

# Programiranje 2, letnji semestar 2023/4

## Potpogrami

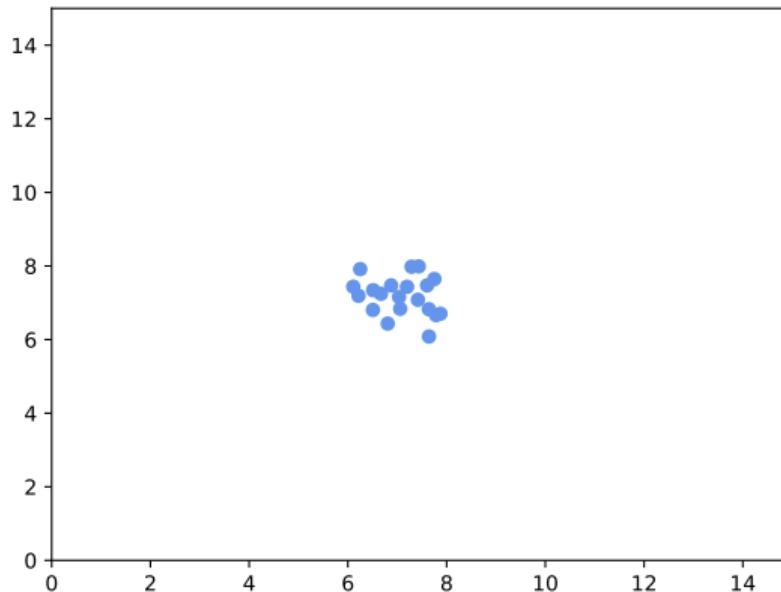
---

Stefan Nikolić

Prirodno-matematički fakultet, Novi Sad

08.04.2024.

Prepostavimo da imamo sledeći problem



Kako da nađemo približan centar grupe tačaka?

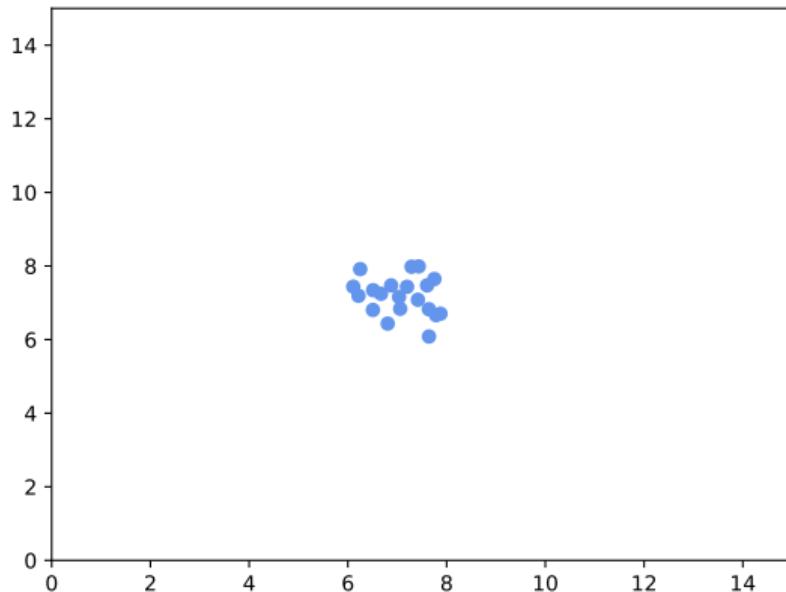
## Prvo pitanje

Kako da predstavimo tačke koristeći samo ono znanje koje smo do sada stekli na predavanjima?

## Odvojeni nizovi koordinata

```
float xs[NUM PTS] = {...  
float ys[NUM PTS] = {...
```

# Da se vratimo na početni problem

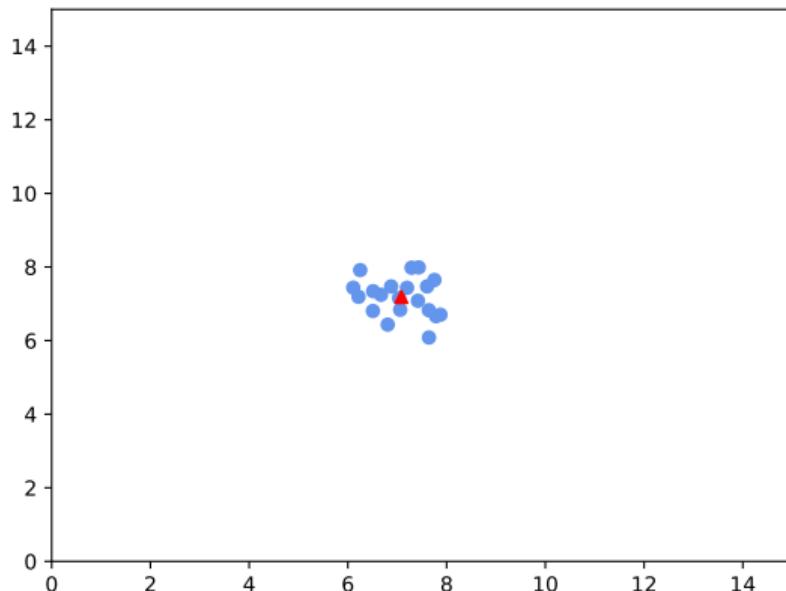


Kako da nađemo približan centar grupe tačaka?

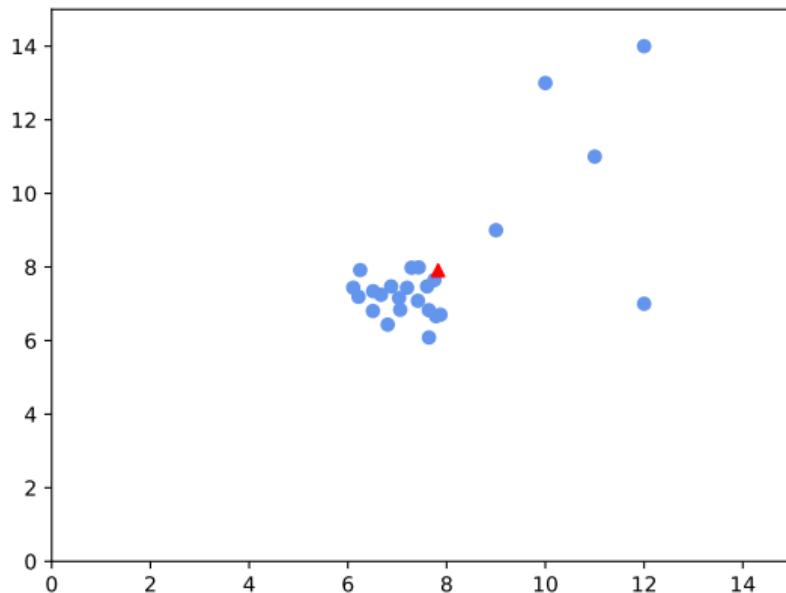
# Prva ideja

centar = (prosek x koordinata svih tačaka, prosek y koordinata svih tačaka)

To je razumna aproksimacija

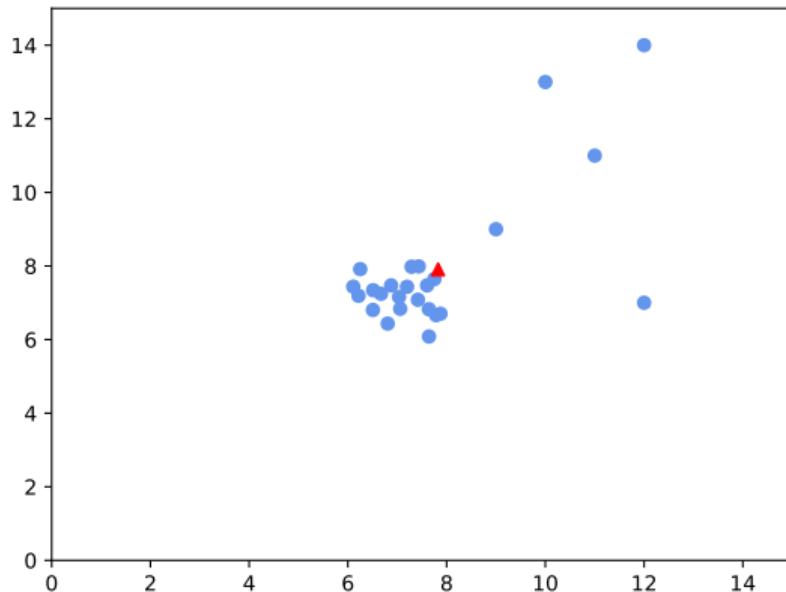


# To je razumna aproksimacija



Kada nema udaljenih tačaka koje nisu deo grupe (eng. *outliers*)

# Da li možemo da smislimo nešto robusnije?

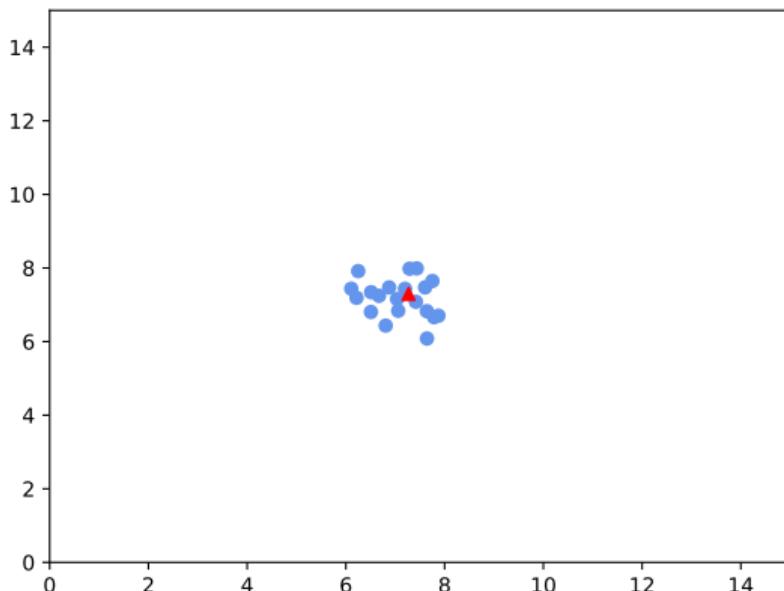


Tako da smanjimo uticaj udaljenih tačaka

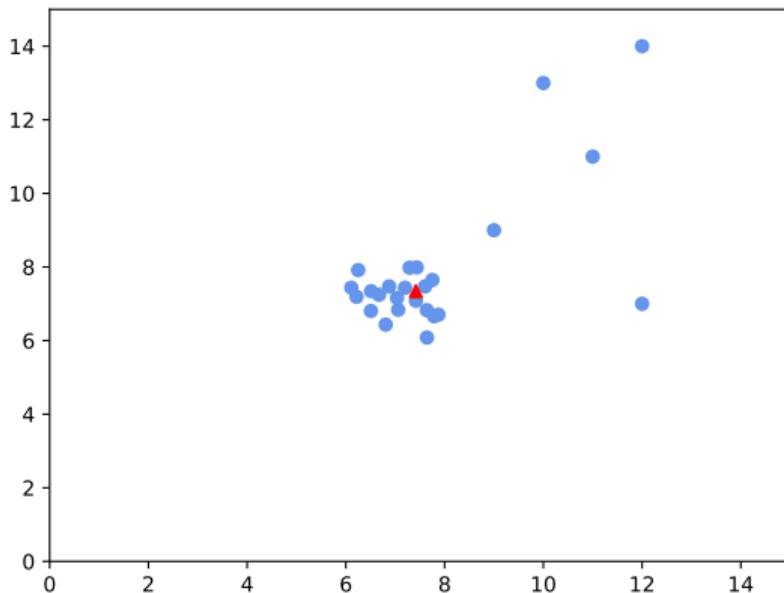
## Druga ideja

centar = (medijana x koordinata svih tačaka, medijana y koordinata svih tačaka)

I to je razumna aproksimacija



I to je razumna aproksimacija



otpornija na prisustvo udaljenih tačaka

## Kako računamo medijanu?

```
float xs[5] = {1.2, 2.7, 0.3, 4.2, 9.1}
```

Prvi korak: sortiramo niz

{0.3, 1.2, 2.7, 4.2, 9.1}

## Drugi korak: biramo element u sredini

{0.3, 1.2, **2.7**, 4.2, 9.1}

Šta radimo ako imamo paran broj elemenata?

# Šta radimo ako imamo paran broj elemenata?

{0.3, 1.2, 2.7, 3.1, 4.2, 9.1}

$$\text{medijana} = \frac{2.7+3.1}{2} = 2.9$$

Kako to možemo implementirati u C++-u, koristeći dosadašnja znanja?

# Najpre inicializujemo nizove koordinata

```
1 #include <iostream>
2 #define NUM PTS 25
3
4 using namespace std;
5
6 int main()
7 {
8     float xs[NUM PTS] = {7.7857203028720035, 7.642458246115663, 6.215313359871936, 7.059634724385643, 6.670815698546658, 6.876282852217749, 7.036072
823998538, 7.290710191791188, 7.639716393843683, 7.752535310413255, 6.1089490156447095, 7.6043411248489505, 7.418263501562334, 6.2496483487612196, 6.886
5125974698395, 7.434551692906639, 6.51122811120167, 7.198011826278233, 7.874699069289809, 6.507268194942792, 12, 9, 12, 11, 10};
9
10    float ys[NUM PTS] = {6.663959610602355, 6.0833932514505, 7.1901041284124805, 6.837614857113309, 7.24503886439218, 7.47176421269738, 7.1577172004
52314, 7.988448542746833, 6.826401869632361, 7.647518865141043, 7.437274473128388, 7.472813287622878, 7.081873708605856, 7.915294590939337, 6.4339023206
4712, 7.988414876843875, 7.3426188638785606, 7.434664291111027, 6.703619537079404, 6.804945018073546, 14, 9, 7, 11, 13};
11
```

Zatim koristimo insertion sort sa prošlog časa da sortiramo x koordinate

```
13     //Koristimo insertion sort sa proslog casa da sortiramo x koordinate
14     for(unsigned i = 1; i < NUM PTS; ++i)
15     {
16         float val = xs[i];
17         unsigned j = i;
18         while((j > 0) && (val < xs[j - 1]))
19         {
20             xs[j] = xs[j - 1];
21             --j;
22         }
23         xs[j] = val;
24     }
```

## Pa izračunamo medijanu

```
26     float median_x = 0;
27     if(NUM PTS % 2)
28         median_x = xs[NUM PTS / 2];
29     else
30         median_x = 0.5 * (xs[NUM PTS / 2] + xs[NUM PTS / 2 + 1]);
31
```

# Ponovimo sve to za y koordinate

```
32      //Koristimo insertion sort sa proslog casa da sortiramo y koordinate
33      for(unsigned i = 1; i < NUM PTS; ++i)
34      {
35          float val = ys[i];
36          unsigned j = i;
37          while((j > 0) && (val < ys[j - 1]))
38          {
39              ys[j] = ys[j - 1];
40              --j;
41          }
42          ys[j] = val;
43      }
44
45      float median_y = 0;
46      if(NUM PTS % 2)
47          median_y = ys[NUM PTS / 2];
48      else
49          median_y = 0.5 * (ys[NUM PTS / 2] + ys[NUM PTS / 2 + 1]);
50
51
52      cout << "centar = (" << median_x << ", " << median_y << ")\n";
53
54      return 0;
55 }
```

I u samo 55 linija koda dobijamo rezultat

```
centar = (7.41826, 7.34262)
```

Da li možemo da pojednostavimo program?

# Izdvojimo računanje medijane u potprogram

```
1 #include <iostream>
2 #define NUM PTS 25
3
4 using namespace std;
5
6 float median(float a[], unsigned len)
7 {
8     //Koristimo insertion sort sa proslog casa da sortiramo niz
9     for(unsigned i = 1; i < len; ++i)
10    {
11        float val = a[i];
12        unsigned j = i;
13        while((j > 0) && (val < a[j - 1]))
14        {
15            a[j] = a[j - 1];
16            --j;
17        }
18        a[j] = val;
19    }
20
21    if(len % 2)
22        return a[len / 2];
23
24    return 0.5 * (a[len / 2] + a[len / 2 + 1]);
25 }
26
```

Zatim umesto dve kopije istog koda, pozivamo dva puta istu funkciju

```
27 int main()
28 {
29     float xs[NUM PTS] = {7.7857203028720035, 7.642458246115663, 6.215313359871936, 7.059634724385643, 6.670815698546658, 6.876282852217749, 7.036072
823998538, 7.290710191791188, 7.639716393843683, 7.752535310413255, 6.1089490156447095, 7.6043411248489505, 7.418263501562334, 6.2496483487612196, 6.806
5125974698395, 7.434551692906639, 6.51122811120167, 7.198011826278233, 7.874699069289809, 6.507268194942792, 12, 9, 12, 11, 10};
30
31     float ys[NUM PTS] = {6.663959610602355, 6.0833932514505, 7.1901041284124805, 6.837614857113309, 7.24503886439218, 7.47176421269738, 7.1577172004
52314, 7.980448542746833, 6.826401869632361, 7.647518865141043, 7.437274473128388, 7.472813287622878, 7.081873708605056, 7.915294590939337, 6.4339023206
4712, 7.988414876843875, 7.3426188638785606, 7.434664291111027, 6.703619537079404, 6.804945018073546, 14, 9, 7, 11, 13};
32
33
34     cout << "centar = (" << median(xs, NUM PTS) << ", " << median(ys, NUM PTS) << ")\n";
35
36     return 0;
37 }
```

U 37 linija koda dobijamo isti rezultat

```
centar = (7.41826, 7.34262)
```

# Nešto o terminologiji

Potprogram (eng. *subprogram*) je za nas skup naredbi sa jasno definisanim ulazom i izlazom, koje je moguće ponovo koristiti, nad proizvoljnim podacima odgovarajućeg tipa.

## Nešto o terminologiji

Osim termina potprogram, za takav skup naredbi se još koriste i nazivi funkcija (eng. *function*), metoda (eng. *method*), rutina (eng. *routine*), podrutina (eng. *subroutine*), procedura (eng. *procedure*)...

## Nešto o terminologiji

Neki autori insistiraju na tome da ovi termini nisu ekvivalentni (na primer, da je procedura isključivo potrprogram koji ne vraća vrednost, ili da je metoda funkcija unutar klase), no oko toga često ne postoji konsenzus.

# Nešto o terminologiji

U svakom slučaju, to su sve detalji koji za nas nisu bitni. Ono što je bitno je razumeti **SUŠTINU** pojma potprograma.

