

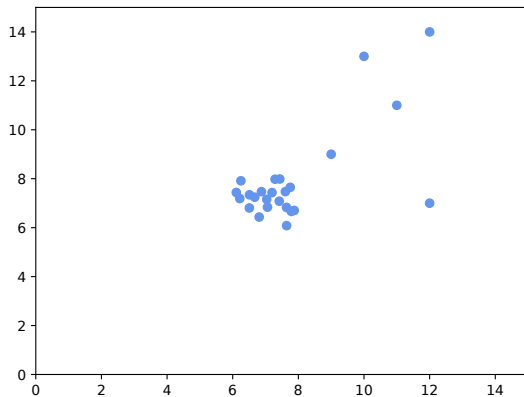
Programiranje 2, letnji semestar 2023/4

Složeni tipovi podataka i uvod u
objektno-orijentisano programiranje (OOP)

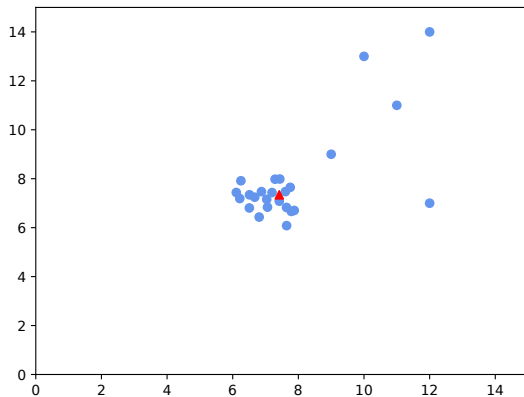
Stefan Nikolić

Prirodno-matematički fakultet, Novi Sad

15.04.2024.

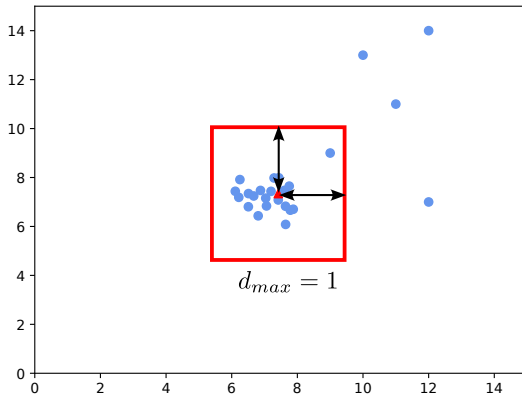


Imamo zadat skup tačaka P



Imamo zadat skup tačaka P i jednu tačku c

Zadatak



Za dati parametar $d_{max} \in \mathbb{R}^+$, izdvojiti sve tačke $p \in P$ čija je Čebiševljeva razdaljina od tačke $c \leq d_{max}$

Čebiševljeva razdaljina

$$d((x_1, y_1), (x_2, y_2)) = \max(|x_2 - x_1|, |y_2 - y_1|)$$

Prvo pitanje (kao i na prošlom času)

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] = {...
```

Šta je potrebno da implementiramo?

1. Korisnik unosi koordinate tačka iz P na tastaturi
2. Kada upiše sve tačke, unosi i koordinate tačke c
3. Najzad, korisnik unosi d_{max} i dobija nazad izabrane tačke

Jedna implementacija

```
#ifndef DISTANCES_H
#define DISTANCES_H

//Funkcija racuna Cebisevljevu razdaljinu tacke (x, y) od tacke (center_x, center_y)
float cheby_dist(float x, float y, float center_x, float center_y);

#endif
~
~
~
distances.h 1,1
```

Jedna implementacija

```
#include <iostream>
#include "distances.h"

float cheby_dist(float x, float y, float center_x, float center_y)
{
    return std::max(abs(x - center_x), abs(y - center_y));
}

~
~
```

distances.cpp

1,1

Jedna implementacija

```
#include <iostream>
#include "distances.h"

#define MAX_CNT 100

using namespace std;

int main()
{
    float xs[MAX_CNT];
    float ys[MAX_CNT];

    unsigned cnt = 0;

    do
    {
        cout << "Unesite broj tacaka u P: ";
        cin >> cnt;

        if(cnt >= MAX_CNT)
            cout << "Maksimalan broj tacaka (" << MAX_CNT << ") je ucitan!\n";
    } while(cnt >= MAX_CNT);
```

Jedna implementacija

```
for(unsigned i = 0; i < cnt; ++i)
{
    cout << "Unesite x koordinatu tacke " << i << ": ";
    cin >> xs[i];

    cout << "Unesite y koordinatu tacke " << i << ": ";
    cin >> ys[i];
}
```

Jedna implementacija

```
float c_x = 0;
float c_y = 0;

cout << "Unesite x koordinatu tacke c: ";
cin >> c_x;
cout << "Unesite y koordinatu tacke c: ";
cin >> c_y;

cout << "Unesite maksimalnu Cebisevljevu razdaljinu: ";
unsigned max_cheby;
cin >> max_cheby;
```

Jedna implementacija

