

Programiranje 2, letnji semestar 2023/4

Tipovi podataka, operatori, izrazi

Stefan Nikolić

Prirodno-matematički fakultet, Novi Sad

04.03.2024.



Na prošlom predavanju smo polazeći od jednog Ojlerovog problema došli do principa rada fon Nojmanovog računara

Da bismo naučili da ga programiramo u C++-u, potreban nam je novi polazni problem

Neka je n proizvoljan broj iz $\mathbb{N}$

Neka su a i b nezavisno i uniformno odabrani iz $[1, n]$

Neka je p_n verovatnoća da su a i b uzajamno prosti

Da li postoji granična vrednost $\lim_{n \rightarrow \infty} p_n$ i ako postoji, koja joj je vrednost?

Da li neko ima ideju?

Osnovni pristup

1. Izlistamo sve parove brojeva $\{a, b\} \in [1, n]^2$
2. Prebrojimo koliko ih je uzajamno prostih
3. Podelimo taj broj sa n^2 , da dobijemo p_n
4. Proverimo kako se p_n menja sa uvećanjem n i postavljamo hipotezu koju posle dokazujemo

Šta nam je sve potrebno za to?

Šta nam je sve potrebno za to?

- Moramo nekako da predstavimo brojeve
- Moramo da proverimo da li su uzajamno prosti
- Moramo da prebrojimo sve koji jesu
- Moramo da nađemo odnos tog broja i n^2

Šta nam je sve potrebno za to?

- Moramo nekako da predstavimo brojeve
- Moramo da proverimo da li su uzajamno prosti
 - Jedan način: proveravamo da li je $\text{NZD}(a, b) > 1$
 - Na primer, pitamo se da li $(\exists i > 1)(i \mid a \wedge i \mid b)$
- Moramo da prebrojimo sve koji jesu
- Moramo da nađemo odnos tog broja i n^2

Promenljive

Moramo nekako da predstavimo brojeve

Deklaracija promenljive:

tip identifikator;

Primer deklaracije promenljive

```
tip  identifikator ;  
int      a  ;
```

O tipovima ćemo uskoro detaljno govoriti

Identifikatori

Identifikator je niz karaktera gde je prvi karakter iz

$$S = [a, z] \cup [A, Z] \cup _$$

a svi ostali iz $S \cup [0, 9]$

Primeri ispravnih identifikatora

broj, nzd, i_1_2, uzajamno_prosti

Primeri neispravnih identifikatora

8_32_A, void

Identifikator se ne sme podudarati sa ključnim rečima jezika. Zbog toga void nije validan identifikator

https://en.cppreference.com/w/cpp/keyword

C++ C++ language Keywords

C++ keywords

This is a list of reserved keywords in C++. Since they are used by the language, these keywords are not available for re-definition or overloading. As an exception, they are not considered reserved in [attributes](#) (excluding attribute argument lists). [\(since C++11\)](#)

A - C	D - P	R - Z
alignas (C++11)	decltype (C++11)	reflexpr (reflection TS)
alignof (C++11)	default (1)	register (2)
and	delete (1)	reinterpret_cast
and_eq	do	requires (C++20)
asm	double	return
atomic_cancel (TM TS)	dynamic_cast	short
atomic_commit (TM TS)	else	signed
atomic_noexcept (TM TS)	enum (1)	sizeof (1)
auto (1) (2) (3) (4)	explicit	static
bitand	export (1) (3)	static_assert (C++11)
bitor	extern (1)	static_cast
bool	false	struct (1)
break	float	switch
case	for (1)	synchronized (TM TS)
catch	friend	template
char	goto	this (4)
char8_t (C++20)	if (2) (4)	thread_local (C++11)
char16_t (C++11)	inline (1)	throw
char32_t (C++11)	int	true
class (1)	long	try
compl	mutable (1)	typedef
concept (C++20)	namespace	typeid
const	new	typename
constexpr (C++20)	noexcept (C++11)	union
constexpr (C++11)	not	unsigned
constinit (C++20)	not_eq	using (1)
const_cast	nullptr (C++11)	virtual
continue	operator (4)	void
co_await (C++20)	or	volatile
co_return (C++20)	or_eq	wchar_t
co_yield (C++20)	private (3)	while
	protected	xor
	public	xor_eq

- (1) — meaning changed or new meaning added in C++11.
- (2) — meaning changed or new meaning added in C++17.
- (3) — meaning changed or new meaning added in C++20.
- (4) — new meaning added in C++23.

Većina editora će ih obojiti posebnom bojom

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

int main()
{
    int solid = 0;
    int void = 0;
    cout << "Unesite zapreminu praznine: ";
    cin >> void;

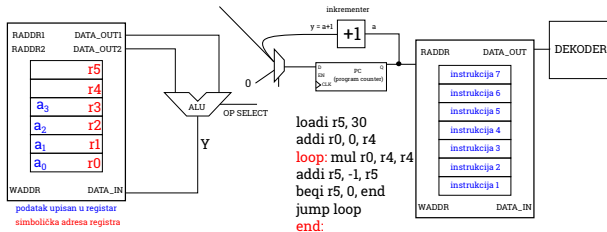
    cout << "Zapremina praznine je " << void << endl;

    return 0;
}
```

A kompajler će se uvek buniti ako napravimo grešku

```
keyword_conflict.cpp: In function 'int main()':
keyword_conflict.cpp:7:18: error: expected unqualified-id before '=' token
   7 |         int void = 0;
     |                   ^
keyword_conflict.cpp:9:16: error: expected primary-expression before 'void'
   9 |         cin >> void;
     |                ^~~~
keyword_conflict.cpp:11:45: error: expected primary-expression before 'void'
  11 |         cout << "Zapremina praznine je " << void << endl;
     |                                           ^~~~~
```

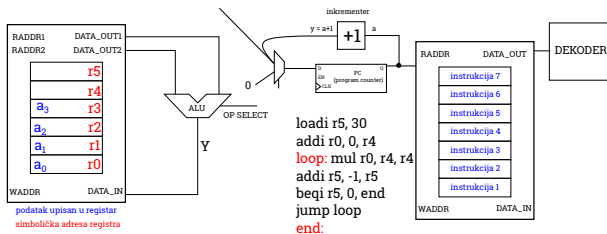
Šta je zapravo promenljiva?



Promenljiva je ništa drugo do ime za neku memorijsku lokaciju

Ona omogućuje da staranje o tome gde se koji podatak nalazi u memoriji prepustimo kompajleru

Šta je zapravo promenljiva?

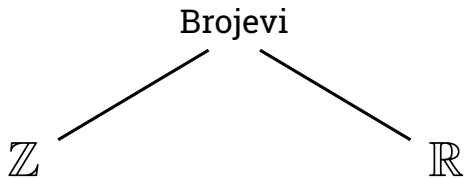


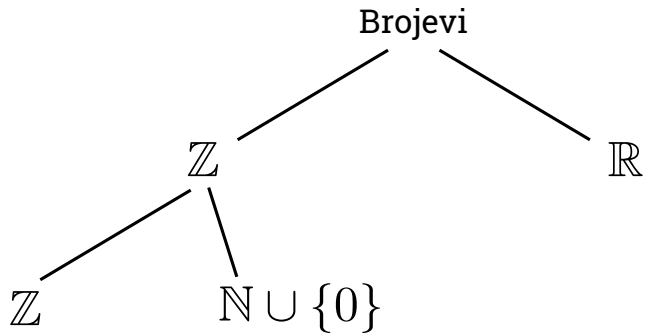
Na primer, ne moramo da znamo da se `a3` nalazi u registru `r3`, već svuda u kodu naprosto koristimo identifikator `a3`, a deklaracija promenljive će osigurati da kompajler rezerviše odgovarajuće memorijske lokacije (jednu ili više; o tome uskoro)

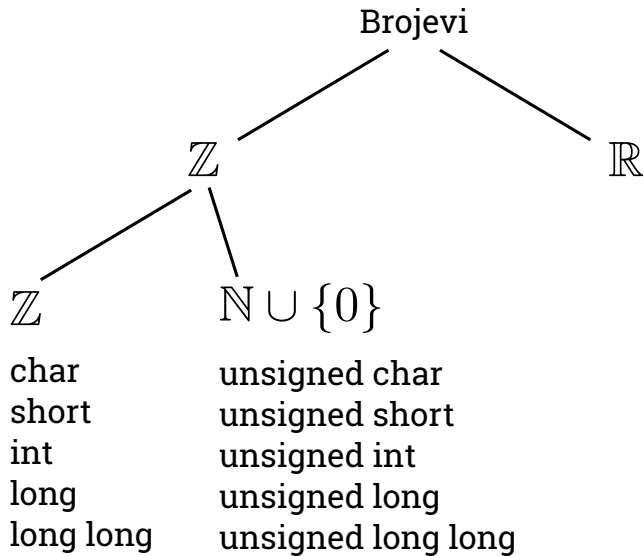
Detaljniji memorijski model ćemo razviti kada budemo stigli do potprograma (funkcija)

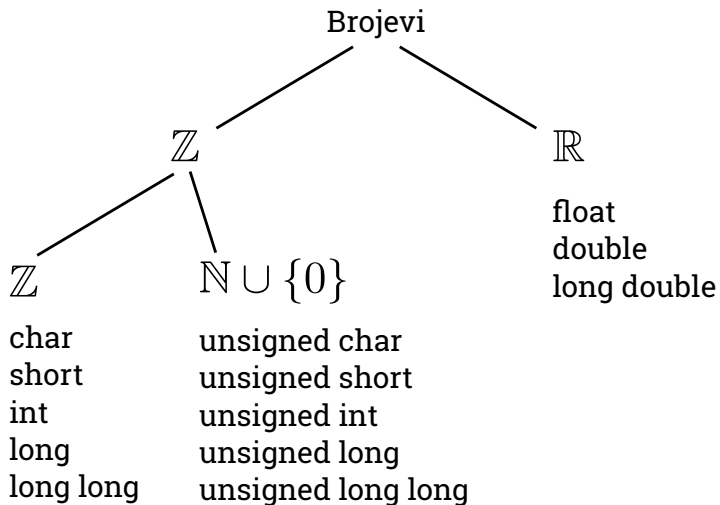
Tipovi podataka

Brojevi









Čemu ta podela?

$$\mathbb{N} \cup \{0\} \subset \mathbb{Z} \subset \mathbb{R}$$

Ako znamo da predstavimo brojeve iz $\mathbb{R}$, čemu svi ovi drugi tipovi?

Jedan primer

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

int main()
{
    int milion = 1000000;

    int bilion = milion * milion;

    cout << "1000000 ** 2 = " << bilion << endl;

    return 0;
}
```

Šta će biti ispisano na ekranu?

a) 10000000000000

b) 1e12

c) nešto treće

Jedan primer

```
1000000 ** 2 = -727379968
```

Čemu ta podela?

$$\mathbb{N} \cup \{0\} \subset \mathbb{Z} \subset \mathbb{R}$$

Ako znamo da predstavimo brojeve iz $\mathbb{R}$, čemu svi ovi drugi tipovi?