## Zašto koristimo potprograme?

---

# Dobijamo kraći izvorni kod

```
17         }
18         a[j] = val;
19     }
20
21     if(len % 2)
22         return a[len / 2];
23
24     return 0.5 * (a[len / 2] + a[len / 2 + 1]);
25 }
26
27 int main()
28 {
29     float xs[NUM PTS] = {7.7857283028720035, 7.642458246115663, 6.215313359871936,
30     , 7.059634724385643, 6.670815698546658, 6.876282852217749, 7.036072823998538, 7.29071
0191791188, 7.639716393843683, 7.752535310413255, 6.1089490156447095, 7.6043411248489
505, 7.418263581562334, 6.2496483487612196, 6.8865125974698395, 7.434551692906639, 6.
51122811120167, 7.198011826278233, 7.874699069289809, 6.507268194942792, 12, 9, 12, 1
1, 10};
31     float ys[NUM PTS] = {6.663959610602355, 6.0833932514505, 7.1901041284124805,
32     6.837614857113309, 7.24503886439218, 7.47176421269736, 7.157717200452314, 7.98044852
746833, 6.826401869632361, 7.647518865141043, 7.437274473128388, 7.472813287622878, 7
.081873708605056, 7.915294590939337, 6.43390232064712, 7.988414876843875, 7.342618863
8785606, 7.434664291111027, 6.703619537079404, 6.804945018073546, 14, 9, 7, 11, 13};
33
34     cout << "centar = (" << median(xs, NUM PTS) << ", " << median(ys, NUM PTS) <<
35     ")\\n";
36
37     return 0;
38 }
39 //32% kraci kod
center_with_function.cpp [+] [R0] 39,1 Bot center.cpp [R0] 55,1 Bot
```

# Ali i kraći mašinski kod

The screenshot shows the Compiler Explorer interface with two panes. The left pane displays the C++ source code, and the right pane shows the generated RISC-V assembly code.

**C++ Source Code:**

```
33     for(unsigned i = 1; i < NUM PTS; ++i)
34     {
35         float val = ys[i];
36         unsigned j = i;
37         while((j > 0) && (val < ys[j - 1]))
38         {
39             ys[j] = ys[j - 1];
40             --j;
41         }
42         ys[j] = val;
43     }
44
45     float median_y = 0;
46     if(NUM PTS % 2)
47         median_y = ys[NUM PTS / 2];
48     else
49         median_y = 0.5 * (ys[NUM PTS / 2] + ys[NUM PTS
50
51
52     cout << "centar = (" << median_x << ", " << median
53
54     return 0;
55 }
```

**RISC-V Assembly Output:**

```
127    fcvt.d.s    fa0,fs0
128    mv    a0,s0
129    call   std::basic_ostream<char, std::char_traits<char>>::operator<<(_Lc4)
130    lui    a1,%hi(_Lc4)
131    addi   a1,a1,%lo(_Lc4)
132    li    a2,2
133    call   std::basic_ostream<char, std::char_traits<char>>::operator<<(_Lc4)
134    ld    ra,232(sp)
135    ld    s0,224(sp)
136    fld   fs0,216(sp)
137    fld   fs1,208(sp)
138    li    a0,0
139    addi   sp,sp,240
140    jr    ra
141    .L25:
142    mv    a5,a4
143    j    .L9
144    .L24:
145    mv    a5,a4
146    j    .L5
```

Asemblji bez upotrebe funkcije za izračunavanje medijane

# Ali i kraći mašinski kod

The screenshot shows the Compiler Explorer interface with two tabs: C++ source #1 and RISC-V (64-bits) gcc (trunk). The C++ source code implements an insertion sort algorithm for finding the median of an array. The assembly output shows the generated RISC-V assembly code, which includes calls to std::basic\_ostream<char> for printing the result.

```
C++ source #1:
1 #include <iostream>
2 #define NUM PTS 25
3
4 using namespace std;
5
6 float median(float a[], unsigned len)
7 {
8     //Koristimo insertion sort sa proslog casa da sort
9     for(unsigned i = 1; i < len; ++i)
10    {
11        float val = a[i];
12        unsigned j = i;
13        while((j > 0) && (val < a[j - 1]))
14        {
15            a[j] = a[j - 1];
16            --j;
17        }
18        a[j] = val;
19    }
20
21    if(len % 2)
22        return a[len / 2];
23
24    return 0.5 * (a[len / 2] + a[len / 2 + 1]);
25 }
```

```
RISC-V (64-bits) gcc (trunk):
109    addi   a0,sp,104
110    li    a1,25
111    call   median(float*, unsigned int)
112    fcvt.d.s    fa0,fa0
113    mv    a0,s0
114    call   std::basic_ostream<char>, std::cha
115    lui   a1,%hi(.LC5)
116    addi  a1,a1,%lo(.LC5)
117    li    a2,2
118    call   std::basic_ostream<char>, std::cha
119    ld    ra,216(sp)
120    ld    s0,208(sp)
121    li    a0,0
122    addi sp,sp,224
123    jr    ra
124 .LC2:
125 .word  1056964608
```

Asemblji sa upotrebom funkcije za izračunavanje medijane

O tome kako su potprogrami zapravo implementirani čemo više reći uskoro

# Održavanje je često bitnije od razvoja



Analogija: nije problem samo rezbariti drvo, već i brisati prašinu

# Potprogrami nam omogućuju modularni razvoj softvera

Menjamo implementaciju funkcije  
ali njen interfejs ostaje isti

```
7 void sort(float a[], unsigned len)
8 {
9     //Koristimo insertion sort sa proslog casa da sortiramo niz
10    for(unsigned i = 1; i < len; ++i)
11    {
12        float val = a[i];
13        unsigned j = i;
14        while((j > 0) && (val < a[j - 1]))
15        {
16            a[j] = a[j - 1];
17            --j;
18        }
19        a[j] = val;
20    }
21 }
22 float median(float a[], unsigned len)
23 {
24     sort(a, len);
25
26     if(len % 2)
27         return a[len / 2];
28
29     return 0.5 * (a[len / 2] + a[len / 2 + 1]);
30 }
31
32 int main()
33 {
34 }
```

```
void sort(float a[], unsigned len)
{
    //Koristimo Shell sort sa proslog casa da sortiramo niz
    unsigned gap = 1;
    do
    {
        gap *= 3;
        ++gap;
    } while(gap <= len);

    do
    {
        gap /= 3;
        for(unsigned i = gap; i < len; ++i)
        {
            float val = a[i];
            unsigned j = i;
            while(val < a[j - gap])
            {
                a[j] = a[j - gap];
                j -= gap;
                if(j < gap)
                    break;
            }
            a[j] = val;
        }
    } while(gap > 1);
}

float median(float a[], unsigned len)
{
    sort(a, len);
```

Samim tim ne moramo  
menjati ni logiku preostalog  
dela programa

## Neke od prednosti modularnosti

- Prvo možemo napisati najjednostavniju moguću implementaciju (na primer selection sort)
- Ako se ispostavi da nije dovoljno dobra za naše potrebe, onda možemo utrošiti više vremena na pisanje bolje ali složenije implementacije (na primer Shell sort)

## Neke od prednosti modularnosti

Modularnost je veoma korisna za timski razvoj: dok jedna osoba piše jedan potprogram, druga može da piše neki drugi

# Neke od prednosti modularnosti

The screenshot shows a web browser displaying the Cplusplus.com reference page for the `atan` function from the `cmath` library. The URL is <https://cplusplus.com/reference/cmath/atan/>. The left sidebar has a 'Reference' section with a tree view, where the `<cmath>` node is currently selected. The main content area shows the `atan` function definition: `double atan (double x); float atan (float x); long double atan (long double x);`. It also includes a brief description: "Compute arc tangent" and "Returns the principal value of the arc tangent of  $x$ , expressed in radians." A note at the bottom states: "Notice that because of the sign ambiguity, the function cannot determine with certainty in which quadrant the angle falls only by its tangent value. See [atan2](#) for an alternative that takes a fractional argument instead."

Vrlo često, ni ne moramo pisati nove funkcije, već možemo koristiti one koje su drugi napisali, ili koje smo mi ranije napisali za neki drugi program

# Nije dobro izmišljati toplu vodu, ali

Alan Mishchenko

## ABC: The Way It Should Have Been Designed



Thursday, 28 September 2017 at 11:00 in room BC 420

Research Scientist  
Department of Electrical Engineering and Computer Science  
University of California, Berkeley  
Berkeley, California, USA



### Abstract:

Twelve years ago, in September 2005, the first public version of ABC was released. It featured technology-independent synthesis by DAG-aware rewriting, technology mapping for standard cells and lookup tables, and simple combinational equivalence checking, all based on the And-Inverter Graphs (AIG) data-structure used to unify the computation flow. In the coming years ABC has been adopted as an optimization engine and a research environment by a number of academic and industrial users. The use that followed exposed a number of shortcomings in the original design of ABC. This talk focuses on what is present and, more importantly, what is missing in ABC, and how ABC could be redesigned to make it more versatile and user-friendly. The motivation for this talk is to help academic researchers maximize the usefulness of their tools and set a new standard for future versions of ABC.

## ABC: The Way It Should Have Been Designed

Alan Mishchenko

Department of EECS, UC Berkeley



Jedna od glavnih poruka: ne treba se suzdržavati od pisanja ključnih delova programa iznova i iznova

## Nečto o paradigmama

---

# Šta smo radili do sada?

Način programiranja koji smo do sada proučavali nazivamo *proceduralnim programiranjem*

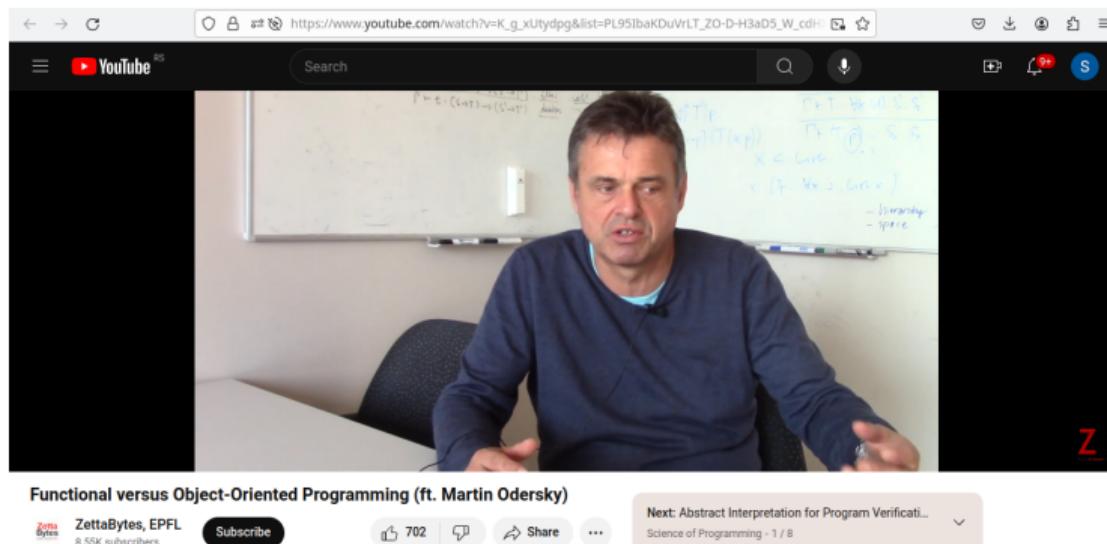
## Šta smo radili do sada?

Proceduralno programiranje se zasniva na podeli problema na potprograme (funkcije, procedure)

## Šta ćemo raditi od sledeće nedelje?

Počevši od sledeće nedelje, bavićemo se objektno-orientisanim programiranjem, u kom potprograme pridružujemo podacima nad kojima operišu, stvarajući posebnu vrstu modula koju nazivamo *objektima*

# Postoji i funkcionalno programiranje



[https://www.youtube.com/watch?v=K\\_g\\_xUtydp&list=PL95IbaKDvrlT\\_ZO-D-H3aD5\\_W\\_cdHxEq](https://www.youtube.com/watch?v=K_g_xUtydp&list=PL95IbaKDvrlT_ZO-D-H3aD5_W_cdHxEq)

# I proceduralno i funkcionalno programiranje sežu u 1930-te

**Alan Turing**

OBE FRS



Turing in 1936

<b>Born</b>	Alan Mathison Turing 23 June 1912 <a href="#">Maida Vale</a> , London, England
<b>Died</b>	7 June 1954 (aged 41) <a href="#">Wilmslow</a> , Cheshire, England

**Alonzo Church**



<b>Born</b>	June 14, 1903 <a href="#">Washington, D.C.</a> , U.S.
<b>Died</b>	August 11, 1995 (aged 92) <a href="#">Hudson, Ohio</a> , U.S.

Proceduralno programiranje

Funktionalno programiranje

## Nagradno pitanje

Šta su Tjuring i Čerč bili po obrazovanju?

---

# **Alan Mathison Turing**

[Biography](#) [MathSciNet](#)

---

Ph.D. Princeton University 1938



Dissertation: *Systems of Logic Based on Ordinals*

Advisor: [Alonzo Church](#)

## Siméon Denis Poisson

[Biography MathSciNet](#)

Ph.D. École Polytechnique 1800 

Dissertation:

Advisor 1: [Joseph Louis Lagrange](#)

Advisor 2: [Pierre-Simon Laplace](#)



## Michel Chasles

[Biography MathSciNet](#)

Ph.D. École Polytechnique 1814 

Dissertation:

Advisor: [Siméon Denis Poisson](#)



## H. A. (Hubert Anson) Newton

B.A. Yale University 1850 

Dissertation:

Advisor: [Michel Chasles](#)



## E. H. (Eliakim Hastings) Moore

[Biography MathSciNet](#)

Ph.D. Yale University 1885 

Dissertation: *Extensions of Certain Theorems of Clifford and Cayley in the Geometry of n Dimensions*

Advisor: [H. A. \(Hubert Anson\) Newton](#)



## Oswald Veblen

[Biography MathSciNet](#)



Ph.D. The University of Chicago 1903 

Dissertation: *A System of Axioms for Geometry*

Advisor: [E. H. \(Eliakim Hastings\) Moore](#)



## Alonzo Church

[Biography MathSciNet](#)



Ph.D. Princeton University 1927 

Dissertation: *Alternatives to Zermelo's Assumption*

Mathematics Subject Classification: 06—Order, lattices, ordered algebraic structures

Advisor: [Oswald Veblen](#)



## Alan Mathison Turing

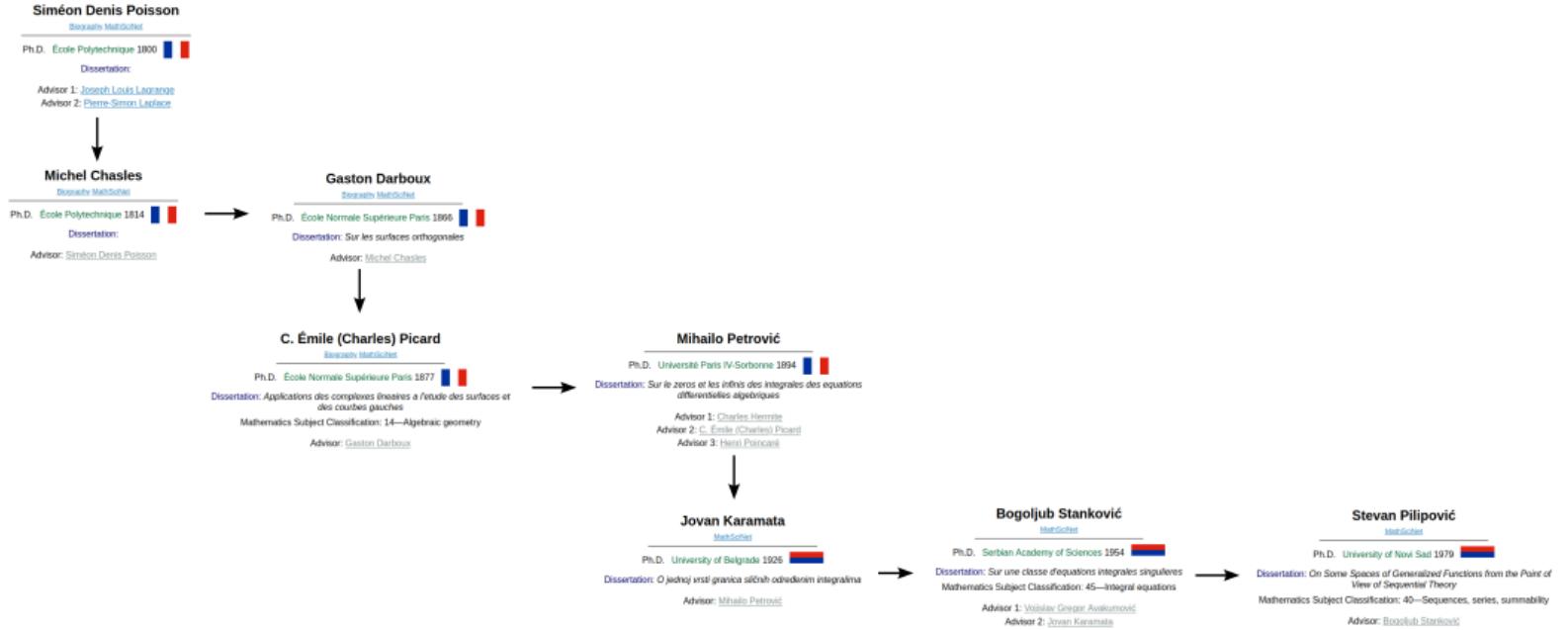
[Biography MathSciNet](#)



Ph.D. Princeton University 1938 

Dissertation: *Systems of Logic Based on Ordinals*

Advisor: [Alonzo Church](#)



## Vratimo se na potprograme

Šta nam oni još omogućuju?

# Verifikacija

---

Neophodno je da osiguramo da naš program ispravno radi

# Dva osnovna pristupa verifikaciji

1. Funkcionalna verifikacija (testiranje): za niz ulaznih podataka **PROVERAVAMO** da li je izlaz očekivan
2. Formalna verifikacija (često je nazivamo samo „verifikacija”): **DOKAZUJEMO** da je naš program ispravan za **SVE** ulazne podatke

# Funkcionalna verifikacija (testiranje)

Screenshot of the Wikipedia article on the Pentium FDIV bug:

The page title is "Pentium FDIV bug". The main content describes the bug as a hardware bug affecting the floating-point unit (FPU) of early Intel Pentium processors. It was discovered in 1994 by Thomas R. Nicely. The bug involves missing values in a lookup table used for floating-point division, leading to small errors in calculations. The severity of the bug is debated, with estimates ranging from 1 in 9 billion divides being inaccurate to 1 in 4 billion. Intel initially handled the issue poorly, leading to a full recall of the processor in December 1994.

On the right side of the page, there is an image of an Intel Pentium processor chip. The chip is yellow with the "intel® pentium™" logo in the center. Below the chip, the caption reads: "66 MHz Intel Pentium (sSpec=SX837) with the FDIV bug".

