ότι θέλω να υπολογίσω αναδρομικά το 4!

Συνάρτηση	Επιστροφή	Κατάσταση
factorial(4)	4 * factorial(3)	ανοικτή
factorial(3)	3 * factorial(2)	ανοικτή
factorial(2)	2 * factorial(1)	ανοικτή
factorial(1)	1 * factorial(0)	ανοικτή
factorial(0)	1	ολοκληρωμένη

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

Αναδρομή

Αναδρομική Συνάρτηση

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

Έστω ότι θέλω να υπολογίσω αναδρομικά το 4!

Συνάρτηση	Επιστροφή	Κατάσταση
factorial(4)	4 * factorial(3)	ανοικτή
factorial(3)	3 * factorial(2)	ανοικτή
factorial(2)	2 * factorial(1)	ανοικτή
factorial(1)	1 * 1	ολοκληρωμένη

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

Αναδρομή

Αναδρομική Συνάρτηση

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

Έστω ότι θέλω να υπολογίσω αναδρομικά το 4!

Συνάρτηση	Επιστροφή	Κατάσταση
factorial(4)	4 * factorial(3)	ανοικτή
factorial(3)	3 * factorial(2)	ανοικτή
factorial(2)	2 * 1	ολοκληρωμένη

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

Αναδρομή

Αναδρομική Συνάρτηση

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

Έστω ότι θέλω να υπολογίσω αναδρομικά το 4!

Συνάρτηση	Επιστροφή	Κατάσταση
factorial(4)	4 * factorial(3)	ανοικτή
factorial(3)	3 * 2	ολοκληρωμένη

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

Αναδρομή

Αναδρομική Συνάρτηση

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

Έστω ότι θέλω να υπολογίσω αναδρομικά το 4!

Συνάρτηση	Επιστροφή	Κατάσταση
factorial(4)	4 * 6	ολοκληρωμένη

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

Αναδρομή vs Επανάληψη

Αναδρομή

```c
int factorial(int n){
   if (n == 0)
      return 1;
   else
      return n * factorial(n-1);
}

int main(){
   printf("%d", factorial(4));
   return 0;
}
```

Επανάληψη

```c
int factorial(int n){
   int res = 1;
   while(n != 0){
      res = res *n;
      n--;
   }
   return res;
}

int main(){
   printf("%d", factorial(4));
   return 0;
}
```

Αναδρομή vs Επανάληψη

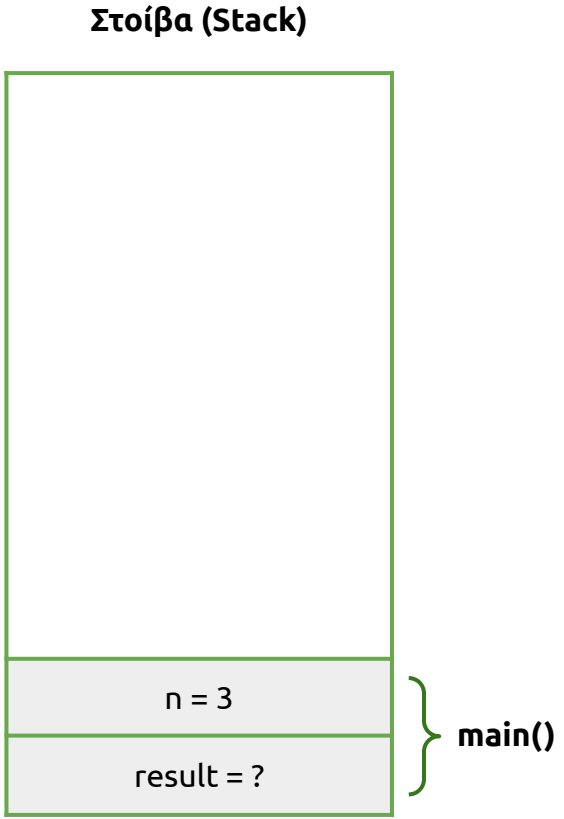
- Κάθε πρόβλημα μπορεί να μοντελοποιηθεί και με αναδρομή αλλά και με επανάληψη.
- Η αναδρομή συνήθως απαιτεί τη συγγραφή μικρότερου σε έκταση κώδικα.
- Η αναδρομή δεσμεύει περισσότερη μνήμη.