Lakše pitanje: čemu svi ti tipovi za neoznačene brojeve?

unsigned char

unsigned short

unsigned int ¹

unsigned long

unsigned long long

¹Dovoljno je da napišemo i samo "unsigned"

Mali podsetnik na pozicione (težinske) brojne sisteme

Primer:

$$1748 = 8 \times 10^0 + 4 \times 10^1 + 7 \times 10^2 + 1 \times 10^3$$

Mali podsetnik na pozicione (težinske) brojne sisteme

Primer:

$$1748 = 8 \times 10^0 + 4 \times 10^1 + 7 \times 10^2 + 1 \times 10^3$$

Obeležimo redom cifre sa desna (najmanje vredna) na levo (najvrednija):

d_3	d_2	d_1	d_0
1	7	4	8

Mali podsetnik na pozicione (težinske) brojne sisteme

Primer:

$$1748 = 8 \times 10^0 + 4 \times 10^1 + 7 \times 10^2 + 1 \times 10^3$$

Obeležimo redom cifre sa desna (najmanje vredna) na levo (najvrednija):

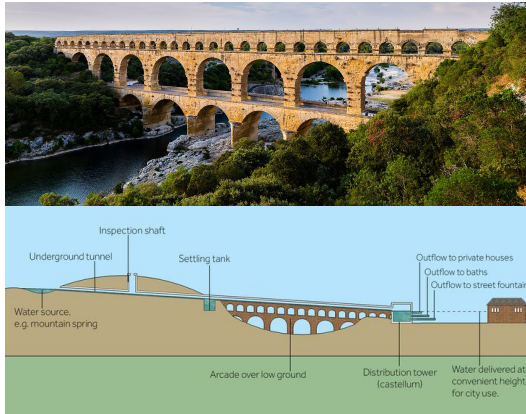
d_3	d_2	d_1	d_0
1	7	4	8

$$m = \sum_{i=0}^3 a_i \times 10^i, \quad (\forall i \in [0, 3])(a_i \in [0, 10))$$

Pozicioni brojni sistemi nisu neophodni, ali znatno olakšavaju račun

1748 = MDCCXLVIII

= 1000 + 500 + 100 + 100 + (50 - 10) + 5 + 1 + 1 + 1



B —osnova

$$m = \sum_{i=0}^{n-1} a_i \times B^i, \quad (\forall i \in [0, n))(a_i \in [0, B))$$

m je n -tocifreni broj u pozicionom sistemu sa osnovom B

Brojevi u pozicionom brojnom sistemu sa $B = 2 \implies (\forall i)(a_i \in \{0, 1\})$

Primer

$2^{10} 2^9 2^8 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0$

$$11011010100_2 = 2^2 + 2^4 + 2^6 + 2^7 + 2^9 + 2^{10} \\ = 4 + 16 + 64 + 128 + 512 + 1024 = 1748_{10}$$

$$\begin{array}{l} 1748_{10} : 2 = 874 + 0 \quad 2^0 \\ \phantom{1748_{10}} : 2 = 437 + 0 \quad 2^1 \\ \phantom{1748_{10}} : 2 = 218 + 1 \quad 2^2 \\ \phantom{1748_{10}} : 2 = 109 + 0 \quad 2^3 \\ \phantom{1748_{10}} : 2 = 54 + 1 \quad 2^4 \\ \phantom{1748_{10}} : 2 = 27 + 0 \quad 2^5 \\ \phantom{1748_{10}} : 2 = 13 + 1 \quad 2^6 \\ \phantom{1748_{10}} : 2 = 6 + 1 \quad 2^7 \\ \phantom{1748_{10}} : 2 = 3 + 0 \quad 2^8 \\ \phantom{1748_{10}} : 2 = 1 + 1 \quad 2^9 \\ \phantom{1748_{10}} : 2 = 0 + 1 \quad 2^{10} \end{array}$$

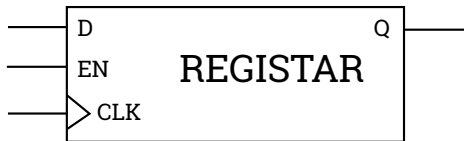
Malo terminologije

jedna binarna cifra = 1 bit

osam binarnih cifara = 1 bajt (eng. *byte*)

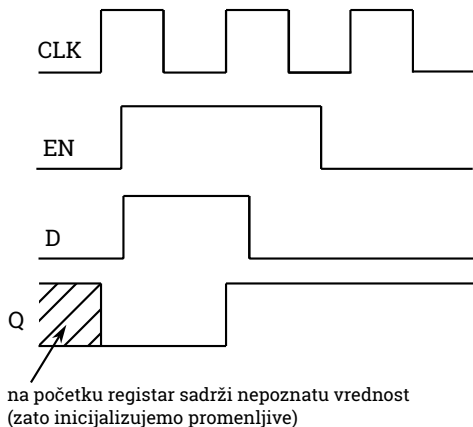
manje bitno: četiri binarne cifre = 1 nibl (eng. *nibble*)

Uprošćenje sa prethodnog predavanja



Prepostavili smo da registar može da smesti proizvoljan broj bita

No, u prirodi je sve ograničeno, pa i veličina registara



Uprošćenje sa prethodnog predavanja

U današnjim računarima, registri najčešće sadrže 64 bita, no još uvek ima i procesora sa 32-bitnim, 16-bitnim, pa i 8-bitnim registrima

Pored registara, postoji čitava hijerarhija memorije



Procesor:	Intel Core i5 Deca Core Processor 1235U Brzina: 3.30 GHz (Turbo do 4.40 GHz) Keš memorija: 12MB
Ekran:	14", Full HD, IPS, 250 niti, Anti-glare, 45% NTSC Rezolucija: 1920 x 1080
Grafička kartica:	Integrirana Intel Iris Xe Graphics sa deljenom sistemskom memorijom
Tip HDD:	NVMe SSD
Memorija:	8GB (1 x 8GB) DDR4 3200MHz RAM, 2 SODIMM slot Maksimalno podržano do 64GB
Čitač kartica:	microSD čitač kartica

Tu je najčešće moguće adresirati svaki bajt posebno
(jedan bajt = jedna adresibilna lokacija)

Zato su veličine promenljivih izražene u bajtovima

Pravila:

$\text{veličina(char)} \leq \text{veličina(short)} \leq \text{veličina(int)} \leq \text{veličina(long)} \leq \text{veličina(long long)}$

minimalna veličina(char) = 1 bajt

minimalna veličina(short) = 2 bajta

minimalna veličina(int) = 2 bajta

minimalna veličina(long) = 4 bajta

minimalna veličina(long long) = 8 bajta

Ponovo ne moramo pamtiti

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

int main()
{
    cout << "sizeof(char) = " << sizeof(char)
        << " bytes <= sizeof(short) = " << sizeof(short)
        << " bytes <= sizeof(int) = " << sizeof(int)
        << " bytes <= sizeof(long) = " << sizeof(long)
        << " bytes <= sizeof(long long) = " << sizeof(long long) << " bytes\n";

    return 0;
}
```

Na ovom računaru (na drugim može biti drugačije)