Testiranje nam može pomoći da otkrijemo da naš program ne radi, ali ako ne prođemo sve kombinacije ulaznih podataka, testiranjem ne можемо dokazati da ispravno radi

Podjela programa na potprograme smanjuje broj kombinacija ulaza u svakom od njih

Tada ograničen broj testova pokriva veći broj mogućih kombinacija i povećava verovatnoću da smo otkrili sve greške

# Kako izgleda testiranje?

ulazi  
a = {2, 5, 7, 1, 4}  
a = {1, 1, 8, 6, 1}  
a = {13, 7}

## Funkcija koju testiramo

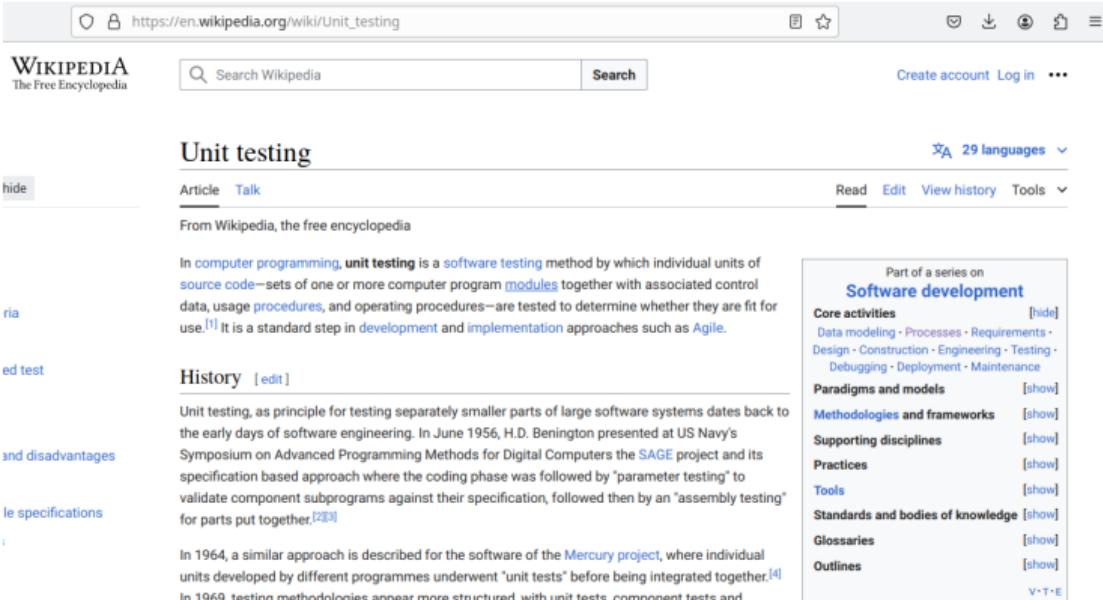
```
23 float median(float a[], unsigned len)
24 {
25     sort(a, len);
26
27     if(len % 2)
28         return a[len / 2];
29
30     return 0.5 * (a[len / 2] + a[len / 2 + 1]);
31 }
```

proizvedeni  
rezultat

očekivani rezultati  
? 4  
1  
10

Ako uočimo da se rezultat razlikuje od očekivanog, detektovali smo grešku  
(ali je još nismo locirali)

# Lociranje grešaka je lakše ako najpre testiramo svaku funkciju posebno



The screenshot shows the Wikipedia article on Unit testing. The page title is "Unit testing". The main content discusses what unit testing is, its history, and its significance in software development. A sidebar on the right is titled "Part of a series on Software development" and lists various sub-topics under "Core activities", "Paradigms and models", "Methodologies and frameworks", "Supporting disciplines", "Practices", "Tools", "Standards and bodies of knowledge", "Glossaries", and "Outlines".

Unit testing

From Wikipedia, the free encyclopedia

In computer programming, **unit testing** is a software testing method by which individual units of source code—sets of one or more computer program [modules](#) together with associated control data, usage procedures, and operating procedures—are tested to determine whether they are fit for use.<sup>[1]</sup> It is a standard step in [development](#) and [implementation](#) approaches such as [Agile](#).

**History** [edit]

Unit testing, as principle for testing separately smaller parts of large software systems dates back to the early days of software engineering. In June 1956, H.D. Benington presented at US Navy's Symposium on Advanced Programming Methods for Digital Computers the [SAGE](#) project and its specification based approach where the coding phase was followed by "parameter testing" to validate component subprograms against their specification, followed then by an "assembly testing" for parts put together.<sup>[2][3]</sup>

In 1964, a similar approach is described for the software of the [Mercury](#) project, where individual units developed by different programmes underwent "unit tests" before being integrated together.<sup>[4]</sup> In 1969, testing methodologies appear more structured, with unit tests, component tests and

Part of a series on  
**Software development**  
Core activities [hide]  
Data modeling · Processes · Requirements · Design · Construction · Engineering · Testing · Debugging · Deployment · Maintenance  
Paradigms and models [show]  
Methodologies and frameworks [show]  
Supporting disciplines [show]  
Practices [show]  
Tools [show]  
Standards and bodies of knowledge [show]  
Glossaries [show]  
Outlines [show]  
V-T-E

Logika: ako funkcija nije ispravna, neće biti ni druga koja je koristi

# Testove bi trebalo automatski pokretati pri svakoj izmeni

The screenshot shows a browser window displaying the English Wikipedia article on "Regression testing". The page title is "Regression testing". The main content discusses what regression testing is, its purpose, and how it is performed. A sidebar on the right is titled "Software development" and lists various sub-topics under this category. The sidebar also includes a "Part of a series on" section with a link to "Software development". The footer of the page contains a "V-T-E" link.

WIKIPEDIA  
The Free Encyclopedia

Search Wikipedia

Regression testing

23 languages

Read Edit View history Tools

From Wikipedia, the free encyclopedia

This article is about software development. For the statistical analysis process, see [Regression analysis](#).

**Regression testing** (rarely, *non-regression testing*)<sup>[1]</sup> is re-running [functional](#) and [non-functional](#) tests to ensure that previously developed and tested software still performs as expected after a change.<sup>[2]</sup> If not, that would be called a [regression](#).

Changes that may require regression testing include [bug fixes](#), software enhancements, configuration changes, and even substitution of [electronic components \(hardware\)](#).<sup>[3]</sup> As regression test suites tend to grow with each found defect, test automation is frequently involved. The evident exception is the [GUIs](#) regression testing, which normally must be executed manually. Sometimes a [change impact analysis](#) is performed to determine an appropriate subset of tests (*non-regression analysis*).<sup>[4]</sup>

**Background** [edit]

As software is updated or changed, or reused on a modified target, emergence of new faults and/or re-emergence of old faults is quite common.

Part of a series on  
**Software development**

Core activities [show]  
Paradigms and models [show]  
Methodologies and frameworks [show]  
Supporting disciplines [show]  
Practices [show]  
Tools [show]  
Standards and bodies of knowledge [show]  
Glossaries [show]  
Outlines [show]  
V-T-E

Da bismo bili sigurni da nova izmena nije pokvarila nešto što smo nakon prethodnog testiranja prepostavili da je ispravno

# Kako nalazimo očekivane rezultate?

ulazi  
a = {2, 5, 7, 1, 4}  
a = {1, 1, 8, 6, 1}  
a = {13, 7}



## Funkcija koju testiramo

```
23 float median(float a[], unsigned len)
24 {
25     sort(a, len);
26
27     if(len % 2)
28         return a[len / 2];
29
30     return 0.5 * (a[len / 2] + a[len / 2 + 1]);
31 }
```



proizvedeni  
rezultat

očekivani rezultati  
? 4  
1  
10

## Kako nalazimo očekivane rezultate?

Jedna mogućnost je da ih ručno izračunamo

# Kako nalazimo očekivane rezultate?

## Funkcija koju testiramo

ulazi

a = {2, 5, 7, 1, 4}

a = {1, 1, 8, 6, 1}

a = {13, 7}

```
23 float median(float a[], unsigned len)
24 {
25     sort(a, len);
26
27     if(len % 2)
28         return a[len / 2];
29
30     return 0.5 * (a[len / 2] + a[len / 2 + 1]);
31 }
```

proizvedeni  
rezultat

?

Referentni model

očekivani  
rezultat

Druga je da imamo referentni model u čiju ispravnost verujemo i sa kojim poredimo ponašanje testirane funkcije

# Šta može biti referentni model?

ulazi

a = {2, 5, 7, 1, 4}

a = {1, 1, 8, 6, 1}

a = {13, 7}

## Funkcija koju testiramo

```
23 float median(float a[], unsigned len)
24 {
25     sort(a, len);
26
27     if(len % 2)
28         return a[len / 2];
29
30     return 0.5 * (a[len / 2] + a[len / 2 + 1]);
31 }
```

proizvedeni  
rezultat

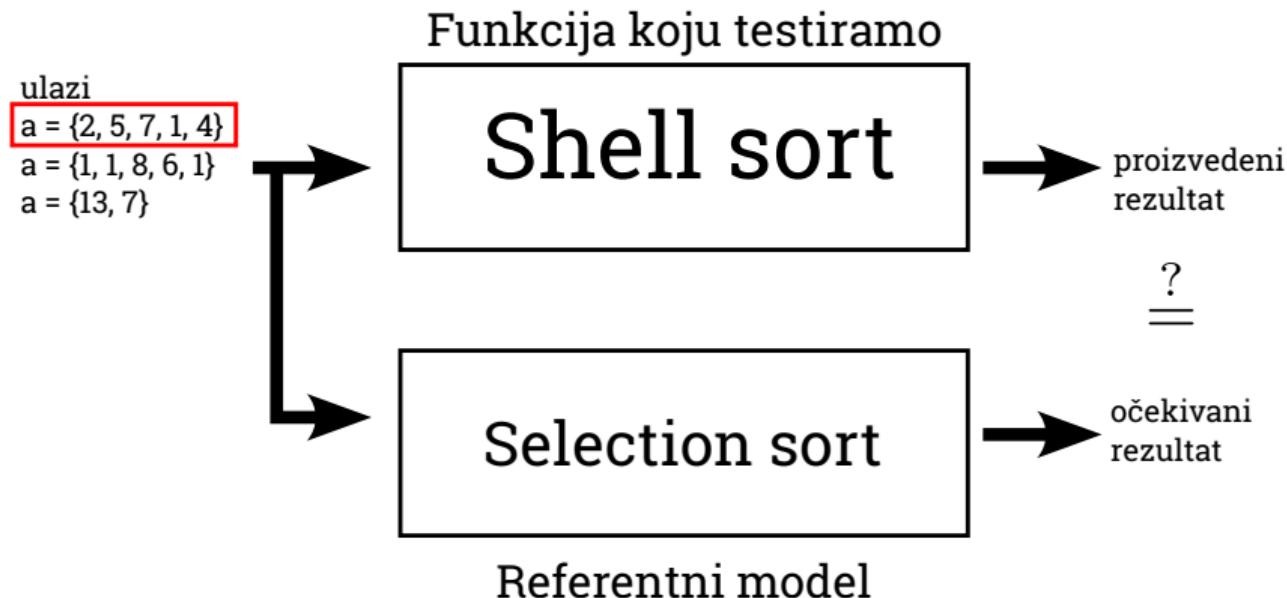
?

---

Referentni model

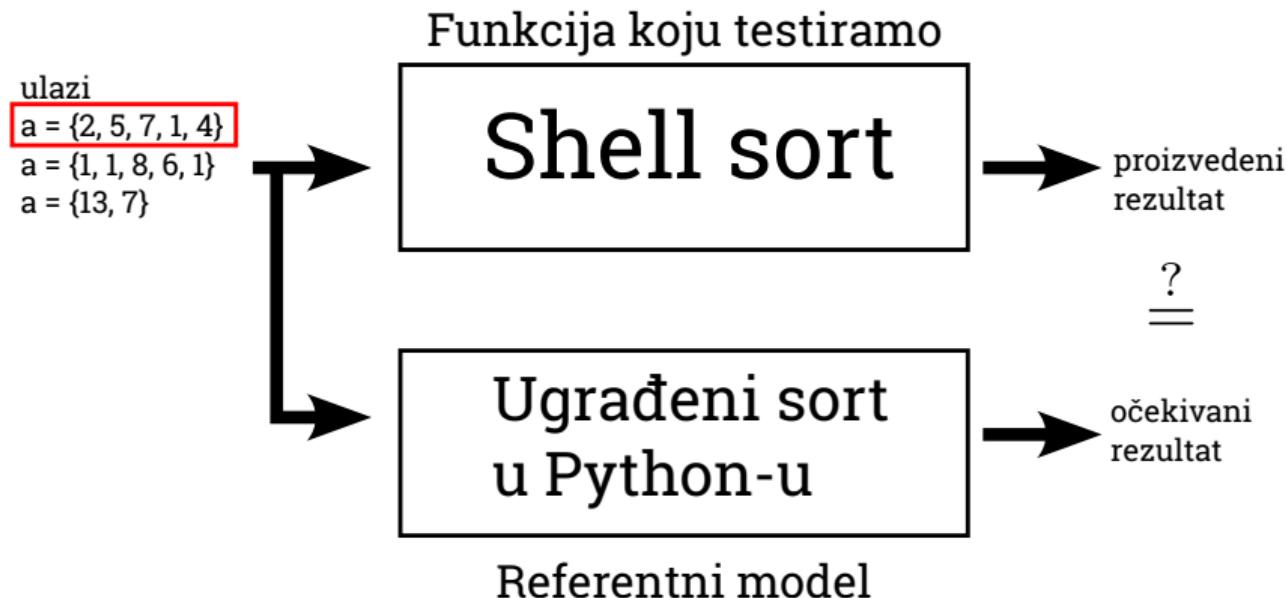
očekivani  
rezultat

# Šta može biti referentni model?



Referentni model je najčešće sporija ali znatno jednostavnija implementacija iste funkcije

# Šta može biti referentni model?

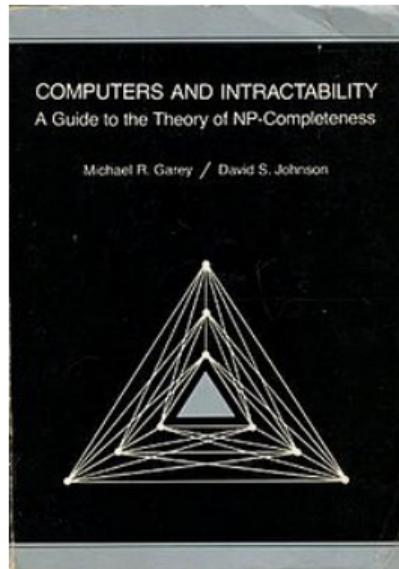


Referentni model je najčešće sporija ali znatno jednostavnija implementacija iste funkcije

## Testiranje upotrebom referentnog modela

Ulazne podatke možemo generisati i nasumično, a očekivane rezultate možemo sačuvati u fajl, da ne bismo mnogo puta tokom razvoja čekali na evaluaciju spore implementacije korišćene u modelu

# Opšti princip: upotreba testiranja za procenu optimalnosti



Stephen Cook  
OC 00nt

Cook in 2008

**Born** Stephen Arthur Cook  
December 14, 1939 (age 84)  
Buffalo, New York

**Alma mater** Harvard University  
University of Michigan

**Known for** NP-completeness  
Propositional proof complexity  
Cook–Levin theorem

Леонид Анатольевич Левин

Дата рождения 2 ноября 1948 (75 лет)  
Место рождения Днепропетровск,  
Украинская ССР, СССР  
Страна СССР, США  
Научная сфера информатика  
Место работы Бостонский университет  
Альма-матер МГУ (мехмат)

Nekada su problemi koje rešavamo toliko složeni da ne znamo ni jedan algoritam pomoću kog možemo da nađemo tačno rešenje za proizvoljnu instancu problema u razumnom vremenu

## Jedan konkretan primer

Problem: Imamo graf i potrebno je da svakom čvoru dodelimo koordinate u 2D ravni, tako da se nijedna dva čvora ne preklapaju a da zbir Euklidovih rastojanja između svaka dva čvora među kojima postoji grana bude minimalan.

U opštem slučaju, ne znamo kako da nađemo optimalno rešenje.

Ali možemo da osmislimo algoritme koji konstruišu prihvatljiva rešenja

[https:](https://github.com/verilog-to-routing/vtr-verilog-to-routing)

[//github.com/verilog-to-routing/vtr-verilog-to-routing](https://github.com/verilog-to-routing/vtr-verilog-to-routing)

## Legalno naspram optimalnog rešenja

Postoje uslovi koje rešenje mora da zadovolji da bi bilo **LEGALNO**

Na primer, u prethodnom problemu, svako rešenje u kom su svim čvorovima dodeljene koordinate i nijedna dva čvora se ne preklapaju je legalno

## Legalno naspram optimalnog rešenja

Optimalno rešenje je ono legalno rešenje koje je najbolje spram postavljenog kriterijuma

U prethodnom primeru, to je rešenje sa minimalnom zbirnom dužinom grana

Optimalno rešenje ne mora biti jedinstveno

Kako da proverimo koliko je naš algoritam daleko od optimuma?

Konstruišemo instance problema za koje unapred znamo optimalno rešenje

The screenshot shows a web browser window with a PDF viewer. The URL in the address bar is [https://websrv.cecs.uci.edu/~papers/compendium94-03/papers/2003/aspdac03/pdf files/07b\\_1.pdf](https://websrv.cecs.uci.edu/~papers/compendium94-03/papers/2003/aspdac03/pdf files/07b_1.pdf). The title of the document is "Optimality and Scalability Study of Existing Placement Algorithms" by Chin-Chih Chang<sup>1</sup>, Jason Cong, Min Xie. The authors are affiliated with the Department of Computer Science, University of California at Los Angeles, Los Angeles, CA 90095, and their email is {cchang, cong, xie} @cs.ucla.edu.