Αναδρομή - Παράδειγμα


```c
#include <stdio.h>

int fun(int n){
   if(n == 1)
      return 1;
   else
      return 1 + fun(n-1);
}

int main() {
   int n = 3;
   int result = fun(n);
   printf("Αποτέλεσμα: %d\n", result);
   return 0;
}
```

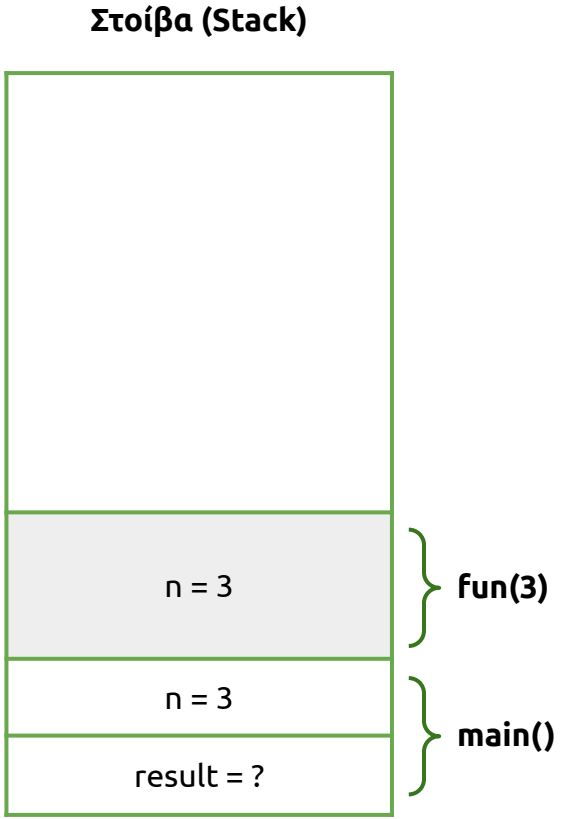


Αναδρομή - Παράδειγμα


```c
#include <stdio.h>

int fun(int n){
   if(n == 1)
      return 1;
   else
      return 1 + fun(n-1);
}

int main() {
   int n = 3;
   int result = fun(n);
   printf("Αποτέλεσμα: %d\n", result);
   return 0;
}
```

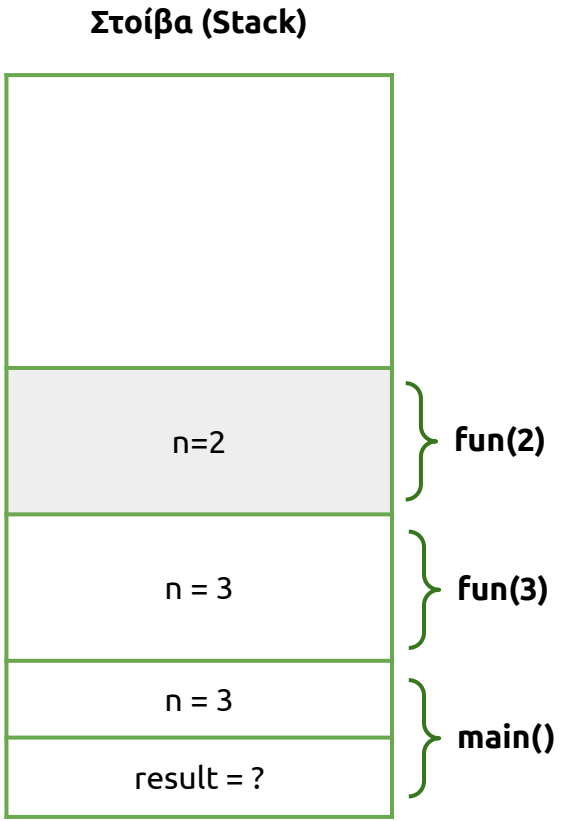


Αναδρομή - Παράδειγμα


```c
#include <stdio.h>

int fun(int n){
   if(n == 1)
      return 1;
   else
      return 1 + fun(n-1);
}

int main() {
   int n = 3;
   int result = fun(n);
   printf("Αποτέλεσμα: %d\n", result);
   return 0;
}
```

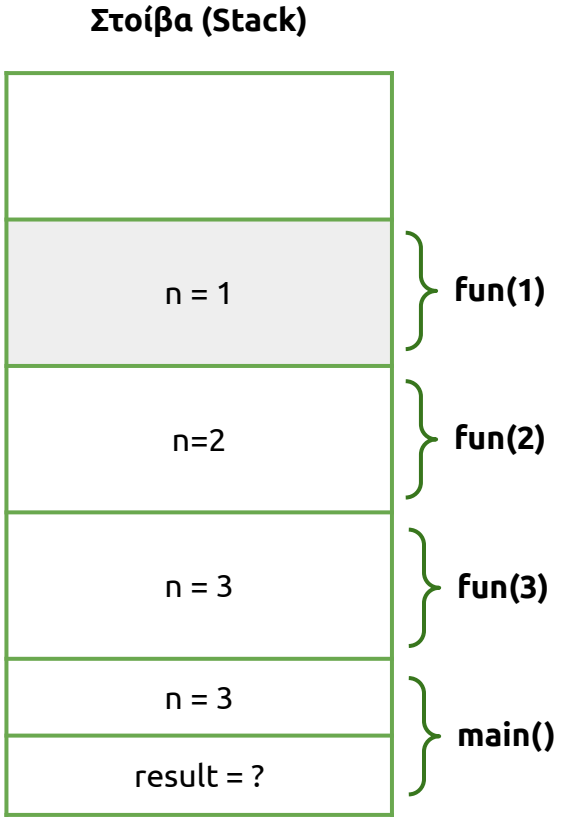


Αναδρομή - Παράδειγμα


```c
#include <stdio.h>

int fun(int n){
   if(n == 1)
      return 1;
   else
      return 1 + fun(n-1);
}

int main() {
   int n = 3;
   int result = fun(n);
   printf("Αποτέλεσμα: %d\n", result);
   return 0;
}
```

