Programiranje 2
1. predavanje

Saša Singer

singer@math.hr
web.math.hr/~singer

PMF – Matematički odjel, Zagreb

Dobar dan, dobro došli

Sadržaj predavanja (početak)

- Uvod u kolegij:
  - Tko sam, što sam i kako do mene.
  - Pravila lijepog ponašanja.
  - Računarski kolegiji na preddiplomskom studiju.
  - Cilj kolegija "Programiranje 2".
  - Pregled sadržaja kolegija.
  - Ostale važne informacije o kolegiju. Posebno:
    - "Pravila igre" ili način polaganja ispita.
    - Literatura.
    - Korisni linkovi — službena web stranica kolegija.

Sadržaj predavanja (nastavak)

- Funkcije (ponavljanje):
  - Načini prijenosa argumenata:
    - "po vrijednosti", "po adresi".
  - Prijenos argumenata po vrijednosti u C-u.
  - Prijenos adresa — "varijabilni" argumenti.
- Rekurzivne funkcije.
  - Fibonaccijevi brojevi — NE TAKO.
  - QuickSort algoritam.

Informacije

Ovaj semestar imamo

- samo 13 tjedana nastave, umjesto ranijih 14.

Zato

- preskačemo bivše prvo predavanje — ponavljanje gradiva iz Prog1 (visi na webu kao nulto predavanje),
- sva ranija predavanja se pomiču za jedno predavanje ranije.

Vodite računa o tome, ako štampate unaprijed!

Uvod u kolegij

Sadržaj

- Uvod u kolegij:
  - Tko sam, što sam i kako do mene.
  - Pravila lijepog ponašanja.
  - Računarski kolegiji na preddiplomskom studiju.
  - Cilj kolegija "Programiranje 2".
  - Pregled sadržaja kolegija.
  - Ostale važne informacije o kolegiju. Posebno:
    - "Pravila igre" ili način polaganja ispita.
    - Literatura.
    - Korisni linkovi — službena web stranica kolegija.

Na samom početku

- Moja malenkost (u punom "sjaju"):

  doc. dr. sc. Saša Singer

- Službeni osobni podaci:
  - ured (soba, kabinet): 227, drugi kat,
  - e–mail: singer@math.hr
    (Molim plain text poruke.)
  - web stranica: `http://web.math.hr/~singer/`
    (ona "službena": `http://www.math.hr/~singer/`
    je, uglavnom, beskorisna)
- Konzultacije (službeno):
  - petak, 12–14 sati, ili — po dogovoru.

Osnovna pravila "lijepog" ponašanja

Imam nekoliko lijepih zamolbi u rubrici "kultura".

- Prva i osnovna je

  razumna tišina,

  tj. da pričanjem ne ometate izvođenje nastave.
- Zatim, ne kasnite na predavanje.
- Održavajte razuman red u predavaonici.
- Mobilne telefone, molim, utišajte.

Ukratko o kolegijima iz računarstva

Programiranje 2 — skraćeno = P2, je drugi od (barem) 4 računarska kolegija na preddiplomskom studiju Matematika:

- Programiranje 1 (P1) (prije: Uvod u računarstvo (UuR)),
- Programiranje 2 (P2) (prije: Programiranje (C)),
- Strukture podataka i algoritmi (SPA),
- Računarski praktikum I (RP1).

Napomena: Raniji kolegiji su preduvjet za kasnije (navedenim redom, od 1. do 4. semestra).

P2 je drugi osnovni kolegij iz računarstva. Dakle, ne šalite se.

- Tko ima problema s P2, vrlo će teško "preživjeti" ostatak.

Cilj kolegija Programiranje 2

Ovaj kolegij,

- kao nastavak na Prog1 i preduvjet za SPA,

ima 2 osnovna cilja:

- savladavanje osnovnih tehnika programiranja, tj. realizacija osnovnih algoritama,
- učenje konkretnog programskog jezika — C, koji je sredstvo za realizaciju tih algoritama.

Cilj kolegija Programiranje 2 (nastavak)

Očekivana znanja i vještine — koje Vi trebate steći:

- razumijevanje koncepata i praktični rad s
  - funkcijama,
  - pokazivačima,
  - složenim strukturama podataka (polja, strukture, vezane liste),
  - datotekama,
- razumijevanje sintakse i semantike naredbi programskog jezika C,
- sposobnost pisanja osnovnih algoritama u programskom jeziku C.

Pregled sadržaja kolegija

Teme — posložene kao elementi programskog jezika C:

- Ponavljanje gradiva iz kolegija Programiranje 1 — web.
- Funkcije i rekurzivne funkcije.
- Struktura programa.
- Dvodimenzionalna i višedimenzionalna polja.
- Pokazivači. Pokazivači i polja. Pokazivači i funkcije.
- Strukture.
- Datoteke.
- Pre(d)procesorske naredbe.
- Standardna C biblioteka.

Kako položiti Programiranje 2?

Ocjena se formira na temelju zbroja bodova iz 3 dijela:

- 1. kolokvij — ima (najmanje) 40 bodova,
- 2. kolokvij — ima (najmanje) 60 bodova,
- "domaće" zadaće (ukupno n, svaka nosi m bodova) — još nisu "žive", pratite obavijesti.

Nije greška — zaista se može osvojiti preko 100 bodova.

Za prolaz je potrebno:

- zaraditi ukupno barem 45 bodova iz kolokvija (bilo kako, nema uvjeta).

Zadaće ne ulaze u granicu (45 bodova) za prolaz i ne zbrajaju se na popravnom!

Polaganje ispita (nastavak)

Nadalje, zadaće nisu obavezne, ali su vrlo korisne

- kao bonus za "dizanje" ocjene!

U načelu — usmenog ispita NEMA. Mogući izuzeci su:

- po želji — ako niste zadovoljni ocjenom,
- po kazni — nastavnik IMA PRAVO pozvati studenta na usmeni ispit (na pr. zbog prepisivanja na kolokviju).

Napomena: usmeni je praktični (za računalom).

Više detalja o načinu polaganja ispita možete naći na službenim web stranicama kolegija.

Ovdje ide priča da "nema šale".

Kako položiti ispit — upozorenje!

"Nema šale" $\Longleftrightarrow$ programiranje se uči prvenstveno

- samostalnim pisanjem programa na računalu.

Nema zamjene za to iskustvo!

- Ne može ga netko steći za vas, umjesto vas.

Upozorenje: C nije jednostavan jezik i

- nije izmišljen za učenje programiranja.

Svakako,

- isprobajte programe s predavanja i vježbi.

Sve je dostupno na webu

- službenom i/ili mojem — v. malo dalje.

Literatura za Prog2

Osnovna literatura su, naravno,

- predavanja i vježbe,

s popratnim materijalima (na pr. programi na webu).

Dodatna literatura — ukratko (više riječi je bilo na Prog1):

- Brian W. Kernighan i Dennis M. Ritchie,
  The C Programming Language (second edition),
  Prentice Hall, Upper Saddle River, New Jersey, 1988.
- B. S. Gottfried, Theory and Problems of Programming with C (second edition), Schaum's outline series,
  McGraw-Hill, New York, 1996.
  (Uputa: tražite najnovije izdanje.)

Programska podrška za C

Za praktično programiranje u C-u, možete koristiti

- DevC++, MS Visual Studio, . . . , na Windowsima,
- cc, gcc na Unix/Linux platformi.

Ponavljam:

- isprobajte programe s predavanja i vježbi.

Osim toga, (is)koristite demonstratore.

Korisni linkovi

Službena web stranica kolegija je:

`http://degiorgi.math.hr/prog2/`

Tamo su:

- predavanja prof. Nogo
  (moja predavanja su na mom webu, da ne bude "kaos"),
- vježbe,
- sve bitne obavijesti,
- svašta drugo — pogledajte!

