Uvod u programiranje

- predavanja -

studeni 2025.

12. Funkcije

Vodič „11 prije dvanaestog predavanja”

- Očekuje se da ste savladali (na razini na kojoj je obrađeno u vodiču):
  - Motivacija za korištenje funkcija
  - Jednostavan primjer poziva funkcije, faktorijel

- Pitanja?

Funkcije

nastavak na vodič…

Definicija funkcije

- definicijom funkcije opisuje se
  - tip rezultata funkcije
  - naziv funkcije
  - (opcionalno) lista parametara funkcije, za svaki parametar: tip i naziv
  - tijelo funkcije: definicije i deklaracije, naredbe koje se obavljaju kad se funkcija pozove
  - (opcionalno) naredba za povratak rezultata i programskog slijeda

Opći oblik

```
tip_rezultata naziv_funkcije(lista_parametara) {
   definicije, deklaracije i naredbe;
   return rezultat;
}
```

tijelo funkcije

Definicija funkcije: tip funkcije

- tip rezultata (tip funkcije) mora biti naveden i može biti
  - bilo koji tip podatka **osim polja (*array*)**
    - osnovni tipovi, pokazivači, strukture
  - npr. ako funkcija vraća rezultat tipa int, kažemo: funkcija je tipa int
  - ako funkcija ne vraća rezultat (kažemo: funkcija je tipa void), kao *`tip_rezultata`* mora se navesti ključna riječ *`void`*

Definicija funkcije: parametri

- tip i naziv parametra trebaju biti navedeni za svaki parametar (ako funkcija ima parametre)
  - parametar može biti bilo kojeg tipa podatka **osim polja (*array*)**
    - osnovni tipovi, pokazivači, strukture
  - broj parametara nije ograničen (a može ih i ne biti)
    - ako funkcija nema niti jedan parametar, tada se na mjestu `lista_parametara` mora navesti ključna riječ `void`
  - parametar je *modifiable lvalue*: u tijelu funkcije može se koristiti na isti način kao svaka druga varijabla definirana u tijelu funkcije
    - od varijable definirane u tijelu funkcije parametar se razlikuju samo po tome što mu se na početku izvršavanja funkcije pridružuje vrijednost *argumenta* s kojim je funkcija pozvana

Definicija funkcije: povratak rezultata

- povratak rezultata i programskog slijeda na pozivajuću razinu (na mjesto u izrazu u kojem se nalazi poziv funkcije) obavlja se naredbom *return*
  - Opći oblik naredbe `return` *izraz*$_{\text{opcionalno}}$`;`
  - funkcija može vratiti najviše jednu vrijednost (ili niti jednu)
  - rezultat (*izraz*) može biti bilo kojeg tipa podatka **osim polja (array)**
    - ako se tip izraza razlikuje od tipa funkcije, obavlja se implicitna konverzija vrijednosti rezultata u tip funkcije
- u funkciji koja ne vraća rezultat ova naredba se može
  - izostaviti, u kojem slučaju će se povratak programskog slijeda dogoditi na kraju tijela funkcije
  - ili koristiti bez navođenja izraza koji predstavlja rezultat
- naredba se može navesti više puta unutar iste funkcije
  - iako to nije sasvim u skladu sa strukturiranim programiranjem

Primjer: definicija funkcije

- funkcija x^n, $x \in \mathbb{R}$, $n \in \mathbb{N}^0$

Rezultat funkcije je tipa double.

