



Funkcije – SIU1 – i2 - Rekurzije

I2 – Prepoznati i koristiti rekurziju u rješavanju problema.

Problemski poticaj

- koliko se koraka napravi pri spuštanju s n -te stepenice ako se uvijek spušta jedna po jedna?
Opisati ponavljanje istog problema za $n - 1$.
- Rekurzija je jedan od onih programerskih koncepata koji se **može naizgled razumjeti dok nastavnik objašnjava**, a zatim se potpuno izgubiti kada treba samostalno pratiti izvršavanje.
- razumjeti drugi dio postupka: **nastajanje poziva, čekanje rezultata i povratak kroz prethodne pozive**.
- usmjeriti na **mentalni model izvršavanja**, umjesto na puko pokretanje programa.

- Ključni pojmovi
- **rekurzija, osnovni slučaj, rekurzivni korak, problem manjeg opsega, završavanje**

```
int zbrojDo(int n) { // pretpostavka: n >= 0  
    if (n == 0) return 0;  
    return n + zbrojDo(n - 1);  
}
```

što se događa s izrazom $n + \dots$ dok se pozvana funkcija još nije završila?

Najveća konceptualna prepreka nije pozivanje funkcije same sebe, nego razumijevanje da **svaki poziv ima vlastito stanje i da se izvršavanje nakon osnovnog slučaja vraća unatrag.**