```
for(unsigned i = 0; i < cnt; ++i)
{
    if(cheby_dist(xs[i], ys[i], c_x, c_y) <= max_cheby)
        cout << "tacka " << i << " = (" << xs[i] << ", " << ys[i] << ")\n";
}

return 0;
}
```

Jedna implementacija

```
Unesite broj tacaka u P: 6
Unesite x koordinatu tacke 0: 2
Unesite y koordinatu tacke 0: 3
Unesite x koordinatu tacke 1: 7
Unesite y koordinatu tacke 1: 4
Unesite x koordinatu tacke 2: 1
Unesite y koordinatu tacke 2: 1
Unesite x koordinatu tacke 3: -3
Unesite y koordinatu tacke 3: -6
Unesite x koordinatu tacke 4: 2
Unesite y koordinatu tacke 4: 5
Unesite x koordinatu tacke 5: 17
Unesite y koordinatu tacke 5: 4
Unesite x koordinatu tacke c: 3
Unesite y koordinatu tacke c: 3
Unesite maksimalnu Cebisevljevu razdaljinu: 5
tacka 0 = (2, 3)
tacka 1 = (7, 4)
tacka 2 = (1, 1)
tacka 4 = (2, 5)
```

Malo složeniji zadatak

Pored koordinata, svaka tačka ima i naziv koji je potrebno da zapamtimo i ispišemo nakon filtriranja

Stringovi

Primer

```
#include <string>

using namespace std;

int main()
{
    string ime = "Mileva";

    cout << "Zadato ime je " << ime << endl;

    cout << "Njegova dužina je " << ime.length() << endl;
```

```
Zadato ime je Mileva  
Njegova dužina je 6
```

```
cout << "Da li želite da ga promenite? (unesite d za DA i n za NE)" << endl;
string choice = "none";
do
{
    getline(cin, choice); //getline cita liniju do kraja
} while((choice.length() < 1) || (choice[0] != 'd' && choice[0] != 'n'));
```

```
if(choice[0] == 'd')  
{  
    cout << "Unesite novo ime: ";  
    cin >> ime;  
  
    cout << "Uneto ime je " << ime << endl;
```

```
Da li želite da ga promenite? (unesite d za DA i n za NE)
k
k
d
Unesite novo ime: albert
Uneto ime je albert
```

```
        if(!isupper(ime[0]))  
            cout << "Upozorenje: uneto ime ne pocinje velikim slovom." << endl;  
    }  
    return 0;  
}
```

```
Unesite novo ime: albert  
Uneto ime je albert  
Upozorenje: uneto ime ne pocinje velikim slovom.
```


Stringovi u C++-u

Klasa (uskoro o tome) definisana u biblioteci *string*

Deklaracija stringa

```
string identifikator;
```

Inicijalizacija stringa

string identifikator = drugi string ili string literal;

String literali

String literal je niz karaktera unutar duplih znaka navoda

Primer literala: "Mileva"

Primer inicijalizacije: `string ime = "Mileva";`

U osnovi, stringovi su nizovi char promenljivih

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

int main()
{
    string s = "Mileva";

    for(unsigned i = 0; i <= s.length(); ++i)
        cout << +s[i] << endl;

    return 0;
}
```

U osnovi, stringovi su nizovi char promenljivih

77

105

108

101

118

97

0

U osnovi, stringovi su nizovi char promenljivih

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

77 = 'M'

105 = 'i'

108 = 'l'

101 = 'e'

118 = 'v'

97 = 'a'

0 = '\0'

U C-u su bili samo to i još uvek možemo koristiti C stringove

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

int main()
{
    char s[] = {'M', 'i', 'l', 'e', 'v', 'a', '\\0'};
    puts(s);

    char s2[] = "Mileva";
    puts(s2);

    return 0;
}
```


U C-u su bili samo to i još uvek možemo koristiti C stringove

```
Mileva  
Mileva
```

A na pozadinu stringova se možemo oslanjati i prilikom upotrebe C++ stringova

```
#include <iostream>
#include <string>

std::string ad_hoc_to_upper(const std::string& s)
{
    std::string su = s;

    for(unsigned i = 0; i < s.length(); ++i)
    {
        if(s[i] >= 'a' && s[i] <= 'z')
            su[i] += 'A' - 'a';
    }

    return su;
}

int main()
{
    std::string s = "albert";

    std::cout << ad_hoc_to_upper(s) << std::endl;

    return 0;
}
```

A na pozadinu stringova se možemo oslanjati i prilikom upotrebe C++ stringova

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

'a' = 97 'A' = 65

'p' = 112 'P' = 80

'A' - 'a' = 32 = 'P' - 'p'

A na pozadinu stringova se možemo oslanjati i prilikom upotrebe C++ stringova



ALBERT

No, C++ stringovi obezbeđuju mnogo više od toga