```
sizeof(char) = 1 bytes <= sizeof(short) = 2 bytes <= sizeof(int) = 4 bytes <= sizeof(long) = 8 bytes <= sizeof(long long) = 8 bytes
```

Pretek (eng. *overflow*)

Vratimo se za trenutak na decimalne brojeve

Zamislamo da registri mogu da smeste trocifrene decimalne brojeve

Šta će biti rezultat operacije $123 + 891$?

Pretek (eng. *overflow*)

$$123 + 891 = 1014$$

Pretek (eng. *overflow*)

$123 + 891 = 1$ za ovu cifru nema mesta u registru |014

Pretek (eng. *overflow*)

$$123 + 891 = 14$$

Pretek (eng. *overflow*)

Isto je i sa binarnim brojevima:

Pretpostavimo da su registri osmobitni

Koji će biti rezultat operacije $01101011 + 11101101$?

Pretek (eng. *overflow*)

$$01101011 + 11101110 = 1|01011001 = 01011001$$

$$107 + 238 = 89 = 345 - 256 \text{ (} 2^8 \text{—ukupan broj različitih vrednosti 8-bitnog registra)}$$

Pretek može biti vrlo opasan

← → ↻

🔒 https://en.wikipedia.org/wiki/Year_2000_problem

📄 ☆

📧 ⬇️ 👤

☰



WIKIPEDIA
The Free Encyclopedia

🔍

Search Wikipedia

Search

Create account

Log in

Contents

hide

(Top)

Background

> Programming problem

Programming solutions

> Documented errors

> Government responses

Private sector response

Fringe group responses

Cost

Remedial work organization

> Results

See also

Notes

Year 2000 problem

44 languages

Article

Talk

Read

Edit

View history

Tools

From Wikipedia, the free encyclopedia

"January 1, 2000" redirects here. For the date, see [January 2000](#). For the event, see [Millennium celebrations](#).

The **year 2000 problem**, also commonly known as the **Y2K problem**, **Y2K scare**, **millennium bug**, **Y2K bug**, **Y2K glitch**, **Y2K error**, or simply **Y2K**, refers to potential computer errors related to the [formatting and storage of calendar data](#) for dates in and after the year 2000. Many [programs](#) represented four-digit years with only the final two digits, making the year 2000 indistinguishable from 1900. Computer systems' inability to distinguish dates correctly had the potential to bring down worldwide infrastructures for computer reliant industries.

In the years leading up to the turn of the millennium, the public gradually became aware of the "Y2K scare", and individual companies predicted the global damage caused by the bug would require anything between \$400 million and \$600 billion to rectify.^[1] A lack of clarity regarding the potential dangers of the bug led some to stock up on food, water, and firearms, purchase backup generators, and withdraw large sums of money in anticipation of a computer-induced [apocalypse](#).^[2]

Contrary to published expectations, few major errors occurred in 2000. Supporters of the Y2K remediation effort argued that this was primarily due to the pre-emptive action of many computer programmers and [information](#)



An electronic sign at [École centrale de Nantes](#) incorrectly displaying the year 1900 on 3 January 2000 (On the screen it translates to "Welcome to the Central School of Nantes 12:09 p.m. January 3, 1900.")

Pretek može biti vrlo opasan



Search

[Create account](#) [Log in](#)

Ariane flight V88

5 languages

Contents [hide](#)

[\(Top\)](#)

Launch failure

Payload

Aftermath

See also

References

Further reading

External links

[Article](#) [Talk](#)

From Wikipedia, the free encyclopedia

"Cluster (spacecraft)" redirects here. This article is about the failed Cluster launch. For the successful space mission, see [Cluster II \(spacecraft\)](#).

Ariane flight V88^[1] was the failed maiden flight of the **Arianespace Ariane 5** rocket, vehicle no. 501, on 4 June 1996. It carried the **Cluster** spacecraft, a constellation of four **European Space Agency** research satellites.

The launch ended in failure due to multiple errors in the software design: **dead code**, intended only for Ariane 4, with inadequate protection against **integer overflow** led to an **exception handled** inappropriately, halting the whole otherwise unaffected **inertial navigation system**. This caused the rocket to veer off its flight path 37 seconds after launch, beginning to disintegrate under high aerodynamic forces, and finally self-destructing via its automated **flight termination system**. The failure has become known as one of the most infamous and expensive **software bugs** in history.^[2] The failure resulted in a loss of more than US\$370 million.^[3]

Launch failure [\[edit \]](#)

The Ariane 5 reused the code from the inertial reference platform from the Ariane 4, but the early part of the Ariane 5's flight path differed from the Ariane 4 in having higher horizontal velocity values. This caused an internal value BH (Horizontal Bias) calculated in the alignment function to be unexpectedly high. The alignment function was operative for approximately 40 seconds of flight, which was based on a requirement of Ariane 4, but served no purpose after lift-off on the Ariane 5 [4]. The greater values

Cluster

Mission type	Magnetospheric
Operator	ESA
Spacecraft properties	
Launch mass	1,200 kilograms (2,600 lb)
Start of mission	
Launch date	12:34:06, 4 June 1996 (UTC)
Rocket	Ariane 5G
Launch site	Kourou ELA-3
End of mission	
Disposal	launch failure
Destroyed	4 June 1996



Potrebno je predvideti opseg promenljivih i u skladu sa tim odabrati tip

Nekad nije moguće naprosto odabrati long long, jer je potrebno uskladištiti velik broj podataka, ili postići brže izvršenje programa

Kako da saznamo opseg svakog od tipova?

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

int main()
{
    cout << "Na ovom racunaru:\n";

    cout << "unsigned char je u opsegu [0, " << UCHAR_MAX << "]\n";
    cout << " = [0, 2 ** " << 8 * sizeof(unsigned char) << ")]\n";
    cout << "unsigned short je u opsegu [0, " << USHRT_MAX << "]\n";
    cout << " = [0, 2 ** " << 8 * sizeof(unsigned short) << ")]\n";
    cout << "unsigned je u opsegu [0, " << UINT_MAX << "]\n";
    cout << " = [0, 2 ** " << 8 * sizeof(unsigned) << ")]\n";
    cout << "unsigned long je u opsegu [0, " << ULONG_MAX << "]\n";
    cout << " = [0, 2 ** " << 8 * sizeof(unsigned long) << ")]\n";
    cout << "unsigned long long je u opsegu [0, " << ULLONG_MAX << "]\n";
    cout << " = [0, 2 ** " << 8 * sizeof(unsigned long long) << ")]\n";

    return 0;
}
```

Kako da saznamo opseg svakog od tipova?

Na ovom racunaru:

unsigned char je u opsegu $[0, 255] = [0, 2^{**8})$

unsigned short je u opsegu $[0, 65535] = [0, 2^{**16})$

unsigned je u opsegu $[0, 4294967295] = [0, 2^{**32})$

unsigned long je u opsegu $[0, 18446744073709551615] = [0, 2^{**64})$

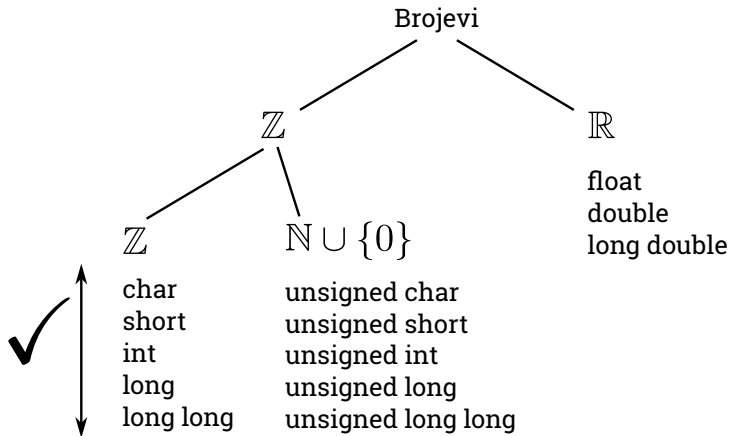
unsigned long long je u opsegu $[0, 18446744073709551615] = [0, 2^{**64})$

Ako je i long long mali za vaše brojeve

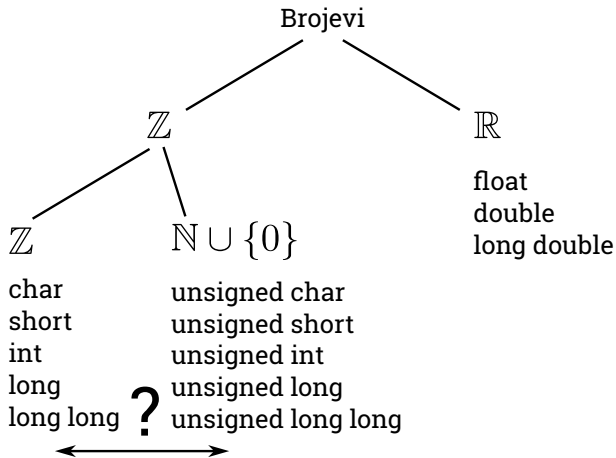


Možete koristiti neku biblioteku za aritmetiku proizvoljne preciznosti. Tada će vaše promenljive zauzimati niz memorijskih reči, u skladu sa potrebom, ali će sve operacije biti sporije, jer se neće izvršavati direktno u hardveru (npr. jedna instrukcija za množenje), već će predstavljati niz instrukcija

Sad znamo čemu svi ti tipovi neoznačenih celih brojeva



No, čemu uopšte podela na označene i neoznačene?



Aditivni komplementi

$$a + -a = 0 \pmod{B} = B \pmod{B}$$

$$-a = B - a$$

Na primer jednocifreni decimalni brojevi

$$-0 = 0$$

$$-1 = 9$$

$$-2 = 8$$

$$-3 = 7$$

$$-4 = 6$$

$$-5 = 5$$

$$-6 = 4$$

$$-7 = 3 \quad // 7 + -7 = 7 + 3 = 0 \pmod{10}$$

$$-8 = 2$$

$$-9 = 1$$

Mapiranje na fiksni opseg

Razmotrimo još malo jednocifrene decimalne brojeve

Na raspolaganju imamo cifre 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Potrebno je da pomoću tih cifara predstavimo neki skup $[0, n]$, tako da $[0, 9]$ za svaki $a \in [0, n]$ sadrži i $-a$

Mapiranje na fiksni opseg

Krećemo redom:

1. Ako ubacimo 0, $-0 = 0$, pa već imamo i komplement
2. Ako ubacimo 1, $-1 = 9$, pa moramo ubaciti i 9
3. Ako ubacimo 2, $-2 = 8$, pa moramo ubaciti i 8
4. Ako ubacimo 3, $-3 = 7$, pa moramo ubaciti i 7
5. Ako ubacimo 4, $-4 = 6$, pa moramo ubaciti i 6
6. Ako ubacimo 5, $-5 = 5$ i to više nije jednoznačno
7. Moramo da se odlučimo da li će 5 biti pozitivno ili negativno
8. Ako uzmemo da je negativno, važiće pravilo da su svi brojevi preko polovine opsega negativni a svi ispod polovine opsega nenegativni
9. Dalje ne možemo, jer smo 6 već iskoristili za -4, 7 za -3...