Parametri funkcije

prog.c

```
double eksp(float x, int n) {
   int i;
   double rez = 1.;
   for (i = 0; i < n; ++i)
      rez *= x;
   return rez;
}
...
int main(void) {
   ...
```

Varijable i, rez su lokalne varijable, vidljive samo u tijelu funkcije

Naredba za povratak (programski slijed i rezultat)

definicija funkcije main

- u programskom jeziku C funkcija se nikad ne definira *unutar* definicije neke druge funkcije

Poziv funkcije: argumenti

- argumenti funkcije su izrazi koji se navode pri pozivu funkcije, u okruglim zagradama iza naziva funkcije, odvojeni zarezima
  - argument može biti bilo koji tip podatka **osim polja (*array*)**
    - osnovni tipovi, pokazivači, strukture
  - broj i redoslijed argumenata treba odgovarati broju i redoslijedu parametara u funkciji koja se poziva
  - tipovi podataka argumenata moraju odgovarati tipovima podataka odgovarajućih parametara ili mora biti moguća konverzija podatka iz tipa podatka argumenta u tip podatka parametra
    - npr. ako je parametar funkcije tipa double, funkcija se može pozvati s argumentom tipa int (vodeći računa o mogućim gubicima informacija koje mogu nastati pri konverziji tipova podataka)
  - ako funkcija nema parametre, tada se pri pozivu funkcije na mjestu liste argumenata ne piše ništa (ne piše se void)

Primjer: poziv funkcije

prog.c

```
#include <stdio.h>                                  zbog printf u funkciji main

double eksp(float x, int n) {
   ...
   return rez;
}
...
int main(void) {
   ...
   double y = eksp(3.f, 4);
   printf("Tri na cetvrtu je %lf", y);
   printf("Dva na petu je %lf", eksp(2.f, 5));
   eksp(3.f, 2);                                    ispravno, ali beskorisno
   y = eksp(3.5f, 2) + 2 * eksp(2.1f, 3);           3.5^2 + 2 · 2.1^3
   y = eksp(eksp(2 * 3.f, 2), 4);                   (6^2)^4
   ...
}
```

Parametri

Argumenti

- argumenti mogu biti izrazi
- rezultat funkcije može se koristiti u raznim izrazima

- za sada, definiciju funkcije *main* uvijek napisati na kraju. Kasnije će biti objašnjeno na koji se način redoslijed pisanja definicija funkcija može promijeniti.

Primjer

- funkcija koja nema parametre, ali vraća rezultat

```
...
int prebroji(void) {
   char c;
   int brojac = 0;
   do {
      scanf("%c", &c);
      ++brojac;
   } while (c != '#');
   return brojac - 1;
}
...
int main(void) {
   printf("%d\n", prebroji());
   printf("%d\n", prebroji());
   ...
```

- broji koliko je znakova upisano preko tipkovnice prije pojave prvog znaka #

```
koliko znakova?#a sada?#↵
15↵
7↵
```

```
##↵
0↵
0↵
```

Primjer

- funkcija koja ima parametre, ali ne vraća rezultat

```
...
void ispisXY(float x, float y) {
   printf("(%.4f, %.4f)", x, y);
   return;
}
...
int main(void) {
   float x1 = 3.25f, y1 = -12.f;
   float x2 = 0.1f, y2 = 4.5f;

   printf("Koordinate T1: ");
   ispisXY(x1, y1);
   printf("\nKoordinate T2: ");
   ispisXY(x2, y2);
   ...
```

može se izostaviti

- ispisuje x, y koordinate unutar okruglih zagrada, s 4 znamenke iza decimalne točke

```
Koordinate T1: (3.2500, -12.0000)
Koordinate T2: (0.1000, 4.5000)
```

Primjer

- funkcija koja nema parametre i ne vraća rezultat

```
...
void preskoci(void) {
   char c;
   do {
      scanf("%c", &c);
   } while (c != '#');
   return;
}
...
int main(void) {
   int prvi, drugi;
   preskoci();
   scanf("%d", &prvi);
   preskoci();
   scanf("%d", &drugi);
   printf("prvi = %d, drugi = %d", prvi, drugi);
   ...
```

- preskače znakove upisane preko tipkovnice, do znaka #

može se izostaviti

- čita po jedan cijeli broj iza prva dva znaka #

```
preskoci sve ovo#567 preskoci i ovo#98765↵
prvi = 567, drugi = 98765
```