**Optimality and Scalability Study of Existing Placement Algorithms**

Chin-Chih Chang<sup>1</sup>, Jason Cong, Min Xie

Department of Computer Science  
University of California at Los Angeles  
Los Angeles, CA 90095  
Email: {cchang, cong, xie} @cs.ucla.edu

Slično kao kod funkcionalne verifikacije, ni ovde ne možemo da garantujemo optimalnost algoritma u opštem slučaju tako što potvrdimo da daje optimalan odgovor na sve generisane primere

Konstruišemo instance problema za koje unapred znamo optimalno rešenje

The screenshot shows a web browser window with a PDF document loaded. The URL in the address bar is [https://websrv.cec.cs.uci.edu/~papers/compendium94-03/papers/2003/aspdac03/pdf files/07b\\_1.pdf](https://websrv.cec.cs.uci.edu/~papers/compendium94-03/papers/2003/aspdac03/pdf files/07b_1.pdf). The browser interface includes standard controls like back/forward, zoom, and download. The PDF content itself has a title page with the following text:

**Optimality and Scalability Study of Existing Placement Algorithms**  
Chin-Chih Chang<sup>1</sup>, Jason Cong, Min Xie  
Department of Computer Science  
University of California at Los Angeles  
Los Angeles, CA 90095  
Email: { cchang, cong, xie } @cs.ucla.edu

Međutim, ako su primeri koje konstruišemo slični onim koji se javljaju u praksi, ovakvim pristupom možemo da detektujemo da je naš algoritam daleko od optimuma, pa možemo da pokušamo da ga unapredimo

# Formalna verifikacija

---

Formalna verifikacija omogućuje dokazivanje da naš program ispravno radi za **SVE** kombinacije ulaza

Da bismo znali šta dokazujemo, potrebna nam je formalna specifikacija programa



Za to često upotrebljavamo Horovu logiku

Za kratku ilustraciju čemo pozajmiti jedan primer iz Švedske

Part II. Hoare Logic and Program  
Verification

Dilian Gurov

## Example

- Program *Abs*

```
if (x > 0) {
    y = x;
} else {
    y = -x;
}
```

can be specified with

preduslov →  $\langle x = x_0 \rangle$  Abs  $\langle y = |x_0| \rangle$   
 naredba →  $\langle y = |x_0| \rangle$   
 postuslov →  $\langle x = x_0 \rangle$

37

## Proof Tableau

$\langle x = x_0 \rangle$	Precondition
if ( $x > 0$ ) {	If
$\langle x = x_0 \wedge x > 0 \rangle$	Implied( )
$\langle x =  x_0  \rangle$	
y = x;	Assignment
$\langle y =  x_0  \rangle$	
} else {	If
$\langle x = x_0 \wedge \neg(x > 0) \rangle$	Implied( )
$\langle \neg x =  x_0  \rangle$	
y = -x;	Assignment
$\langle y =  x_0  \rangle$	
}	
$\langle y =  x_0  \rangle$	Postcondition

38

Ako se program neposredno pre izvršenja naredbe nalazi u stanju u kom je preduslov zadovoljen, nakon što izvršimo naredbu, mora preći u stanje u kom je zadovoljen postuslov

$\langle x = x_0 \rangle$  preduslov za čitav program; pretpostavimo da važi

```
if (x > 0) {
    ⟨x = x₀ ∧ x > 0⟩
    ⟨x = | x₀ |⟩
    y = x;
    ⟨y = | x₀ |⟩
} else {
    ⟨x = x₀ ∧ ¬(x > 0)⟩
    ⟨¬x = | x₀ |⟩
    y = -x;
    ⟨y = | x₀ |⟩
}
⟨y = | x₀ |⟩ postuslov za čitav program
```

$\langle x = x_0 \rangle$  preduslov za čitav program; pretpostavimo da važi

if ( $x > 0$ ) {

$\langle x = x_0 \wedge x > 0 \rangle$

$\langle x = |x_0| \rangle$

$y = x;$

$\langle y = |x_0| \rangle$

} else {

$\langle x = x_0 \wedge \neg(x > 0) \rangle$

$\langle -x = |x_0| \rangle$

$y = -x;$

$\langle y = |x_0| \rangle$

}

$\langle y = |x_0| \rangle$  postuslov za čitav program

Ako smo ušli u granu if( $x > 0$ ),  
x mora biti veće od nule

$\langle x = x_0 \rangle$  preduslov za čitav program; pretpostavimo da važi

```
if (x > 0) {
    ⟨x = x₀ ∧ x > 0⟩
    ⟨x = |x₀|⟩
    y = x;
    ⟨y = |x₀|⟩
} else {
    ⟨x = x₀ ∧ ¬(x > 0)⟩
    ⟨¬x = |x₀|⟩
    y = -x;
    ⟨y = |x₀|⟩
}
⟨y = |x₀|⟩ postuslov za čitav program
```

Iz prethodnog iskaza, ovo važi po definiciji

$\langle x = x_0 \rangle$  preduslov za čitav program; pretpostavimo da važi

```
if (x > 0) {  
    ⟨x = x₀ ∧ x > 0⟩  
    ⟨x = | x₀ |⟩  
    y = x;  
    ⟨y = | x₀ |⟩  
} else {  
    ⟨x = x₀ ∧ ¬(x > 0)⟩  
    ⟨-x = | x₀ |⟩  
    y = -x;  
    ⟨y = | x₀ |⟩  
}  
⟨y = | x₀ |⟩ postuslov za čitav program
```

Prosta posledica dodele

$\langle x = x_0 \rangle$  preduslov za čitav program; pretpostavimo da važi

```
if (x > 0) {  
    ⟨x = x₀ ∧ x > 0⟩  
    ⟨x = |x₀|⟩  
    y = x;  
    ⟨y = |x₀|⟩  
} else {  
    ⟨x = x₀ ∧ ¬(x > 0)⟩  
    ⟨¬x = |x₀|⟩  
    y = -x;  
    ⟨y = |x₀|⟩  
}  
⟨y = |x₀|⟩ postuslov za čitav program
```

Ako smo u else grani, x je  $\leq 0$

$\langle x = x_0 \rangle$  preduslov za čitav program; pretpostavimo da važi

```
if (x > 0) {  
    ⟨x = x₀ ∧ x > 0⟩  
    ⟨x = |x₀|⟩  
    y = x;  
    ⟨y = |x₀|⟩  
} else {  
    ⟨x = x₀ ∧ ¬(x > 0)⟩  
    ⟨-x = |x₀|⟩  
    y = -x;  
    ⟨y = |x₀|⟩  
}  
⟨y = |x₀|⟩ postuslov za čitav program
```

Tada ovo važi po definiciji apsolutne vrednosti

$\langle x = x_0 \rangle$  preduslov za čitav program; pretpostavimo da važi

if ( $x > 0$ ) {

$\langle x = x_0 \wedge x > 0 \rangle$

$\langle x = |x_0| \rangle$

$y = x;$

$\langle y = |x_0| \rangle$

} else {

$\langle x = x_0 \wedge \neg(x > 0) \rangle$

$\langle -x = |x_0| \rangle$

$y = -x;$

$\langle y = |x_0| \rangle$

A ovo je posledica dodele

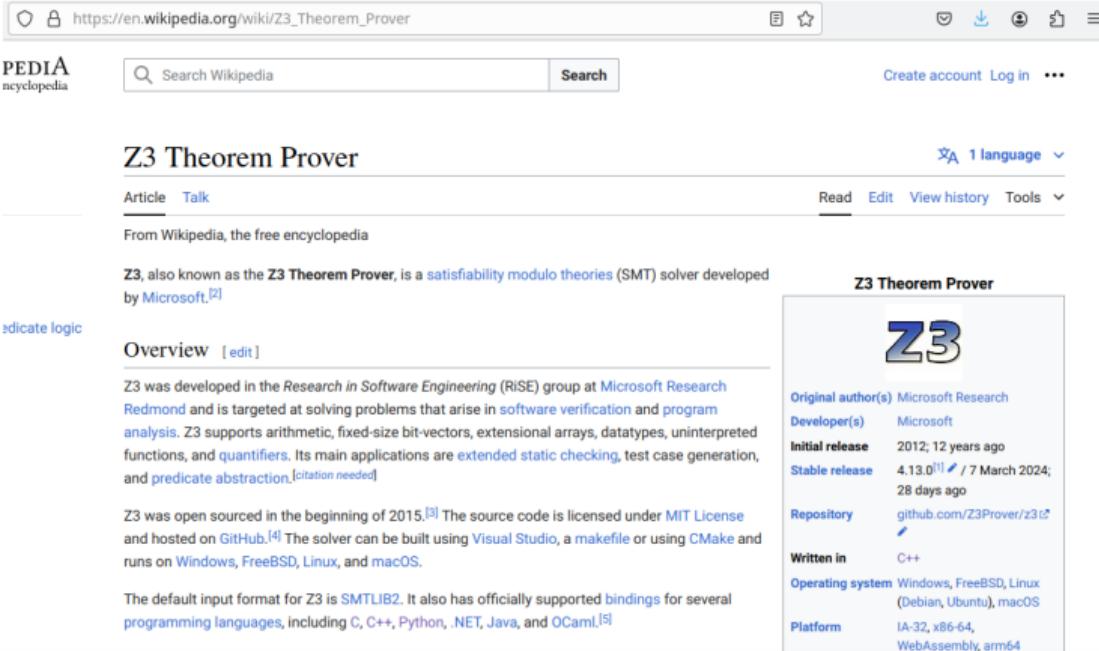
}

$\langle y = |x_0| \rangle$  postuslov za čitav program

$\langle x = x_0 \rangle$  preduslov za čitav program; pretpostavimo da važi  
**if** ( $x > 0$ ) {  
      $\langle x = x_0 \wedge x > 0 \rangle$   
      $\langle x = |x_0| \rangle$   
      $y = x;$   
      $\boxed{\langle y = |x_0| \rangle}$   
  } **else** {  
      $\langle x = x_0 \wedge \neg(x > 0) \rangle$   
      $\langle -x = |x_0| \rangle$   
      $y = -x;$   
      $\boxed{\langle y = |x_0| \rangle}$   
  }  
 $\langle y = |x_0| \rangle$  postuslov za čitav program

Pošto smo dokazali da postuslov proističe iz preduslova za sve moguće putanje izvršenja programa, dokazali smo da je program tačan

# Dokaze ne moramo sprovoditi ručno



The screenshot shows the English Wikipedia page for "Z3 Theorem Prover". The page title is "Z3 Theorem Prover". The main content area includes sections for "Overview", "History", "Features", "Usage", and "Implementation". A sidebar on the left contains links related to predicate logic. On the right, there is a "Z3 Theorem Prover" box containing developer information and a GitHub repository link.

**Z3 Theorem Prover**

From Wikipedia, the free encyclopedia

**Z3**, also known as the **Z3 Theorem Prover**, is a **satisfiability modulo theories** (SMT) solver developed by Microsoft.<sup>[2]</sup>

**Overview** [edit]

Z3 was developed in the *Research in Software Engineering* (RISE) group at Microsoft Research Redmond and is targeted at solving problems that arise in **software verification** and **program analysis**. Z3 supports arithmetic, fixed-size bit-vectors, extensional arrays, datatypes, uninterpreted functions, and quantifiers. Its main applications are **extended static checking**, test case generation, and **predicate abstraction**.<sup>[citation needed]</sup>

Z3 was open sourced in the beginning of 2015.<sup>[3]</sup> The source code is licensed under **MIT License** and hosted on **GitHub**.<sup>[4]</sup> The solver can be built using **Visual Studio**, a **makefile** or using **CMake** and runs on **Windows**, **FreeBSD**, **Linux**, and **macOS**.

The default input format for Z3 is **SMTLIB2**. It also has officially supported **bindings** for several programming **languages**, including **C**, **C++**, **Python**, **.NET**, **Java**, and **OCaml**.<sup>[5]</sup>

**Z3 Theorem Prover**

<b>Z3</b>	
Original author(s)	Microsoft Research
Developer(s)	Microsoft
Initial release	2012; 12 years ago
Stable release	4.13.0 <sup>[1]</sup> / 7 March 2024; 28 days ago
Repository	<a href="https://github.com/Z3Prover/z3">github.com/Z3Prover/z3</a> <sup>[2]</sup>
Written in	C++ <sup>[3]</sup>
Operating system	Windows, FreeBSD, Linux (Debian, Ubuntu), macOS
Platform	IA-32, x86-64, WebAssembly, arm64

Međutim, automatski dokazivači imaju ograničenu moć

## Prednost modularnosti

Što je veći broj stanja u programu i putanja između njih, automatski dokazivač mora da pretraži veći prostor, pa je manje verovatno da će naći dokaz u razumnom vremenu

## Prednost modularnosti

Potprogrami imaju manji prostor stanja od celog programa, pa je lakše verifikovati neke od njih (ciljamo na one kod kojih greške imaju najopasnije posledice)

# Više o verifikaciji možemo čuti od Viktora Kunčaka

The screenshot shows a YouTube video player. The video title is "Basics of Program Verification (ft. Viktor Kuncak)". The channel is "ZettaBytes, EPFL" with 8.55K subscribers. There is a "Subscribe" button. Below the video, there are interaction buttons for likes (62), dislikes, share, and more. A navigation bar includes "All", "From ZettaBytes, EPFL", "Presentations", and a search bar with the text "Basics of verification and". The video frame itself shows a man with short dark hair, wearing a light blue striped shirt, sitting in front of a whiteboard. The whiteboard has several mathematical equations written on it, including:  
-  $\text{panke } \frac{2}{3} = 2$   
-  $\text{Telatar } \ln \frac{(-1)^{\frac{1}{2}}}{\sqrt{v}} = \ln \frac{1}{\sqrt{v}}$   
-  $\text{Rimol } \frac{1}{2} \mu - 2v = (-1)^{\frac{1}{2}} \cdot (-1)^{\frac{1}{2}} \sum_{i=1}^n \alpha_i$   
-  $\text{Hilbert } \frac{1}{2} \mu - 2v = \frac{1}{2} \sum_{i=1}^n \alpha_i$

<https://www.youtube.com/watch?v=20loktdryYM>



- | + Automatic Zoom ▾

## Early Achievements

- Best Student of University of Novi Sad in Class of 2000.
- Aleksandar Popović Award for Best Science Project (*Modular Interpreters in Haskell*, Advisor: Prof. Mirjana Ivanović), University of Novi Sad, 2000.
- Student of the Year of Faculty of Science, University of Novi Sad, 2000.
- Mileva Marić-Einstein Award for accomplishments in Computer Science, University of Novi Sad, 1999
- Awards of Excellence for Student Projects (*Early Deadlock Prevention*, Advisor: Prof. Zoran Budimac), University of Novi Sad, 1999; (*Herbrand's Theorem and the Resolution Method*, Advisor: Prof. Gradimir Vojvodić), University of Novi Sad, 1998
- Fellowship of the Serbian Foundation for Scientific Youth Development, 1995-1998
- University of Novi Sad Fellowship, 1998-2000
- Honorable Mention, 27th Int. Physics Olympiad, Oslo, Norway, 1996, First prizes on National Physics Competition of FR Yugoslavia in 1992, 1994 and 1996.

# Mala digresija

Sir  
**Tony Hoare**  
FRS FREng



Tony Hoare in 2011

Pronunciation [/tənihɔ:/r/](#)

**Leonid Anatolievich Levin**



Leonid Levin in 2010

<b>Born</b>	November 2, 1948 (age 75) Dnipropetrovsk, Ukrainian SSR, Soviet Union
<b>Alma mater</b>	Moscow University Massachusetts Institute of

Šta su zajedničko imali Ljevin i Hor?

# Mala digresija

## Mathematics Genealogy Project

C. A. R. (Tony) Hoare

---

Ph.D.

Dissertation:

Advisor 1: [Andrei Nikolayevich Kolmogorov](#)

Advisor 2: [Leslie Fox](#)

## Mathematics Genealogy Project

Leonid Anatolievich Levin

---

[MathSciNet](#)

Ph.D. Lomonosov Moscow State University 1972 

Dissertation: *Some Theorems on the Algorithmic Approach to Probability Theory and Information Theory (unsuccessful defence for political reasons)*

Advisor 1: [Andrei Nikolayevich Kolmogorov](#)

Ph.D. Massachusetts Institute of Technology 1979 

Dissertation: *A General Notion of Independence of Mathematical Objects: Its Applications to Some Problems of Probability Theory, Mathematical Logic and Algorithm Theory*

Mathematics Subject Classification: 03—Mathematical logic and foundations

Advisor 1: [Albert Ronald da Silva Meyer](#)

Андреј Колмогоров



Лични подаци

Датум рођења	25. април 1903.
Место рођења	Тамбов, Руска Империја
Датум смрти	26. октобар 1987. (84 год.)
Место смрти	Москва, СССР
Држављанство	СССР

# Mala digresija

Mathematical analysis of quicksort shows that, on average, the algorithm takes  $O(n \log n)$  comparisons to sort  $n$  items. In the worst case, it makes  $O(n^2)$  comparisons.

## History [edit]

The quicksort algorithm was developed in 1959 by Tony Hoare while he was a visiting student at Moscow State University. At that time, Hoare was working on a machine translation project for the National Physical Laboratory. As a part of the translation process, he needed to sort the words in Russian sentences before looking them up in a Russian-English dictionary, which was in alphabetical order on magnetic tape.<sup>[5]</sup> After recognizing that his first idea, insertion sort, would be slow, he came up with a new idea. He wrote the partition part in Mercury Autocode but had trouble dealing with the list of unsorted segments. On return to England, he was asked to write code for Shellsort. Hoare mentioned to his boss that he knew of a faster algorithm and his boss bet a sixpence that he did not. His boss ultimately accepted that he had lost the bet. Hoare published a paper about his algorithm in *The Computer Journal Volume 5, Issue 1, 1962, Pages 10–16*.<sup>[6]</sup> Later, Hoare learned about ALGOL and its ability to do recursion that enabled him to publish an improved version of the algorithm in ALGOL in *Communications of the Association for Computing Machinery*, the premier computer science journal of the time.<sup>[2][6]</sup> The ALGOL code is published in *Communications of the ACM (CACM), Volume 4, Issue 7 July 1961, pp 321 Algorithm 63: partition and Algorithm 64: Quicksort*.

Quicksort gained widespread adoption, appearing, for example, in Unix as the default library sort subroutine. Hence, it lent its name to the C standard library subroutine `qsort`<sup>[7]</sup> and in the reference implementation of Java.

Robert Sedgewick's PhD thesis in 1975 is considered a milestone in the study of Quicksort where he resolved many open problems related to the analysis of various pivot selection schemes including Samplesort, adaptive partitioning by Van Emden<sup>[8]</sup> as well as derivation of expected number of comparisons and swaps.<sup>[7]</sup> Jon Bentley and Doug McIlroy in 1993 incorporated various improvements for use in programming libraries,

Best-case performance	$O(n \log n)$ (simple partition) or $O(n)$ (three-way partition and equal keys)
Average performance	$O(n \log n)$
Worst-case space complexity	$O(n)$ auxiliary (naive) $O(\log n)$ auxiliary (Hoare 1962)
Optimal	No

# A govorili smo i o Dancigu i rešavanju otvorenih problema za domaći

**Андрей Николаевич Колмогоров**



Имя при рождении фр. *Andreï Nikolaïevitch Kolmogorov*  
Дата рождения 25 апреля 1903<sup>[1][2]</sup>  
Место рождения Тамбов, Российской империи<sup>[5]</sup>  
Дата смерти 20 октября 1987<sup>[3][4]..</sup>  
(84 года)

**Карацуба Анатолий Алексеевич**



Дата рождения 31 января 1937  
Место рождения Грозный  
Дата смерти 28 сентября 2008  
(71 год)  
Место смерти Москва, Россия  
Страна СССР, Россия  
Научная сфера математика  
Место работы МИАН, МГУ  
Альма-матер МГУ ( мехмат)  
Учёная степень доктор физико-

[https://en.wikipedia.org/wiki/Karatsuba\\_algorithm](https://en.wikipedia.org/wiki/Karatsuba_algorithm)

## History [edit]

The standard procedure for multiplication of two  $n$ -digit numbers requires a number of elementary operations proportional to  $n^2$ , or  $O(n^2)$  in big-O notation. Andrey Kolmogorov conjectured that the traditional algorithm was *asymptotically optimal*, meaning that any algorithm for that task would require  $\Omega(n^2)$  elementary operations.