Uopštenje na višecifrene brojeve

Neka a ima n cifara

$$a + -a = 0 \pmod{B^n} = B^n \pmod{B^n}$$

$$-a = B^n - a$$

U $\mathbb{Z}_2$ se taj DRUGI aditivni komplement jako lako računa

$$-a = 2^n - a = (2^n - 1) - a + 1$$

$(2^n - 1) - a$ nazivamo **PRVI KOMPLEMENT** ili **KOMPLEMENT JEDINICE** (eng. **ONE'S COMPLEMENT**)

$2^n - 1$ je niz od n jedinica, pa $(2^n - 1) - a$ računamo tako što svaki bit u a koji je 1 pretvorimo u 0 i obrnuto

$2^n - a$ nazivamo **DRUGI KOMPLEMENT** ili **KOMPLEMENT DVOJKE** (eng. **TWO'S COMPLEMENT**)

Primer (osmocifreni binarni brojevi)

$$\begin{array}{ccccccc} -00000111 & = & 100000000 & - & 00000111 & = & 11111111 - 00000111 + 00000001 = 11111001 \\ -7 & & = 256 & & - 7 & & = 255 & & - 7 & & + 1 & & = 249 \end{array}$$

Da vidimo kako izgledaju binarni zapisi brojeva u C++-u

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

int main()
{
    char a = 7;
    cout << "a u binarnom zapisu: " << bitset<8 * sizeof(char)>(a) << endl;
    cout << "a u decimalnom zapisu: " << a << endl;
    cout << "-a u binarnom zapisu: " << bitset<8 * sizeof(char)>(-a) << endl;
    cout << "-a u decimalnom zapisu: " << -a << endl;
    cout << "binarni zapis -a interpretiran kao neoznacen broj: "
        << bitset<8 * sizeof(unsigned char)>((unsigned char)(-a)) << endl;
    cout << "-a interpretiran kao neoznacen broj: " << +(unsigned char)(-a) << endl;

    return 0;
}
```


Da vidimo kako izgledaju binarni zapisi brojeva u C++-u

```
a u binarnom zapisu: 00000111
a u decimalnom zapisu:
-a u binarnom zapisu: 11111001
-a u decimalnom zapisu: -7
binarni zapis -a interpretiran kao neoznačen broj: 11111001
-a interpretiran kao neoznačen broj: 249
```

Zapazimo da označeni -7 i neoznačeni +249 imaju isti binarni zapis nad osam bita: 11111001

Dakle, sve je stvar interpretacije. Označeni brojevi sa vodećom jedinicom su uvek u gornjoj polovini opsega, što znači da su negativni. Provera znaka je dakle trivijalna, što, na primer, znatno olakšava implementaciju skokova

Sumarno o označenim brojevima

- prva polovina opsega je rezervisana za, redom, brojeve iz $[0, 2^{n-1})$
- ti brojevi imaju vodeću nulu u binarnom zapisu
- druga polovina opsega je rezervisana za, redom, brojeve iz $[-2^{n-1}, -1]$
- ti brojevi imaju vodeću jedinicu u binarnom zapisu
- sve u svemu je moguće predstaviti brojeve iz $[-2^{n-1}, 2^{n-1})$

Šta se onda ovde dogodilo?

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

int main()
{
    int milion = 1000000;

    int bilion = milion * milion;

    cout << "1000000 ** 2 = " << bilion << endl;

    return 0;
}
```

Uzmimo manji primer za ilustraciju

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

int main()
{
    unsigned char a = 72;
    unsigned char b = 2 * a;

    cout << +a << " * 2 = " << +b << " = " << bitset<8 * sizeof(char)>(b) << endl;

    char c = 72;
    char d = 2 * c;

    cout << +c << " * 2 = " << +d << " = " << bitset<8 * sizeof(char)>(d) << endl;

    return 0;
}
```

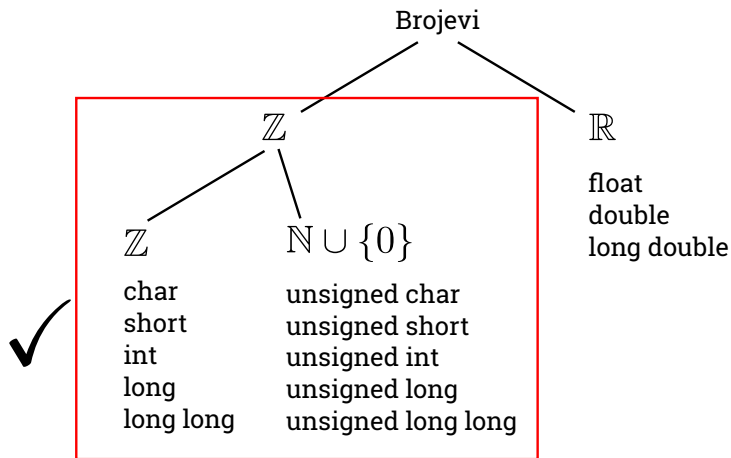
Uzmimo manji primer za ilustraciju

$$72 * 2 = 144 = 10010000$$

$$72 * 2 = -112 = 10010000$$

Ako znamo da će naši brojevi biti iz $\mathbb{N} \cup \{0\}$, unsigned verzije celobrojnih tipova će nam obezbediti duplo veći opseg vrednosti

Time smo završili celobrojne tipove



Operatori

Vratimo se na početni problem

Neka je n proizvoljan broj iz $\mathbb{N}$

Neka su a i b nezavisno i uniformno odabrani iz $[1, n]$

Neka je p_n verovatnoća da su a i b uzajamno prosti

Da li postoji granična vrednost $\lim_{n \rightarrow \infty} p_n$ i ako postoji, koja joj je vrednost?

Osnovni pristup

1. Izlistamo sve parove brojeva $\{a, b\} \in [1, n]^2$
2. Prebrojimo koliko ih je uzajamno prostih
3. Podelimo taj broj sa n^2 , da dobijemo p_n
4. Proverimo kako se p_n menja sa uvećanjem n i postavljamo hipotezu koju posle dokazujemo

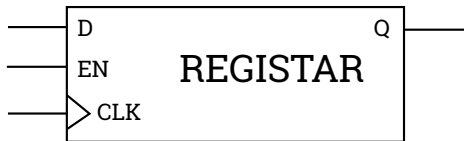
Šta nam je sve potrebno za to?

- Moramo nekako da predstavimo brojeve ✓
- Moramo da proverimo da li su uzajamno prosti
 - Jedan način: proveravamo da li je $\text{NZD}(a, b) > 1$
 - Na primer, pitamo se da li $(\exists i > 1)(i \mid a \wedge i \mid b)$
- Moramo da prebrojimo sve koji jesu
- Moramo da nađemo odnos tog broja i n^2

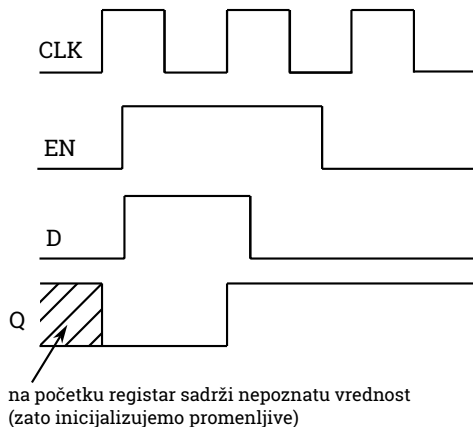
Do sad smo videli samo deklaraciju promenljive

```
unsigned a;
```

Do sad smo videli samo deklaracijo promenljive



No, kako da dodelimo vrednost promenljivoj?



Operator dodele: =

Kao i u Python-u, operator = ne znači proveru jednakosti levog i desnog operanda, već dodelu vrednosti (eng. **ASSIGNMENT**) desnog operanda levom operandu

Primer: `unsigned a = 0; //Deklaracija promenljive a sa inicijalizacijom na 0`

Operator dodele: =

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

int main()
{
    unsigned char a = 7;
    cout << "a = " << +a << endl;

    int b = -7;
    cout << "b = " << b << endl;

    a = b;
    cout << "a = " << +a << endl;

    return 0;
}
```

Ukoliko je tip izraza sa desne strane operatora = (eng. **RIGHT-HAND SIDE** (RHS)) drugačiji od tipa promenljive sa leve strane operatora (eng. **LEFT-HAND SIDE** (LHS)), kompajler će izvršiti implicitnu konverziju u tip sa leve strane

Operator dodele: =

```
a = 7  
b = -7  
a = 249
```

Ukoliko je tip izraza sa desne strane operatora = (eng. **RIGHT-HAND SIDE** (RHS)) drugačiji od tipa promenljive sa leve strane operatora (eng. **LEFT-HAND SIDE** (LHS)), kompajler će izvršiti implicitnu konverziju u tip sa leve strane

Operator dodele: =

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

int main()
{
    unsigned char a = 0;
    int* p = NULL; //int* je pokazivac na int. Pokazivace cemo raditi kasnije.

    a = p;

    cout << +a << endl;

    return 0;
}
```

Među nekim tipovima, implicitna konverzija nije moguća i kompajler će prilikom pokušaja dodele prijaviti grešku

Operator dodele: =

```
implicit_cast.cpp: In function 'int main()':  
implicit_cast.cpp:9:13: error: invalid conversion from 'int*' to 'unsigned char' [-fpermissive]  
    9 |         a = p;  
      |         ^  
      |         |  
      |         int*  
  
shell returned 1
```

Među nekim tipovima, implicitna konverzija nije moguća i kompajler će prilikom pokušaja dodele prijaviti grešku

Izraz dodele

Izraz je niz operatora i operandada. Svaki izraz u C++-u ima vrednost i tip te vrednosti (to je ono što je malopre bilo implicitno konvertovano)

Kako je i operator dodele = operator, dodela je izraz

Tip tog izraza je tip levog operanda, a vrednost tog izraza je vrednost upravo dodeljena levom operandu (odnosno, vrednost desnog operanda)

Primer

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

int main()
{
    unsigned a = 7;
    unsigned b = 13;
    cout << "a: " << a << ", a = b: " << (a = b) << ", a: " << a << endl;
    //NAPOMENA: Bocni efekat operatora dodele, to jest dodela vrednosti
    //promenljivoj a mora biti izvršen pre kraja naredbe završene terminatorom ;
    //U medjuvremenu moze i ne mora biti izvršen.
    //(u zavisnosti od kompajlera, poslednji ispis a moze biti bilo 13 ili 7)

    return 0;
}
```

```
a: 7, a = b: 13, a: 13
```

Nizovi dodela

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

int main()
{
    unsigned a, b, c, d;
    a = b = c = d = 5;

    //Ovo je ekvivalentno a = (b = (c = (d = 5)));
    //Evalvacija izraza se vrši s desna na levo, da bi vrednosti
    //dodele levim operandima bile pripremljene kada su potrebne

    cout << a << ", " << b << ", " << c << ", " << d << endl;

    return 0;
}
```

Pošto je i sama dodela izraz, ona može postati desni operand novoj dodeli unutar iste naredbe

Nizovi dodela

5, 5, 5, 5

Pošto je i sama dodela izraz, ona može postati desni operand novoj dodeli unutar iste naredbe

Aritmetički operatori

Unarni

- + plus: $y = +a$;
- negacija : $y = -a$;

Binarni

- + sabiranje: $y = a + b$;
- oduzimanje: $y = a - b$;
- * množenje: $y = a * b$;
- / deljenje: $y = a / b$;
- % ostatak pri deljenju: $y = a \% d$;

Aritmetički operatori

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

int main()
{
    short a = -11;
    unsigned b = 8e5;

    unsigned c = a + b;
    //a je implicitno konvertovan u unsigned pa izraz a + b ima odgovarajući opseg
    cout << a << " + " << b << " = " << c << endl;
    //To se događa čak i kada ne dodeljujemo vrednost izraza promenljivoj c, tipa unsigned
    cout << a << " + " << b << " = " << a + b << endl;

    c = b + a;
    //Slično je i kada a postane desni operand sabiranja
    cout << a << " + " << b << " = " << c << endl;
    cout << a << " + " << b << " = " << a + b << endl;

    short d = a + b;
    //Kada pokušamo da vrednost izraza tipa unsigned dodelimo promenljivoj tipa short, može
    //se dogoditi da ona nema odgovarajući opseg
    cout << a << " + " << b << " = " << d << endl;

    return 0;
}
```

Po potrebi ponovo dolazi do implicitnih konverzija tipova operanada, obično tako da se slabiji (manje precizan) tip pretvara u jači (precizniji), tako da i sam izraz dobija taj tip.