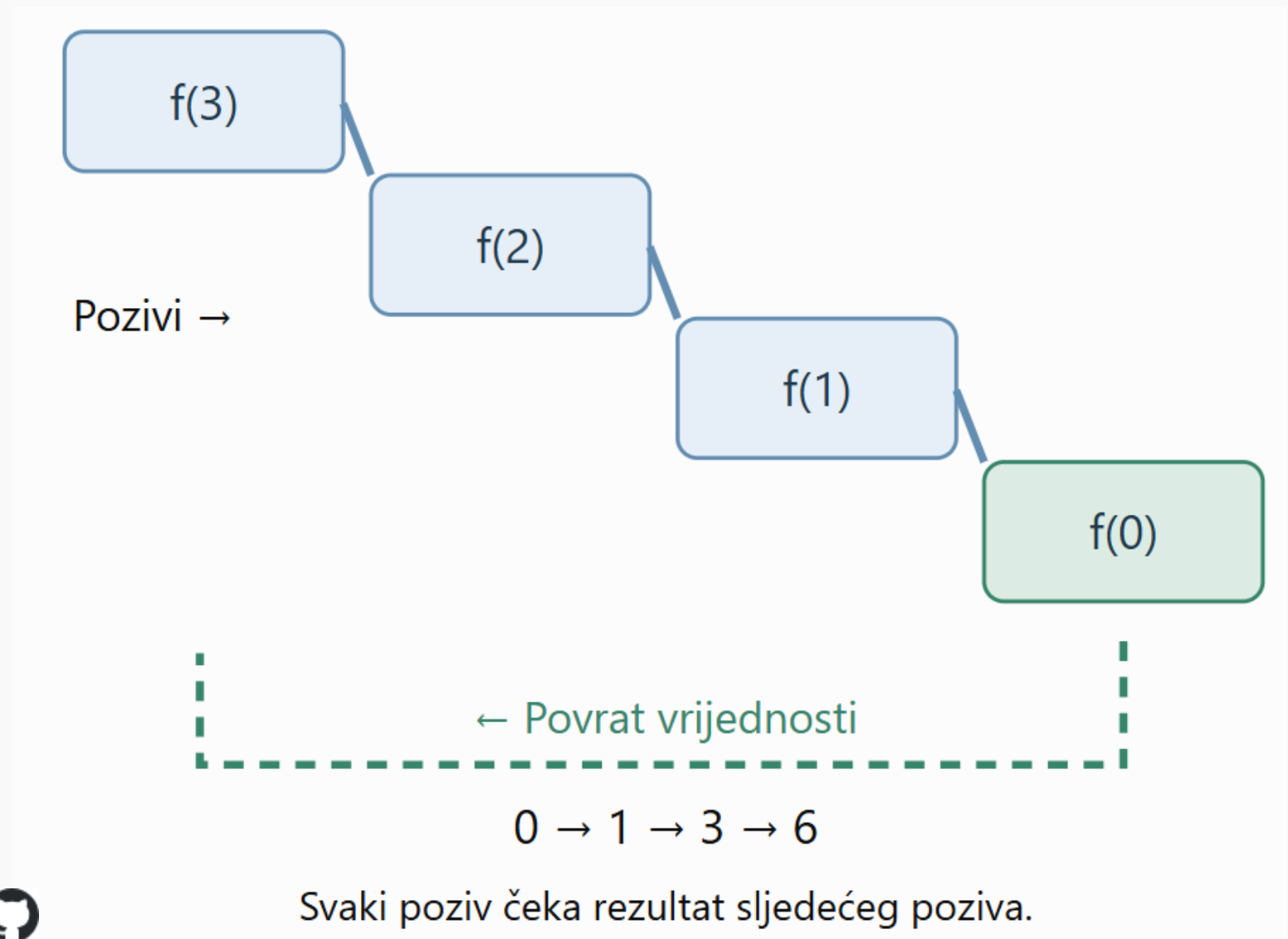
Vizualni prikaz poziva i povrata

<https://n-shovel.github.io/c-recursion-visualizer/#stack>

<https://n-shovel.github.io/c-recursion-visualizer/>

Prikazuje kako se pozivi slažu na stog, koji pozivi čekaju te kako se rezultati vraćaju.

Vizualizacija povezuje fizičko iskustvo s izvršavanjem programa.



- osnovni slučaj mora biti dosegljiv za dopuštene ulaze.
- označiti osnovni slučaj i rekurzivni korak u dvije kratke funkcije;
- pronaći zašto se ne zaustavlja
- provjera: $n = 3, 2, 1, 0$; razlikovati „funkcija se poziva” i „funkcija vraća”.
- navedi dvije nužne sastavnice ispravne rekurzije i objasni zašto se argument mora približavati osnovnom slučaju.



Funkcije – SIU1 – i2 – Rekurzije – 2. sat

I2 – Što se događa tijekom poziva i povratka?

- Vizualizacija poziva i povrata

- $\text{zbrojDo}(3) \rightarrow \text{zbrojDo}(2) \rightarrow \text{zbrojDo}(1) \rightarrow \text{zbrojDo}(0)$, zatim $0 \rightarrow 1 \rightarrow 3 \rightarrow 6$.

Poziv	Što čeka?	Vraćena vrijednost
$\text{zbrojDo}(3)$	$3 + \text{zbrojDo}(2)$	6
$\text{zbrojDo}(2)$	$2 + \text{zbrojDo}(1)$	3
$\text{zbrojDo}(1)$	$1 + \text{zbrojDo}(0)$	1
$\text{zbrojDo}(0)$	Osnovni slučaj	0

- Zašto $\text{zbrojDo}(3)$ ne vraća odmah broj 3?

```
int potencija(int a, int n) { // n >= 0
    if (n == 0) return 1;
    return a * potencija(a, n - 1);
}
```

- tablica s dva stupca „pozivi” i „povrati”.
- Pratiti funkciju potencija(2, 4), zapisati sve pozive i povratne vrijednosti; usporediti načine zapisa.
- Napomena: **svaki poziv ima vlastite parametre i lokalne varijable; dublji pozivi troše dodatnu memoriju.**
- <https://pythontutor.com/cpp.html#mode=edit>

```
int potencija(int a, int n) { // n
    >= 0
    if (n == 0) return 1;
    return a * potencija(a, n - 1);
}
```

Poziv	Što čeka?	Vraćena vrijednost
potencija(2, 8)	2*potencija(2,7)	256
potencija(2, 7)	2*potencija(2,6)	128
potencija(2, 6)	2*potencija(2,5)	64
potencija(2, 5)	2*potencija(2,4)	32
potencija(2, 4)	2*potencija(2,3)	16
potencija(2, 3)	2*potencija(2,2)	8
potencija(2, 2)	2*potencija(2,1)	4
potencija(2, 1)	2*potencija(2,0)	2
potencija(2, 0)	Osnovni slučaj	1

Analiza mogućih pogrešaka:

- izostavljen osnovni slučaj;
- argument $n + 1$ umjesto $n - 1$;
- pogrešan rezultat osnovnog slučaja.

provjera:

- što vraća potencija(3, 2)?
- Koji je prvi poziv koji vraća vrijednost bez novog poziva?