Primjer

- funkcija u kojoj se na više mjesta koristi naredba return

```
...
double apsolut(double x) {
   if (x < 0) {
      return -x;
   } else {
      return x;
   }
}
...
int main(void) {
   double x = -3.5;
   printf("abs(%lf) = %lf", x, apsolut(x));
   ...
```

- izračunava apsolutnu vrijednost realnog broja

```
abs(-3.500000) = 3.500000
```

Primjer

- implicitna konverzija argumenata

```
...
double eksp(float x, int n) {
   ...
   return rez;
}


...
int main(void) {
   ...
   double y = eksp(3, 4);
   printf("%lf", y);
   ...
```

int → float

bez konverzije, jer su argument i parametar istog tipa

- slično, npr. funkcija `sqrt` iz `<math.h>`, čiji je parametar tipa `double`, vratit će ispravan rezultat i onda kada se kao argument koristi cijeli broj

Primjer

- implicitna konverzija rezultata

```
...
char malo_u_veliko(char c) {
   if (c >= 'a' && c <= 'z')
      return c - ('a' - 'A');
   else
      return c;
}

...
int main(void) {
   printf("%c", malo_u_veliko('f'));
   printf("%c", malo_u_veliko('B'));
   printf("%c", malo_u_veliko('*'));
```

int → char, jer c - ('a' - 'A') je tipa int

bez konverzije, jer c već jest tipa char

```
F
B
*
```

Primjer

- funkcija **ne može** izmjenom vrijednosti parametara promijeniti vrijednosti argumenata
  - jer parametar sadrži *kopiju* vrijednosti argumenta

```
#include <stdio.h>
void pokusajPromijenitiArgument(int n) {
   n = 10;
   printf("Funkcija je parametar promijenila u n = %d\n", n);
   return;
}

int main(void) {
   int n = 5;
   printf("Funkcija je pozvana s argumentom n = %d\n", n);
   pokusajPromijenitiArgument(n);
   printf("Ali argument je ostao n = %d", n);
   return 0;
}
```

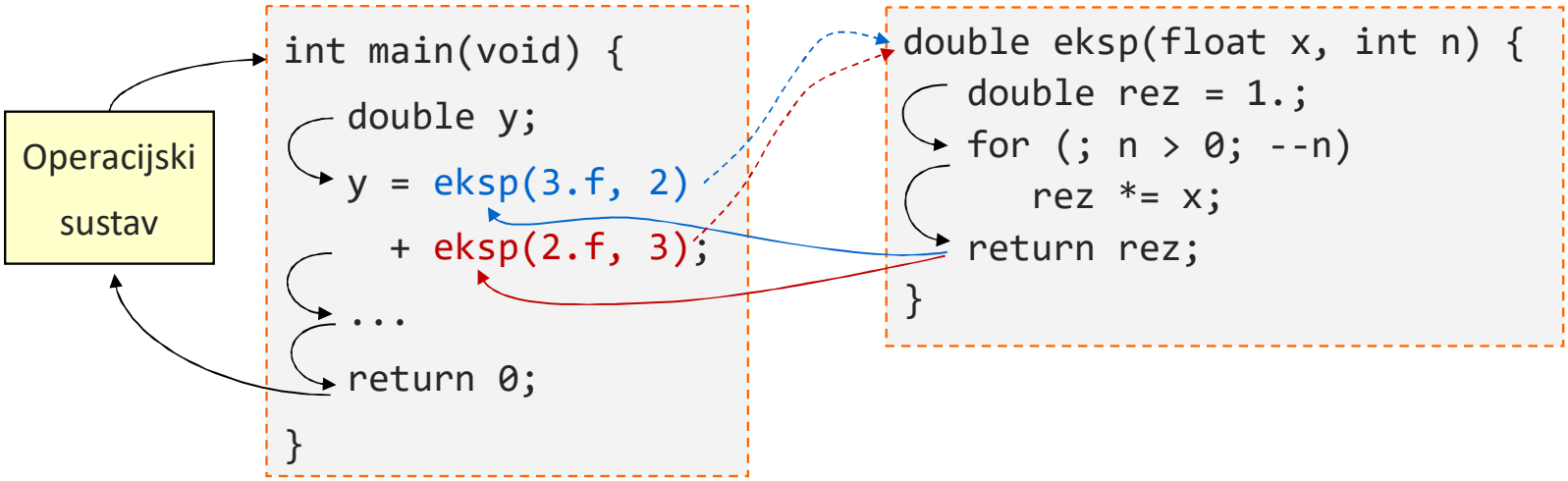
```
Funkcija je pozvana s argumentom n = 5↵
Funkcija je parametar promijenila u n = 10↵
Ali argument je ostao n = 5
```