```
#include <iostream>
#include <string>

std::string ad_hoc_to_upper(const std::string& s)
{
    std::string su = s;
    for(unsigned i = 0; i < s.length(); ++i)
    {
        if(s[i] >= 'a' && s[i] <= 'z')
            su[i] += 'A' - 'a';
    }

    return su;
}

int main()
{
    std::string s = "albert";

    std::cout << ad_hoc_to_upper(s) << std::endl;

    return 0;
}
```

automatsko kopiranje sadržaja

No, C++ stringovi obezbeđuju mnogo više od toga

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string ime = "Mileva";
    ime += " Maric";
    ime += " Ajnstajn";

    cout << ime << endl;

    return 0;
}
```

konkatenaciju i automatsko proširenje
rezervisanog prostora

No, C++ stringovi obezbeđuju mnogo više od toga

Mileva Maric Ajnstajn

No, C++ stringovi obezbeđuju mnogo više od toga

 https://en.cppreference.com/w/cpp/string/basic_string 

<code>copy</code>	(public member function)
<code>resize</code>	changes the number of characters stored (public member function)
<code>resize_and_overwrite</code> (C++23)	changes the number of characters stored and possibly overwrites indeterminate contents via user-provided operation (public member function)
<code>swap</code>	swaps the contents (public member function)
Search	
<code>find</code>	finds the first occurrence of the given substring (public member function)
<code>rfind</code>	find the last occurrence of a substring (public member function)
<code>find_first_of</code>	find first occurrence of characters (public member function)
<code>find_first_not_of</code>	find first absence of characters (public member function)
<code>find_last_of</code>	find last occurrence of characters (public member function)
<code>find_last_not_of</code>	find last absence of characters (public member function)
Operations	
<code>compare</code>	compares two strings (public member function)
<code>starts_with</code> (C++20)	checks if the string starts with the given prefix (public member function)
<code>ends_with</code> (C++20)	checks if the string ends with the given suffix (public member function)

Setimo se da pri pozivu funkcije dolazi do kopiranja parametara

C string je niz, pa će prosleđivanje C stringa dovesti samo do kopiranja pokazivača na prvi element

Međutim, C++ string je objekat i on će ceo biti kopiran: što je string duži, to će kopiranje duže trajati

Kako to da predupredimo?

Kako to da predupredimo?

Prosleđivanjem po adresi, uz upotrebu pokazivača na konstantu vrednost, gde god je to primereno

C++ nudi i jedno rešenje koje je često elegantnije i bezbednije, iako se upozadini uglavnom opet prevodi u konstantan pokazivač

Reference

Šta je referenca?

tip& identifikator = identifikator objekta kog referencira;

Referencu možemo shvatiti kao drugo ime za neki objekat ili kao konstantan pokazivač sa automatskim dereferenciranjem

referenca na konstantni string




```
std::string ad_hoc_to_upper(const std::string& s)
```

Primer prosleđivanja po referenci

Zbog automatskog dereferenciranja,
prosleđujemo identifikator a ne adresu

```
int main()  
{  
    std::string s = "albert";  
    std::cout << ad_hoc_to_upper(s) << std::endl;  
    return 0;  
}
```



Neke odlike referenci

Pošto se referenca **uvek** ponaša kao konstantan pokazivač¹, moramo je inicijalizovati

Dereferenciranje je automatsko (nema potrebe da pišemo * ispred identifikatora da bismo izvukli vrednost sa adrese, niti & da bismo dobili referencu na objekat)

Referenca nikada ne može da pokazuje na NULL, već na neki konkretan objekat

To je bezbednije, ali ako uklonimo objekat iz memorije, referenca opet može da pokazuje na nešto što nema smisla

¹Zapamtimo da to nije isto što i pokazivač na konstantnu vrednost

Setimo se primera sa prošlog časa

```
#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;
}
```

Setimo se primera sa prošlog časa

```
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;  
      |                ^NNN  
wrong_ret.cpp:6:13: note: declared here  
    6 |         int sum = a + b;  
      |             ^NN  
/bin/bash: line 1: 30593 Segmentation fault      (core dumped) ./run
```

Kada pokazivač zamenimo referencom, problem ostaje

```
#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;
}
```

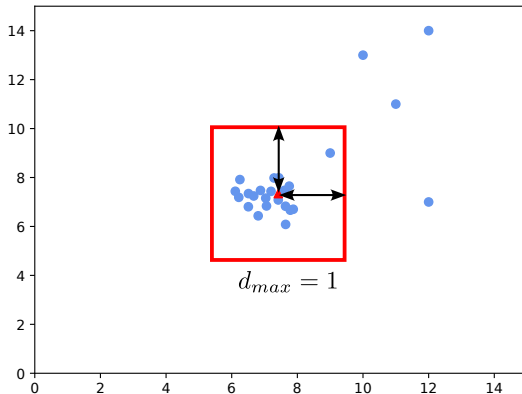
Kada pokazivač zamenimo referencom, problem ostaje

```
ref_stack.cpp: In function 'int& return_sum_address(int, int)':
ref_stack.cpp:8:16: warning: reference to local variable 'sum' returned [-Wreturn-local-addr]
    8 |         return sum;
      |         ^~~~
ref_stack.cpp:6:13: note: declared here
    6 |         int sum = a + b;
      |         ^~~~
/bin/bash: line 1: 5207 Segmentation fault      (core dumped) ./run

real    0m0.005s
user    0m0.002s
sys     0m0.000s

shell returned 139
```

Vratimo se sada na naš zadatak



Pored koordinata, svaka tačka ima i naziv koji je potrebno da zapamtimo i ispišemo nakon filtriranja

Kako to da izvedemo?