Korisni linkovi (nastavak)

Isplati se relativno često svratiti, jer se

- sve važne stvari prvo pojave na webu.

Na primjer, rezultati kolokvija!

Ako mislite da bi na službenom webu, trebalo biti još nešto, slobodno predložite!

- Ideja je da tamo bude sve što vam može pomoći.

Molba: Ako nešto ne radi, odmah javite meni ili asistentima.

- Najbolje, Vedranu Šegi (na `vsego`), jer on vodi brigu o webu.

Ima li pitanja?

Slušam ...

Funkcije

Sadržaj

- Funkcije:
  - Načini prijenosa argumenata:
    - "po vrijednosti", "po adresi".
  - Prijenos argumenata po vrijednosti u C-u.
  - Prijenos adresa — "varijabilni" argumenti.
  - Rekurzivne funkcije.
    - Fibonaccijevi brojevi.
    - QuickSort algoritam.

Definicija funkcije — ponavljanje

Funkcija je programska cjelina koja

- uzima neke ulazne podatke,
- izvršava određeni niz naredbi,
- i vraća rezultat svog izvršavanja na mjesto poziva.

Definicija funkcije ima oblik:

```
tip_podatka ime_funkcije(tip_1 arg_1,
                         ..., tip_n arg_n)
{
    tijelo funkcije
}
```

Načini prijenosa argumenata

Formalni i stvarni argumenti (ili parametri):

- Argumenti deklarirani u definiciji funkcije nazivaju se formalni argumenti.
- Izrazi koji se pri pozivu funkcije nalaze na mjestima formalnih argumenata nazivaju se stvarni argumenti.

Veza između formalnih i stvarnih argumenata uspostavlja se

- prijenosom argumenata prilikom poziva funkcije.

Sasvim općenito, postoje dva načina prijenosa (ili predavanja) argumenata prilikom poziva funkcije:

- prijenos vrijednosti argumenata — engl. "call by value",
- prijenos adresa argumenata — engl. "call by reference".

Prijenos argumenata po vrijednosti

Kod prijenosa vrijednosti argumenata

- funkcija prima kopije vrijednosti stvarnih argumenata,

što znači da

- funkcija ne može izmijeniti stvarne argumente.

Stvarni argumenti mogu biti izrazi. Prilikom poziva funkcije,

- prvo se izračuna vrijednost tog izraza,
- a zatim se ta vrijednost prenosi u funkciju,
- i kopira u odgovarajući formalni argument.

Prijenos argumenata po adresi

Kod prijenosa adresa argumenata

- funkcija prima adrese stvarnih argumenata,

što znači da

- funkcija može izmijeniti stvarne argumente, tj. sadržaje na tim adresama.

Stvarni argumenti, u principu, ne mogu biti izrazi,

- već samo varijable,
- odnosno, objekti koji imaju adresu.

Prijenos argumenata u C-u

U C-u postoji samo prijenos argumenata po vrijednosti.

- Svaki formalni argument ujedno je i lokalna varijabla u toj funkciji.
- Stvarni argumenti u pozivu funkcije su izrazi (izračunaj vrijednost, kopiraj ju u formalni argument).

Ako funkcijom želimo promijeniti vrijednost nekog podatka,

- pripadni argument treba biti pokazivač na taj podatak,

tj. njegova adresa!

- Tada se adresa prenosi po vrijednosti — kopira u funkciju,
- ali smijemo promijeniti sadržaj na toj adresi, koristeći operator dereferenciranja `*`.

Prijenos vrijednosti argumenata

Primjer. Prijenos vrijednosti argumenata (kvad_1.c).

```
#include <stdio.h>

void kvadrat(int x, int y)
{
    y = x*x;
    printf("Unutar funkcije: x = %d, y = %d.\n",
            x, y);
    return;
}
```

Kvadrat od x sprema se u lokalnoj varijabli y, pa nema traga izvan funkcije kvadrat.

Prijenos vrijednosti argumenata (nastavak)

```
int main(void) {
    int x = 3, y = 5;

    printf("Prije poziva: x = %d, y = %d.\n", x, y);
    kvadrat(x, y);
    printf("Nakon poziva: x = %d, y = %d.\n", x, y);
    return 0;
}
```

Rezultat izvršavanja programa je:

```
Prije poziva: x = 3, y = 5.
Unutar funkcije: x = 3, y = 9.
Nakon poziva: x = 3, y = 5.
```

Prijenos adresa argumenata

Primjer. Prijenos adresa argumenata (kvad_2.c).

```
#include <stdio.h>

void kvadrat(int *x, int *y)
{
    *y = *x**x;  /* = (*x) * (*x). */
    printf("Unutar funkcije: x = %d, y = %d.\n",
            *x, *y);
    return;
}
```

Kvadriramo sadržaj od x i spremamo ga u sadržaj od y, pa ostaje trag izvan funkcije kvadrat — mijenja se *y.

Prijenos adresa argumenata (nastavak)

```
int main(void) {
    int x = 3, y = 5;

    printf("Prije poziva: x = %d, y = %d.\n", x, y);
    kvadrat(&x, &y);
    printf("Nakon poziva: x = %d, y = %d.\n", x, y);
    return 0;
}
```

Rezultat izvršavanja programa je:

```
Prije poziva: x = 3, y = 5.
Unutar funkcije: x = 3, y = 9.
Nakon poziva: x = 3, y = 9.
```

Napomene uz primjer

U prvom primjeru

- `void kvadrat(int x, int y)`

`x` i `y` su lokalne varijable tipa `int`.

U drugom primjeru

- `void kvadrat(int *x, int *y)`

`x` i `y` su lokalne varijable tipa `int *`, tj. pokazivači na `int`.

Nije lijepo da se razne stvari isto zovu! Recimo, `px` i `py` bi bilo bolje u drugom primjeru.

"Prava" realizacija bi bila

- `void kvadrat(int x, int *py)`

jer `x` ne mijenjamo!

Korektni prijenos argumenata

Primjer. Korektni prijenos argumenata — y je "varijabilni" argument, pa prenosimo adresu py (kvad_3.c).

```
#include <stdio.h>

void kvadrat(int x, int *py)
{
    *py = x*x;
    printf("Unutar funkcije: x = %d, y = %d.\n",
           x, *py);
    return;
}
```

Kvadrat od x spremamo u sadržaj od py, pa ostaje trag izvan funkcije kvadrat — mijenja se *py.

Korektni prijenos argumenata (nastavak)

```
int main(void) {
    int x = 3, y = 5;

    printf("Prije poziva: x = %d, y = %d.\n", x, y);
    kvadrat(x, &y);
    printf("Nakon poziva: x = %d, y = %d.\n", x, y);
    return 0;
}
```

Rezultat izvršavanja programa je:

```
Prije poziva: x = 3, y = 5.
Unutar funkcije: x = 3, y = 9.
Nakon poziva: x = 3, y = 9.
```

Korektni prijenos argumenata (nastavak)

Potpuni pregled stanja stvari dobivamo ispisom adresa, vrijednosti i sadržaja na adresama (`kvad_p3.c`):

```
U glavnom programu (funkcija main):
     adresa od x (&x) = 0012FF5C
     adresa od y (&y) = 0012FF60
Prije poziva funkcije:
  vrijednost od x (x) = 3
  vrijednost od y (y) = 5
Unutar funkcije kvadrat:
       adresa od x (&x) = 0012FF28
     adresa od py (&py) = 0012FF2C
    vrijednost od x (x) = 3
  vrijednost od py (py) = 0012FF60
    sadrzaj od py (*py) = 9
```

Korektni prijenos argumenata (nastavak)

```
Nakon poziva funkcije:
  vrijednost od x (x) = 3
  vrijednost od y (y) = 9
```

Rekurzivne funkcije

Rekurzivne funkcije

Programski jezik C dozvoljava tzv. rekurzivne funkcije, tj.