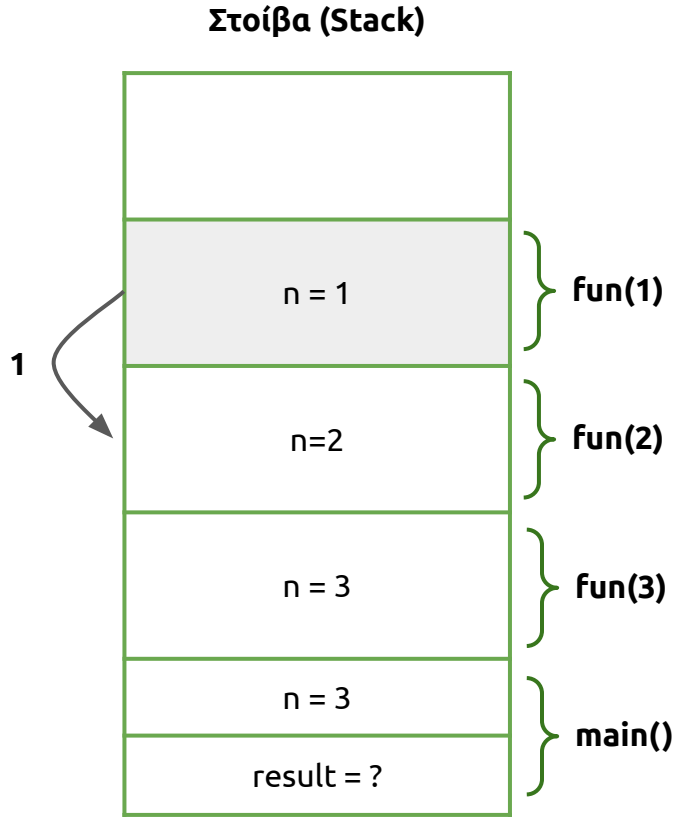


Αναδρομή - Παράδειγμα


```c
#include <stdio.h>

int fun(int n){
   if(n == 1)
      return 1;
   else
      return 1 + fun(n-1);
}

int main() {
   int n = 3;
   int result = fun(n);
   printf("Αποτέλεσμα: %d\n", result);
   return 0;
}
```

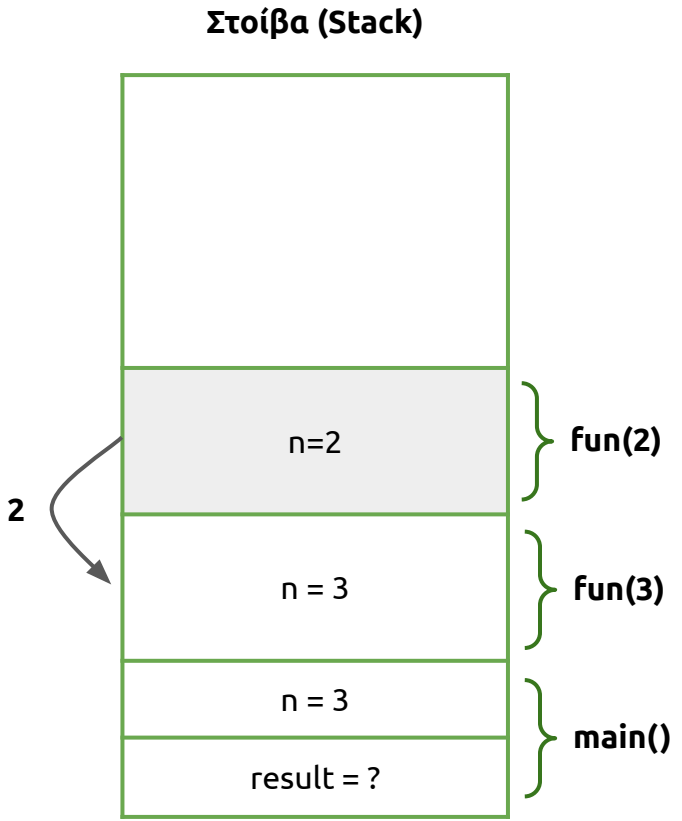


Αναδρομή - Παράδειγμα


```c
#include <stdio.h>

int fun(int n){
   if(n == 1)
      return 1;
   else
      return 1 + fun(n-1);
}

int main() {
   int n = 3;
   int result = fun(n);
   printf("Αποτέλεσμα: %d\n", result);
   return 0;
}
```