Funkcije – SIU1 – i2 – Rekurzije – 3. sat

I2 – Što se događa tijekom poziva i povratka?

- Problemska situacija
- **iz cijelog nenegativnog broja izdvojiti i zbrojiti znamenke.**
- kako se problem smanjuje dijeljenjem s 10.
- odredi osnovni slučaj, napiši rekurzivni korak i predvidi rezultat za zadani mali ulaz.
- konstrukcija funkcije `zbrojZnamenaka(n)`,
 - osnovni slučaj $n < 10$, rekurzivni korak $n \% 10 + \text{zbrojZnamenaka}(n / 10)$.
- ili
- samostalni zadatak na papiru: dopuniti praznine u funkciji za broj znamenaka pozitivnog broja;
- Usporedba s iterativnom idejom: kako bismo isti zadatak riješili ponavljanjem $n /= 10$?

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

Dodatno:

- trag poziva za 234;
- objasniti zašto funkcija za $n = 0$ vraća 0.
- provjeriti za 7, 80 i 205.



Funkcije – SIU1 – i6 – Rekurzije – 4. sat

I2 – Isti rezultat, različiti troškovi

Usporedba - pitanje: dvije funkcije vraćaju isti zbroj od 1 do n; znači li to da su jednako prikladne?

```
int zbrojIter(int n) { // n >= 0
    int s = 0;
    for (int i = 1; i <= n; i++) s += i;
    return s;
}
```

```
int zbrojRek(int n) { // n >= 0
    if (n == 0) return 0;
    return n + zbrojRek(n - 1);
}
```

Usporedna tablica

- Usporedba iterativne i rekurzivne inačice zbrojDo(n):

inačica	čitljivost	broj iteracija/poziva	dodatna memorija	
iterativna				
rekurzivna				

- Predviđanje za $n = 5$ i za vrlo velik n : koje rješenje može imati problem s dubinom poziva?
- Što znači da su oba slučaja linearna po broju koraka?
- kada rekurzija olakšava opis problema, a kada petlja ima manje dodatne memorijske zahtjeve?
- označiti rješenje s manjom dodatnom memorijom i obrazložiti izbor.

- Zaključak
- oba postupka imaju $O(n)$ koraka; prikazana iteracija troši $O(1)$ dodatne memorije, a izravna rekurzija $O(n)$ zbog dubine poziva.
- To je analiza ovih konkretnih rješenja, a ne tvrdnja da je svaka rekurzija sporija.



Funkcije – SIU1 – i6 – Rekurzije – 5. sat

I2 – iterativno i rekurzivno

Usporedba tri problema:

	pristup	razlog
zbroy brojeva do n		
obilazak ugniježđenih mapa		
ispis znamenaka obrnutim redom		

matrica odluke: struktura problema, jednostavnost,
broj koraka, memorija, mogućnost vrlo velikog ulaza

usporedba

- dva rješenja za potenciju s nenegativnim eksponentom:
 - jednostavna iteracija i jednostavna rekurzija;
- identificirati prednosti i ograničenja
- Individualna provjera:
 - (1) objasni razliku memorijskih zahtjeva;
 - (2) odaberi pristup za zadani slučaj;
 - (3) obrazloži zašto.

detaljna usporedba prednosti i nedostataka rekurzije:

Prednost	Opis
Čitljivost i elegancija	Kod je često puno kraći, čišći i lakši za razumijevanje nego kada se koriste komplicirane petlje (<code>for</code> ili <code>while</code>).
Prirodno rješenje za složene strukture	Idealna je za probleme koji se prirodno dijele na manje podprobleme (npr. <i>podijeli pa vladaj</i> algoritmi kao što su QuickSort i MergeSort).
Lakši rad s grafovima i stablima	Kretanje kroz strukture podataka poput binarnih stabala ili direktorija na računalu puno je jednostavnije implementirati rekurzivno.