Aritmetički operatori

```
-11 + 800000 = 799989  
-11 + 800000 = 799989  
-11 + 800000 = 799989  
-11 + 800000 = 799989  
-11 + 800000 = 13557
```

Po potrebi ponovo dolazi do implicitnih konverzija tipova operanada, obično tako da se slabiji (manje precizan) tip pretvara u jači (precizniji), tako da i sam izraz dobija taj tip.

EksPLICITNA konverzija tipova

```
#include <iostream>

using namespace std;

int main()
{
    unsigned a = 4e9;
    unsigned b = 2e9;

    //Ako su oba operanda istog tipa, neće doći do konverzije, bez obzira
    //na to da li rezultat operacije prevazilazi opseg tipa izraza
    cout << a << " + " << b << " = " << a + b << endl;

    unsigned long x = 4e9;
    unsigned y = 2e9;

    //Dovoljno je da jedan operand ima dovoljan opseg za rezultat i
    //implicitna konverzija će osigurati da izraz dobije odgovarajući tip
    cout << x << " + " << y << " = " << x + y << endl;
    cout << x << " + " << y << " = " << y + x << endl;

    //Moguće je koristiti i eksplisitnu konverziju: (željeni tip)(vrednost za konvertovanje)
    //Nisu neophodne sve te zagrade, ali čine konverziju citljivijom
    cout << a << " + " << b << " = " << (unsigned long)(x) + y << endl;
    cout << a << " + " << b << " = " << x + (unsigned long)(y) << endl;

    return 0;
}
```

EksPLICITNA konverzija tipova

```
40000000000 + 20000000000 = 1705032704  
40000000000 + 20000000000 = 60000000000  
40000000000 + 20000000000 = 60000000000  
40000000000 + 20000000000 = 60000000000  
40000000000 + 20000000000 = 60000000000
```

 https://en.cppreference.com/w/cpp/language/implicit_conversion#Integral_promotion



80%



Integral promotion

prvalues of small integral types (such as **char**) and unscoped enumeration types may be converted to prvalues of larger integral types (such as **int**). In particular, **arithmetic operators** do not accept types smaller than **int** as arguments, and integral promotions are automatically applied after lvalue-to-rvalue conversion, if applicable. This conversion always preserves the value.

The following implicit conversions in this section are classified as *integral promotions*.

Pri aritmetičkim operacijama, celobrojni tipovi manji od int (char i short) su automatski konvertovani u int, čak i kada su oba operanda istog tipa.² To je zbog širine operanada na aritmetičko-logičkim jedinicama savremenih procesora

²Isto važi i za unsigned verzije)

Rang operatora

Rang aritmetičkih operatora je kao i u matematici, ali se zbog čitljivosti preporučuje upotreba zagrada, čak i kada nisu neophodne

Komutativnost i asocijativnost su takođe kao u matematici

Rang operatora

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

int main()
{
    int a = 16;
    int b = 4;
    int c = 7;
    int d = 2;

    cout << a % d + c * d << endl;      //0 + 7 * 2
    cout << a % d + d * c << endl;      //0 + 7 * 2
    cout << d * c + a % d << endl;      //7 * 2 + 0
    cout << (a % b) + (c * d) << endl;  //0 + 7 * 2
    cout << a % (b + c) * d << endl;    //(16 % 11) * 2 = 5 * 2
    cout << (a % (b + c)) * d << endl;  //(16 % 11) * 2 = 5 * 2

    return 0;
}
```

Rezultat

14
14
14
14
10
10

Kombinovane dodele

Često imamo potrebu da napišemo naredbu oblika $a = a + 4$;
(levi operand operatora dodele je operand aritmetičke operacije sa desne strane)

C++ omogućuje skraćenje takve naredbe kombinovanjem dodele i aritmetičke operacije:

$a += 4$

Slično postoje i $-=$, $*=$, $/=$, $\%=$

Jedan malo složeniji primer

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

int main()
{
    unsigned short a;

    cout << "Unesite pozitivan broj manji od " << USHRT_MAX << ": ";
    cin >> a;

    unsigned short sum = 0;
    unsigned short digit_weight = 1;

    cout << "Binarni zapis broja " << a << " je " << bitset<8 * sizeof(unsigned short)>(a) << endl;
    cout << "Obrnut binarni zapis broja " << a << " je ";
    for(unsigned i = 0; i < 8 * sizeof(unsigned short); ++i)
    {
        unsigned char digit = a % 2;
        cout << +digit;
        a /= 2;
        sum += digit * digit_weight;

        digit_weight *= 2;
    }

    cout << "\n\n A vrednost tog broja izracunata konverzijom binarnog zapisa je: " << sum << endl;

    return 0;
}
```

Jedan malo složeniji primer

```
Unesite pozitivan broj manji od 65535: 47  
Binarni zapis broja 47 je 00000000000101111  
Obrnut binarni zapis broja 47 je 1111010000000000
```

```
A vrednost tog broja izracunata konverzijom binarnog zapisa je: 47
```

Inkrement i dekrement

Često imamo i potrebu da uvećamo ili umanjimo promenljivu za 1

C++ za to obezbeđuje posebne operatore inkrementa i dekrementa:

`i++`, `++i`, `i--`, `--i`

Post- i pre-inkrement (dekrement)

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

int main()
{
    unsigned i = 6;

    unsigned j = i++;
    cout << "i = " << i << ", j = " << j << endl;

    j = ++i;
    cout << "i = " << i << ", j = " << j << endl;

    j = i--;
    cout << "i = " << i << ", j = " << j << endl;

    j = --i;
    cout << "i = " << i << ", j = " << j << endl;

    return 0;
}
```

Rezultat

```
i = 7, j = 6
i = 8, j = 8
i = 7, j = 8
i = 6, j = 6
```

Operacije nad bitovima (eng. BITWISE OPERATIONS)

C++ omogućuje i direktnu manipulaciju bitova neke promenljive

~ bitwise not	$\sim 10010110 = 01101001$
& bitwise and	$10010110 \& 11110000 = 10010000$
bitwise or	$10010110 11110000 = 11110110$
^ bitwise xor	$10010110 \wedge 11110000 = 01100110$
« left shift	$10010110 \ll 2 = 01011000$
» right shift	$10010110 \gg 3 = 11110010^3$

³Za označene brojeve, vršimo proširenje znakom (prvom cifrom), a za neoznačene nulom.

Jedan primer

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

int main()
{
    unsigned short a = 7;
    short neg_a = ~a + 1;

    cout << "Prvi nacin racunanja drugog komplementa:\n";
    cout << a << " : " << neg_a << endl;

    neg_a = (a ^ USHRT_MAX) + 1;

    cout << "Drugi nacin racunanja drugog komplementa:\n";
    cout << a << " : " << neg_a << endl;

    //Siftovanje (mnozenje stepenima dvojke)
    a <<= 4;
    cout << "a = " << a << ", a / 16 = " << a / 16 << endl;

    a = 13274;
    cout << "a u binarnom zapisu: " << bitset<8 * sizeof(unsigned short)>(a) << endl;

    //Da vidimo "Maskiranje"
    cout << "Najnizi nibl broja a: " << bitset<8 * sizeof(unsigned short)>(a & 0x0F) << endl;
    //0x0F = 0000 1111 u heksadecimalnom zapisu
    cout << "Treci nibl s desna broja a: " << bitset<8 * sizeof(unsigned short)>((a & (0x0F << 8)) >> 8) << endl;

    return 0;
}
```

Jedan primer

Prvi nacin racunanja drugog komplementa:

7 : -7

Drugi nacin racunanja drugog komplementa:

7 : -7

$a = 112$, $a / 16 = 7$

a u binarnom zapisu: 0011001111011010

Najnizi nibl broja a: 00000000000001010

Treci nibl s desna broja a: 000000000000000011

Manipulacija bitovima omogućuje veoma gusto pakovanje podataka

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

int main()
{
    unsigned prikaz_proseka = 0;
    cout << "Ukljuci prikaz proseka (1 = DA, 0 = NE)? ";
    cin >> prikaz_proseka;

    unsigned prikaz_zbira = 0;
    cout << "Ukljuci prikaz zbira (1 = DA, 0 = NE)? ";
    cin >> prikaz_zbira;

    unsigned prikaz_maksimuma = 0;
    cout << "Ukljuci prikaz maksimuma (1 = DA, 0 = NE)? ";
    cin >> prikaz_maksimuma;

    unsigned prikaz_minimuma = 0;
    cout << "Ukljuci prikaz minimuma (1 = DA, 0 = NE)? ";
    cin >> prikaz_minimuma;