Kako to da izvedemo?

```
#include <iostream>
#include <string>
#include "distances.h"

#define MAX_CNT 100

using namespace std;

int main()
{
    float xs[MAX_CNT];
    float ys[MAX_CNT];
    string labels[MAX_CNT];

    unsigned cnt = 0;
```


Kako to da izvedemo?

```
for(unsigned i = 0; i < cnt; ++i)
{
    cout << "Unesite ime tacke " << i << ": ";
    getline(cin >> ws, labels[i]);

    cout << "Unesite x koordinatu tacke " << i << ": ";
    cin >> xs[i];

    cout << "Unesite y koordinatu tacke " << i << ": ";
    cin >> ys[i];
}
```

Kako to da izvedemo?

```
for(unsigned i = 0; i < cnt; ++i)
{
    if(cheby_dist(xs[i], ys[i], c_x, c_y) <= max_cheby)
        cout << "tacka " << labels[i] << " = (" << xs[i] << ", " << ys[i] << ")\n";
}
```

```
return 0;
```

```
}
```

Kako to da izvedemo?

```
Unesite broj tacaka u P: 6
Unesite ime tacke 0: Novi Sad
Unesite x koordinatu tacke 0: 13
Unesite y koordinatu tacke 0: 4
Unesite ime tacke 1: Sombor
Unesite x koordinatu tacke 1: 12
Unesite y koordinatu tacke 1: 9
Unesite ime tacke 2: Becej
Unesite x koordinatu tacke 2: 9
Unesite y koordinatu tacke 2: 7
Unesite ime tacke 3: Sabac
Unesite x koordinatu tacke 3: 7
Unesite y koordinatu tacke 3: 1
Unesite ime tacke 4: Donji Milanovac
Unesite x koordinatu tacke 4: 22
Unesite y koordinatu tacke 4: 2
Unesite ime tacke 5: Bujanovac
Unesite x koordinatu tacke 5: 18
Unesite y koordinatu tacke 5: -7
Unesite x koordinatu tacke c: 18
Unesite y koordinatu tacke c: 3
Unesite maksimalnu Cebisevljevu razdaljinu: 10
tacka Novi Sad = (13, 4)
tacka Sombor = (12, 9)
tacka Becej = (9, 7)
tacka Donji Milanovac = (22, 2)
tacka Bujanovac = (18, -7)
```

Poziciono kombinovanje podataka smeštenih u različite nizove je moguće

Ali brzo postaje teško za održavanje i podložno greškama

Na primer, ako u jednom nizu zamenimo mesta vrednostima na dve pozicije, a u drugom to zaboravimo, naši podaci više neće biti ispravni

Srećom, C/C++ pruža mogućnost znatno fleksibilnijeg kombinovanja različitih podataka, nezavisno od tipa

Strukture

Deklaracija struktura

```
struct identifikator  
{  
    deklaracija polja 1;  
    deklaracija polja 2;  
    ...  
};
```

```
struct Point
{
    float x;
    float y;
    std::string label;
};
```


Nakon deklaracije strukture, ona postaje novi raspoloživi tip

Promenljivu tipa struct Point možemo deklarirati na sledeći način:

```
struct Point p;
```

Poljima pristupamo korišćenjem operatora ”

Na primer, ako želimo da polje x smestimo u neku spoljnu promenljivu, možemo napisati sledeće:

```
float x = p.x;
```

Šta je zapravo struktura?

Da li neko ima ideju?

Šta je zapravo struktura?

Struktura je suštinski niz čiji elementi imaju proizvoljan tip i dužinu

To možemo i da proverimo

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

typedef struct Triple
{
    float a;
    float b;
    float c;
    unsigned long accesses_to_a;
    unsigned long accesses_to_b;
    unsigned long accesses_to_c;
} Triple;

int main()
{
    Triple t;

    t.a = t.b = t.c = -1;
    t.accesses_to_a = t.accesses_to_b = t.accesses_to_c = 0;

    cout << "Addr(t.b) - Addr(t.a) = " << long(&t.b) - long(&t.a) << endl;
    cout << "Addr(t.c) - Addr(t.b) = " << long(&t.c) - long(&t.b) << endl;
    cout << "sizeof(float) = " << sizeof(float) << endl;

    cout << "Addr(t.accesses_to_b) - Addr(t.accesses_to_a) = " << long(&t.accesses_to_b) - long(&t.accesses_to_a) << endl;
    cout << "sizeof(unsigned long) = " << sizeof(unsigned long) << endl;

    return 0;
}
```

To možemo i da proverimo

```
Addr(t.b) - Addr(t.a) = 4  
Addr(t.c) - Addr(t.b) = 4  
sizeof(float) = 4  
Addr(t.accesses_to_b) - Addr(t.accesses_to_a) = 8  
sizeof(unsigned long) = 8
```

Iz različitih razloga, nekada između uzastopnih polja mogu postojati i nekorišćene adrese, ali su polja u memoriji gotovo uvek smeštena u istom poretku u kom su i deklarirana

Naš problem uz upotrebu struktura

```
#ifndef POINT_H
#define POINT_H

#include <string>

struct Point
{
    float x;
    float y;

    std::string label;
};