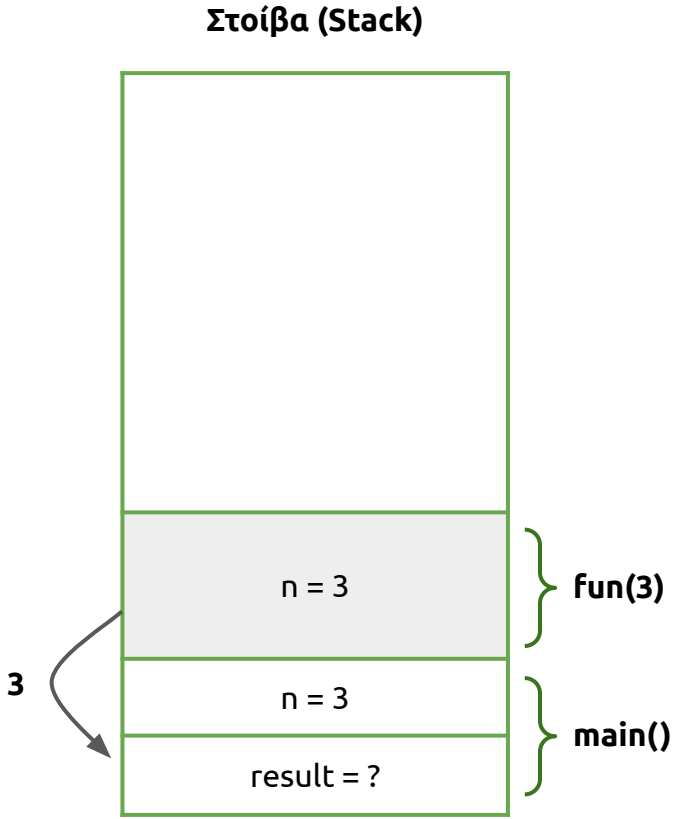


Αναδρομή - Παράδειγμα


```c
#include <stdio.h>

int fun(int n){
   if(n == 1)
      return 1;
   else
      return 1 + fun(n-1);
}

int main() {
   int n = 3;
   int result = fun(n);
   printf("Αποτέλεσμα: %d\n", result);
   return 0;
}
```

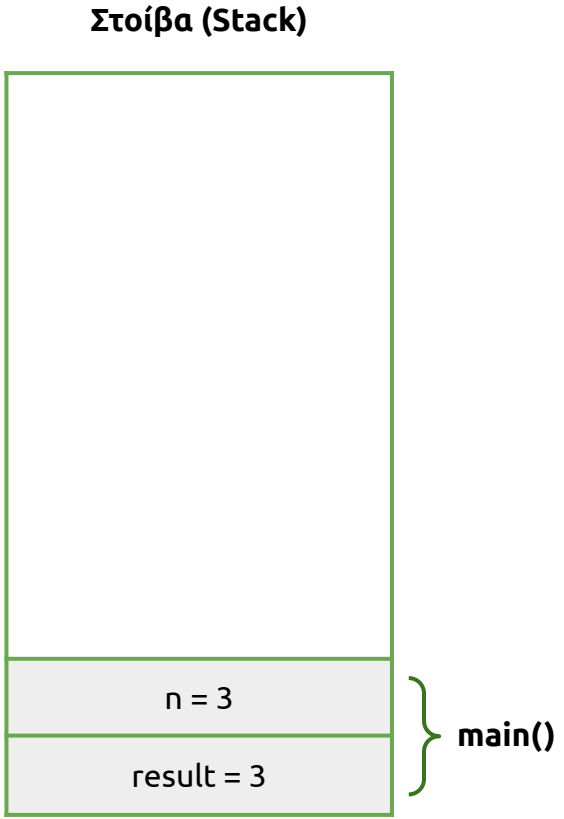


Αναδρομή - Παράδειγμα


```c
#include <stdio.h>

int fun(int n){
   if(n == 1)
      return 1;
   else
      return 1 + fun(n-1);
}

int main() {
   int n = 3;
   int result = fun(n);
   printf("Αποτέλεσμα: %d\n", result);
   return 0;
}
```



Άσκηση 7.1

Χρησιμοποιώντας αναδρομή, να γραφεί μία συνάρτηση σε γλώσσα C για την εύρεση του αθροίσματος όλων των φυσικών αριθμών μέχρι ένα γνωστό όριο `n`
(δηλ. το άθροισμα `1+2+...+n`).

Η συνάρτηση στη συνέχεια θα καλείται από τη `main()`.

Άσκηση 7.1


```c
#include <stdio.h>
int sum(int n) {
    if (n == 1)
        return 1;
    return n + sum(n - 1);  // Αναδρομική κλήση
}
int main() {
    int n;
    do {
        printf("Δώσε έναν φυσικό αριθμό: ");
        scanf("%d", &n);
    } while (n < 0);
    printf("Το άθροισμα των αριθμών 1 έως %d είναι: %d\n", n, sum(n));
    return 0;
}
```

Άσκηση 7.2

Χρησιμοποιώντας αναδρομή, να γραφεί μία συνάρτηση σε γλώσσα C για την εύρεση του παραγοντικού ενός φυσικού αριθμού.

Η συνάρτηση στη συνέχεια θα καλείται από τη `main()`.