In 1960, Kolmogorov organized a seminar on mathematical problems in [cybernetics](#) at the [Moscow State University](#), where he stated the  $\Omega(n^2)$  conjecture and other problems in the complexity of computation. Within a week, Karatsuba, then a 23-year-old student, found an algorithm that multiplies two  $n$ -digit numbers in  $O(n^{\log_2 3})$  elementary steps, thus disproving the conjecture. Kolmogorov was very excited about the discovery; he

**Priča o Karacubinom množenju:  
nekad se isplatići i protiv autoriteta**

# Veštačka inteligencija nas polako sustiže

https://deepmind.google/discover/blog/discovering-novel-algorithms-with-alphatensor/

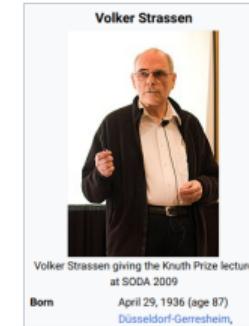
Google DeepMind About Research Technologies Impact Discover

Overview Blog The Podcast Visualising AI

Current state      AlphaTensor      Algorithmic instruction      State update      New state

Repeat

Single-player game played by AlphaTensor, where the goal is to find a correct matrix multiplication algorithm. The state of the game is a cubic array of numbers (shown as grey for 0, blue for 1, and green for -1), representing the remaining work to be done.



Volker Strassen  
Born April 29, 1936 (age 87)  
Düsseldorf-Gerresheim,

Beyond this example, AlphaTensor's algorithm improves on Strassen's two-level algorithm in a finite field for the first time since its discovery 50 years ago. These algorithms for multiplying small matrices can be used as primitives to multiply much larger matrices of arbitrary size.

Moreover, AlphaTensor also discovers a diverse set of algorithms with state-of-the-art complexity – up to thousands of matrix multiplication algorithms for each size, showing that the space of matrix multiplication algorithms is richer than previously thought.

Dovoljno smo odlutali; vratimo se na potprograme

# Sintaksa potprogramma

---

## Deklaracija funkcije

```
povratni_tip identifikator_funkcije(niz_parametara);
```

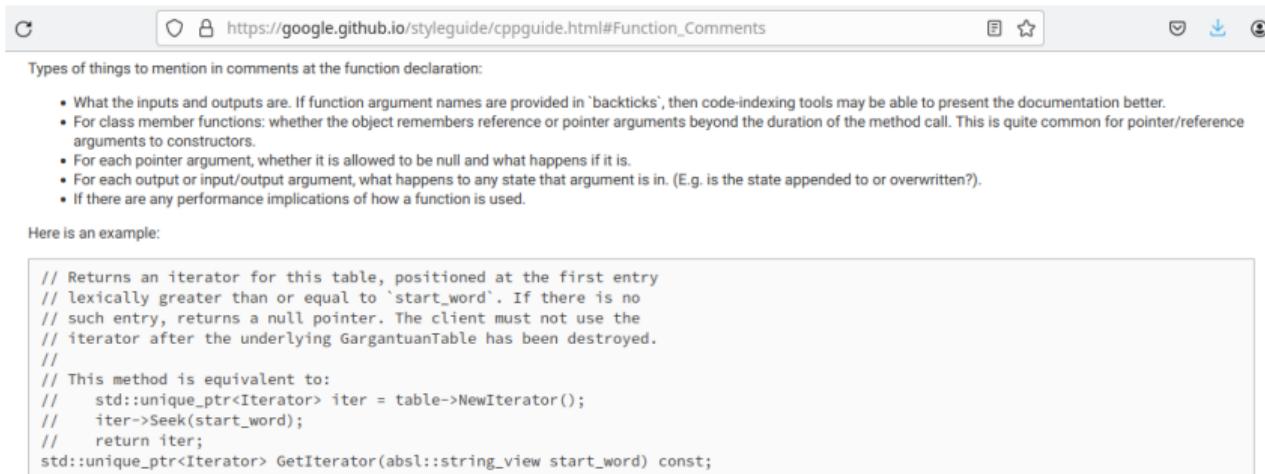
Deklaracija specificira interfejs funkcije sa ostalim funkcijama, ali ne i njenu implementaciju

(govori nam kako da koristimo funkciju, ali ne i kako ona radi)

# Primer

```
int abs(int x);
```

# Preporuka



The screenshot shows a browser window with the URL [https://google.github.io/styleguide/cppguide.html#Function\\_Comments](https://google.github.io/styleguide/cppguide.html#Function_Comments). The page content discusses what types of things to mention in comments at the function declaration, including inputs/outputs, pointer/reference semantics, and performance implications. It includes a code example and a note about avoiding unnecessary verbosity.

Types of things to mention in comments at the function declaration:

- What the inputs and outputs are. If function argument names are provided in ‘backticks’, then code-indexing tools may be able to present the documentation better.
- For class member functions: whether the object remembers reference or pointer arguments beyond the duration of the method call. This is quite common for pointer/reference arguments to constructors.
- For each pointer argument, whether it is allowed to be null and what happens if it is.
- For each output or input/output argument, what happens to any state that argument is in. (E.g. is the state appended to or overwritten?).
- If there are any performance implications of how a function is used.

Here is an example:

```
// Returns an iterator for this table, positioned at the first entry
// lexically greater than or equal to 'start_word'. If there is no
// such entry, returns a null pointer. The client must not use the
// iterator after the underlying GargantuanTable has been destroyed.
//
// This method is equivalent to:
//     std::unique_ptr<Iterator> iter = table->NewIterator();
//     iter->Seek(start_word);
//     return iter;
std::unique_ptr<Iterator> GetIterator(absl::string_view start_word) const;
```

However, do not be unnecessarily verbose or state the completely obvious.

When documenting function overrides, focus on the specifics of the override itself, rather than repeating the comment from the overridden function. In many of these cases, the override needs no additional documentation and thus no comment is required.

When commenting constructors and destructors, remember that the person reading your code knows what constructors and destructors are for, so comments that just say something like “destroys this object” are not useful. Document what constructors do with their arguments (for example, if they take ownership of pointers), and what cleanup the destructor does. If this is trivial, just skip the comment. It is quite common for destructors not to have a header comment.

Preporuka je ostaviti komentar sa opisom šta deklarisana funkcija radi, kako je treba pozvati...

## Definicija funkcije

```
povratni_tip identifikator_funkcije(niz_parametara)
{
    telo_funkcije;
}
```

Definicija funkcije specificira njenu implementaciju (govori nam i kako funkcija radi)

# Primer

```
1  /* Copyright (C) 1991-2024 Free Software Foundation, Inc.  
2   This file is part of the GNU C Library.  
3  
4   The GNU C Library is free software; you can redistribute it and/or  
5   modify it under the terms of the GNU Lesser General Public  
6   License as published by the Free Software Foundation; either  
7   version 2.1 of the License, or (at your option) any later version.  
8  
9   The GNU C Library is distributed in the hope that it will be useful,  
10  but WITHOUT ANY WARRANTY; without even the implied warranty of  
11  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU  
12  Lesser General Public License for more details.  
13  
14  You should have received a copy of the GNU Lesser General Public  
15  License along with the GNU C Library; if not, see  
16  <https://www.gnu.org/licenses/>. */  
17  
18 #include <stdlib.h>  
19  
20 #undef abs  
21  
22 /* Return the absolute value of I. */  
23 int  
24 abs (int i)  
25 {  
26     return i < 0 ? -i : i;  
27 }  
28
```

## Definicija funkcije

```
povratni_tip identifikator_funkcije(niz_parametara)
{
    telo_funkcije;
}
```

Primetimo da je po formi početak definicije identičan deklaraciji

Zašto su nam onda uopšte potrebne deklaracije?

# Posredna rekurzija

```
#include <iostream>
#define MAX_STEP 20
using namespace std;

void state_read(unsigned step)
{
    if(step == MAX_STEP)
        return;

    cout << "Unesite broj izmedju 10 i 20: ";
    int a;
    cin >> a;

    if(a < 10 || a > 20)
    {
        cout << "Uneti broj nije u opsegu [10, 20].\n";
        state_read(step + 1);
    }
    else
        state_compute(step, a);
}

void state_compute(unsigned step, unsigned a)
{
    if(step == MAX_STEP)
        return;

    cout << "a ^ 2 = " << a * a << endl;
    state_read(step + 1);
}

int main()
{
    state_read(0);

    return 0;
```

# Posredna rekurzija

```
circular_dependency.cpp: In function 'void state_read(unsigned int)':  
circular_dependency.cpp:20:17: error: 'state_compute' was not declared in this scope  
  20 |           state_compute(step, a);  
     |           ^~~~~~  
shell returned 1
```

# Posredna rekurzija

```
#include <iostream>
#define MAX_STEP 20
using namespace std;

void state_compute(unsigned, unsigned);

void state_read(unsigned step)
{
    if(step == MAX_STEP)
        return;

    cout << "Unesite broj izmedju 10 i 20: ";
    int a;
    cin >> a;

    if(a < 10 || a > 20)
    {
        cout << "Uneti broj nije u opsegu [10, 20].\n";
        state_read(step + 1);
    }
    else
        state_compute(step, a);
}

void state_compute(unsigned step, unsigned a)
{
    if(step == MAX_STEP)
        return;

    cout << "a ^ 2 = " << a * a << endl;
    state_read(step + 1);
}

int main()
{
    state_read(0);
```

## Posredna rekurzija

```
Unesite broj izmedju 10 i 20: 7
Uneti broj nije u opsegu [10, 20].
Unesite broj izmedju 10 i 20: 13
a ^ 2 = 169
Unesite broj izmedju 10 i 20: 12
a ^ 2 = 144
Unesite broj izmedju 10 i 20: 22
Uneti broj nije u opsegu [10, 20].
Unesite broj izmedju 10 i 20: 19
a ^ 2 = 361
Unesite broj izmedju 10 i 20: █
```

## Posredna rekurzija

Ukoliko funkcija nije eksplicitno deklarisana, njena definicija će poslužiti i kao deklaracija

Pre prvog poziva bilo koje funkcije, ona mora biti deklarisana, da bi kompjuter mogao da izvrši prevođenje u jednom prolazu

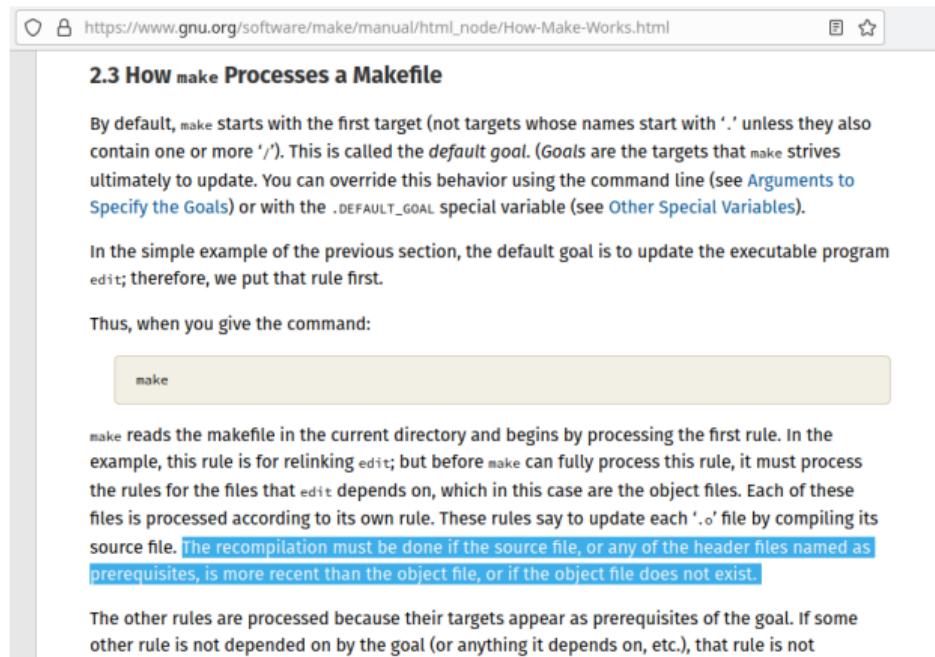
## Graf poziva

Funkcije možemo predstaviti čvorovima u grafu, a činjenica da jedna poziva drugu usmerenim granama između odgovarajućih čvorova

Ako je takav *graf poziva* (eng. *call graph*) acikličan, onda možemo da složimo definicije u topološkom poretku i izbegnemo upotrebu eksplisitnih deklaracija

U suprotnom, to nije moguće

# Ipak, čak i kada nisu potrebne, deklaracije su preporučljive



The screenshot shows a web browser window displaying the Gnu Make manual page at [https://www.gnu.org/software/make/manual/html\\_node/How-Make-Works.html](https://www.gnu.org/software/make/manual/html_node/How-Make-Works.html). The page title is "2.3 How make Processes a Makefile". The text discusses the default behavior of make, which starts with the first target unless specified otherwise. It also mentions how make handles dependencies and prerequisites. A code block shows the command "make" being run.

By default, `make` starts with the first target (not targets whose names start with '.' unless they also contain one or more '/'). This is called the *default goal*. (Goals are the targets that `make` strives ultimately to update. You can override this behavior using the command line (see [Arguments to Specify the Goals](#)) or with the `.DEFAULT_GOAL` special variable (see [Other Special Variables](#)).

In the simple example of the previous section, the default goal is to update the executable program `edit`; therefore, we put that rule first.

Thus, when you give the command:

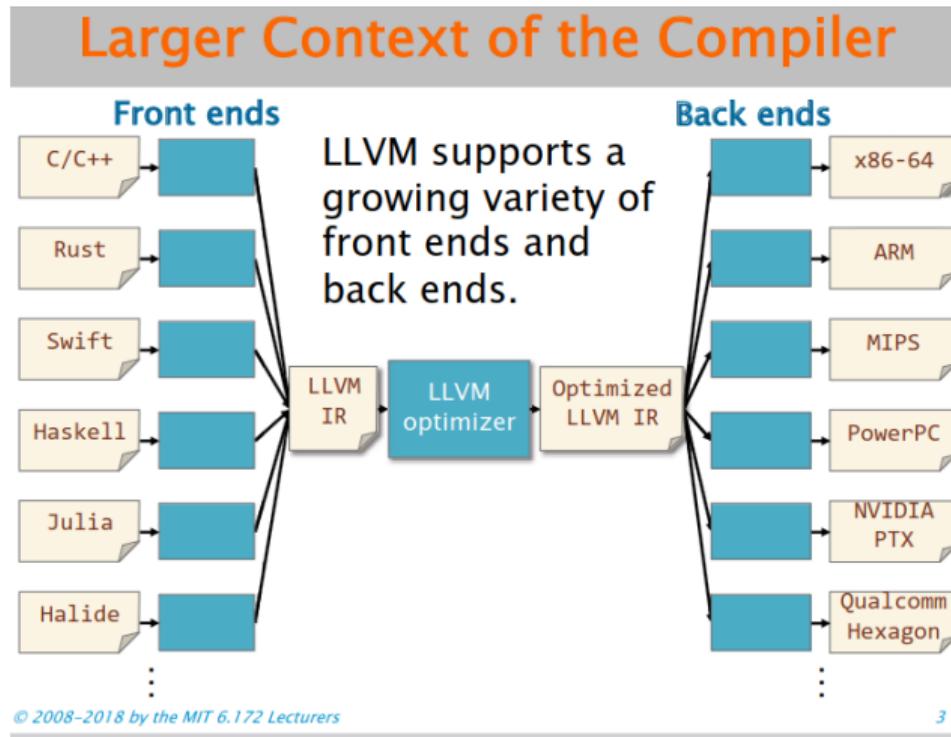
```
make
```

`make` reads the makefile in the current directory and begins by processing the first rule. In the example, this rule is for relinking `edit`; but before `make` can fully process this rule, it must process the rules for the files that `edit` depends on, which in this case are the object files. Each of these files is processed according to its own rule. These rules say to update each `'.o'` file by compiling its source file. [The recompilation must be done if the source file, or any of the header files named as prerequisites, is more recent than the object file, or if the object file does not exist.](#)

The other rules are processed because their targets appear as prerequisites of the goal. If some other rule is not depended on by the goal (or anything it depends on, etc.), that rule is not

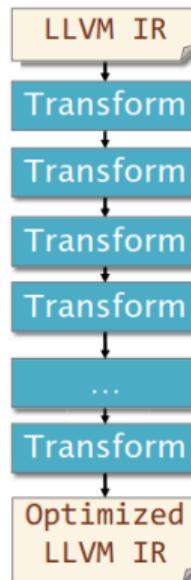
Podjela programa na više fajlova omogućuje jednostavnu inkrementalnu kompilaciju

# Kako izgleda proces kompilacije?



# Kako izgleda proces kompilacije?

## Simple Model of the Compiler



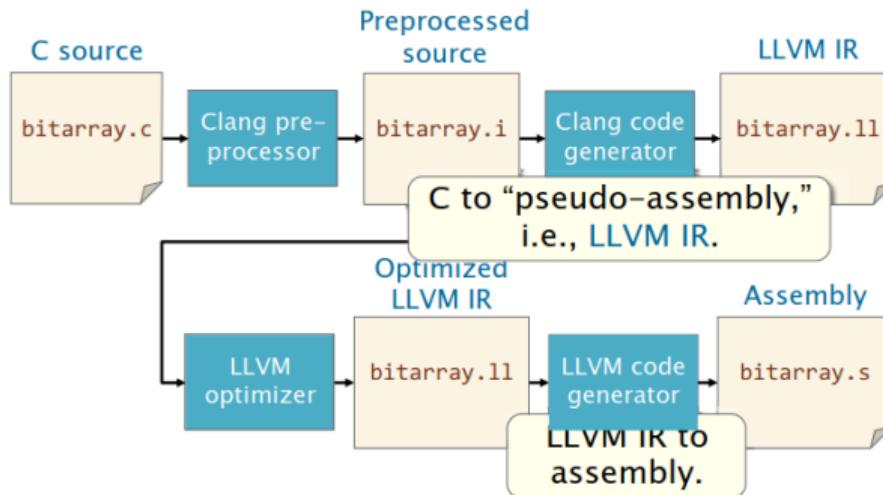
An optimizing compiler performs a sequence of **transformation passes** on the code.

- Each transformation pass **analyzes and edits** the code to try to **optimize** the code's performance.
- A transformation pass might run **multiple times**.
- Passes run in a **predetermined order** that seems to work well most of the time.