//Funkcija racuna Cebisevljevu razdaljinu tacke p od tacke c
float cheby_dist(const struct Point& p, const struct Point& c);

#endif
```

Naš problem uz upotrebu struktura

```
#include <algorithm>
#include "point.h"

float cheby_dist(const struct Point& p, const struct Point& c)
{
    return std::max(abs(p.x - c.x), abs(p.y - c.y));
}
```



~
~
~
~
~
~
~

point.cpp

Naš problem uz upotrebu struktura

```
#include <iostream>
#include <string>
#include "point.h"

#define MAX_CNT 100

using namespace std;

int main()
{
    struct Point points[MAX_CNT];
    unsigned cnt = 0;
```

Naš problem uz upotrebu struktura

```
for(unsigned i = 0; i < cnt; ++i)
{
    cout << "Unesite ime tacke " << i << ": ";
    getline(cin >> ws, points[i].label);

    cout << "Unesite x koordinatu tacke " << i << ": ";
    cin >> points[i].x;

    cout << "Unesite y koordinatu tacke " << i << ": ";
    cin >> points[i].y;
}
```

Naš problem uz upotrebu struktura

```
struct Point c;  
  
cout << "Unesite x koordinatu tacke c: ";  
cin >> c.x;  
cout << "Unesite y koordinatu tacke c: ";  
cin >> c.y;  
  
cout << "Unesite maksimalnu Cebisevljevu razdaljinu: ";  
unsigned max_cheby;  
cin >> max_cheby;
```

Naš problem uz upotrebu struktura

```
for(unsigned i = 0; i < cnt; ++i)
{
    if(cheby_dist(points[i], c) <= max_cheby)
        cout << "tacka " << points[i].label << " = (" << points[i].x << ", " << points[i].y << ")\n";
}
return 0;
}
```

Naš problem uz upotrebu struktura

```
Unesite broj tacaka u P: 6
Unesite ime tacke 0: Novi Sad
Unesite x koordinatu tacke 0: 13
Unesite y koordinatu tacke 0: 4
Unesite ime tacke 1: Sombor
Unesite x koordinatu tacke 1: 12
Unesite y koordinatu tacke 1: 9
Unesite ime tacke 2: Becej
Unesite x koordinatu tacke 2: 9
Unesite y koordinatu tacke 2: 7
Unesite ime tacke 3: Sabac
Unesite x koordinatu tacke 3: 7
Unesite y koordinatu tacke 3: 1
Unesite ime tacke 4: Donji Milanovac
Unesite x koordinatu tacke 4: 22
Unesite y koordinatu tacke 4: 2
Unesite ime tacke 5: Bujanovac
Unesite x koordinatu tacke 5: 18
Unesite y koordinatu tacke 5: -7
Unesite x koordinatu tacke c: 18
Unesite y koordinatu tacke c: 3
Unesite maksimalnu Cebisevljevu razdaljinu: 10
tacka Novi Sad = (13, 4)
tacka Sombor = (12, 9)
tacka Becej = (9, 7)
tacka Donji Milanovac = (22, 2)
tacka Bujanovac = (18, -7)
```

typedef

Prefiks typedef nam omogućuje da određenom tipu pridružimo drugi naziv

Na primer:

```
typedef struct Point
{
    float x;
    float y;
    std::string label;
} Point;
```

nam omogućuje da svuda gde bismo inače pisali struct Point pišemo samo Point

Apstrakcija podataka

Problemi sa vidljivim podacima

```
typedef struct Point
{
    float x;
    float y;
    std::string label;
} Point;
```

Kada dopustimo direktno pristupanje podacima, to dovodi do nekoliko problema

Problemi sa vidljivim podacima

```
typedef struct Point
{
    float x;
    float y;
    std::string label;
} Point;
```

1) Promena načina skladištenja podataka postaje komplikovana, jer moramo izmeniti i sav kod koji im pristupa

Problemi sa vidljivim podacima

```
typedef struct Point
{
    float x;
    float y;
    std::string label;
} Point;
```

2) Ako neko drugi koristi naš kod, mora da razume i kako su podaci organizovani, a ne samo kako da im pristupi

Problemi sa vidljivim podacima

```
typedef struct Point
{
    float x;
    float y;
    std::string label;
} Point;
```

3) Neograničen pristup iz bilo kog dela programa dovodi do značajnog povećanja rizika od unošenja greške

Kako da izbegnemo te probleme?

Razdvojimo organizaciju podataka od načina na koji im je moguće pristupiti

Kako to izgleda na našem primeru?

Najpre deklariramo interfejse za uzimanje podataka

```
#ifndef POINT_H
#define POINT_H

#include <string>

typedef struct Point
{
    float x;
    float y;

    std::string label;
} Point;

//Getters

//Funkcija vraca x-koordinatu tacke p
float get_x(const Point& p);
//Funkcija vraca y-koordinatu tacke p
float get_y(const Point& p);
//Funkcija vraca naziv tacke p
const std::string& get_label(const Point& p);
```

Zatim deklariramo interfejs za izmenu podataka