Άσκηση 7.2


```c
#include <stdio.h>

long long factorial(int n) {
    if (n == 0 || n == 1)
        return 1;
    return n * factorial(n - 1);  // Αναδρομική κλήση
}
int main() {
    int n;
    do {
        printf("Δώσε έναν φυσικό αριθμό: ");
        scanf("%d", &n);
    } while (n < 0);
    printf("%d! = %lld\n", n, factorial(n));
    return 0;
}
```

Άσκηση 7.3

Η ακολουθία `Fibonacci` είναι μια αριθμητική ακολουθία όπου κάθε όρος προκύπτει από το άθροισμα των δύο προηγούμενων όρων:

`F(0) = 0, F(1) = 1`

Για `n ≥ 2`, κάθε επόμενος όρος δίνεται από τον τύπο:

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

Παραδείγματα των πρώτων όρων της ακολουθίας:

`0,1,1,2,3,5,8,13,21,34,55,...`

Χρησιμοποιώντας αναδρομή, να γραφεί μία συνάρτηση σε γλώσσα C για την εμφάνιση όλων των όρων της ακολουθίας `Fibonacci` μέχρι να συμπληρωθεί ένα γνωστό πλήθος `n` όρων της ακολουθίας.

Η συνάρτηση στη συνέχεια θα καλείται από τη `main()`.

Άσκηση 7.3 - fibonacci


```c
#include <stdio.h>

int fibonacci(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibonacci(n - 1) + fibonacci(n - 2);  // Αναδρομική κλήση
}
```

Άσκηση 7.3 - main


```c
int main() {
    int n;

    do {
        printf("Δώσε τον αριθμό των όρων της ακολουθίας: ");
        scanf("%d", &n);

    } while (n <= 0);

    printf("\nΗ ακολουθία Fibonacci μέχρι τον %d-οστό όρο είναι:\n", n);
    for (int i = 0; i < n; i++) {
        printf("F(%d) = %d\n", i, fibonacci(i));
    }
    return 0;
}
```

Άσκηση 7.4

Χρησιμοποιώντας αναδρομή, να γραφεί μία συνάρτηση σε γλώσσα C για την εύρεση του αθροίσματος όλων των ψηφίων ενός θετικού ακεραίου αριθμού.

Η συνάρτηση στη συνέχεια θα καλείται από τη `main()`.

Άσκηση 7.4 - sum_digits


```c
#include <stdio.h>

int sum_digits(int n) {
    if (n == 0) {
        return 0;
    }
    return (n % 10) + sum_digits(n / 10);
}
```

Άσκηση 7.4 - main


```c
int main() {
    int number;

    do {
        printf("Δώσε έναν θετικό ακέραιο αριθμό: ");
        scanf("%d", &number);
    } while (number <= 0);

    int result = sum_digits(number);
    printf("Το άθροισμα των ψηφίων του αριθμού %d είναι: %d\n", number, result);

    return 0;
}
```

Άσκηση 7.5

Χρησιμοποιώντας αναδρομή, να γραφεί μία συνάρτηση σε γλώσσα C για τον υπολογισμό της δύναμης οποιουδήποτε αριθμού.

Η συνάρτηση στη συνέχεια θα καλείται από τη `main()`.

Άσκηση 7.5

Χρησιμοποιώντας αναδρομή, να γραφεί μία συνάρτηση σε γλώσσα C για τον υπολογισμό της δύναμης οποιουδήποτε αριθμού.

Η συνάρτηση στη συνέχεια θα καλείται από τη `main()`.

Παράδειγμα:

$2^n = 2 \times 2 \times 2 \times \ldots \times 2$

$2^{(n-1)}$

Άσκηση 7.5

Χρησιμοποιώντας αναδρομή, να γραφεί μία συνάρτηση σε γλώσσα C για τον υπολογισμό της δύναμης οποιουδήποτε αριθμού.

Η συνάρτηση στη συνέχεια θα καλείται από τη `main()`.

Παράδειγμα:

$$2^n = 2 \times \underbrace{2 \times 2 \times \ldots \times 2}_{2^{(n-1)}}$$

Αναδρομική Συνάρτηση

$$2^n = \begin{cases} 2^n = 2 \times 2^{(n-1)} \\ \\ 1, \text{ αν } n = 0 \end{cases}$$

Άσκηση 7.5

Αναδρομή για τον υπολογισμό της δύναμης 2^3:

1. Πρώτη κλήση:
   `power(2, 3)`
   Η συνάρτηση δεν είναι στην βασική περίπτωση και ο εκθέτης είναι μεγαλύτερος από 0, οπότε καλείται αναδρομικά με τον εκθέτη μειωμένο κατά 1:
   `2 * power(2, 2)`

2. Δεύτερη κλήση:
   `power(2, 2)` → `2 * power(2, 1)`

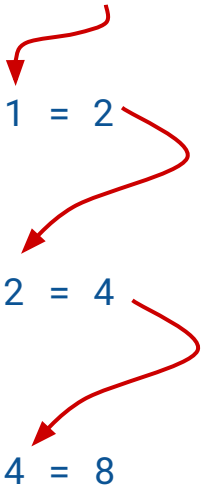
3. Τρίτη κλήση:
   `power(2, 1)` → `2 * power(2, 0)`

4. Τέταρτη κλήση (Βασική περίπτωση):
   `power(2, 0)` Εδώ, η συνάρτηση φτάνει στην βασική περίπτωση και επιστρέφει 1.