# Kako izgleda proces kompilacije?

## Clang/LLVM Compilation Pipeline

To understand this translation process, let us see how the **compiler** reasons about it.



© 2008–2018 by the MIT 6.172 Lecturers

6

# Linker povezuje više objektnih fajlova u jedan izvršni

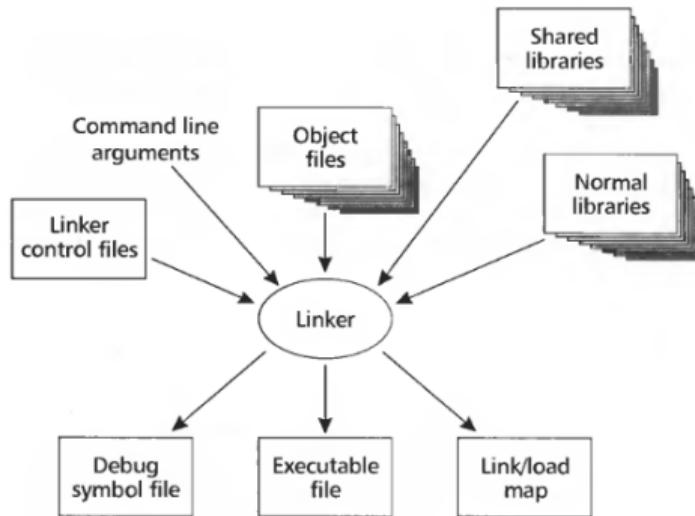


FIGURE 1.1 • The linker process.

Ilustracije iz knjige John R. Levine, "Linkers and Loaders"

# Zaglavlja (eng. *header* fajlovi)

```
struct tacka
{
    int x;
    int y;
};
```

```
tacka.h
#include "tacka.h"
```

```
void ispisi_tacku(const struct tacka* t);
```

```
tacka_2.h
```

```
#include "tacka.h"
#include "tacka_2.h"
#include <iostream>

using namespace std;

void ispisi_tacku(const struct tacka* t)
{
    cout << "(" << t->x << ", " << t->y << ")\n";
}
```

```
tacka.cpp
#include "tacka.h"
#include "tacka_2.h"
```

```
int main()
{
    struct tacka A;
    A.x = -6;
    A.y = 11;

    ispisi_tacku(&A);

    return 0;
}
```

```
include_1.cpp
```

## Zaglavlja (eng. *header* fajlovi)

```
In file included from tacka_2.h:1,
                 from tacka.cpp:2:
tacka.h:1:8: error: redefinition of 'struct tacka'
  1 | struct tacka
    |           ^~~~~~
In file included from tacka.cpp:1:
tacka.h:1:8: note: previous definition of 'struct tacka'
  1 | struct tacka
    |           ^~~~~~
In file included from tacka_2.h:1,
                 from tacka.cpp:2:
tacka.h:1:8: error: redefinition of 'struct tacka'
  1 | struct tacka
    |           ^~~~~~
In file included from tacka.cpp:1:
tacka.h:1:8: note: previous definition of 'struct tacka'
  1 | struct tacka
    |           ^~~~~~
```

## Zaštita od višestrukog uključenja (eng. *include guards*)

Pri nailasku na direktivu #include, pretprocesor će iskopirati sadržaj fajla iz direktive u trenutni fajl

No, i u fajlu koji uključujemo možemo imati pretprocesorske direkitive

Kako bismo osigurali da se neki deo petlje izvrši samo jednom?

# Kako bismo osigurali da se neki deo petlje izvrši samo jednom?

```
bool flag = false;  
while(uslov)  
{  
    if(!flag)  
    {  
        flag = true;  
        naredbe;  
    }  
}
```

# Isti princip primenjujemo i ovde

```
#ifndef TACKA_H_
#define TACKA_H_

struct tacka
{
    int x;
    int y;
};

#endif
```

```
tacka.h          10,6      All
```

```
#include "tacka.h"
```

```
void ispisi_tacku(const struct tacka* t);
```

```
tacka_2.h        3,41     All
```

```
#include "tacka.h"
#include "tacka_2.h"
#include <iostream>

using namespace std;

void ispisi_tacku(const struct tacka* t)
{
    cout << "(" << t->x << ", " << t->y << ")\n";
}
```

```
tacka.cpp        10,1      All
```

```
int main()
{
    struct tacka A;
    A.x = -6;
    A.y = 11;

    ispisi_tacku(&A);

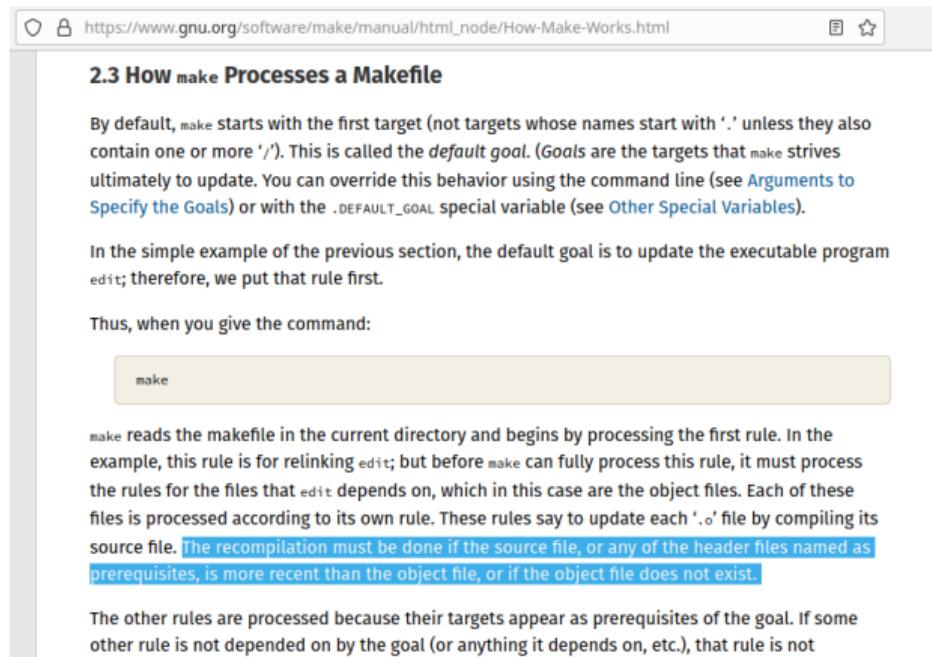
    return 0;
}
```

```
include_1.cpp     3,0-1    All
```

Isti princip primenjujemo i ovde

$$(-6, 11)$$

# Čak i kada nisu potrebne, deklaracije su preporučljive



The screenshot shows a web browser window displaying the official GNU Make documentation at [https://www.gnu.org/software/make/manual/html\\_node/How-Make-Works.html](https://www.gnu.org/software/make/manual/html_node/How-Make-Works.html). The specific section shown is "2.3 How make Processes a Makefile". The text discusses how make starts with the first target by default, unless specified otherwise, and how it processes rules for dependencies. A code block shows the command "make" being run.

By default, `make` starts with the first target (not targets whose names start with '.' unless they also contain one or more '/'). This is called the *default goal*. (Goals are the targets that `make` strives ultimately to update. You can override this behavior using the command line (see [Arguments to Specify the Goals](#)) or with the `.DEFAULT_GOAL` special variable (see [Other Special Variables](#)).

In the simple example of the previous section, the default goal is to update the executable program `edit`; therefore, we put that rule first.

Thus, when you give the command:

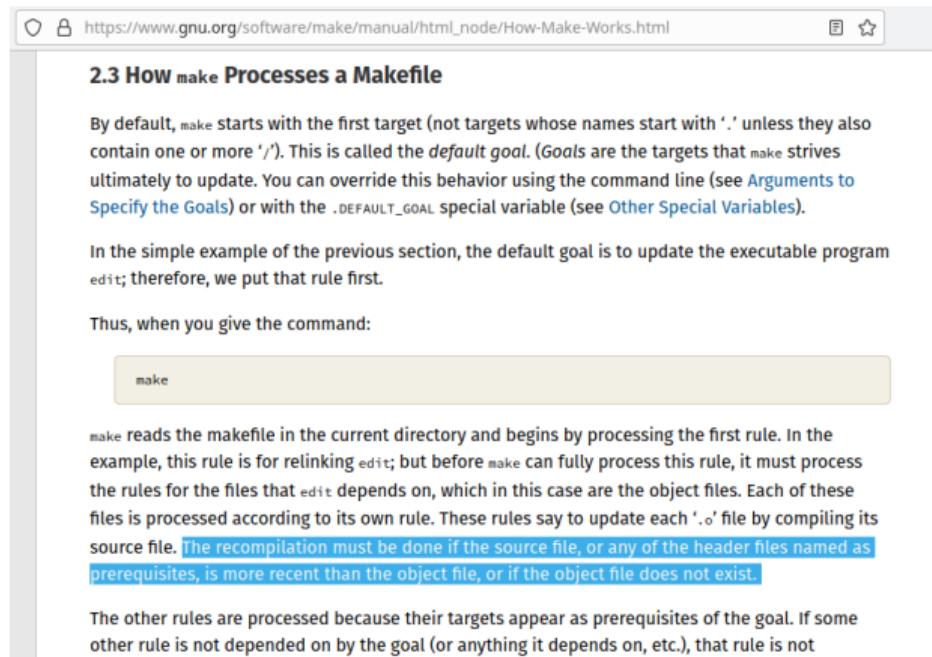
```
make
```

`make` reads the makefile in the current directory and begins by processing the first rule. In the example, this rule is for relinking `edit`; but before `make` can fully process this rule, it must process the rules for the files that `edit` depends on, which in this case are the object files. Each of these files is processed according to its own rule. These rules say to update each `'.o'` file by compiling its source file. [The recompilation must be done if the source file, or any of the header files named as prerequisites, is more recent than the object file, or if the object file does not exist.](#)

The other rules are processed because their targets appear as prerequisites of the goal. If some other rule is not depended on by the goal (or anything it depends on, etc.), that rule is not

Kada pokrenemo bildovanje, make proverava da li je izvorni fajl menjan od poslednje kompilacije

# Čak i kada nisu potrebne, deklaracije su preporučljive



The screenshot shows a web browser window with the URL [https://www.gnu.org/software/make/manual/html\\_node/How-Make-Works.html](https://www.gnu.org/software/make/manual/html_node/How-Make-Works.html). The page title is "2.3 How make Processes a Makefile". The text discusses the default goal behavior of make, mentioning the .DEFAULT\_GOAL variable. It then provides an example where make processes rules for dependencies before reaching the final target. A command-line interface box shows the word "make" being typed.

By default, `make` starts with the first target (not targets whose names start with `.` unless they also contain one or more `/`). This is called the *default goal*. (Goals are the targets that `make` strives ultimately to update. You can override this behavior using the command line (see [Arguments to Specify the Goals](#)) or with the `.DEFAULT_GOAL` special variable (see [Other Special Variables](#)).

In the simple example of the previous section, the default goal is to update the executable program `edit`; therefore, we put that rule first.

Thus, when you give the command:

```
make
```

`make` reads the makefile in the current directory and begins by processing the first rule. In the example, this rule is for relinking `edit`; but before `make` can fully process this rule, it must process the rules for the files that `edit` depends on, which in this case are the object files. Each of these files is processed according to its own rule. These rules say to update each `'.o'` file by compiling its source file. [The recompilation must be done if the source file, or any of the header files named as prerequisites, is more recent than the object file, or if the object file does not exist.](#)

The other rules are processed because their targets appear as prerequisites of the goal. If some other rule is not depended on by the goal (or anything it depends on, etc.), that rule is not

Ako nije, preskače ga i time štedi značajno vreme

Ta ušteda može biti zaista značajna

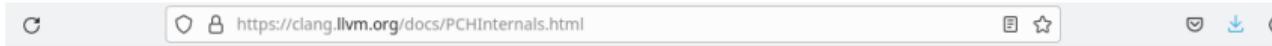


## Ne zaboravimo, #include kopira ceo sadržaj fajla

U principu bismo mogli da koristimo i #include "fajl.cpp"

Ali onda više ne bismo imali mogućnost inkrementalne kompilacije

# Ponekad i kompilacija zaglavlja oduzima zнатно vreme



## Using Precompiled Headers with clang

The Clang compiler frontend, `clang -cc1`, supports two command line options for generating and using PCH files.

To generate PCH files using `clang -cc1`, use the option `-emit-pch`:

```
$ clang -cc1 test.h -emit-pch -o test.h.pch
```

This option is transparently used by `clang` when generating PCH files. The resulting PCH file contains the serialized form of the compiler's internal representation after it has completed parsing and semantic analysis. The PCH file can then be used as a prefix header with the `-include-pch` option:

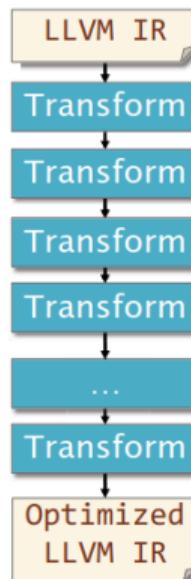
```
$ clang -cc1 -include-pch test.h.pch test.c -o test.s
```

## Design Philosophy

Precompiled headers are meant to improve overall performance concerns. The use case for precompiled headers is relatively simple: when there is a common set of headers that is included in nearly every source file in the project, we *precompile* that bundle of headers into a single precompiled header (PCH file). Then, when compiling the source files in the project, we load the PCH file first (as a prefix header), which acts as a stand-in for that bundle of headers.

Tada i njih можемо unapred kompajlirati

## Simple Model of the Compiler



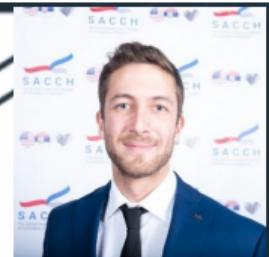
An optimizing compiler performs a sequence of **transformation passes** on the code.

- Each transformation pass **analyzes and edits** the code to try to **optimize** the code's performance.
- A transformation pass might run **multiple times**.
- Passes run in a **predetermined order** that seems to work well most of the time.

Mašinsko učenje nam danas dozvoljava da otklonimo ovo ograničenje



Dejan će početkom juna na PMF-u održati predavanje na ovu temu



# Optimizing Compiler Heuristics with Machine Learning

Ph.D. Candidate: Dejan Grubišić

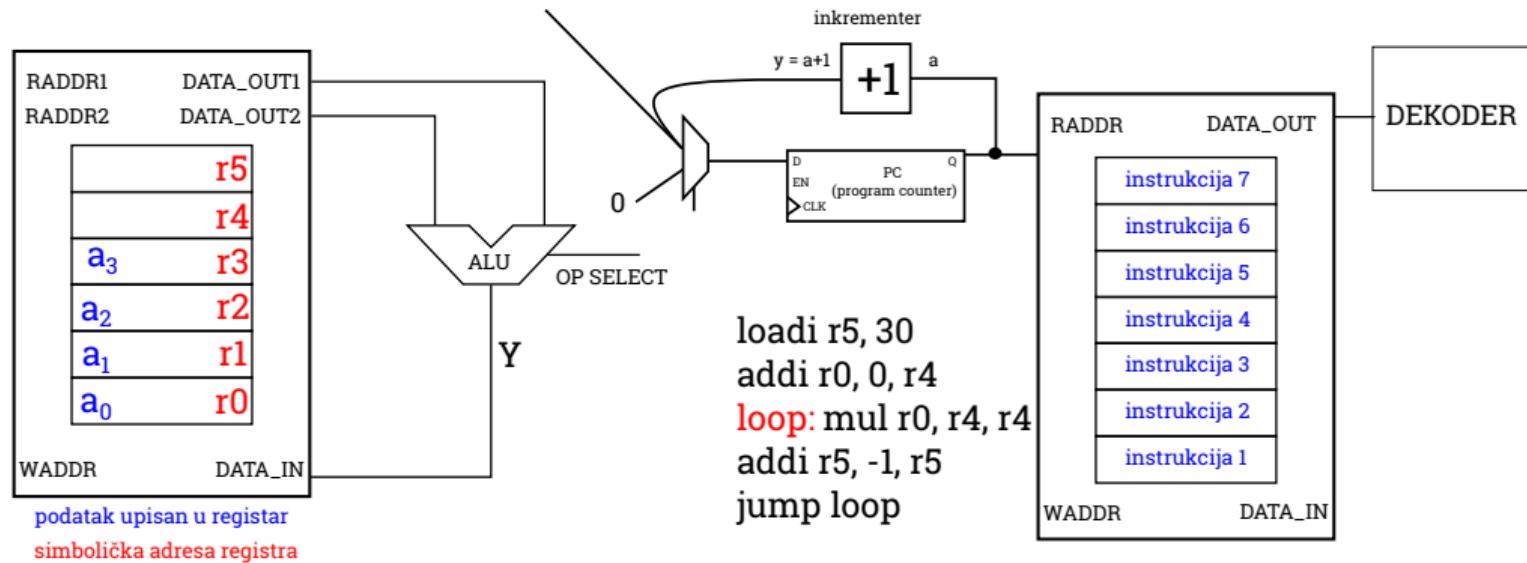
Thesis Committee:  
John Mellor-Crummey (chair)  
Christopher Jermaine  
Ray Simar



## Kako su funkcije implementirane?

---

# Setimo se principa rada fon Nojmanovog računara



## Setimo se još i opseg vidljivosti promenljivih

Promenljive su vidljive samo u bloku u kom su deklarisane, uključujući i sve blokove unutar tog bloka

## I argumenti funkcije su takođe lokalni

Pri pozivu funkcije, pravi se kopija svih prosleđenih parametara koje funkcija može da menja

# Kada ne bi bilo tako, rekurzija ne bi bila moguća

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

## Kako izgleda kopiranje?

To zavisi od konkretne arhitekture procesora

## No opšti princip je sledeći

Procesor ima određen broj registara koji služe za prenos parametara i povratnih vrednosti<sup>1</sup>

---

<sup>1</sup>Povratna vrednost je samo jedna, ali procesor može imati više registara u kojima može biti smeštena

## No opšti princip je sledeći

Pozivna funkcija kopira parametre u registre za prenos parametara

Po završetku računanja, pozvana funkcija kopira povratnu vrednost u registar za vraćanje vrednosti

# To je princip koji nam je poznat i iz svakodnevnog života

Pozivna funkcija : poštar  
Pozvana funkcija: građanin

Poštari ostavljaju pismo tamo gde zna da će ga građanin naći



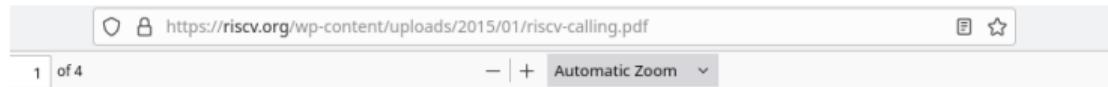
Građanin ostavlja pismo tamo gde zna da će ga poštari naći

Kako poštari i građanin znaju gde da ostave pisma?