- da funkcija poziva samu sebe.

U pravilu,

- rekurzivni algoritmi su kraći,
- ali izvođenje, u načelu, traje dulje.

Katkad — puno dulje, ako puno puta računamo istu stvar.
Zato oprez!

Napomena. Svaki rekurzivni algoritam mora imati

- "nerekurzivni" dio, koji omogućava prekidanje rekurzije.

Najčešće je to neki `if` u inicijalizaciji rekurzije.

Fibonaccijevi brojevi

Fibonaccijevi brojevi

Primjer. Drugi standardni primjer rekurzivne funkcije (osim faktorijela) su Fibonaccijevi brojevi, definirani rekurzijom

$$F_n = F_{n-1} + F_{n-2}, \quad n \geq 2, \quad \text{uz} \quad F_0 = 0,\ F_1 = 1.$$

Po definiciji, možemo napisati rekurzivnu funkciju:

```
long int fib(int n)
{
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fib(n - 1) + fib(n - 2);
}
```

Ali, ovo nemojte raditi. Zabranjujem!

Fibonaccijevi brojevi (nastavak)

Ovdje je broj rekurzivnih poziva ogroman i veći od samog broja F_n.
Ne vjerujete? Dodajmo funkciji globalni brojač poziva `broj_poziva` (`fib_r.c`).

```
long int fib(int n)
{
    ++broj_poziva;  /* globalni brojac poziva */
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fib(n - 1) + fib(n - 2);
}
```

Za $n = 20$ rezultat je $F_{20} = 6765$, a za računanje treba 21891 poziv funkcije!

Fibonaccijevi brojevi petljom

I ovo ide puno brže običnom petljom:

- novi član je zbroj prethodna dva, uz "pomak" članova.

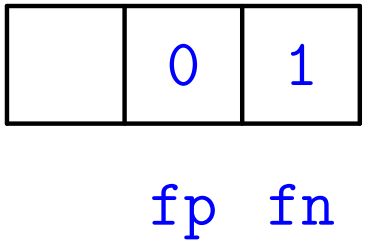
Za realizaciju tog algoritma trebamo "prozor" od samo 3 člana niza:

- `fn` = novi član,
- `fp` = prošli član,
- `fpp` = pretprošli član.

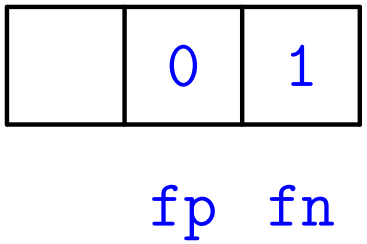
Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 "putuje" nizom:



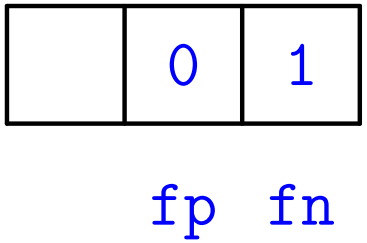
Što se stvarno zbiva s prozorom:



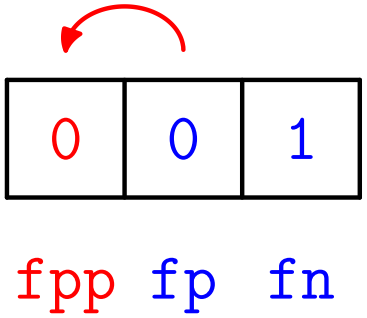
Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 "putuje" nizom:



Što se stvarno zbiva s prozorom:

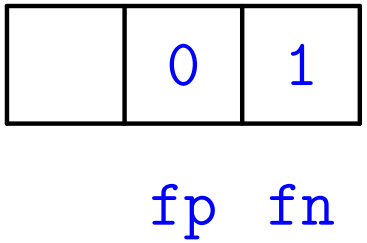



```
fpp = fp
```

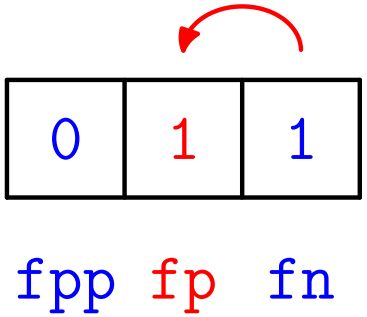
Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 "putuje" nizom:



Što se stvarno zbiva s prozorom:




```
fp = fn
```

Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 "putuje" nizom:

0	1	1
fpp	fp	fn

Što se stvarno zbiva s prozorom:

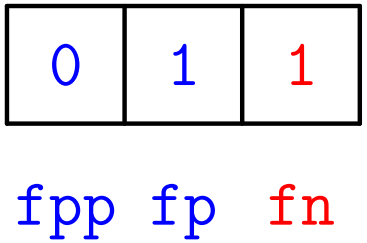
0	1	1
fpp	fp	fn

```
fn = fp + fpp
```

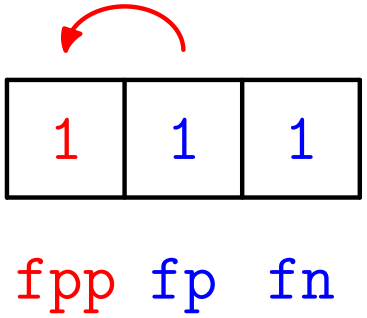
Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 "putuje" nizom:



Što se stvarno zbiva s prozorom:

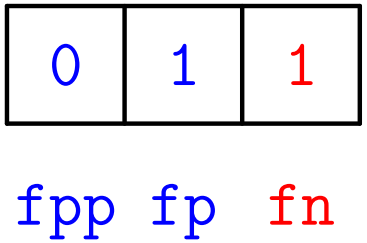



```
fpp = fp
```

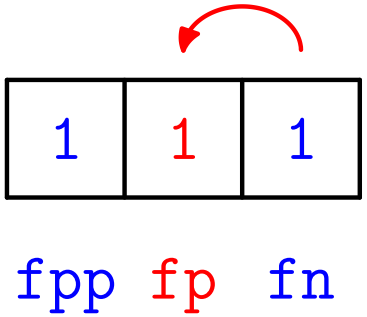
Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 "putuje" nizom:



Što se stvarno zbiva s prozorom:




```
fp = fn
```

Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 "putuje" nizom:

0	1	1	2

fpp fp fn

Što se stvarno zbiva s prozorom:

1	1	2

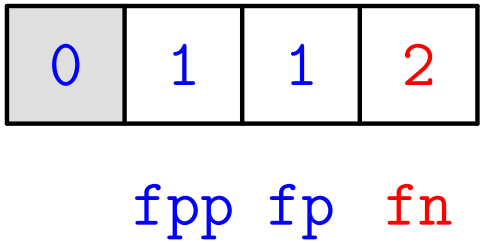
fpp fp fn

```
fn = fp + fpp
```

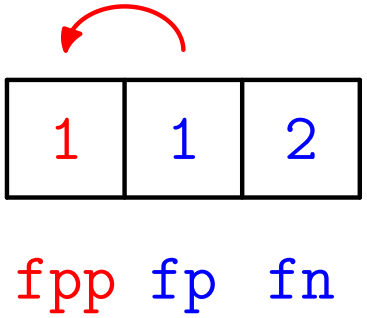
Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 "putuje" nizom:



Što se stvarno zbiva s prozorom:

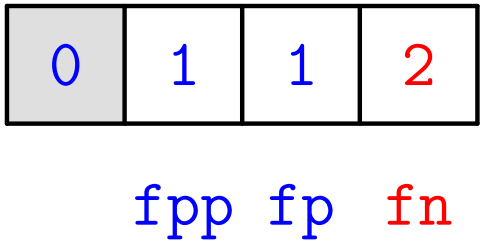



```
fpp = fp
```

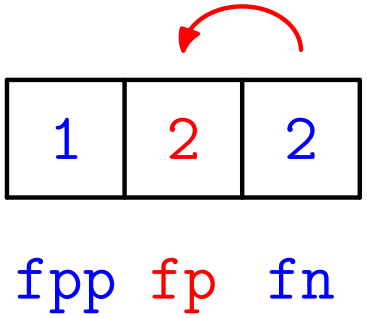
Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 "putuje" nizom:



Što se stvarno zbiva s prozorom:




```
fp = fn
```

Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 "putuje" nizom:

0	1	1	2	3
	fpp	fp		fn

Što se stvarno zbiva s prozorom:

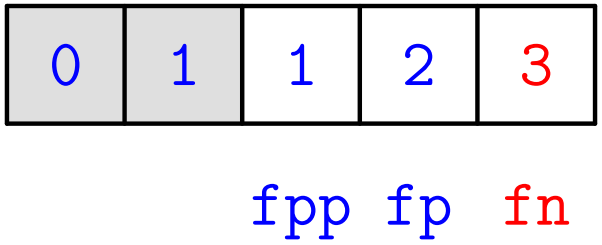
1	2	3
fpp	fp	fn

```
fn = fp + fpp
```

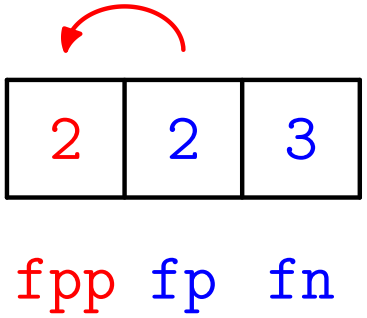
Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 "putuje" nizom:



Što se stvarno zbiva s prozorom:

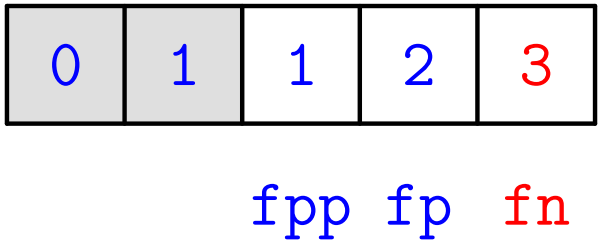



```
fpp = fp
```

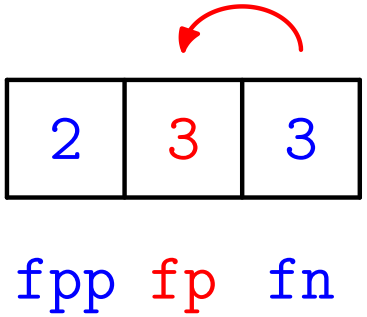
Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 "putuje" nizom:



Što se stvarno zbiva s prozorom:




```
fp = fn
```

Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 "putuje" nizom:

0	1	1	2	3	5
			fpp	fp	fn

Što se stvarno zbiva s prozorom:

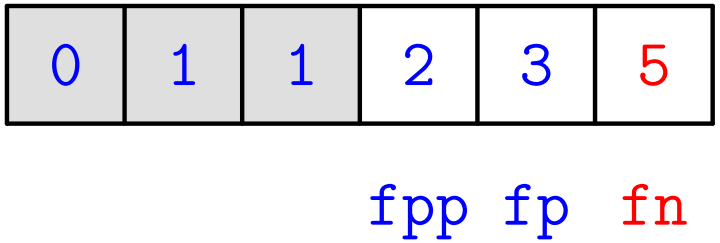
2	3	5
fpp	fp	fn

```
fn = fp + fpp
```

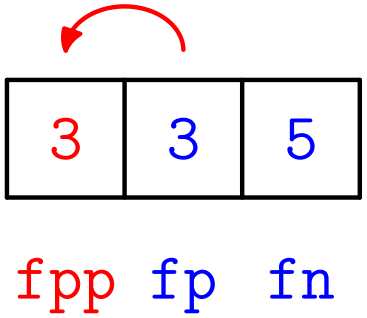
Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 "putuje" nizom:



Što se stvarno zbiva s prozorom:

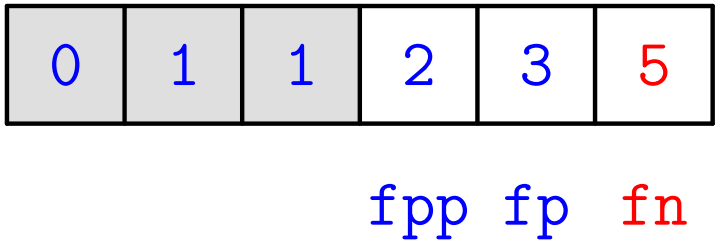



```
fpp = fp
```

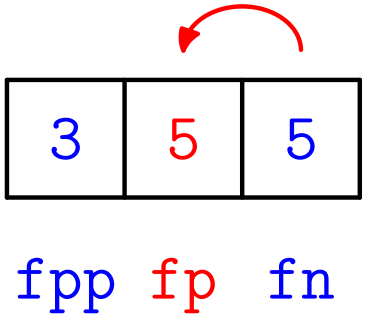
Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 "putuje" nizom:



Što se stvarno zbiva s prozorom:




```
fp = fn
```

Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 "putuje" nizom:

0	1	1	2	3	5	8
				fpp	fp	fn

Što se stvarno zbiva s prozorom:

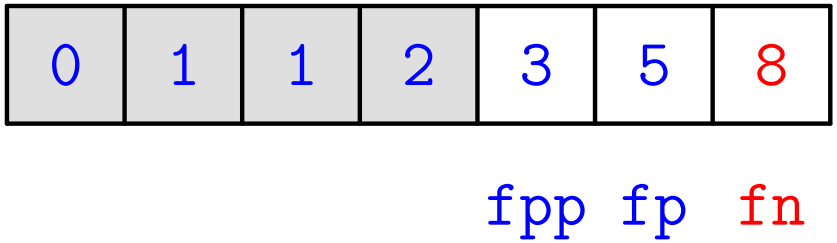
3	5	8
fpp	fp	fn

```
fn = fp + fpp
```

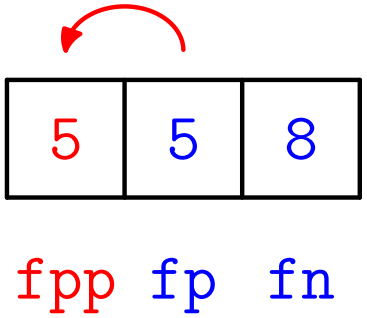
Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 "putuje" nizom:



Što se stvarno zbiva s prozorom:

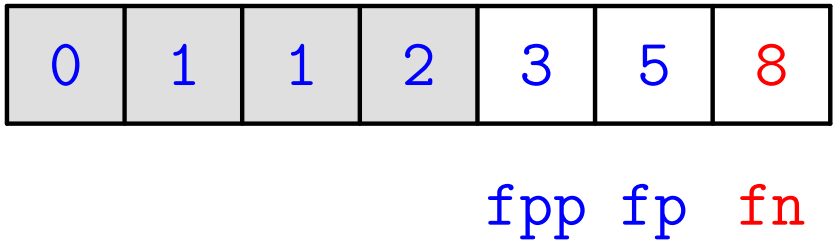



```
fpp = fp
```

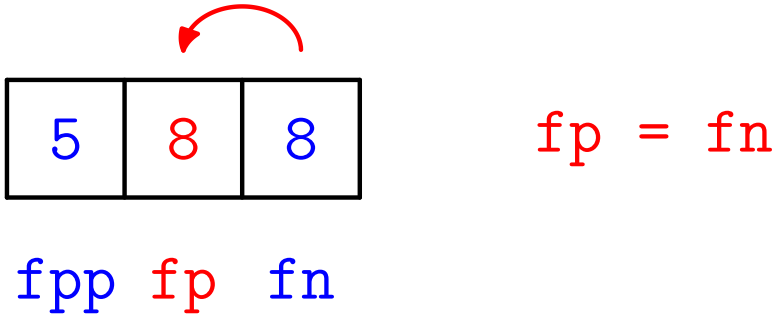
Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 "putuje" nizom:



Što se stvarno zbiva s prozorom:



Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 "putuje" nizom:

0	1	1	2	3	5	8	13
					fpp	fp	fn

Što se stvarno zbiva s prozorom:

5	8	13
fpp	fp	fn

```
fn = fp + fpp
```

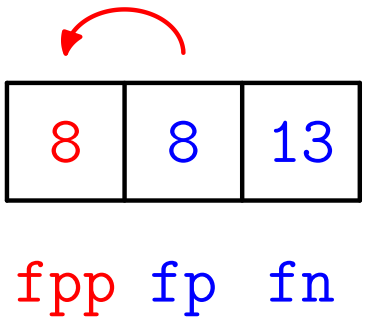
Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 "putuje" nizom:

0	1	1	2	3	5	8	13
					fpp	fp	fn

Što se stvarno zbiva s prozorom:




```
fpp = fp
```

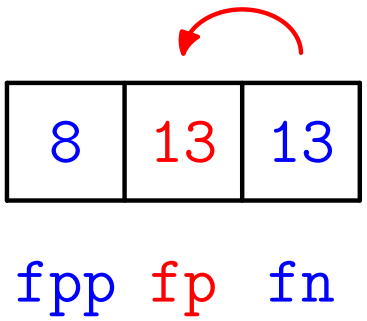
Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 "putuje" nizom:

0	1	1	2	3	5	8	13
					fpp	fp	fn

Što se stvarno zbiva s prozorom:




```
fp = fn
```

Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 "putuje" nizom:

0	1	1	2	3	5	8	13	21
						fpp	fp	fn

Što se stvarno zbiva s prozorom:

8	13	21
fpp	fp	fn

```
fn = fp + fpp
```

Fibonaccijevi brojevi petljom (nastavak)

Iterativna (nerekurzivna) verzija funkcije za Fibonaccijeve brojeve (`fib_a.c`).

```
long int fibonacci(int n)
{
    long int f_n, f_p, f_pp;  /* Namjerno NE inic. */
    int i;

    if (n == 0) return 0;  /* F[0] */
    if (n == 1) return 1;  /* F[1] */

        /* Sad inicijaliziramo prva dva.
           Inicijalizacija odgovara
           stanju za n = 1 (a ne 2). */
```

Fibonaccijevi brojevi petljom (nastavak)

```
    f_p = 0;  /* Prosli F[0] */
    f_n = 1;  /* Ovaj   F[1] */

    for (i = 2; i <= n; ++i) {
        f_pp = f_p;              /* F[i - 2] */
        f_p  = f_n;              /* F[i - 1] */
        f_n  = f_p + f_pp;       /* F[i]     */
    }

    return f_n;
}
```

Fibonaccijevi brojevi (kraj)

Ima još puno brži algoritam za računanje F_n,

- složenost mu je $O(\log n)$, a ne $O(n)$,

ali se ne isplati za male n.

Naime, najveći prikazivi Fibonaccijev broj

- na 32 bita, u tipu `int` i u tipu `long int`,

je $F_{46} = 1836311903$.

Dakle, korektne rezultate dobivamo samo za $n \leq 46$, a tad je dovoljno brz i obični aditivni algoritam.

QuickSort algoritam

QuickSort — uvod i skica algoritma

QuickSort se temelji na principu podijeli pa vladaj.

- Uzmemo jedan element x_k iz niza i dovedemo ga na njegovo pravo mjesto.
- Lijevo od njega ostavimo elemente koji su manji ili jednaki njemu (u bilo kojem poretku).
- Desno od njega ostavimo elemente koji su veći od njega (u bilo kojem poretku). Možemo i tu dozvoliti jednakost.
- Ako smo dobro izabrali, tj. ako je mjesto x_k blizu sredine, onda ćemo morati sortirati dva polja polovične duljine.
- U najgorem slučaju, ako smo izabrali "krivi" x_k, morat ćemo sortirati polje duljine $n - 1$.

QuickSort — razrada algoritma

U danom trenutku, rekurzivna funkcija za QuickSort treba sortirati nesređeni dio niza

- između "lijevog" indeksa l i "desnog" indeksa d.

Ta dva indeksa (i polje) su argumenti funkcije.

Posla ima ako i samo ako taj dio niza ima barem 2 elementa, tj. ako je $l < d$.

Za tzv. ključni element, najčešće se uzima $k = l$, tj.

- "prvi" element x_l treba dovesti na njegovo pravo mjesto u tom komadu niza.

QuickSort — razrada algoritma (nastavak)

Dogovor:

- lijevo (ili ispred) njega stavljamo elemente koji su manji ili jednaki x_l,
- desno (ili iza) njega stavljamo elemente koji su strogo veći od x_l.

Tada će pravo mjesto elementa x_l biti zadnje u lijevom dijelu.

Kako se traži "pravo" mjesto?

- Dvostranim pretraživanjem po ostatku niza.
- Sa svake strane (lijeve i desne) tražimo prvi sljedeći element koji "ne spada" na tu stranu niza.
- Ako nađemo takav par — zamijenimo im mjesta!

QuickSort — razrada algoritma (nastavak)

Algoritam za dvostrano pretraživanje:

```
if (l < d) {
    i = l + 1;
    j = d;

        /* Prolaz mora i za i == j */
    while (i <= j) {
        while (i <= d && x[i] <= x[l]) ++i;
        while (x[j] > x[l]) --j;
        if (i < j) swap(&x[i], &x[j]);
    }
```

QuickSort — razrada algoritma (nastavak)

Iza toga treba još:

- dovesti element x_l na njegovo pravo mjesto — indeks tog mjesta je j,
- rekurzivno sortirati lijevi i desni podniz.

```
        if (l < j) swap(&x[j], &x[l]);
        quick_sort(x, l, j - 1);
        quick_sort(x, j + 1, d);
    }
```

QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

42	12	55	94	18	44	67

QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

42	12	55	94	18	44	67

`l = 0`

`i = 1`

`d = 6`

`j = 6`

QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

42	12	55	94	18	44	67

`l = 0`

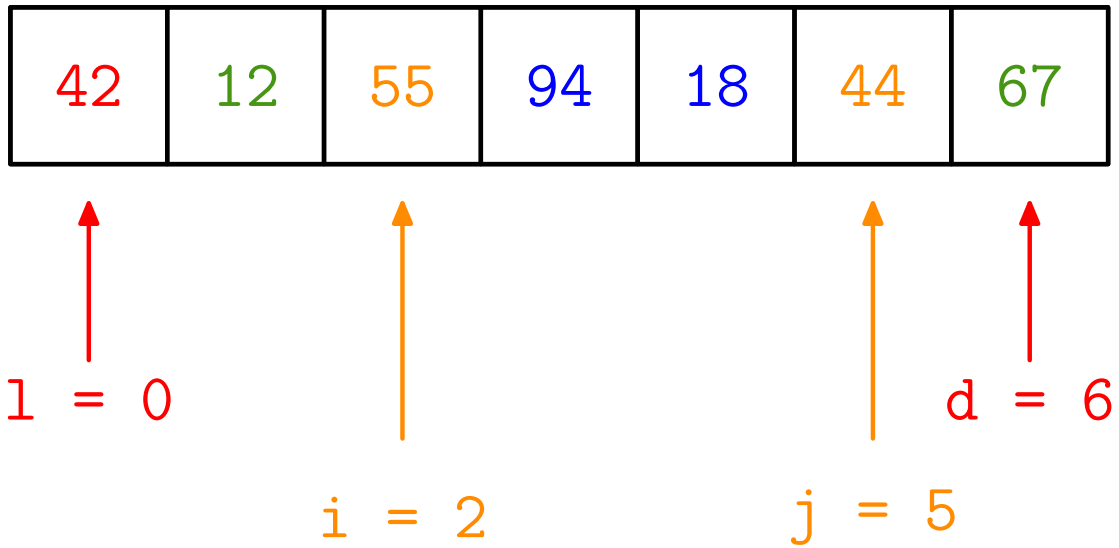
`i = 2`

`d = 6`

`j = 6`

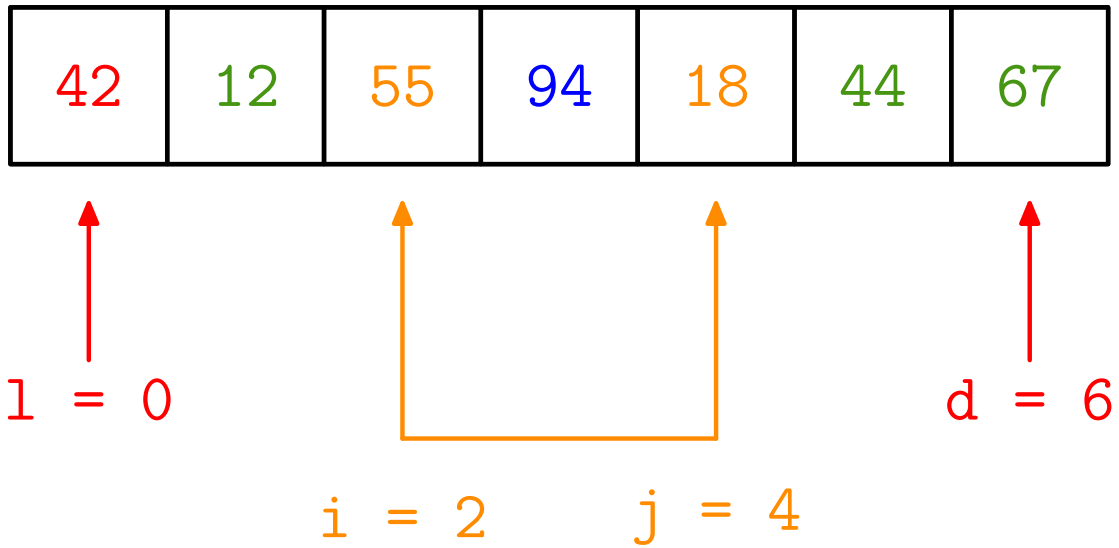
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



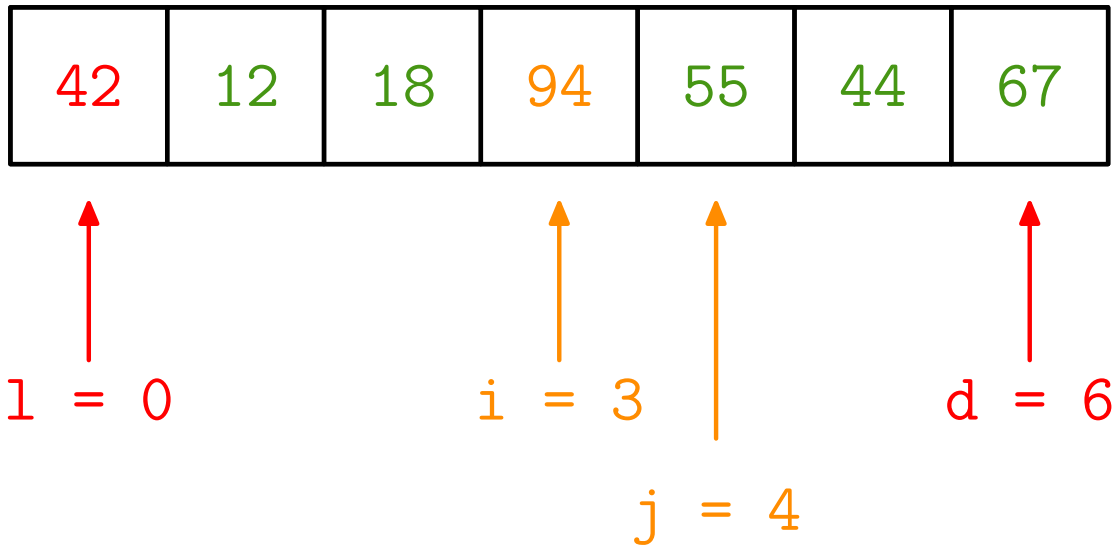
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

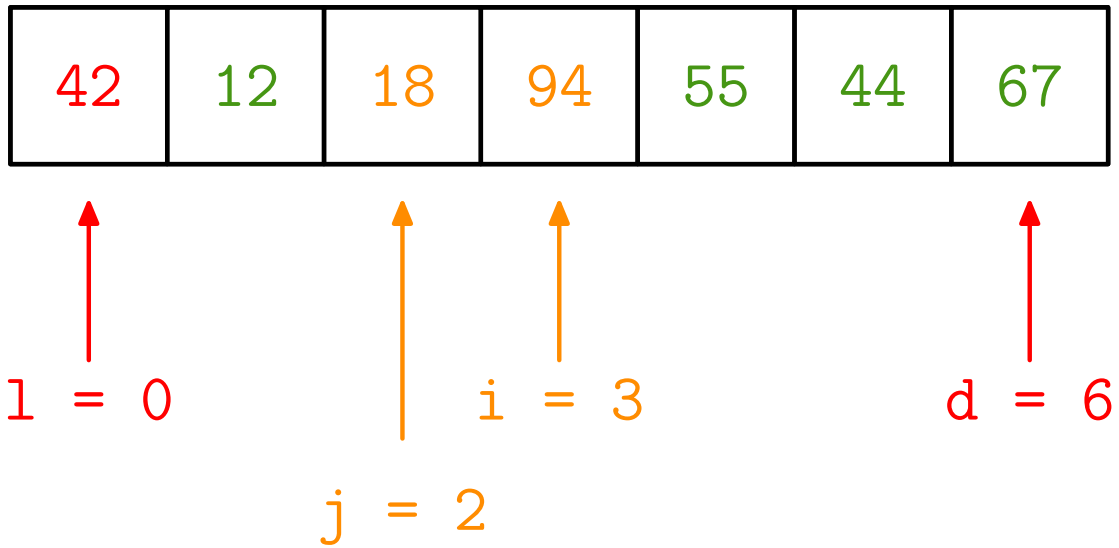
42	12	18	94	55	44	67

`l = 0` `i = 3` `d = 6`

`j = 3`

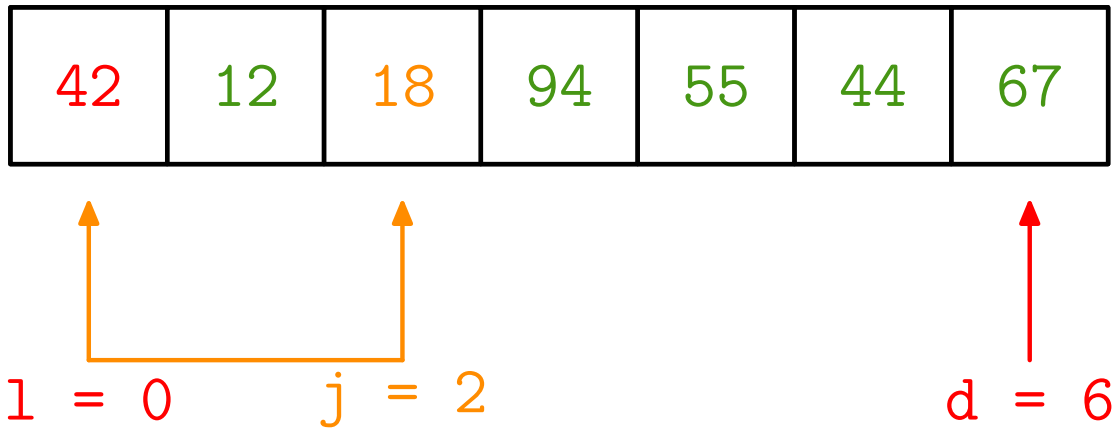
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