    //Ispisi konfiguraciju:

    cout << "\n\nBice prikazane sledece stavke:\n";
    if(prikaz_proseka)
        cout << "Prosek\n";
    if(prikaz_zbira)
        cout << "Zbir\n";
    if(prikaz_maksimuma)
        cout << "Maksimum\n";
    if(prikaz_minimuma)
        cout << "Minimum\n";

    return 0;
}
```


Manipulacija bitovima omogućuje veoma gusto pakovanje podataka

```
Ukljuci prikaz proseka (1 = DA, 0 = NE)? 1  
Ukljuci prikaz zbira (1 = DA, 0 = NE)? 0  
Ukljuci prikaz maksimuma (1 = DA, 0 = NE)? 1  
Ukljuci prikaz minimuma (1 = DA, 0 = NE)? 0
```

Bice prikazane sledece stavke:

Prosek

Maksimum

Manipulacija bitovima omogućuje veoma gusto pakovanje podataka

```
#include <iostream>

#define PROSEK 1 << 0
#define ZBIR 1 << 1
#define MAX 1 << 2
#define MIN 1 << 3
#define BLANK 0

//define je pretprocesorska direktiva koja nalaze da identifikator posle
//kljucne reci define bude zamenjem nizom karaktera koji slede nakon razmaka
//pre nego sto se pristupi kompilaciji. Na primer, svako pojavljivanje reci ZBIR
//ce biti zamenjeno izrazom 1 << 1
//
//1 << 3 = 0...000001 << 3 = 0...001000

using namespace std;

int main()
{
    unsigned config = 0;
    unsigned read = 0;

    cout << "Ukljuci prikaz proseka (1 = DA, 0 = NE)? ";
    cin >> read;
    config |= read ? PROSEK : BLANK;

    cout << "Ukljuci prikaz zbira (1 = DA, 0 = NE)? ";
    cin >> read;
    config |= read ? ZBIR : BLANK;

    cout << "Ukljuci prikaz maksimuma (1 = DA, 0 = NE)? ";
    cin >> read;
    config |= read ? MAX : BLANK;
```

```
    cout << "Ukljuci prikaz minimuma (1 = DA, 0 = NE)? ";
    cin >> read;
    config |= read ? MIN : BLANK;

    //Ispisi konfiguraciju:

    cout << "\n\nBice prikazane sledece stavke:\n";
    if(config & PROSEK)
        cout << "Prosek\n";
    if(config & ZBIR)
        cout << "Zbir\n";
    if(config & MAX)
        cout << "Maksimum\n";
    if(config & MIN)
        cout << "Minimum\n";

    return 0;
```

Manipulacija bitovima omogućuje veoma gusto pakovanje podataka

```
Ukljuci prikaz proseka (1 = DA, 0 = NE)? 1
Ukljuci prikaz zbira (1 = DA, 0 = NE)? 0
Ukljuci prikaz maksimuma (1 = DA, 0 = NE)? 1
Ukljuci prikaz minimuma (1 = DA, 0 = NE)? 0
```

Bice prikazane sledece stavke:

Prosek

Maksimum

Učitavanje podataka iz memorije je danas daleko skuplje od prostih računskih operacija

ISSCC 2014 / SESSION 1 / PLENARY / 1.1

1.1 Computing's Energy Problem (and what we can do about it)

Mark Horowitz

Departments of Electrical Engineering and Computer Science,
Stanford University, Stanford, CA

Integer	
Add	
8 bit	0.03pJ
32 bit	0.1pJ
Mult	
8 bit	0.2pJ
32 bit	3.1pJ

FP	
FAdd	
16 bit	0.4pJ
32 bit	0.9pJ
FMult	
16 bit	1.1pJ
32 bit	3.7pJ

Memory	
Cache (64bit)	
8KB	10pJ
32KB	20pJ
1MB	100pJ
DRAM	1.3-2.6nJ

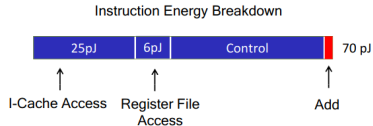


Figure 1.1.9: Rough energy costs for various operations in 45nm 0.9V.

Relacione i logičke operatore ćemo upoznati na sledećem času, kada budemo radili naredbe grananja

No, vratimo se sada na polazni problem

Problem

Neka je n proizvoljan broj iz $\mathbb{N}$

Neka su a i b nezavisno i uniformno odabrani iz $[1, n]$

Neka je p_n verovatnoća da su a i b uzajamno prosti

Da li postoji granična vrednost $\lim_{n \rightarrow \infty} p_n$ i ako postoji, koja joj je vrednost?

Osnovni pristup

1. Izlistamo sve parove brojeva $\{a, b\} \in [1, n]^2$
2. Prebrojimo koliko ih je uzajamno prostih
3. Podelimo taj broj sa n^2 , da dobijemo p_n
4. Proverimo kako se p_n menja sa uvećanjem n i postavljamo hipotezu koju posle dokazujemo

Šta nam je sve potrebno za to?

- Moramo nekako da predstavimo brojeve
- Moramo da proverimo da li su uzajamno prosti
 - Jedan način: proveravamo da li je $\text{NZD}(a, b) > 1$
 - Na primer, pitamo se da li $(\exists i > 1)(i \mid a \wedge i \mid b)$
- Moramo da prebrojimo sve koji jesu
- Moramo da nađemo odnos tog broja i n^2

Naivno rešenje

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

int main()
{
    unsigned n;
    cout << "Unesite n: ";
    cin >> n;
    unsigned uzajamno_prostih = 0;
    for(unsigned a = 1; a <= n; ++a)
    {
        for(unsigned b = 1; b <= n; ++b)
        {
            unsigned nzd = 1;
            for(unsigned i = 2; i <= a; ++i)
            {
                if(i > b)
                    break;
                if((a % i == 0) && (b % i == 0))
                {
                    nzd = i;
                    break;
                }
            }
            if (nzd == 1)
            {
                uzajamno_prostih++;
            }
        }
    }
    cout << uzajamno_prostih / (n * n) << endl;
    return 0;
}
```

Naivno rešenje

```
Unesite n: 1000  
0
```

Naivno rešenje

```
Unesite n: 1000  
0
```

Da li je moguće da je verovatnoća 0?

Naivno rešenje

Kako to da proverimo?

Da bi verovatnoća bila nula, ne bismo smeli da uočimo ni jedan uzajamno prost par

Printf debug

Da bismo to proverili, koristimo takozvani **PRINTF** debug⁴

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

int main()
{
    unsigned n;
    cout << "Unesite n: ";
    cin >> n;
    unsigned uzajamno_prostih = 0;
    for(unsigned a = 1; a <= n; ++a)
    {
        for(unsigned b = 1; b <= n; ++b)
        {
            unsigned nzd = 1;
            for(unsigned i = 2; i <= a; ++i)
            {
                if(i > b)
                    break;
                if((a % i == 0) && (b % i == 0))
                {
                    nzd = i;
                    break;
                }
            }
            if (nzd == 1)
            {
                uzajamno_prostih++;
            }
        }
    }

    cout << "Broj uocenih uzajamno prostih parova: " << uzajamno_prostih << endl;
    cout << uzajamno_prostih / (n * n) << endl;

    return 0;
}
```

⁴printf je u C-u bila funkcija čiju je ulogu u C++-u preuzeo cout «.

Printf debug

```
Unesite n: 1000  
Broj uocenih uzajamno prostih parova: 608383  
0
```

Nešto zaista nije u redu

Ponovo problem sa tipovima

```
    }  
    if (nzd == 1)  
    {  
        uzajamno_prostih++;  
    }  
}
```

```
cout << "Broj uocenih uzajamno prostih parova: " << uzajamno_prostih << endl;  
cout << uzajamno_prostih / (n * n) << endl;
```

unsigned unsigned $\Rightarrow$ **nema implicitne konverzije
pa je i celo deljenje unsigned**

$$\Rightarrow a/b \Leftrightarrow \lfloor a/b \rfloor$$

Predstavljanje realnih brojeva

Zadržimo se na racionalnim brojevima

Pretpostavimo da imamo tačno tri cifre za celobrojni i tri za decimalni deo

$$0,1 + 100,25 = 000,100 + 100,250$$

Operacija je identična operaciji sabiranja celih brojeva (zanemarimo decimalnu zapetu)

$$000100 + 100250 = 100350$$

nakon koje vratimo decimalnu zapetu na odgovarajuće mesto

$$100,350$$

Pošto Englezi umesto , koriste ., ovakav način predstavljanja brojeva nazivamo sistemom **FIKSNE TAČKE** (eng. **FIXED-POINT ARITHMETIC**)

Jedan način da to implementiramo u C++-u

1. Prvo pomnožimo izraz koji računamo sa
 $p = 10^{\text{broj decimalnih mesta koji želimo da sačuvamo}}$
2. Zatim izračunamo vrednost izraza v kao da su svi operandi celi brojevi
3. Celobrojni deo rešenja dobijamo kao v/p
4. Razlomljeni deo dobijamo kao $v \% p$

Kako to izgleda na našem primeru?

```
        }  
        if (nzd == 1)  
        {  
            uzajamno_prostih++;  
        }  
    }  
}  
  
const unsigned preciznost = 1000;  
unsigned rezultat = (uzajamno_prostih * preciznost) / (n * n);  
//(a / b) * preciznost = (a * preciznost) / b  
  
cout << rezultat / preciznost << '.' << rezultat % preciznost << endl;  
  
return 0;
```



Kako to izgleda na našem primeru?

```
Unesite n: 1000  
0.608
```

Šta ako želimo više decimalnih mesta?

```
    }  
    if (nzd == 1)  
    {  
        uzajamno_prostih++;  
    }  
}  
  
const unsigned preciznost = 1000000;  
unsigned rezultat = (uzajamno_prostih * preciznost) / (n * n);  
//(a / b) * preciznost = (a * preciznost) / b  
  
cout << rezultat / preciznost << '.' << rezultat % preciznost << endl;  
  
return 0;
```



Šta ako želimo više decimalnih mesta?

```
Unesite n: 1000  
0.2792
```


Šta ako želimo više decimalnih mesta?