Funkcije

Mehanizmi
prijenosa argumenata i
povrata rezultata

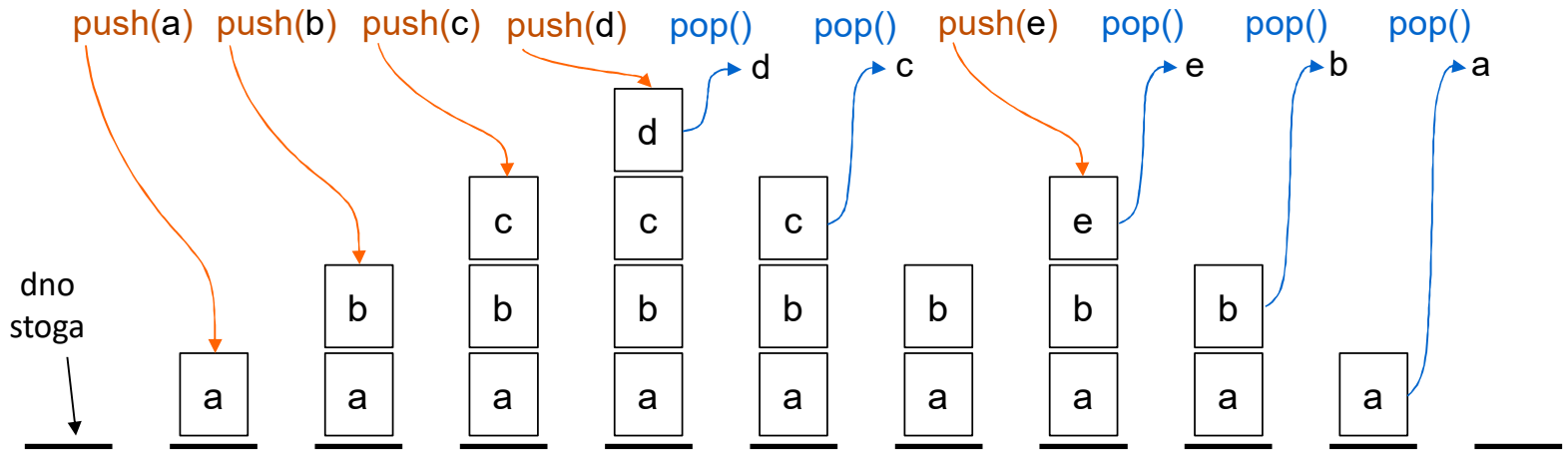
Mehanizam poziva funkcije

- u trenutku poziva funkcije potrebno je rezervirati memoriju za pohranu parametara i adrese instrukcije na koju se treba vratiti (povratne adrese)
- za vrijeme izvršavanja funkcije potrebno je rezervirati memoriju za varijable koje se definiraju u funkciji
- nakon završetka funkcije rezerviranu memoriju treba osloboditi