# Kako poštari i građanin znaju gde da ostave pisma?

Postoji dogovor (konvencija) kog se pridržavaju

# Pozivne konvencije



## Chapter 18

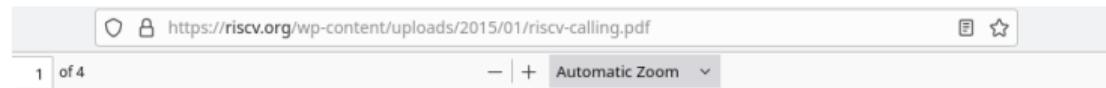
# Calling Convention

This chapter describes the C compiler standards for RV32 and RV64 programs and two calling conventions: the convention for the base ISA plus standard general extensions (RV32G/RV64G), and the soft-float convention for implementations lacking floating-point units (e.g., RV32I/RV64I).

---

*Implementations with ISA extensions might require extended calling conventions.*

I arhitekte procesora pišu pozivne konvencije i određuju „sandučiće” (registre) za prenos



## Chapter 18

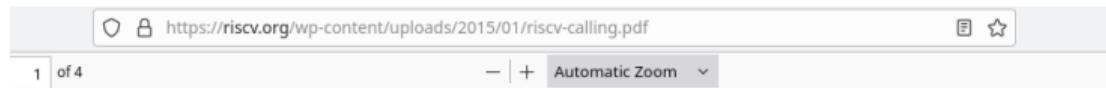
# Calling Convention

This chapter describes the C compiler standards for RV32 and RV64 programs and two calling conventions: the convention for the base ISA plus standard general extensions (RV32G/RV64G), and the soft-float convention for implementations lacking floating-point units (e.g., RV32I/RV64I).

---

*Implementations with ISA extensions might require extended calling conventions.*

Pisci kompjajlera su dužni da osiguraju da su konvencije uvek poštovane



## Chapter 18

# Calling Convention

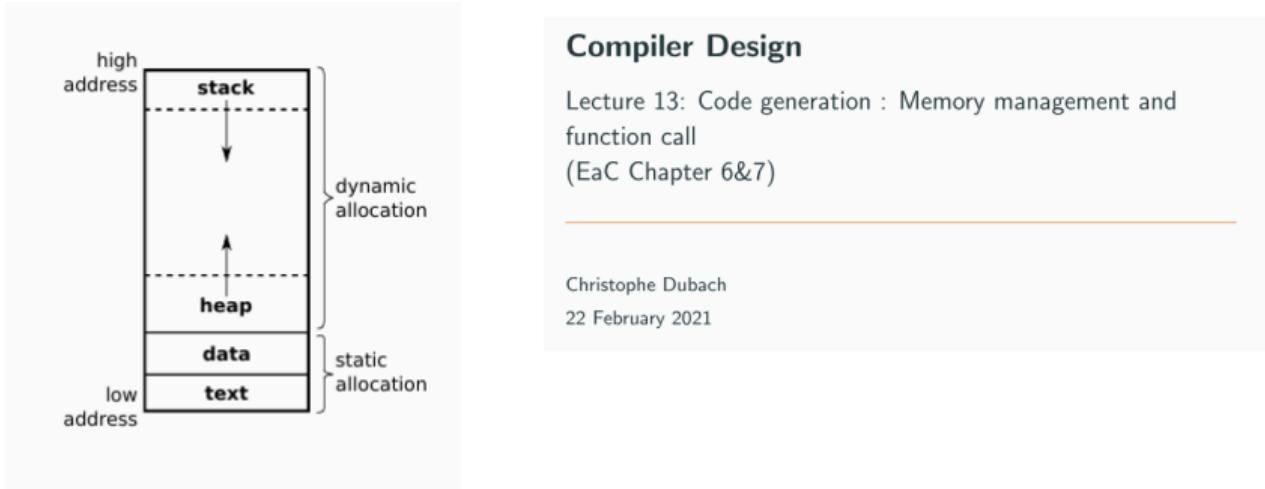
This chapter describes the C compiler standards for RV32 and RV64 programs and two calling conventions: the convention for the base ISA plus standard general extensions (RV32G/RV64G), and the soft-float convention for implementations lacking floating-point units (e.g., RV32I/RV64I).

---

*Implementations with ISA extensions might require extended calling conventions.*

Kompajler je taj koji obezbeđuje primenu konvencije tako što generiše odgovarajuće instrukcije i u pozivnoj i u pozvanoj funkciji

# Kako izgleda kopiranje parametara?



Ako su parametri veći/brojniji od registara za prenos, višak se preliva na **STEK** (eng. stack)

# Jedan konkretan primer

The screenshot shows the Compiler Explorer interface with two panes. The left pane displays the C++ source code, and the right pane shows the generated RISC-V assembly code.

**C++ Source Code (Editor #1):**

```
1 struct s
2 {
3     unsigned a;
4     unsigned b;
5 };
6
7 bool check_s(struct s param)
8 {
9     if(param.a < 10 && param.b < 10)
10        return true;
11
12    return false;
13 }
14
15 bool check_s_ptr(const struct s* param)
16 {
17     if(param -> a < 10 && param -> b < 10)
18        return true;
19
20    return false;
21 }
```

**RISC-V Assembly (Editor #1):**

```
1 check_s(s):
2     addi    sp,sp,-16
3     sd      a0,8(sp)
4     li      a5,9
5     sext.w a0,a0
6     bgtu   a0,a5,.L3
7     lw      a0,12(sp)
8     addi    sp,sp,16
9     sltiu  a0,a0,10
10    jr     ra
11 .L3:
12    li      a0,0
13    addi    sp,sp,16
14    jr     ra
15 check_s_ptr(s const*):
16    lw      a4,0(a0)
17    li      a5,9
18    bgtu   a4,a5,.L8
19    lw      a0,4(a0)
20    sltiu  a0,a0,10
```

The assembly code is annotated with color-coded regions: green for the first function body, yellow for the second function body, and red for the third function body. A tooltip "Spusti stek pokazivač za 16 bajta" is shown over the first red block.

# Jedan konkretan primer

The screenshot shows the Compiler Explorer interface with two panes. The left pane displays the C++ source code, and the right pane shows the generated RISC-V assembly code.

**C++ Source Code:**

```
1 struct s
2 {
3     unsigned a;
4     unsigned b;
5 };
6
7 bool check_s(struct s param)
8 {
9     if(param.a < 10 && param.b < 10)
10        return true;
11    return false;
12 }
13
14 bool check_s_ptr(const struct s* param)
15 {
16     if(param -> a < 10 && param -> b < 10)
17        return true;
18
19    return false;
20 }
```

**RISC-V Assembly Output:**

```
1 check_s(s):
2     addi    sp,sp,-16
3     sd      a0,8(sp) Sačuvaj vrednost registra a0 na stek, 8 bajta iznad sp-a
4     li      a5,9
5     sext.w a0,a0
6     bgtu   a0,a5,.L3 Primetimo da je a0 veličine 8 bajta, a polja strukture imaju po 4, tako da kompjuter može da spakuje celu strukturu u jedan register
7     lw      a0,12(sp)
8     addi   sp,sp,16
9     sltiu  a0,a0,10
10    jr     ra
11 .L3:
12    li      a0,0 Da to nije slučaj, kopiranje bi trajalo duže
13    addi   sp,sp,16
14    jr     ra
15 check_s_ptr(s const*):
16    lw      a4,0(a0)
17    li      a5,9
18    bgtu   a4,a5,.L8
19    lw      a0,4(a0)
20    sltiu  a0,a0,10
```

The assembly code is annotated with explanatory text and highlights. The line `sd a0,8(sp)` is highlighted with a red border and has the annotation "Sačuvaj vrednost registra a0 na stek, 8 bajta iznad sp-a". The block from .L3 to the end of the function is highlighted in pink and has the annotation "Primetimo da je a0 veličine 8 bajta, a polja strukture imaju po 4, tako da kompjuter može da spakuje celu strukturu u jedan register". The line `li a0,0` is highlighted in pink and has the annotation "Da to nije slučaj, kopiranje bi trajalo duže".

# Jedan konkretan primer

The image shows the Compiler Explorer interface on godbolt.org. On the left, the C++ source code is displayed:

```
1 struct s
2 {
3     unsigned a;
4     unsigned b;
5 };
6
7 bool check_s(struct s param)
8 {
9     if(param.a < 10 && param.b < 10)
10        return true;
11
12    return false;
13 }
14
15 bool check_s_ptr(const struct s* param)
16 {
17     if(param -> a < 10 && param -> b < 10)
18        return true;
19
20    return false;
21 }
```

On the right, the generated RISC-V assembly code is shown:

```
1 check_s(s):
2     addi    sp,sp,-16
3     sd     a0,8(sp)
4     li     a5,9      upiši konstantu 9 u
5     sext.w a0,a0      registar a5
6     bgtu   a0,a5,.L3
7     lw     a0,12(sp)
8     addi   sp,sp,16
9     sltiu  a0,a0,10
10    jr    ra
11 .L3:
12    li     a0,0
13    addi   sp,sp,16
14    jr    ra
15 check_s_ptr(s const*):
16    lw     a4,0(a0)
17    li     a5,9
18    bgtu   a4,a5,.L8
19    lw     a0,4(a0)
20    sltiu  a0,a0,10
```

The assembly code is color-coded by line number. Line 4 is highlighted with a red box, and the instruction `li a5,9` is also highlighted with a red box. A tooltip "upiši konstantu 9 u registar a5" is displayed next to the instruction.

# Jedan konkretan primer

The screenshot shows the Compiler Explorer interface with two panes. The left pane displays the C++ source code, and the right pane shows the generated RISC-V assembly code with annotations.

**C++ Source Code:**

```
1 struct s
2 {
3     unsigned a;
4     unsigned b;
5 };
6
7 bool check_s(struct s param)
8 {
9     if(param.a < 10 && param.b < 10)
10        return true;
11
12    return false;
13 }
14
15 bool check_s_ptr(const struct s* param)
16 {
17     if(param -> a < 10 && param -> b < 10)
18        return true;
19
20    return false;
21 }
```

**RISC-V Assembly (Annotations):**

```
1 check_s(s):
2     addi    sp,sp,-16
3     sd     a0,8(sp)
4     li     a5,9
5     sext.w a0,a0          upiši donju reč (4 bajta) u registra a0 (proširenu znakom)
6     bgtu   a0,a5,.L3
7     lw     a0,12(sp)
8     addi   sp,sp,16
9     sltiu  a0,a0,10        Ovo znači da registar a0 sada sadrži polje a strukture param
10    jr    ra
11 .L3:
12    li     a0,0
13    addi   sp,sp,16
14    jr    ra
15 check_s_ptr(s const*):
16    lw     a4,0(a0)
17    li     a5,9
18    bgtu   a4,a5,.L8
19    lw     a0,4(a0)
20    sltiu  a0,a0,10
```

The assembly code is annotated with explanatory text and color-coded regions:

- Line 5: **sext.w a0,a0** upiši donju reč (4 bajta) u registra a0 (proširenu znakom)
- Line 9: Ovo znači da registar a0 sada sadrži polje a strukture param

**Compiler License:** Compiler License

# Jedan konkretan primer

The image shows the Compiler Explorer interface with two panes. The left pane displays the C++ source code, and the right pane shows the generated RISC-V assembly code.

**C++ Source Code (Editor #1):**

```
1 struct s
2 {
3     unsigned a;
4     unsigned b;
5 };
6
7 bool check_s(struct s param)
8 {
9     if(param.a < 10 && param.b < 10)
10        return true;
11
12    return false;
13 }
14
15 bool check_s_ptr(const struct s* param)
16 {
17     if(param -> a < 10 && param -> b < 10)
18        return true;
19
20    return false;
21 }
```

**RISC-V Assembly (Editor #1):**

```
1 check_s(s):
2     addi    sp,sp,-16
3     sd     a0,8(sp)
4     li      a5,9
5     sext.w a0,a0
6     bgtu   a0,a5,.L3  ako je a0 > a5, skoči na a3
7     lw     a0,12(sp)
8     addi   sp,sp,16
9     sltiu  a0,a0,10
10    jr    ra
11 .L3:
12    li     a0,0
13    addi   sp,sp,16
14    jr    ra
15 check_s_ptr(s const*):
16    lw     a4,0(a0)
17    li     a5,9
18    bgtu   a4,a5,.L8
19    lw     a0,4(a0)
20    sltiu  a0,a0,10
```

A red box highlights the instruction `bgtu a0,a5,.L3`, which corresponds to the condition `a0 > a5` in the C++ code. A tooltip "ako je a0 > a5, skoči na a3" is displayed above the box.

# Jedan konkretan primer

The screenshot shows the Compiler Explorer interface with two panes. The left pane displays the C++ source code, and the right pane shows the generated RISC-V assembly code.

**C++ Source Code:**

```
1 struct s
2 {
3     unsigned a;
4     unsigned b;
5 };
6
7 bool check_s(struct s param)
8 {
9     if(param.a < 10 && param.b < 10)
10        return true;
11    return false;
12 }
13
14 bool check_s_ptr(const struct s* param)
15 {
16     if(param -> a < 10 && param -> b < 10)
17        return true;
18    return false;
19 }
20
21 }
```

**RISC-V Assembly Output:**

```
1 check_s(s):
2     addi    sp,sp,-16
3     sd     a0,8(sp)
4     li     a5,9
5     sext.w a0,a0
6     bgtu   a0,a5,.L3
7     lw     a0,12(sp)
8     addi    sp,sp,16
9     sltiu  a0,a0,10
10    jr    ra
11 .L3:
12    li     a0,0
13    addi   sp,sp,16
14    jr    ra
15 check_s_ptr(s const*):
16    lw     a4,0(a0)
17    li     a5,9
18    bgtu   a4,a5,.L8
19    lw     a0,4(a0)
20    sltiu  a0,a0,10
```

A tooltip is displayed over the assembly line `li a0,0`, containing the text:

upiši 0 u registr a0  
u kom će pozivajuća  
funkcija moći da pronađe  
povratnu vrednost

**Compiler License:** Compiler License

# Jedan konkretan primer

The screenshot shows the Compiler Explorer interface with two panes. The left pane displays the C++ source code, and the right pane shows the generated RISC-V assembly code.

**C++ Source Code:**

```
1 struct s
2 {
3     unsigned a;
4     unsigned b;
5 };
6
7 bool check_s(struct s param)
8 {
9     if(param.a < 10 && param.b < 10)
10        return true;
11    return false;
12 }
13
14 bool check_s_ptr(const struct s* param)
15 {
16     if(param -> a < 10 && param -> b < 10)
17        return true;
18    return false;
19 }
20
21 }
```

**RISC-V Assembly Output:**

```
1 check_s(s):
2     addi    sp,sp,-16
3     sd     a0,8(sp)
4     li      a5,9
5     sext.w a0,a0
6     bgtu   a0,a5,.L3
7     lw     a0,12(sp)
8     addi   sp,sp,16
9     sltiu  a0,a0,10
10    jr    ra
11 .L3:
12    li      a0,0
13    addi   sp,sp,16    obriši sve sa steka
14    jr    ra
15 check_s_ptr(s const*):
16    lw     a4,0(a0)
17    li      a5,9
18    bgtu   a4,a5,.L8
19    lw     a0,4(a0)
20    sltiu  a0,a0,10
```

The assembly code is color-coded to highlight specific instructions. A red box highlights the instruction `addi sp,sp,16` at line 13, which is annotated with the text "obriši sve sa steka".

# Jedan konkretan primer

The screenshot shows the Compiler Explorer interface with two panes. The left pane displays the C++ source code, and the right pane shows the generated RISC-V assembly code.

**C++ Source Code (Editor #1):**

```
1 struct s
2 {
3     unsigned a;
4     unsigned b;
5 };
6
7 bool check_s(struct s param)
8 {
9     if(param.a < 10 && param.b < 10)
10        return true;
11    return false;
12 }
13
14 bool check_s_ptr(const struct s* param)
15 {
16     if(param -> a < 10 && param -> b < 10)
17        return true;
18    return false;
19 }
20
21 }
```

**RISC-V Assembly (Editor #1):**

```
1 check_s(s):
2     addi    sp,sp,-16
3     sd     a0,8(sp)
4     li      a5,9
5     sext.w a0,a0
6     bgtu   a0,a5,.L3
7     lw     a0,12(sp)
8     addi   sp,sp,16
9     sltiu  a0,a0,10
10    jr    ra
11 .L3:
12    li     a0,0
13    addi   sp,sp,16
14    jr    ra skoči na povratnu adresu
15 check_s_ptr(s const*):
16    lw     a4,0(a0)
17    li     a5,9
18    bgtu   a4,a5,.L8
19    lw     a0,4(a0)
20    sltiu  a0,a0,10
```

The assembly code is color-coded by section. The first section (lines 1-10) is light blue, the second section (lines 11-14) is light red, and the third section (lines 15-20) is light blue again. The instruction at line 14, `jr ra`, is highlighted with a red box and has a tooltip "skoči na povratnu adresu".

# Jedan konkretan primer

The image shows the Compiler Explorer interface with two panes. The left pane displays the C++ source code, and the right pane shows the generated RISC-V assembly code.

**C++ Source Code (Editor #1):**

```
1 struct s
2 {
3     unsigned a;
4     unsigned b;
5 };
6
7 bool check_s(struct s param)
8 {
9     if(param.a < 10 && param.b < 10)
10        return true;
11
12    return false;
13 }
14
15 bool check_s_ptr(const struct s* param)
16 {
17     if(param -> a < 10 && param -> b < 10)
18        return true;
19
20    return false;
21 }
```

**RISC-V Assembly (Editor #1):**

```
1 check_s(s):
2     addi    sp,sp,-16
3     sd     a0,8(sp)
4     li      a5,9
5     sext.w a0,a0
6     bgtu   a0,a5,.L3
7     lw     a0,12(sp)  učitaj polje b u registar a0
8     addi   sp,sp,16
9     sltiu  a0,a0,10
10    jr    ra
11 .L3:
12    li     a0,0
13    addi   sp,sp,16
14    jr    ra
15 check_s_ptr(s const*):
16    lw     a4,0(a0)
17    li     a5,9
18    bgtu  a4,a5,.L8
19    lw     a0,4(a0)
20    sltiu a0,a0,10
```

The assembly code is color-coded to highlight different sections and instructions. A red box highlights the instruction `lw a0,12(sp)`, which corresponds to the line `lw a0,12(sp) učitaj polje b u registar a0` in the comments above it. This indicates that the compiler has generated assembly code to load the value of variable `b` from memory into register `a0`.

# Jedan konkretan primer

The screenshot shows the Compiler Explorer interface with two panes. The left pane displays the C++ source code, and the right pane shows the generated RISC-V assembly code.