```
const unsigned preciznost = 1000000;  
unsigned long rezultat = (uzajamno_prostih * preciznost) / (n * n);  
//(a / b) * preciznost = (a * preciznost) / b  
  
cout << rezultat / preciznost << '.' << rezultat % preciznost << endl;  
  
return 0;
```



Šiftovanje u levo (množenje nekim stepenom osnove brojnog sistema) čini da broj brzo raste, pa moramo osigurati da prijemna promenljiva ima odgovarajući opseg

Šta ako želimo više decimalnih mesta?

```
Unesite n: 1000  
0.2792
```

Šta ako želimo više decimalnih mesta?

```
    }  
    if (nzd == 1)  
    {  
        uzajamno_prostih++;  
    }  
}  
  
const unsigned preciznost = 1000000;  
unsigned long rezultat = ((unsigned long)(uzajamno_prostih) * preciznost) / (n * n);  
//(a / b) * preciznost = (a * preciznost) / b  
  
cout << rezultat / preciznost << '.' << rezultat % preciznost << endl;  
  
return 0;
```

Ali ne smemo zaboraviti ni tip samog izraza

Šta ako želimo više decimalnih mesta?

```
Unesite n: 1000  
0.608383
```

Naravno, sistem sa fiksnom tačkom je moguće implementirati i nad binarnim brojevima

Na primer sistem 4.4 (4 celobrojne cifre i 4 razlomljene cifre) izgleda ovako (težine cifara):

$$2^3 \ 2^2 \ 2^1 \ 2^0 \ . \ 2^{-1} \ 2^{-2} \ 2^{-3} \ 2^{-4} = 8 \ 4 \ 2 \ 1 \ \frac{1}{2} \ \frac{1}{4} \ \frac{1}{8} \ \frac{1}{16}$$

Konverzija 0.07 u binarni 4.4 sistem: $0.07 \times 2 = 0 + 0.14 \implies a_{-1} = 0$

$$0.14 \times 2 = 0 + 0.28 \implies a_{-2} = 0$$

$$0.28 \times 2 = 0 + 0.56 \implies a_{-3} = 0$$

$$0.56 \times 2 = 1 + 0.12 \implies a_{-4} = 1$$

Nemamo više decimalnih cifara na raspolaganju i naša predstava broja 0.07 je $0000.0001 = 2^{-4} = 0.0625$

Naravno, sistem sa fiksnom tačkom je moguće implementirati i nad binarnim brojevima

Šta da smo izabrali sistem 0.8?

Konverzija 0.07 u binarni 0.8 sistem: $0.07 \times 2 = 0 + 0.14 \implies a_{-1} = 0$

$0.14 \times 2 = 0 + 0.28 \implies a_{-2} = 0$

$0.28 \times 2 = 0 + 0.56 \implies a_{-3} = 0$

$0.56 \times 2 = 1 + 0.12 \implies a_{-4} = 1$

$0.12 \times 2 = 0 + 0.24 \implies a_{-5} = 0$

$0.24 \times 2 = 0 + 0.48 \implies a_{-6} = 0$

$0.48 \times 2 = 0 + 0.96 \implies a_{-7} = 0$

$0.96 \times 2 = 1 + 0.92 \implies a_{-8} = 1$

$a = 00010001 = 2^{-4} + 2^{-8} = 0.06640625$

Bolje, ali još uvek ne idealno. Uopšteno, vrlo mali broj racionalnih brojeva možemo tačno direktno predstaviti pomoću ograničenog broja binarnih cifara.

Šta je glavna mana sistema sa fiksnom tačkom?

Osim što ne mogu tačno da predstavljaju mnoge racionalne i ni jedan iracionalni broj

Šta je glavna mana sistema sa fiksnom tačkom?

Osim što ne mogu tačno da predstavljaju mnoge racionalne i ni jedan iracionalni broj

Veliki brojevi zahtevaju veći broj cifara levo od decimalne tačke, a mali brojevi veći broj desno od decimalne tačke

⇒ Ni jedan sistem sa fiksnom tačkom i dovoljno malim brojem bita ne može adekvatno da predstavi brojeve iz velikog raspona (veoma male i veoma velike)

Kako to rešavamo u decimalnom sistemu?

Jer je i tamo problem isti: 1479321101.0 i 0.0071003249 je moguće zapisati pomoću 10 cifara, ali bi zajednički sistem sa fiksnom tačkom zahtevao 20 cifara (10.10)

Kako to rešavamo u decimalnom sistemu?

Jer je i tamo problem isti: $a = 1479321101.0$ i $b = 0.0071003249$ je moguće zapisati pomoću 10 cifara, ali bi zajednički sistem sa fiksnom tačkom zahtevao 20 cifara (10.10)

$$a = 1.47932110 \times 10^9$$

$$b = 7.10032490 \times 10^{-3}$$

Oba broja smo zapisali u istom brojnom sistemu sa 10 cifara: 9 za mantisu i 1 za eksponent

Broj a je izgubio jednu cifru u tom procesu, zbog odsecanja (zaokruživanja)

Sistem predstave sa pokretnom tačkom (eng. FIXED-POINT ARITHMETIC)

Razlaganje zapisa na mantisu (signifikantu) i eksponent čini da se decimalna tačka „kreće” u skladu sa potrebama broja

IEEE 754 floating point standard:

float = 1 bit za znak, 8 bita za eksponent, 23 bita za mantisu

$$a = (-1)^{\text{znak}} \times 1.\text{mantisa} \times 2^{\text{eksponent}-127}$$

offset od 127 je tu da bi opseg stepena bio $[-127, 128]$

double = 1 bit za znak, 11 bita za eksponent, 52 bita za mantisu

Sistem sa pokretnom tačkom

Otklanja problem sistema sa fiksnom tačkom u pogledu širine opsega

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

int main()
{
    cout << "float moze da predstavi brojeve izmedju " << FLT_MIN << " i " << FLT_MAX << endl;
    cout << "double moze da predstavi brojeve izmedju " << DBL_MIN << " i " << DBL_MAX << endl;

    float a = 1093764251;
    float b = 0.0000700985;

    cout << "Sistem sa pokretnom tackom moze da predstavi i velike (" << a << ") i male (" << b << ") brojeve.\n";

    return 0;
}
```

Sistem sa pokretnom tačkom

Otklanja problem sistema sa fiksnom tačkom u pogledu širine opsega

```
float moze da predstavi brojeve izmedju 1.17549e-38 i 3.40282e+38  
double moze da predstavi brojeve izmedju 2.22507e-308 i 1.79769e+308  
Sistem sa pokretnom tackom moze da predstavi i velike (1.09376e+09) i male (7.00985e-05) brojeve.
```

Sistem sa pokretnom tačkom

Ali zadržava mane konačne reprezentacije

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <bitset>
using namespace std;

int main()
{
    float a = 0.1;
    cout << "a = " << a << endl;

    //Sada podesavamo cout da uvek ispisuje 10 decimalnih mesta.
    cout << fixed;
    cout << setprecision(10);

    cout << "a (malo preciznije) = " << a << endl;

    cout << "Binarni zapis a = " << bitset<8 * sizeof(unsigned)>(*reinterpret_cast<unsigned*>(&a)) << endl;

    return 0;
}
```

Sistem sa pokretnom tačkom

Ali zadržava mane konačne reprezentacije

```
a = 0.1  
a (malo preciznije) = 0.1000000015  
Binarni zapis a = 00111101110011001100110011001101
```

Sistem sa pokretnom tačkom

Ali zadržava mane konačne reprezentacije

←

→

↻

https://www.omnicalculator.com/other/floating-point

≡

omni calculator

We're hiring!

Embed

Share

I want to convert

floating-point to number ▾

Precision

Single-precision (32 bits) ▾

Bit input method

Together ▾

Floating-point binary

00111101110011001100110011001101

Binary representation:

00111101110011001100110011001101₂

• $S = 0_2 = 0$

• $E = 01111011_2 = 123$

• $F = 10011001100110011001101_2 = 5033165$

Hexadecimal representation:

3DCCCCD_H

Result

0.100000001490116119384765625

Created by Rijk de Wet

Reviewed by Steven Wooding

Based on research by [Institute of Electrical and Electronics Engineers](#) [IEEE Standard for Floating-Point Arithmetic](#); IEEE Std 754-2019 (Revision of IEEE 754-2008); 2019

[See 1 more source](#)

Last updated: Jan 18, 2024

♥♥♥♥♥

Cite

Table of contents:

◦ [How to use the floating-point calculator](#)

◦ [What is an IEEE754 floating-point number?](#)

◦ [How are real numbers stored with floating-point representation?](#)

◦ [The single-precision 32-bit float format](#)

◦ [The double-precision 64-bit float format](#)

◦ [How do I convert a number to floating-point?](#)

◦ [Floating-point accuracy](#)

111

Sistem sa pokretnom tačkom

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

int main()
{
    float a = 1e-2;
    float b = 1e-4;
    float c = 1e-5;

    if((a + b + c) == (c + b + a))
        cout << "Prekini trgovanje, gubitak je dosegao kritičnu tačku!.\n";
    else
        cout << "Sve je u redu, nastavi da trguješ.\n";

    return 0;
}
```

Sve je u redu, nastavi da trgujes.

Sistem sa pokretnom tačkom

→ ↺

🔒 https://en.wikipedia.org/wiki/Knight_Capital_Group

📄 120% ☆

📧 ⬇️ 🗑️ 📁

☰

 **WIKIPEDIA**
The Free Encyclopedia

Search

[Create account](#) [Log in](#) ...

Contents hide

(Top)

▼ [Company](#)

[Activities](#)

[Offices](#)

[Operating subsidiaries](#)

[2012 stock trading disruption](#)

[See also](#)

[References](#)

[External links](#)

Knight Capital Group

🌐 4 languages ▼

Article [Talk](#)

[Read](#) [Edit](#) [View history](#) [Tools](#) ▼

From Wikipedia, the free encyclopedia

The **Knight Capital Group** was an American global financial services firm engaging in [market making](#), electronic execution, and institutional sales and trading.^[1] With its [high-frequency trading](#) algorithms Knight was the largest trader in U.S. equities, with a market share of 17.3% on [NYSE](#) and 16.9% on [NASDAQ](#).^[2] The company agreed to be acquired by [Getco LLC](#) in December 2012 after an August 2012 trading error lost \$460 million. The merger was completed in July 2013, forming [KCG Holdings](#).