```
//Setters

//Funkcija postavlja x-koordinatu tacke p
void set_x(Point& p, float x);
//Funkcija postavlja y-koordinatu tacke p
void set_y(Point& p, float y);
//Funkcija postavlja ime tacke p
void set_label(Point& p, const std::string & label);

//Other functions

//Funkcija racuna Cebisevljevu razdaljinu tacke p od tacke c
float cheby_dist(const struct Point& p, const struct Point& c);

#endif
```

Najzad deklariramo funkcije koje implementiraju preostale operacije

```
//Setters
```

```
//Funkcija postavlja x-koordinatu tacke p
```

```
void set_x(Point& p, float x);
```

```
//Funkcija postavlja y-koordinatu tacke p
```

```
void set_y(Point& p, float y);
```

```
//Funkcija postavlja ime tacke p
```

```
void set_label(Point& p, const std::string & label);
```

```
//Other functions
```

```
//Funkcija racuna Cebisevljevu razdaljinu tacke p od tacke c
```

```
float cheby_dist(const struct Point& p, const struct Point& c);
```

```
#endif
```


Na ovaj način, korisnik ima uvid u sve što može da radi sa datom strukturom podataka, bez da zna kako je ona implementirana

Mi naravno sve deklarirane funkcije moramo i da implementiramo

```
#include <algorithm>
#include "point.h"

float get_x(const Point& p)
{
    return p.x;
}

float get_y(const Point& p)
{
    return p.y;
}

const std::string& get_label(const Point& p)
{
    return p.label;
}
```

Mi naravno sve deklarirane funkcije moramo i da implementiramo

```
void set_x(Point& p, float x)
{
    p.x = x;
}

void set_y(Point& p, float y)
{
    p.y = y;
}

void set_label(Point& p, const std::string & label)
{
    p.label = label;
}

float cheby_dist(const struct Point& p, const struct Point& c)
{
    return std::max(abs(get_x(p) - get_x(c)), abs(get_y(p) - get_y(c)));
}
```



Nadalje strukturi pristupamo samo pomoću deklarisanih interfejsa

```
float x;
float y;
string label;
for(unsigned i = 0; i < cnt; ++i)
{
    cout << "Unesite ime tacke " << i << ": ";
    getline(cin >> ws, label);
    set_label(points[i], label);

    cout << "Unesite x koordinatu tacke " << i << ": ";
    cin >> x;
    set_x(points[i], x);

    cout << "Unesite y koordinatu tacke " << i << ": ";
    cin >> y;
    set_y(points[i], y);
}
```

Nadalje strukturi pristupamo samo pomoću deklariranih interfejsa

```
struct Point c;  
  
cout << "Unesite x koordinatu tacke c: ";  
cin >> x;  
set_x(c, x);  
  
cout << "Unesite y koordinatu tacke c: ";  
cin >> y;  
set_y(c, y);
```

Nadalje strukturi pristupamo samo pomoću deklarisanih interfejsa

```
for(unsigned i = 0; i < cnt; ++i)
{
    if(cheby_dist(points[i], c) <= max_cheby)
        cout << "tacka " << get_label(points[i]) << " = (" << get_x(points[i]) << ", " << get_y(points[i]) << ")\n";
}
return 0;
}
```

Nadalje strukturi pristupamo samo pomoću deklarisanih interfejsa

```
Unesite broj tacaka u P: 6
Unesite ime tacke 0: Novi Sad
Unesite x koordinatu tacke 0: 13
Unesite y koordinatu tacke 0: 4
Unesite ime tacke 1: Sombor
Unesite x koordinatu tacke 1: 12
Unesite y koordinatu tacke 1: 9
Unesite ime tacke 2: Becej
Unesite x koordinatu tacke 2: 9
Unesite y koordinatu tacke 2: 7
Unesite ime tacke 3: Sabac
Unesite x koordinatu tacke 3: 7
Unesite y koordinatu tacke 3: 1
Unesite ime tacke 4: Donji Milanovac
Unesite x koordinatu tacke 4: 22
Unesite y koordinatu tacke 4: 2
Unesite ime tacke 5: Bujanovac
Unesite x koordinatu tacke 5: 18
Unesite y koordinatu tacke 5: -7
Unesite x koordinatu tacke c: 18
Unesite y koordinatu tacke c: 3
Unesite maksimalnu Cebisevljevu razdaljinu: 10
tacka Novi Sad = (13, 4)
tacka Sombor = (12, 9)
tacka Becej = (9, 7)
tacka Donji Milanovac = (22, 2)
tacka Bujanovac = (18, -7)
```

Ovo možda još uvek izgleda pomalo apsurdno...

Pozivamo funkcije da bismo pristupili nečem što tačno znamo šta je

No, kao što smo rekli, ovakva apstrakcija nam omogućuje da ne znamo

Pokušajmo na primer da tačke skladištimo pomoću polarnih koordinata

Prelazak sa Dekartovih na polarne koordinate

Koji od ovih fajlova ćemo morati da menjamo?

point.h

point.cpp

cheby.cpp

```
#ifndef POINT_H
#define POINT_H

#include <string>
#include <cmath>

typedef struct Point
{
    float r = 1.0;
    float t = 1.0;

    std::string label;
} Point;