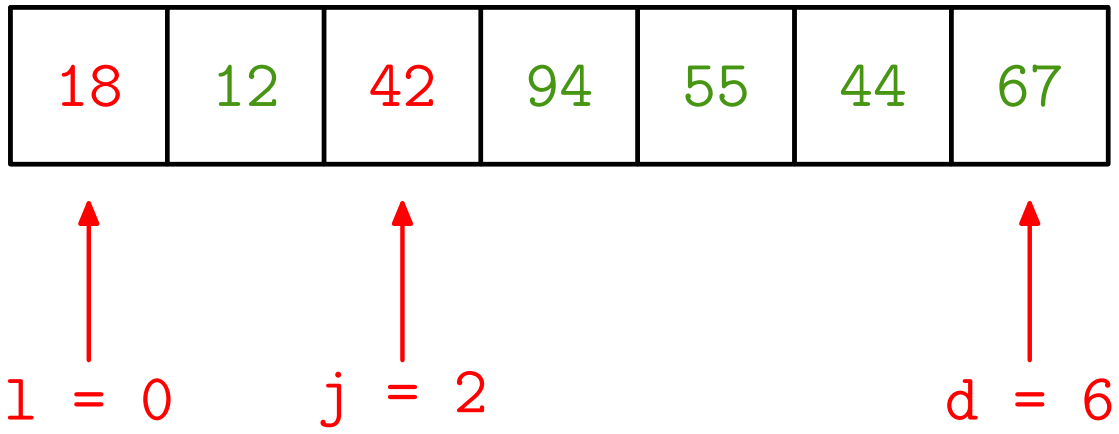
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



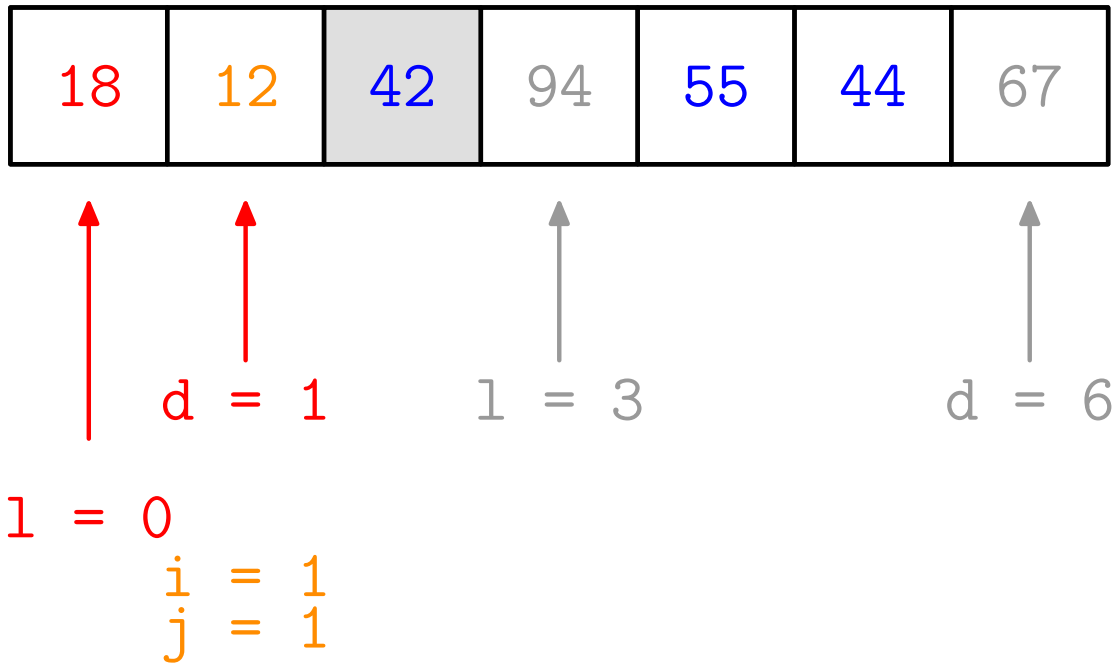
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



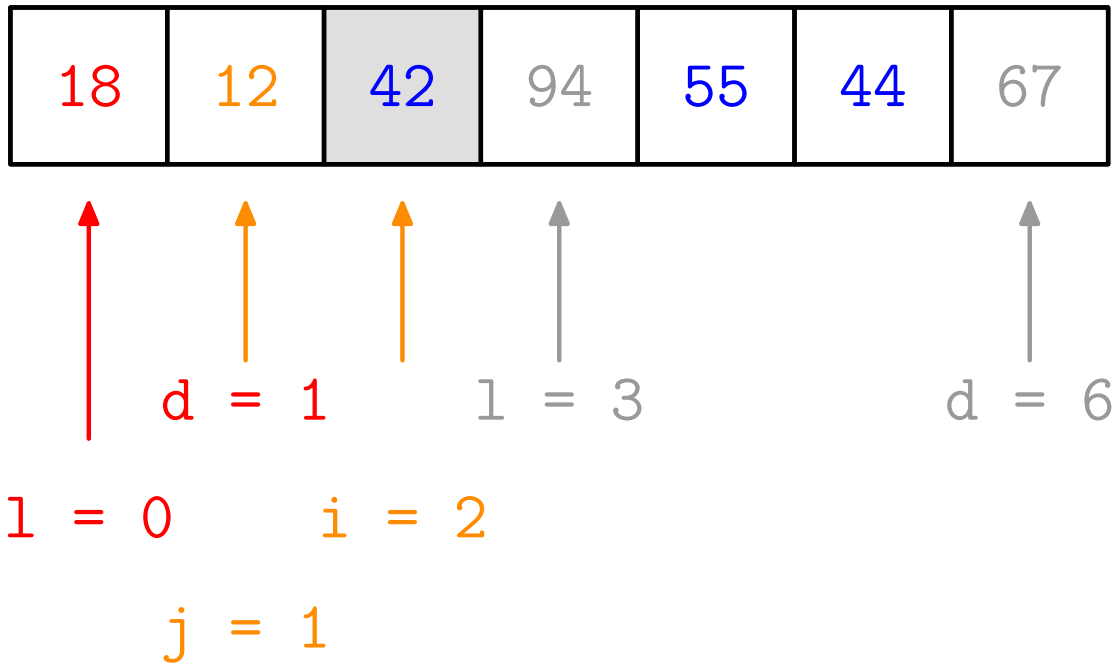
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