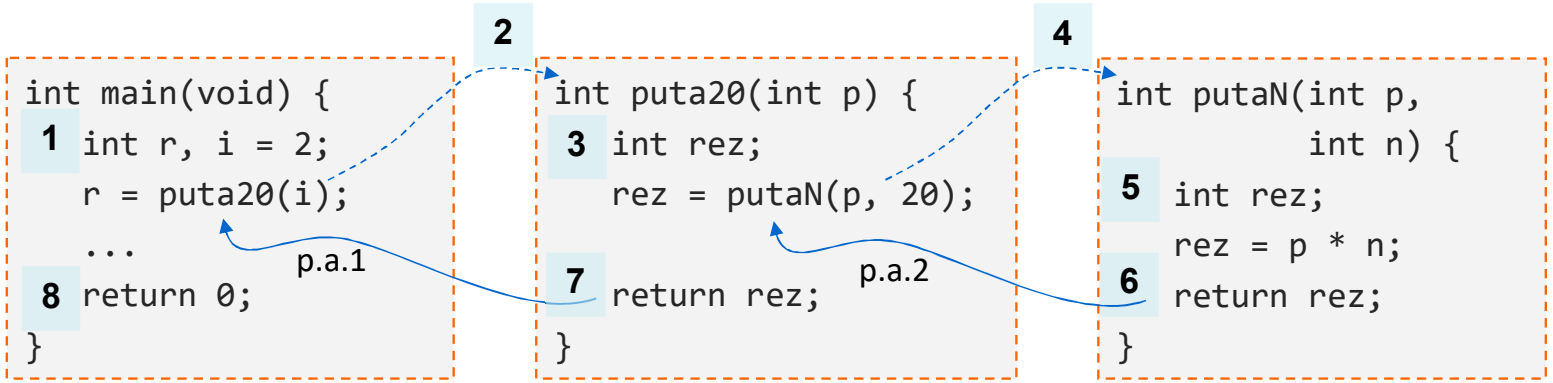


Stog (*stack*) općenito

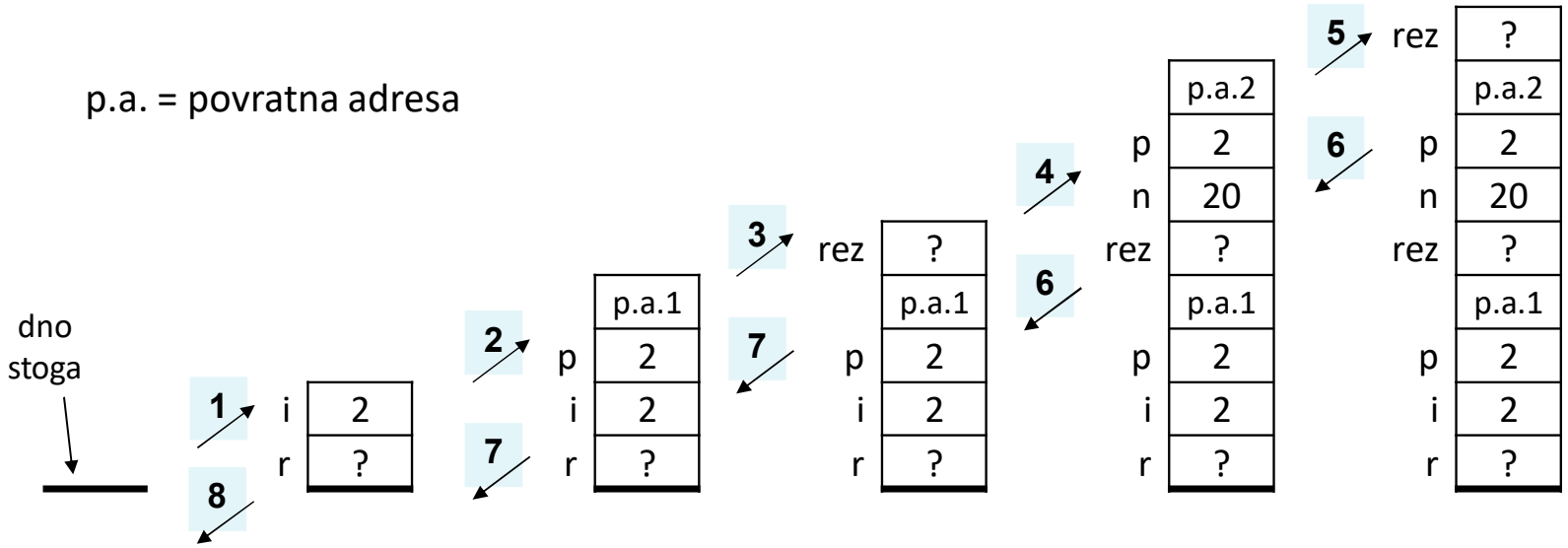
- svaka razina poziva funkcije zahtijeva rezervaciju memorije za *svoje* parametre, varijable i povratnu adresu
  - implementira se pomoću stoga. Osnovne operacije nad stogom su:
    - dodavanje podatka na stog (*push*)
    - uzimanje podatka s vrha stoga (*pop*)
  - posljednji na stog postavljeni podatak prvi je na redu za uzimanje
    - *posljednji unutra, prvi van* (*last in, first out – LIFO*)