Άσκηση 7.5

Αποσύνθεση των κλήσεων:

Αφού η τέταρτη κλήση επιστρέψει 1, οι προηγούμενες κλήσεις αρχίζουν να επιστρέφουν αποτελέσματα:



- `power(2, 1) = 2 * 1 = 2`
- `power(2, 2) = 2 * 2 = 4`
- `power(2, 3) = 2 * 4 = 8`

Άσκηση 7.5 - power


```c
#include <stdio.h>

double power(double base, int exponent) {
    if (exponent == 0) {
        return 1;
    }
    else if (exponent > 0) {
        return base * power(base, exponent - 1);
    }
    else {
        return 1 / power(base, -exponent);
    }
}
```

Άσκηση 7.5 - main


```c
int main() {
    double base;
    int exponent;

    printf("Δώσε τη βάση: ");
    scanf("%lf", &base);

    printf("Δώσε τον εκθέτη: ");
    scanf("%d", &exponent);

    double result = power(base, exponent);
    printf("Το αποτέλεσμα του %.2lf^%d είναι: %.2lf\n", base, exponent, result);

    return 0;
}
```

Άσκηση 7.6

Χρησιμοποιώντας αναδρομή, να γραφεί μία συνάρτηση σε γλώσσα C για την εμφάνιση των στοιχείων ενός μονοδιάστατου πίνακα αριθμών.

Η συνάρτηση στη συνέχεια θα καλείται από τη `main()`.

Άσκηση 7.6


```c
#include <stdio.h>

void printArray(int *arr, int size) {

    if (size == 0)
        return;

    printf("%d ", *arr);
    printArray(arr + 1, size - 1);
}
```

```c
int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int size = sizeof(arr) / sizeof(arr[0]);

    printf("Τα στοιχεία του πίνακα είναι: ");
    printArray(arr, size);

    printf("\n");

    return 0;
}
```

Άσκηση 7.7

Χρησιμοποιώντας αναδρομή, να γραφεί μία συνάρτηση σε γλώσσα C για την εύρεση του μέγιστου στοιχείου ενός μονοδιάστατου πίνακα αριθμών.

Η συνάρτηση στη συνέχεια θα καλείται από τη `main()`.

Άσκηση 7.7 - findMax


```c
#include <stdio.h>

int findMax(int *arr, int size) {

    if (size == 1) {
        return *arr;
    }

    int maxOfRest = findMax(arr + 1, size - 1);

    if (*arr > maxOfRest) {
        return *arr;
    } else {
        return maxOfRest;
    }
}
```

Άσκηση 7.7 - main


```c
int main() {
    int arr[] = {1, 4, 7, 2, 9, 5};
    int size = sizeof(arr) / sizeof(arr[0]);

    int max = findMax(arr, size);

    printf("Το μέγιστο στοιχείο του πίνακα είναι: %d\n", max);

    return 0;
}
```

Άσκηση 7.8 - Έμμεση αναδρομή

Χρησιμοποιώντας αναδρομή, να γραφεί μία συνάρτηση σε γλώσσα C που να δέχεται ως είσοδο έναν αριθμό `n` από το `1` μέχρι και το `10`, και να εκτυπώνει:

- τον επόμενο αριθμό, όταν `n` περιττός (δηλαδή να προσθέτει `1`)
- τον προηγούμενο αριθμό, όταν `n` άρτιος (δηλαδή να αφαιρεί `1`)

Η συνάρτηση στη συνέχεια θα καλείται από τη `main()`.

Παράδειγμα εκτύπωσης: `2, 1, 4, 3, 6, 5, 8, 7, 10, 9`

Άσκηση 7.8


```c
#include <stdio.h>

int n = 1;

void odd() {
   if(n <= 10){
       printf("%d ", n+1);
       n++;
       even();
   }
}
```

```c
void even(){
   if(n <= 10){
       printf("%d ", n-1);
       n++;
       odd();
   }
}

int main() {

   ...
   return 0;
}
```

Άσκηση 7.8


```c
#include <stdio.h>

int n = 1;

void odd() {
   if(n <= 10){
       printf("%d ", n+1);
       n++;
       even();
   }
}
```

```c
void even(){
   if(n <= 10){
       printf("%d ", n-1);
       n++;
       odd();
   }
}

int main() {

   odd();
   return 0;
}
```