//Getters

//Funkcija vraca x-koordinatu tacke p
float get_x(const Point& p);
//Funkcija vraca y-koordinatu tacke p
float get_y(const Point& p);
//Funkcija vraca naziv tacke p
const std::string& get_label(const Point& p);

//Setters

//Funkcija postavlja x-koordinatu tacke p
void set_x(Point& p, float x);
//Funkcija postavlja y-koordinatu tacke p
void set_y(Point& p, float y);
```

Menjamo samo deklaracijo strukture, ali ne i pristupnih interfejsa!

Implementacije funkcija

```
float get_x(const Point& p)
{
    return p.r * cos(p.t);
}

float get_y(const Point& p)
{
    return p.r * sin(p.t);
}

const std::string& get_label(const Point& p)
{
    return p.label;
}

void set_x(Point& p, float x)
{
    float y = p.r * sin(p.t);
    p.r = hypot(x, y);
    p.t = atan2(y, x);
}

void set_y(Point& p, float y)
{
    float x = p.r * cos(p.t);
    p.r = hypot(x, y);
    p.t = atan2(y, x);
}
```

Implementacija funkcija za pristup koordinatama se menja, ali ostatak koda to ne zanima

Pozivi funkcija

```
float x;
float y;
string label;
for(unsigned i = 0; i < cnt; ++i)
{
    cout << "Unesite ime tacke " << i << ": ";
    getline(cin >> ws, label);
    set_label(points[i], label);

    cout << "Unesite x koordinatu tacke " << i << ": ";
    cin >> x;
    set_x(points[i], x);

    cout << "Unesite y koordinatu tacke " << i << ": ";
    cin >> y;
    set_y(points[i], y);
}

struct Point c;

cout << "Unesite x koordinatu tacke c: ";
cin >> x;
set_x(c, x);

cout << "Unesite y koordinatu tacke c: ";
cin >> y;
set_y(c, y);
```

Pošto su deklaracije interfejsa ostale nepromenjene, ni jedan od poziva se ne menja

Dakle, korisnik (to jest preostali kod)

Ni ne zna da se nešto drastično promenilo u načinu na koji skladištimo podatke

Podsetnik sa prošlog časa

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
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 }
32
33 int main()
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

Podsetnik sa prošlog časa

Na prošlom času smo videli kako podela programa na potprograme omogućuje modularne izmene načina na koji vršimo operacije nad podacima

Podsetnik sa prošlog časa

Sada smo videli kako možemo da vršimo modularne izmene načina na koji organizujemo podatke

Ostalo je samo još da povežemo ta dva principa modularizacije

Pridruživanje funkcija podacima

Posmatrajmo opet deklarisanje interfejsa

```
#ifndef POINT_H
#define POINT_H

#include <string>
#include <cmath>

typedef struct Point
{
    float r = 1.0;
    float t = 1.0;

    std::string label;
} Point;

//Getters

//Funkcija vraca x-koordinatu tacke p
float get_x(const Point& p);
//Funkcija vraca y-koordinatu tacke p
float get_y(const Point& p);
//Funkcija vraca naziv tacke p
const std::string& get_label(const Point& p);

//Setters

//Funkcija postavlja x-koordinatu tacke p
void set_x(Point& p, float x);
//Funkcija postavlja y-koordinatu tacke p
void set_y(Point& p, float y);
```

Zamislamo da imamo nekoliko složenijih struktura, pa da i interfejsa ima mnogo više

Korisnik bi se tada suočio sa sledećim problemom

Kako da osiguram da uvek pozivam dobar interfejs za pristup odgovarajućoj strukturi?

Dobro odabrani nazivi, ažurni i dobro formulisani komentari i pažnja prilikom pisanja koda jako mnogo pomažu da ovaj problem bude izbegnut

Ako bismo mogli da pozivamo odgovarajuću funkciju **iz** samog objekta, to bi nam značajno olakšalo razvoj i smanjilo mogućnost uvođenja greške

Pokazivači na funkcije

U C/C++-u sve što zauzima memoriju ima adresu

Pa se tako može smestiti u pokazivač odgovarajućeg tipa

Ni funkcije nisu izuzetak

Deklaracija pokazivača na funkciju

povratni tip (* identifikator)(parametri);

Primer upotrebe

```
#ifndef SORT_H_
#define SORT_H_

//Funkcija sortira niz koriscenjem Insertion Sort algoritma
void insertion_sort(float a[], unsigned len);

//Funkcija sortira niz koriscenjem Shell Sort algoritma
void shell_sort(float a[], unsigned len);

#endif
~
```

Primer upotrebe

```
#include "sort.h"

void insertion_sort(float a[], unsigned len)
{
    for(unsigned i = 1; i < len; ++i)
    {
        float val = a[i];
        unsigned j = i;
        while((j > 0) && (val < a[j - 1]))
        {
            a[j] = a[j - 1];
            --j;
        }
        a[j] = val;
    }
}

void shell_sort(float a[], unsigned len)
{
    unsigned gap = 1;
    do
    {
        gap *= 3;
        ++gap;
    } while(gap <= len);
```

Primer upotrebe