Nedostatak	Opis
Potrošnja memorije (Stack Overflow)	Svaki rekurzivni poziv zauzima mjesto na stogu memorije (<i>call stack</i>). Ako je rekurzija preduboka, program će ostati bez memorije i srušiti se.
Lošije performanse	Rekurzija troši dodatno vrijeme na neprestano pozivanje funkcija i alociranje memorije, zbog čega je obično sporija od iterativnih rješenja (petlji).
Teže otklanjanje pogrešaka (Debugging)	Zbog velikog broja aktivnih poziva funkcija na stogu, praćenje toka programa i pronalaženje bugova može biti vrlo komplicirano.

Glavna prednost rekurzije je **jednostavnost i čitljivost koda** kod rješavanja složenih problema, dok je njezin najveći nedostatak **visoka potrošnja memorije i lošije performanse**.

- **Zaključak:** Rekurziju je najbolje koristiti kada je struktura podataka **po svojoj prirodi rekurzivna** (poput stabala) i kada **dubina poziva nije prevelika**. Ako je bitna maksimalna brzina i ušteda memorije, bolje je odabrati obične petlje (iteraciju).

Napomena o izboru:

- „optimalno” **se procjenjuje u odnosu na konkretan problem, očekivane ulaze i kriterije** (vrijeme, memorija, jasnoća, održavanje).
- **Rekurzija je prirodna za hijerarhijske strukture, ali nije automatski najbolja za sve probleme koji se mogu rekurzivno riješiti.**