Company [\[edit \]](#)

Knight was formerly known as Knight/Trimark Group, Inc. and Knight Trading Group, Inc. Initially, Knight Trading group had multiple offices located in the United States and in

Knight Capital Group



Company type	Subsidiary
Traded as	NYSE: KCG 🔗 , until July 1, 2013
Industry	Financial services
Founded	1995

Možda im nije promakla baš takva greška, ali mogla je

Da vidimo šta se dogodilo u našem primeru

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

int main()
{
    float a = 1e-2;
    float b = 1e-4;
    float c = 1e-5;

    std::cout << std::fixed;
    std::cout << std::setprecision(10);
    cout << "Prvo veci: " << a + b + c << endl;
    cout << "Prvo manji: " << c + b + a << endl;
    cout << "Ocekivani zbir: 0.01011\n";

    return 0;
}
```

Da vidimo šta se dogodilo u našem primeru

```
Prvo veci: 0.0101099992  
Prvo manji: 0.0101100001  
Ocekivani zbir: 0.01011
```

Preporuka: sumirati prvo manje brojeve

Uprošćen primer sa decimalnim brojevima:

Pretpostavimo da imamo 5 cifara na raspolaganju za mantisu

$$\begin{aligned} & 1.2374 \times 10^{-3} + 2.3476 \times 10^1 + 9.2341 \times 10^{-4} \\ &= 0.00012374 \times 10^1 + 2.3476 \times 10^1 + 9.2341 \times 10^{-4} \\ &= (0.0001 + 2.3476) \times 10^1 + 9.2341 \times 10^{-4} \\ &= 2.3477 \times 10^1 + 0.00009341 \times 10^1 = 2.3477 \times 10^1 \end{aligned}$$

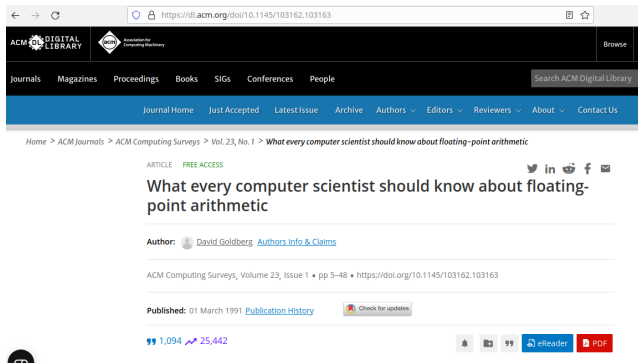
Preporuka: sumirati prvo manje brojeve

Uprošćen primer sa decimalnim brojevima:

Pretpostavimo da imamo 5 cifara na raspolaganju za mantisu

$$\begin{aligned} & 9.2341 \times 10^{-4} + 1.2374 \times 10^{-3} + 2.3476 \times 10^1 \\ &= 0.92341 \times 10^{-3} + 1.2374 \times 10^{-3} + 2.3476 \times 10^1 \\ &= (0.9234 + 1.2374) \times 10^{-3} + 2.3476 \times 10^1 \\ &= 2.1608 \times 10^{-3} + 2.3476 \times 10^1 \\ &= 0.00021608 \times 10^1 + 2.3476 \times 10^1 = (0.0002 + 2.3476) \times 10^1 = 2.3478 \end{aligned}$$

Sistemi sa pokretnom tačkom su mnogo složeniji



Primer sa prethodnog slajda veoma grubo ilustruje intuiciju iza problema zaokruživanja (eng. **ROUND-OFF ERROR**). Mi se nećemo detaljnije baviti tim problemima, ali je bitno biti svestan njihovog postojanja i po potrebi se obratiti odgovarajućoj literaturi

Još jedna preporuka: nikada ne proveravati jednakost brojeva sa pokretnom tačkom

```
#include <iostream>
#define TOL 1e-9
using namespace std;

int main()
{
    float a = 1e-2;
    float b = 1e-4;
    float c = 1e-5;

    if(abs((a + b + c) - (c + b + a)) < TOL)
        cout << "Prekini trgovanje, gubitak je dosegao kriticnu tacku!\n";
    else
        cout << "Sve je u redu, nastavi da trgujes.\n";

    return 0;
}
```

Još jedna preporuka: nikada ne proveravati jednakost brojeva sa pokretnom tačkom

Prekini trgovanje, gubitak je dosegao kritičnu tačku!

Ukoliko nam je bitno da tačno predstavimo racionalne decimalne brojeve

```
    }  
    if (nzd == 1)  
    {  
        uzajamno_prostih++;  
    }  
}  
  
const unsigned preciznost = 1000000;  
unsigned long rezultat = ((unsigned long)(uzajamno_prostih) * preciznost) / (n * n);  
//(a / b) * preciznost = (a * preciznost) / b  
  
cout << rezultat / preciznost << '.' << rezultat % preciznost << endl;  
  
return 0;
```

Možemo koristiti decimalni sistem sa fiksnom tačkom (iznad je samo opis principa; postoje daleko naprednije implementacije)

Ta potreba često postoji u svetu finansija

← → ↺ <https://en.wikipedia.org/wiki/GnuCash> 120%

WIKIPEDIA
The Free Encyclopedia

Q Search Wikipedia **Search** [Create account](#) [Log in](#) ...

GnuCash [25 languages](#)

[Article](#) [Talk](#) [Read](#) [Edit](#) [View history](#) [Tools](#)

From Wikipedia, the free encyclopedia

GnuCash is an [accounting program](#) that implements a [double-entry bookkeeping system](#). It was initially aimed at developing capabilities similar to [Intuit, Inc.'s Quicken](#) application,^[1] but also has features for [small business](#) accounting.^[10] Recent development has been focused on adapting to modern desktop support-library requirements.

GnuCash is part of the [GNU Project](#),^{[11][12]} and runs on [Linux](#), [GNU](#), [OpenBSD](#), [FreeBSD](#), [Solaris](#), [macOS](#), and other [Unix-like](#) platforms.^[13] A [Microsoft Windows](#) (2000 or newer) port was made available starting with the 2.2.0 series.^[14]

GnuCash includes scripting support via [scheme](#), mostly used for creating custom reports.^[15]

Small business accounting features [\[edit \]](#)

- Invoicing and Credit Notes (Credit note functionality was added with version 2.6)^[22]
- Accounts Receivable (A/R)
- Accounts Payable (A/P) including bills due reminders
- Employee expense voucher
- Limited Payroll Management through the use of A/Receivable and A/Payable accounts.^[23]
- Depreciation
- Mapping to income tax schedules and TXF export for import into tax prep software (US)
- Setting up tax tables and applying sales tax on invoices

Technical design [\[edit \]](#)

GnuCash is written primarily in [C](#), with a small fraction in [Scheme](#).^[1] One of the available features is pure [fixed-point arithmetic](#) to avoid rounding errors which would arise with [floating-point arithmetic](#). This feature was introduced with version 1.6.^[24]

GnuCash



Ta potreba često postoji u svetu finansija

← → ↻

🔒 <https://stackoverflow.com/questions/149033/best-way-to-store-currency-values-in-c>

 [About](#) [Products](#) [For Teams](#)

🏠 Home

❓ Questions

🏷 Tags

👤 Users

🏢 Companies

LABS ?

💬 Discussions NEW

COLLECTIVES +

🌟 Explore Collectives

TEAMS

Best way to store currency values in C++

Asked 15 years, 5 months ago Modified 1 year, 6 months ago Viewed 59k times

▲

70

▼

🔖

🕒

c++

currency

Share Improve this question Follow

 **Bo Persson**
91.4k • 31 • 148 • 205

 **Javier Ramos**

I know that a float isn't appropriate to store currency values because of rounding errors. Is there a standard way to represent money in C++?

I've looked in the boost library and found nothing about it. In java, it seems that BigInteger is the way but I couldn't find an equivalent in C++. I could write my own money class, but prefer not to do so if there is something tested.

edited Jun 18, 2012 at 16:19

asked Sep 29, 2008 at 14:53

119

Podsetnik za problem sa opsegom promenljivih

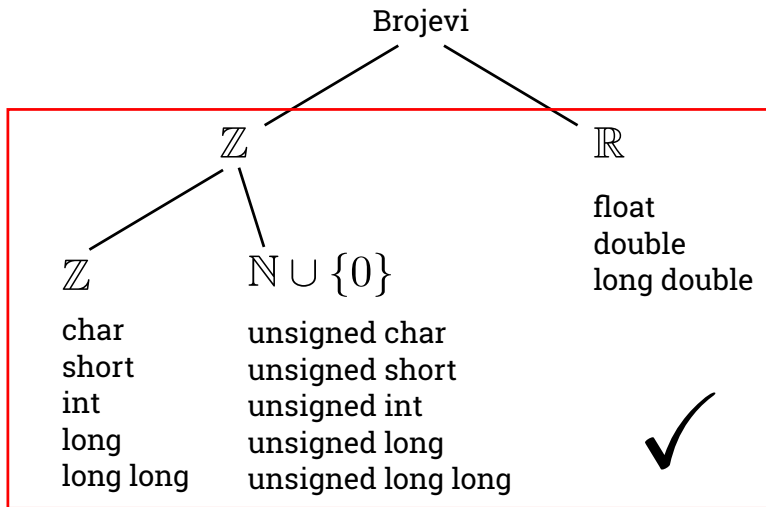


U našem problemu je double sasvim u redu

```
    }  
    cout << double(uzajamno_prostih) / (n * n) << endl;  
    return 0;  
}
```

EksPLICITNO konvertujemo jedan operand u double, da bi i deljenje bilo double

To bi bilo sve za danas



Na sledećem času ćemo nastaviti sa rešavanjem ovog problema

Neka je n proizvoljan broj iz $\mathbb{N}$

Neka su a i b nezavisno i uniformno odabrani iz $[1, n]$

Neka je p_n verovatnoća da su a i b uzajamno prosti

Da li postoji granična vrednost $\lim_{n \rightarrow \infty} p_n$ i ako postoji, koja joj je vrednost?

Hvala na pažnji