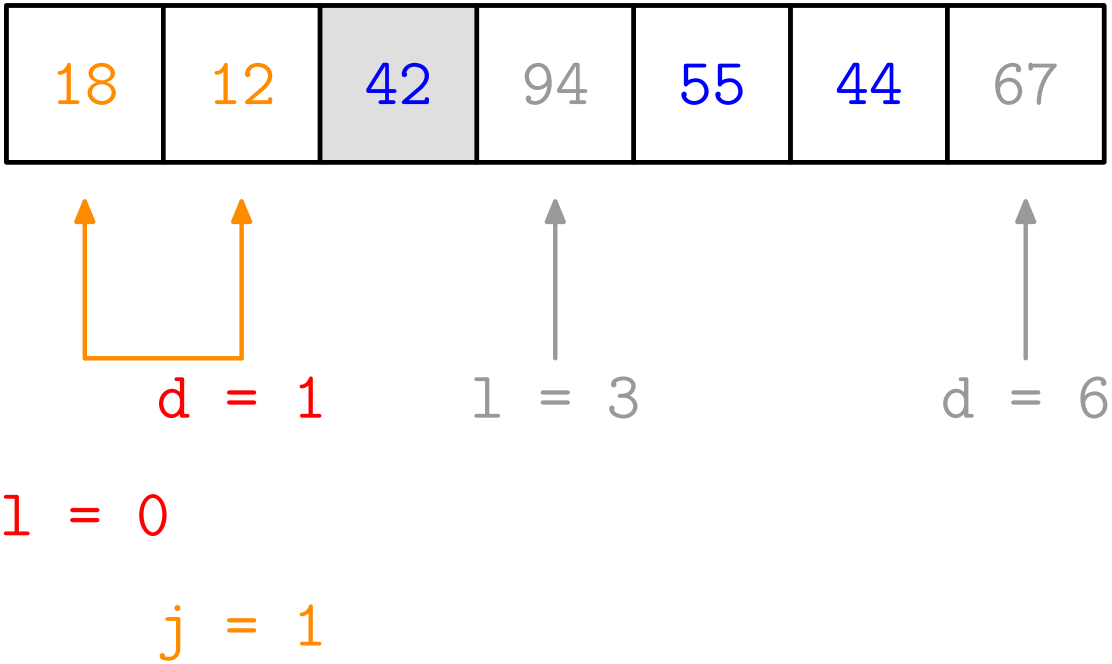
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



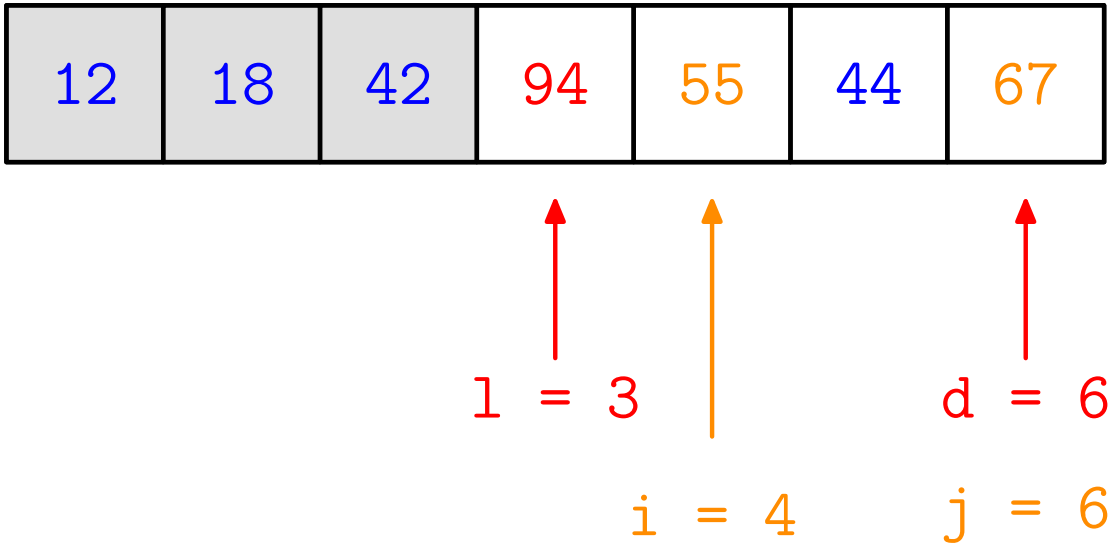
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

12	18	42	94	55	44	67

l = 3

d = 6

j = 6

i = 5

QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

12	18	42	94	55	44	67

`l = 3`

`d = 6`

`j = 6`

`i = 6`

QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

12	18	42	94	55	44	67

l = 3

d = 6

j = 6

i = 7

QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

12	18	42	94	55	44	67

`l = 3` `d = 6`

`j = 6`

QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

12	18	42	67	55	44	94

l = 3

d = 5

j = 5

i = 4

QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

12	18	42	67	55	44	94

`l = 3` `d = 5`

`j = 5`

`i = 5`

QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

12	18	42	67	55	44	94

l = 3 d = 5

j = 5

i = 6

QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

12	18	42	67	55	44	94

l = 3 d = 5

j = 5

QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

12	18	42	44	55	67	94

d = 4

l = 3

i = 4

j = 4

QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

12	18	42	44	55	67	94

d = 4

l = 3

i = 4

j = 3

QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

12	18	42	44	55	67	94

QuickSort — složenost

Za složenost vrijedi:

- prosječna složenost $= O(n \log_2 n)$, za slučajne dobro razbacane nizove,
- složenost u najgorem slučaju $= O(n^2)$, za već sortirani i naopako sortirani niz.

Autor QuickSort-a je C. A. R. Hoare, 1962. godine.

QuickSort — funkcija swap

```
#include <stdio.h>

  /* Sortiranje niza QuickSort algoritmom.
     x[l] je kljucni element - dovodimo
     ga na pravo mjesto u polju. */

void swap(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
    return;
}
```

QuickSort — funkcija quick_sort

```
void quick_sort(int x[], int l, int d)
{
    int i, j;

    if (l < d) {
        i = l + 1;
        j = d;
            /* Prolaz mora i za i == j */
        while (i <= j) {
            while (i <= d && x[i] <= x[l]) ++i;
            while (x[j] > x[l]) --j;
            if (i < j) swap(&x[i], &x[j]);
        }
```

QuickSort — funkcija quick_sort (nastavak)

```
        if (l < j) swap(&x[j], &x[l]);
        quick_sort(x, l, j - 1);
        quick_sort(x, j + 1, d);
    }

    return;
}
```

QuickSort — glavni program

```
int main(void) {
    int i, n;
    int x[] = {42, 12, 55, 94, 18, 44, 67};

    n = 7;
    quick_sort(x, 0, n - 1);

    printf("\n sortirano polje x\n");
    for (i = 0; i < n; ++i) {
        printf(" x[%d] = %d\n", i, x[i]);
    }
    return 0;
}
```

Sortiranje i pretraživanje u standardnoj biblioteci

U standardnoj C biblioteci — datoteka zaglavlja `<stdlib.h>`, postoje i sljedeće dvije funkcije:

- `qsort` — QuickSort algoritam za općenito sortiranje niza podataka,
- `bsearch` — Binarno traženje zadanog podatka u (sortiranom) nizu.

U ovim funkcijama možemo sami zadati

- funkciju za uspoređivanje podataka u nizu.

O njima će biti više riječi na zadnjem predavanju, kad naučimo još neke potrebne stvari o pokazivačima. Na primjer,

- kako se jedna funkcija šalje drugoj funkciji kao argument.

Funkcije `qsort` *i* `bsearch`

Prototip funkcije `qsort` za sortiranje niza:

```
void qsort(void *base, size_t n, size_t size,
           int (*comp) (const void *, const void *));
```

Prototip funkcije `bsearch` za binarno traženje zadanog podatka u sortiranom nizu:

```
void *bsearch(const void *key, const void *base,
              size_t n, size_t size,
              int (*comp) (const void *, const void *));
```

Vraća pokazivač na nađeni podatak (ako ga ima), ili `NULL`.

Zadnji argument u obje funkcije je pokazivač na funkciju za uspoređivanje članova niza.