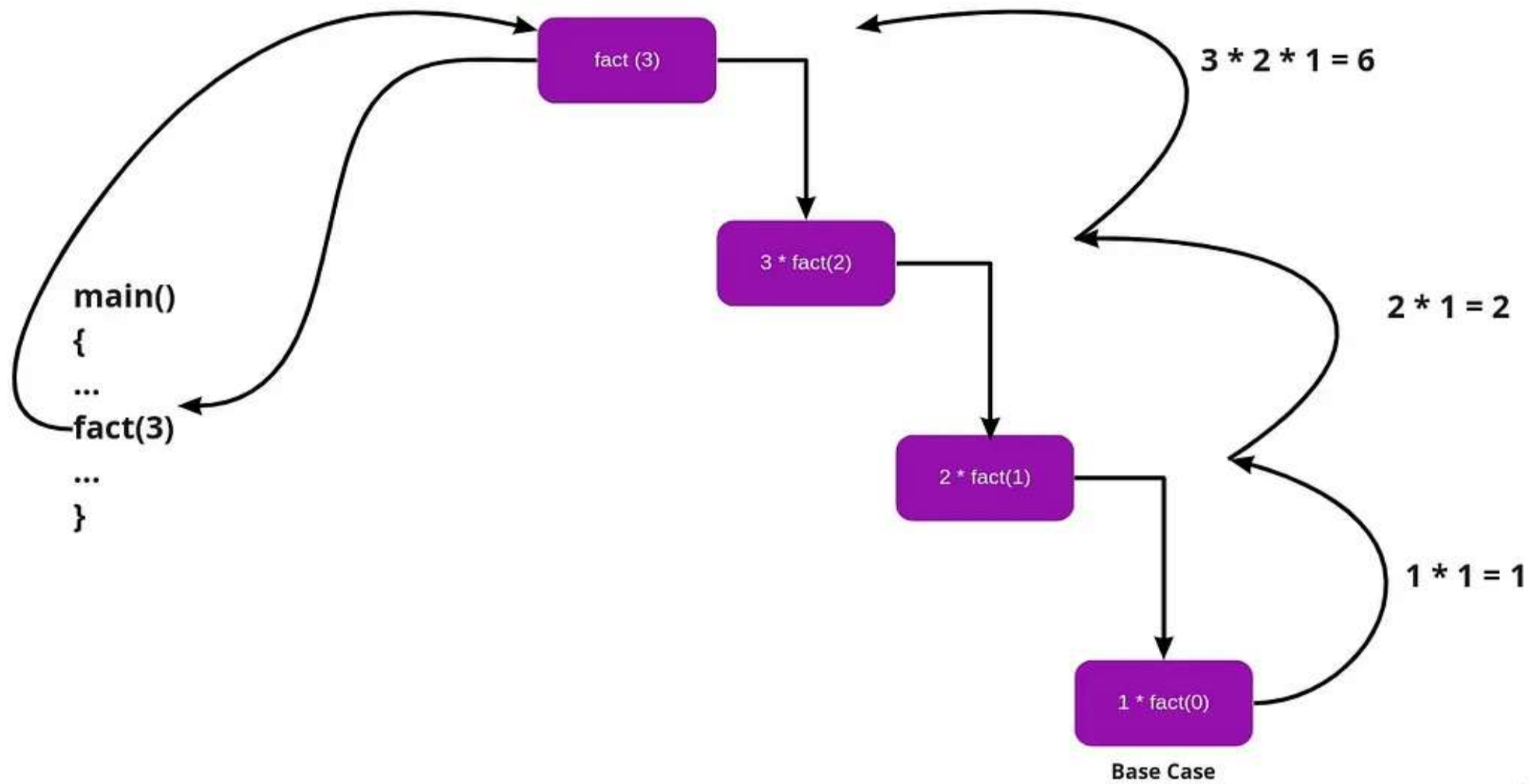
Poveznice

- <https://pythontutor.com/visualize.html#mode=display> (potencija)
- <https://studyalgorithms.com/theory/recursion-and-memory-vizualization/> (print, faktorijeli)
- <https://n-shovel.github.io/c-recursion-visualizer/#stack>



The screenshot shows the Python Tutor website interface. At the top, there's a browser address bar with the URL `pythontutor.com/visualize.html#mode=edit`. Below it, there's a navigation bar with buttons for "Tutor", "Java Tutor", "C Tutor", "C++ Tutor", and "JavaScript Tutor". The "C++ Tutor" button is selected. Below the navigation bar, there's a section titled "Write code in" with a dropdown menu showing "C++ (C++20 + GNU extensions)". The code editor displays the following C++ code:

```
1 #include <iostream>
2 using namespace std;
3
4 int zbrojDo(int n) { // pretpostavka: n >= 0
5     if (n == 0) return 0;
6     return n + zbrojDo(n - 1);
7 }
8
9 int main() {
10     cout<<zbrojDo(5);
11     return 0;
12 }
```





Funkcije – SIU1 – i2 – rekurzije - LV

I2, i6

Zadatak 1 – digitalno zrcalo

```
#include <iostream>
using namespace std;

void zrcalo(unsigned int n) {
    cout << n % 10;

    if (n < 10)
        return;

    zrcalo(n / 10);
}

int main() {
    unsigned int n;
    cin >> n;

    zrcalo(n);
    cout << '\n';

    return 0;
}
```

Objašnjenje: Funkcija najprije ispisuje posljednju znamenku, a zatim rekurzivno obrađuje preostale znamenke. Osnovni slučaj je `n < 10`. Rješenje ispravno čuva nule nastale obrtanjem, primjerice `1200 → 0021`.



Potruga kroz tunel

```
int brojPozitivnih(const vector<int>& a, int i) {
    if (i == static_cast<int>(a.size()))
        return 0;

    int trenutni = (a[i] > 0) ? 1 : 0;

    return trenutni + brojPozitivnih(a, i + 1);
}

int main() {
    int n;
    cin >> n;

    vector<int> a(n);

    for (int i = 0; i < n; i++)
        cin >> a[i];

    cout << brojPozitivnih(a, 0) << '\n';

    return 0;
}
```

Objašnjenje: Svaki rekursivni poziv obrađuje jedan element vektora. Ako je pozitivan, pridonosi rezultatu vrijednošću 1. Osnovni slučaj nastupa kada indeks dosegne veličinu vektora.

Put do cilja

```
long long nacini(int n) {  
    if (n == 0)  
        return 1;  
  
    if (n < 0)  
        return 0;  
  
    return nacini(n - 1) + nacini(n - 2);  
}  
  
int main() {  
    int n;  
    cin >> n;  
  
    cout << nacini(n) << '\n';  
  
    return 0;  
}
```

Objašnjenje: Do polja `n` možemo doći posljednjim potezom duljine 1 ili 2. Zato je ukupan broj načina zbroj brojeva načina dolaska do polja `n - 1` i `n - 2`.

Osnovni slučajevi su `n == 0` (jedan način) i `n < 0` (nijedan način).

Ovo je ispravno osnovno rekurzivno rješenje za zadano ograničenje `n ≤ 30`, iako ponavlja izračun istih potproblema.