**C++ Source Code (Editor #1):**

```
1 struct s
2 {
3     unsigned a;
4     unsigned b;
5 };
6
7 bool check_s(struct s param)
8 {
9     if(param.a < 10 && param.b < 10)
10        return true;
11
12    return false;
13 }
14
15 bool check_s_ptr(const struct s* param)
16 {
17     if(param -> a < 10 && param -> b < 10)
18        return true;
19
20    return false;
21 }
```

**RISC-V Assembly (Editor #1):**

```
1 check_s(s):
2     addi    sp,sp,-16
3     sd     a0,8(sp)
4     li      a5,9
5     sext.w a0,a0
6     bgtu   a0,a5,.L3
7     lw     a0,12(sp)
8     addi    sp,sp,16 obriši sve sa steka
9     sltiu  a0,a0,10
10    jr     ra
11 .L3:
12    li      a0,0
13    addi    sp,sp,16
14    jr     ra
15 check_s_ptr(s const*):
16    lw     a4,0(a0)
17    li      a5,9
18    bgtu   a4,a5,.L8
19    lw     a0,4(a0)
20    sltiu  a0,a0,10
```

The assembly code is annotated with a red box around the instruction `addi sp,sp,16`, which is labeled "obriši sve sa steka" (erase everything from the stack).

# Jedan konkretan primer

The screenshot shows the Compiler Explorer interface with two panes. The left pane displays the C++ source code, and the right pane shows the generated RISC-V assembly code.

**C++ Source Code (Editor #1):**

```
1 struct s
2 {
3     unsigned a;
4     unsigned b;
5 };
6
7 bool check_s(struct s param)
8 {
9     if(param.a < 10 && param.b < 10)
10        return true;
11    return false;
12 }
13
14 bool check_s_ptr(const struct s* param)
15 {
16     if(param -> a < 10 && param -> b < 10)
17        return true;
18
19    return false;
20 }
```

**RISC-V Assembly (Editor #1):**

```
1 check_s(s):
2     addi    sp,sp,-16
3     sd     a0,8(sp)
4     li     a5,9
5     sext.w a0,a0
6     bgtu   a0,a5,.L3
7     lw     a0,12(sp)
8     addi    sp,sp,16
9     sltiu  a0,a0,10  aka je a0 < 10, upiši 1 u a0,
10    jr    ra      u suprotnom upiši 0 u a0
11 .L3:
12    li     a0,0
13    addi    sp,sp,16
14    jr    ra
15 check_s_ptr(s const*):
16    lw     a4,0(a0)
17    li     a5,9
18    bgtu   a4,a5,.L8
19    lw     a0,4(a0)
20    sltiu  a0,a0,10
```

A tooltip is visible over the assembly instruction `sltiu a0,a0,10`, stating: "aka je a0 < 10, upiši 1 u a0, u suprotnom upiši 0 u a0".

# Jedan konkretan primer

The screenshot shows the Compiler Explorer interface with two panes. The left pane displays the C++ source code, and the right pane shows the generated RISC-V assembly code.

**C++ Source Code (Editor #1):**

```
1 struct s
2 {
3     unsigned a;
4     unsigned b;
5 };
6
7 bool check_s(struct s param)
8 {
9     if(param.a < 10 && param.b < 10)
10        return true;
11    return false;
12 }
13
14 bool check_s_ptr(const struct s* param)
15 {
16     if(param -> a < 10 && param -> b < 10)
17        return true;
18
19    return false;
20 }
```

**RISC-V Assembly (Editor #1):**

```
1 check_s(s):
2     addi   sp,sp,-16
3     sd    a0,8(sp)
4     li    a5,9
5     sext.w a0,a0
6     bgtu a0,a5,.L3
7     lw    a0,12(sp)
8     addi   sp,sp,16
9     sltiu a0,a0,10
10    jr    ra      skoči na povratnu adresu
11 .L3:
12    li    a0,0
13    addi   sp,sp,16
14    jr    ra
15 check_s_ptr(s const*):
16    lw    a4,0(a0)
17    li    a5,9
18    bgtu a4,a5,.L8
19    lw    a0,4(a0)
20    sltiu a0,a0,10
```

The assembly code is color-coded: green for labels and instructions, yellow for comments, red for memory operations (lw, ld), and pink for jumps (jr). A specific jump instruction at line 10 is highlighted with a red box and annotated with the text "skoči na povratnu adresu" (jump to return address).

# Zašto ovo moramo dobro da zapamtimo?

The screenshot shows the Compiler Explorer interface with two panes. The left pane displays the C++ source code, and the right pane shows the generated RISC-V assembly code.

**C++ Source Code:**

```
1 struct s
2 {
3     unsigned a;
4     unsigned b;
5 };
6
7 bool check_s(struct s param)
8 {
9     if(param.a < 10 && param.b < 10)
10        return true;
11    return false;
12 }
13
14 bool check_s_ptr(const struct s* param)
15 {
16     if(param -> a < 10 && param -> b < 10)
17        return true;
18
19    return false;
20 }
```

**RISC-V Assembly Output:**

```
1 check_s(s):
2     addi    sp,sp,-16
3     sd     a0,8(sp)
4     li      a5,9
5     sext.w a0,a0
6     bgtu   a0,a5,.L3
7     lw     a0,12(sp)
8     addi   sp,sp,16
9     sltiu  a0,a0,10
10    jr    ra
11 .L3:
12    li      a0,0
13    addi   sp,sp,16    obriši sve sa steka
14    jr    ra
15 check_s_ptr(s const*):
16    lw     a4,0(a0)
17    li      a5,9
18    bgtu   a4,a5,.L8
19    lw     a0,4(a0)
20    sltiu  a0,a0,10
```

The assembly code is color-coded to highlight specific instructions. The instruction `addi sp,sp,16` at line 13 is highlighted in red and has a tooltip "obriši sve sa steka" (erase everything from the stack).

## Lokalne promenljive nestaju nakon završetka funkcije

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

int* return_sum_address(int a, int b)
{
    int sum = a + b;

    return &sum;
}

int main()
{
    cout << *return_sum_address(6, 5) << endl;

    return 0;
}
```

# Lokalne promenljive nestaju nakon završetka funkcije

```
wrong_ret.cpp: In function 'int* return_sum_address(int, int)':  
wrong_ret.cpp:8:16: warning: address of local variable 'sum' returned [-Wreturn-local-addr]  
  8 |         return &sum;  
   |         ^~~~  
wrong_ret.cpp:6:13: note: declared here  
  6 |         int sum = a + b;  
   |         ^~~~  
/bin/bash: line 1: 30593 Segmentation fault      (core dumped) ./run
```

## Malo više o kopiranjima

Potrogram je u punom smislu te reči potprogram

Pri pozivu potprograma, predaje mu se upravljanje čitavim procesorom

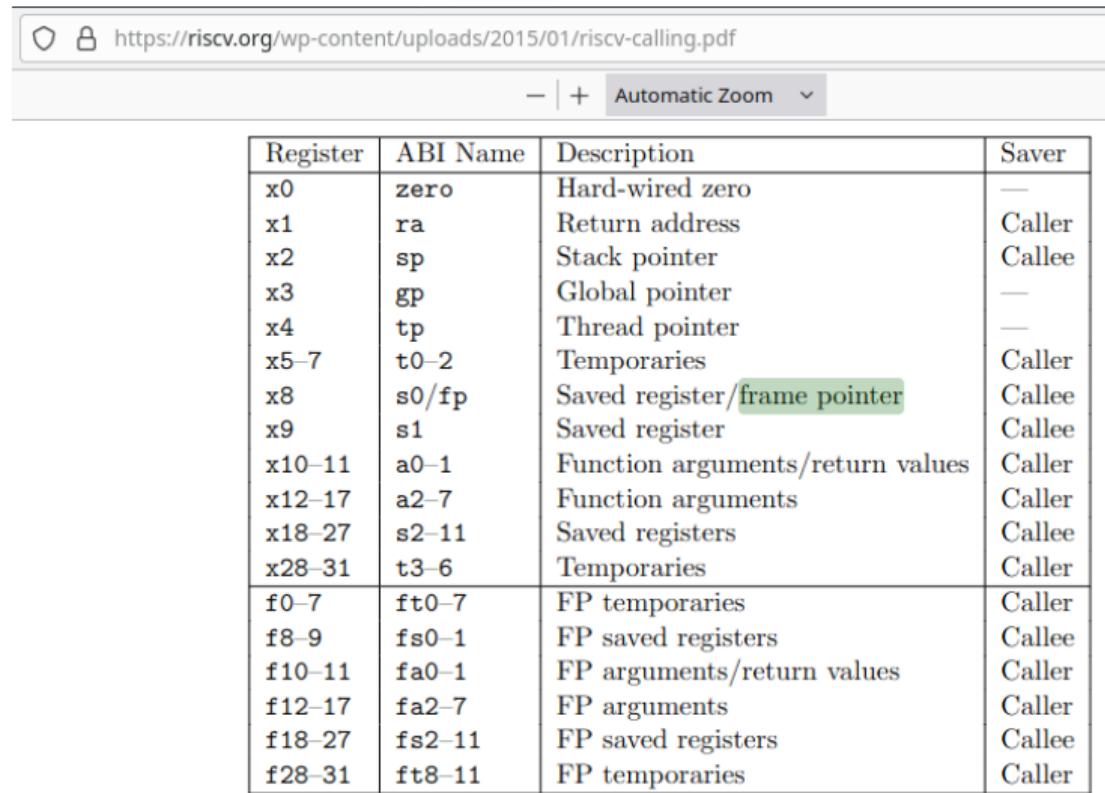
Bitno je da svi registri koje potprogram koristi tokom izvršenja na kraju budu vraćeni u prvobitno stanje

## Malo više o kopiranju

Prethodne vrednosti registara se takođe čuvaju na steku

Pozivna konvencija određuje da li je za čuvanje i ponovno postavljanje vrednosti nekog регистра zadužena pozivna ili pozvana funkcija

# Čuvanje i ponovno postavljanje registara



The screenshot shows a table from the RISC-V calling convention page on riscv.org. The table maps RISC-V registers to ABI names and describes their purpose, indicating whether the caller or callee saves them.

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5–7	t0–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

# Jedan konkretan primer

https://inst.eecs.berkeley.edu/~cs61c/resources/RISCV\_Calling\_Convention.pdf

- | + 110% sum\_squares .

```
prologue:
    addi sp sp -16
    sw s0 0(sp)
    sw s1 4(sp)
    sw s2 8(sp)
    sw ra 12(sp)

    li s0 1
    mv s1 a0
    mv s2 0

loop_start:
    bge s0 s1 loop_end
    mv a0 s0
    jal square
    add s2 s2 a0
    addi s0 s0 1
    j loop_start

loop_end:
    mv a0 s2

epilogue:
    lw s0 0(sp)
    lw s1 4(sp)
    lw s2 8(sp)
    lw ra 12(sp)
    addi sp sp 16
    jr ra
```

na početku koda svake funkcije se nalazi prolog u kom se vrši čuvanje registara za koje je pozvana funkcija zadužena (sw instrukcija znači store word)

na kraju koda svake funkcije se nalazi epilog u kom se vrši vraćanje vrednosti sa steka u registre (lw instrukcija znači load word)

Notice that we store values in the a registers because we need those values for

Zbog svega ovoga, poziv funkcije je skup

No, to ne znači da treba izbegavati podelu programa na funkcije

Prednosti takve podele su, kao što smo već videli, izuzetne

## Ključna reč **INLINE**

Dodavanje ključne reči **INLINE** na početku deklaracije i definicije funkcije (ispred povratnog tipa), učiniće da kompjuter pokuša da izbegne poziv funkcije prostim ubacivanjem njenog tela u telo pozivne funkcije, umesto naredbe poziva

# Ključna reč INLINE



<https://gcc.gnu.org/onlinedocs/gcc/Inline.html>



Next: [Const and Volatile Functions](#), Previous: [Determining the Alignment of Functions, Types or Variables](#), Up: [Extensions to the C Language Family](#) [Contents][Index]

## 6.45 An Inline Function is As Fast As a Macro

By declaring a function inline, you can direct GCC to make calls to that function faster. One way GCC can achieve this is to integrate that function's code into the code for its callers. This makes execution faster by eliminating the function-call overhead; in addition, if any of the actual argument values are constant, their known values may permit simplifications at compile time so that not all of the inline function's code needs to be included. The effect on code size is less predictable; object code may be larger or smaller with function inlining, depending on the particular case. You can also direct GCC to try to integrate all "simple enough" functions into their callers with the option `-finline-functions`.

## Ključna reč **INLINE**

Međutim, zamena poziva telom funkcije nije uvek isplativa, zbog složenosti današnjih memorijskih sistema

U principu, veće funkcije je bolje zadržati kao obične

Svakako je najbolje proveriti kako se vreme izvršenja programa menja nakon deklaracije funkcije kao inline, pa izabrati bolju opciju

## I nikada ne treba zaboraviti

Optimizacija performansi koda uvek ima neku cenu: povećava naše vreme razvoja, otežava održavanje...

Zato kod optimizujemo preko neke razumne mere samo kada se uverimo da je to neophodno

Kako možemo da vratimo pozivnoj funkciji više od jedne vrednosti?

## Prosleđivanje povratne vrednosti preko adrese

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

int add_sub(int a, int b, int* diff)
{
    *diff = a - b;

    return a + b;
}

int main()
{
    int a = 7;
    int b = 5;

    int amb = 0;
    int apb = add_sub(a, b, &amb);

    cout << a << " + " << b << " = " << apb << endl;
    cout << a << " - " << b << " = " << amb << endl;

    return 0;
}
```

## Prosleđivanje povratne vrednosti preko adrese

$$7 + 5 = 12$$

$$7 - 5 = 2$$

Šta možemo da uradimo ako je neophodno da menjamo vrednost parametra i van funkcije?

# I tu koristimo prosleđivanje po adresi

```
int no_ptr_inc(int a, int b)
{
    a += b;

    return a;
}

int with_ptr_inc(int* a, int b)
{
    *a += b;

    return *a;
}

int main()
{
    int a = 7;
    int b = 5;

    cout << a << " + " << b << " = " << no_ptr_inc(a, b) << endl;
    cout << "a = " << a << endl;

    cout << a << " + " << b << " = " << with_ptr_inc(&a, b) << endl;
    cout << "a = " << a << endl;

    return 0;
}
```

I tu koristimo prosleđivanje po adresi

$$7 + 5 = 12$$

$$a = 7$$

$$7 + 5 = 12$$

$$a = 12$$

Ne zaboravimo da je i ime niza zapravo adresa njegovog prvog elementa

```
int main()
{
    float xs[NUM PTS] = {7.7857203028720035, 7.642458246115663, 6.215313359871936, 7.059634724385643, 6.670815698546658,
6.876282852217749, 7.036072823998538, 7.290710191791188, 7.639716393843683, 7.752535310413255, 6.1089490156447095, 7.60434112
48489505, 7.418263501562334, 6.2496483487612196, 6.8065125974698395, 7.434551692906639, 6.51122811120167, 7.198011826278233,
7.874699069289809, 6.507268194942792, 12, 9, 12, 11, 10};

    float ys[NUM PTS] = {6.663959610602355, 6.0833932514505, 7.1901041284124805, 6.837614857113309, 7.24503886439218, 7.4
7176421269738, 7.157717200452314, 7.980448542746833, 6.826401869632361, 7.647518865141043, 7.437274473128388, 7.4728132876228
78, 7.081873708605056, 7.915294590939337, 6.43390232064712, 7.988414876843875, 7.3426188638785606, 7.43466429111027, 6.70361
9537079404, 6.804945018073546, 14, 9, 7, 11, 13};

    for(unsigned i = 0; i < NUM PTS; ++i)
        cout << ys[i] << ' ';
    cout << endl;

    cout << "centar = (" << median(xs, NUM PTS) << ", " << median(ys, NUM PTS) << ")\n";

    for(unsigned i = 0; i < NUM PTS; ++i)
        cout << ys[i] << ' ';
    cout << endl;

    return 0;
}
```

Ne zaboravimo da je i ime niza zapravo adresa njegovog prvog elementa

```
6.66396 6.08339 7.1901 6.83762 7.24504 7.47176 7.15772 7.98045 6.8264 7.64752 7.43727 7.47281 7.08187 7.91529 6.4339 7.98841  
7.34262 7.43466 6.70362 6.80494 14 9 7 11 13  
centar = (7.41826, 7.34262)  
6.08339 6.4339 6.66396 6.70362 6.80494 6.8264 6.83762 7 7.08187 7.15772 7.1901 7.24504 7.34262 7.43466 7.43727 7.47176 7.47281  
1 7.64752 7.91529 7.98045 7.98841 9 11 13 14
```

Prenos po adresi štedi vreme kada funkcija ima veće parametre, jer nema kopiranja

The image shows the Compiler Explorer interface with two panes. The left pane displays C++ source code, and the right pane shows the generated RISC-V assembly code.

**C++ Source Code:**

```
1 struct s
2 {
3     unsigned a;
4     unsigned b;
5 };
6
7 bool check_s(struct s param)
8 {
9     if(param.a < 10 && param.b < 10)
10        return true;
11    return false;
12 }
13
14 bool check_s_ptr(const struct s* param)
15 {
16     if(param -> a < 10 && param -> b < 10)
17        return true;
18
19    return false;
20 }
```

**RISC-V Assembly Output:**

```
8 auui    sp,sp,10
9 sltiu   a0,a0,10
10 jr      ra
11 .L3:
12 li      a0,0
13 addi   sp,sp,16
14 jr      ra
15 check_s_ptr(s const*):
16 lw      a4,0(a0)
17 li      a5,9
18 bgtu   a4,a5,.L8
19 lw      a0,4(a0)
20 sltiu   a0,a0,10
21 ret
22 .L8:
23 li      a0,0
24 ret
```

A red box highlights the assembly code for the `check_s_ptr` function. A callout box points to the first instruction (`lw a4,0(a0)`) and contains the following text:

Pokazivač se prosleđuje u registru a0, pa nema potrebe za kopiranjem parametra na stek

# Ipak, i dereferenciranje pokazivača košta

The screenshot shows the Compiler Explorer interface on godbolt.org. On the left, the C++ source code is displayed:

```
1 int no_ptr_inc(int a, int b)
2 {
3     a += b;
4
5     return a;
6 }
7
8 int with_ptr_inc(int* a, int b)
9 {
10    *a += b;
11
12    return *a;
13 }
```

On the right, the generated RISC-V assembly code is shown:

```
1 no_ptr_inc(int, int):
2     addw    a0,a0,a1
3     ret
4 with_ptr_inc(int*, int):
5     lw      a5,0(a0)
6     mv      a4,a0
7     addw   a0,a5,a1
8     sw      a0,0(a4)
9     ret
```

No, o svemu tome više u sredu

Hvala na pažnji