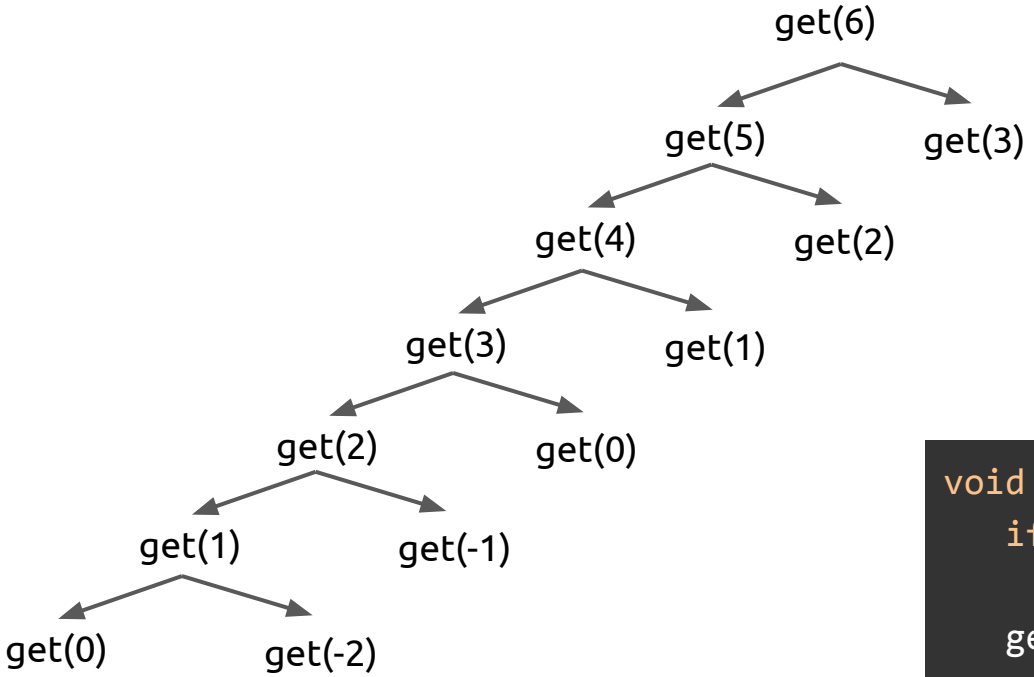
Άσκηση 7.9

Έστω η ακόλουθη αναδρομική συνάρτηση:

```
void get(int n){
   if(n < 1)
      return;
   get(n-1);
   get(n-3);
   printf("%d", n);
}
```

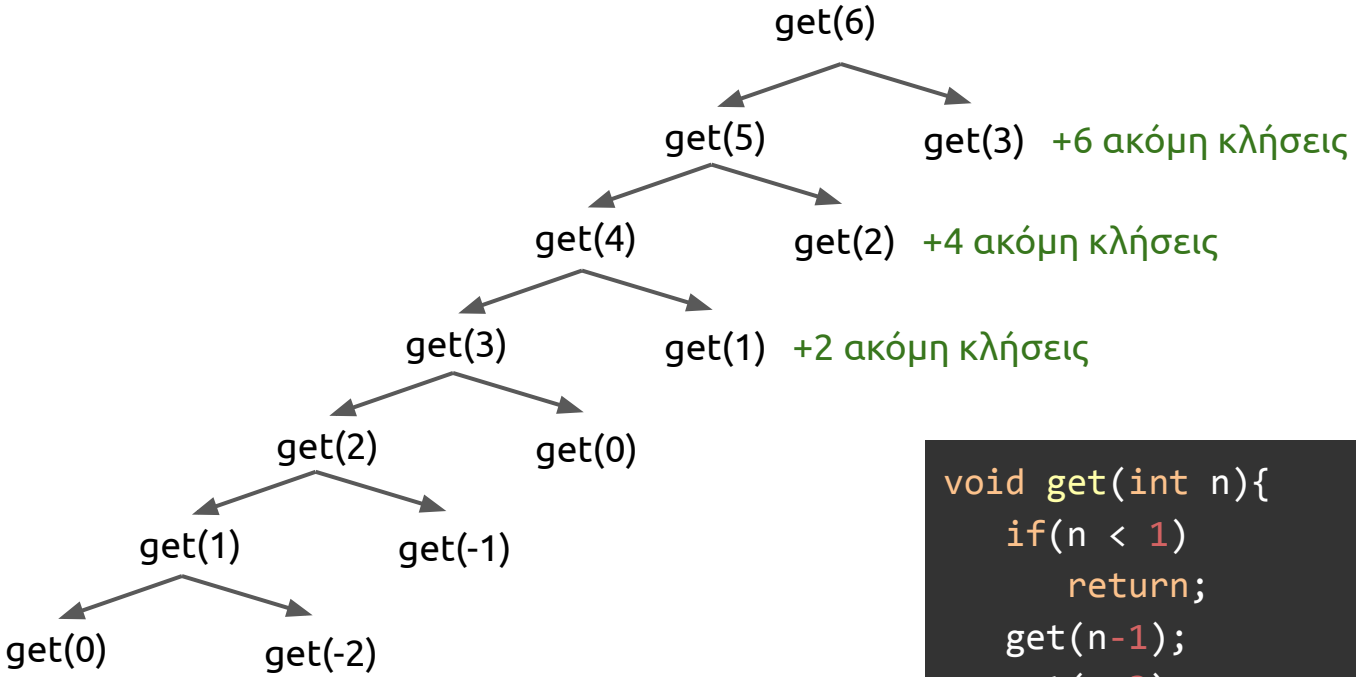
Αν η συνάρτηση `get(6)` κληθεί μέσα στη `main()`, πόσες φορές θα εκτελεστεί η `get()` (συμπεριλαμβανομένων των αναδρομικών κλήσεών της) πριν ολοκληρωθεί και επιστρέψει στη `main()`;