U programskom jeziku C (pojednostavljeno)



p.a. = povratna adresa



Objašnjenje

1. na stog: prostor za lokalne varijable `r`, `i`
2. na stog: vrijednost za parametar `p` i povratnu adresu p.a.1
3. na stog: prostor za lokalnu varijablu `rez`
4. na stog: vrijednosti za parametre `p`, `n` i povratnu adresu p.a.2
5. na stog: prostor za lokalnu varijablu `rez`
6. uklanjanje lokalnih varijabli, parametara i povratne adrese sa stoga nakon povratka na p.a.2
7. uklanjanje lokalnih varijabli, parametara i povratne adrese nakon povratka na p.a.1
8. uklanjanje lokalnih varijabli sa stoga pri završetku programa

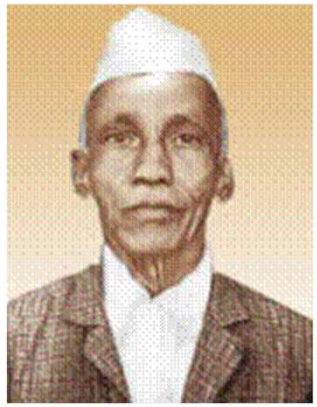
Zadatak – Kaprekarova konstanta

6174

18 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia



The number **6174** is known as **Kaprekar's constant**[1][2][3] after the Indian mathematician D. R. Kaprekar. This number is renowned for the following rule:

1. Take any four-digit number, using at least two different digits (leading zeros are allowed).
2. Arrange the digits in descending and then in ascending order to get two four-digit numbers, adding leading zeros if necessary.
3. Subtract the smaller number from the bigger number.
4. Go back to step 2 and repeat.

The above process, known as Kaprekar's routine, will always reach its fixed point, 6174, in at most 7 iterations.[4] Once 6174 is reached, the process will continue yielding 7641 – 1467 = 6174. For example, choose 1459:

9541 – 1459 = 8082
8820 – 0288 = 8532
8532 – 2358 = 6174
7641 – 1467 = **6174**

The only four-digit numbers for which Kaprekar's routine does not reach 6174 are repdigits such as 1111, which give the result 0000 after a single iteration. All other four-digit numbers eventually reach 6174 if leading zeros are used to keep the number of digits at 4. For numbers with three identical digits and a fourth digit that is one higher or lower (such as 2111), it is essential to treat 3-digit numbers with a leading zero; for example: 2111 – 1112 = 0999; 9990 – 999 = 8991; 9981 – 1899 = 8082; 8820 – 288 = 8532; 8532 – 2358 = 6174.[5]

- Programski provjerite je li istina…

Prije sljedećeg predavanja

- Edgar:
  - Tutorial: **nema** ☺
  - **12. vježbe uz predavanja**