Άσκηση 7.9



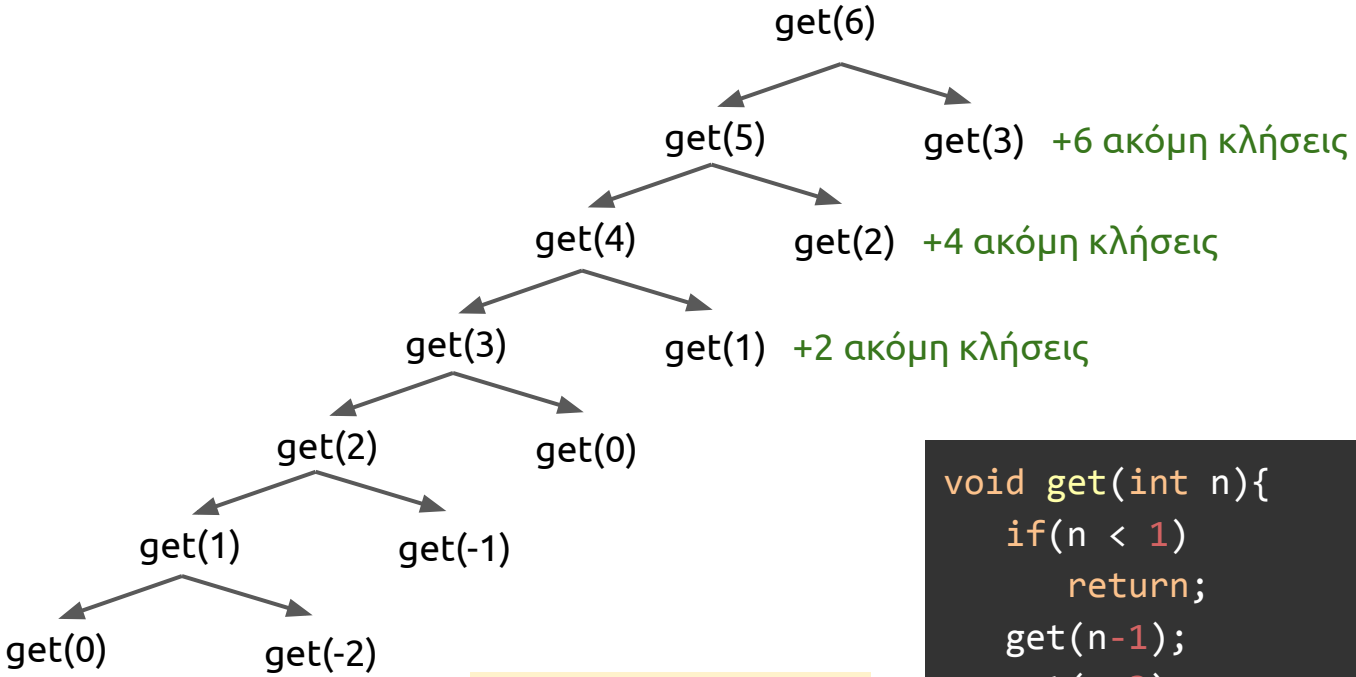

```
void get(int n){
   if(n < 1)
      return;
   get(n-1);
   get(n-3);
   printf("%d", n);
}
```

Άσκηση 7.9




```
void get(int n){
   if(n < 1)
      return;
   get(n-1);
   get(n-3);
   printf("%d", n);
}
```

Άσκηση 7.9



Σύνολο: **25 κλήσεις**

```
void get(int n){
   if(n < 1)
      return;
   get(n-1);
   get(n-3);
   printf("%d", n);
}
```

Άσκηση 7.10

Έστω η ακόλουθη αναδρομική συνάρτηση:

```
void fun1(int n){
   if(n > 1)
      fun1(n-1)
   for(int i = 0; i < n; i++)
      printf(" * ");
}
```

Αν η συνάρτηση `fun1(3)` κληθεί μέσα στη `main()`, πόσες αστεράκια θα εκτυπωθούν στην οθόνη;

Άσκηση 7.10

Έστω η ακόλουθη αναδρομική συνάρτηση:

```
void fun1(int n){
   if(n > 1)
      fun1(n-1)
   for(int i = 0; i < n; i++)
      printf(" * ");
}
```

Αν η συνάρτηση `fun1(3)` κληθεί μέσα στη `main()`, πόσες αστεράκια θα εκτυπωθούν στην οθόνη;

Θα εκτυπωθούν 6 αστεράκια