```
#include <iostream>
#include "sort.h"
#define NUMPTS 25
#define INS_BETTER_THAN_SHELL 8

using namespace std;

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

    if(len % 2)
        return a[len / 2];

    return 0.5 * (a[len / 2] + a[len / 2 + 1]);
}
```

Primer upotrebe

```
int main()
{
    float xs[NUM_PTS] = {7.7857203028720035, 7.642458246115663, 6.215313359871936, 7.059634724385643, 6.670815698546658, 6.876282
852217749, 7.036072823998538, 7.290710191791188, 7.639716393843683, 7.752535310413255, 6.1089490156447095, 7.6043411248489505, 7.4182
63501562334, 6.2496483487612196, 6.8065125974698395, 7.434551692906639, 6.51122811120167, 7.198011826278233, 7.874699069289809, 6.507
268194942792, 12, 9, 12, 11, 10};

    float ys[NUM_PTS] = {6.663959610602355, 6.0833932514505, 7.1901041284124805, 6.837614857113309, 7.24503886439218, 7.471764212
69738, 7.157717200452314, 7.980448542746833, 6.826401869632361, 7.647518865141043, 7.437274473128388, 7.472813287622878, 7.0818737086
05056, 7.915294590939337, 6.43390232064712, 7.988414876843875, 7.3426188638785606, 7.434664291111027, 6.703619537079404, 6.8049450180
73546, 14, 9, 7, 11, 13};

    if(NUM_PTS < INS_BETTER_THAN_SHELL)
        cout << "centar = (" << median(xs, NUM_PTS, insertion_sort) << ", " << median(ys, NUM_PTS, insertion_sort) << ")\n";
    else
        cout << "centar = (" << median(xs, NUM_PTS, shell_sort) << ", " << median(ys, NUM_PTS, shell_sort) << ")\n";

    return 0;
}
```

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

Primer upotrebe

```
#include "sort.h"
#include <iostream>

void insertion_sort(float a[], unsigned len)
{
    std::cout << "Pozvan je insertion sort.\n";
    for(unsigned i = 1; i < len; ++i)
    {
        float val = a[i];
        unsigned j = i;
        while((j > 0) && (val < a[j - 1]))
        {
            a[j] = a[j - 1];
            --j;
        }
        a[j] = val;
    }
}

void shell_sort(float a[], unsigned len)
{
    std::cout << "Pozvan je shell sort.\n";
    unsigned gap = 1;
    do
    {
```



```
centar = (Pozvan je shell sort.  
7.41826, Pozvan je shell sort.  
7.34262)
```

Primer upotrebe

```
#include <iostream>
#include "sort.h"
#define NUM_PTS 25
#define INS_BETTER_THAN_SHELL 30

using namespace std;
```

```
centar = (Pozvan je insertion sort.  
7.41826, Pozvan je insertion sort.  
7.34262)
```

I struktura može sadržati pokazivač na funkciju

```
#ifndef POINT_H
#define POINT_H

#include <string>
#include <cmath>

typedef struct Point
{
    float r = 1.0;
    float t = 1.0;

    std::string label;

    //Getters
    float (* get_x)(const Point& p);
    float (* get_y)(const Point& p);
    const std::string& (* get_label)(const Point& p);

    //Setters
    void (* set_x)(Point& p, float x);
    void (* set_y)(Point& p, float y);
    void (* set_label)(Point& p, const std::string& label);
} Point;
```

Kako to koristimo?

```
float x;
float y;
string label;
for(unsigned i = 0; i < cnt; ++i)
{
    points[i].get_x = get_x;
    points[i].get_y = get_y;
    points[i].get_label = get_label;
    points[i].set_x = set_x;
    points[i].set_y = set_y;
    points[i].set_label = set_label;

    cout << "Unesite ime tacke " << i << ": ";
    getline(cin >> ws, label);
    points[i].set_label(points[i], label);

    cout << "Unesite x koordinatu tacke " << i << ": ";
    cin >> x;
    points[i].set_x(points[i], x);

    cout << "Unesite y koordinatu tacke " << i << ": ";
    cin >> y;
    points[i].set_y(points[i], y);
}
```

Kako to koristimo?

```
struct Point c;  
c.get_x = get_x;  
c.get_y = get_y;  
c.get_label = get_label;  
c.set_x = set_x;  
c.set_y = set_y;  
c.set_label = set_label;
```

```
cout << "Unesite x koordinatu tacke c: ";  
cin >> x;  
c.set_x(c, x);
```

```
cout << "Unesite y koordinatu tacke c: ";  
cin >> y;  
c.set_y(c, y);
```

```
cout << "Unesite maksimalnu Cebisevljevu razdaljinu: ";  
unsigned max_cheby;  
cin >> max_cheby;
```

Kako to koristimo?

```
for(unsigned i = 0; i < cnt; ++i)
{
    if(cheby_dist(points[i], c) <= max_cheby)
        cout << "tacka " << points[i].get_label(points[i])
                << " = (" << points[i].get_x(points[i]) << ", " << points[i].get_y(points[i]) << ")\n";
}
return 0;
}
```

Kako to koristimo?

```
Unesite broj tacaka u P: 6
Unesite ime tacke 0: Novi Sad
Unesite x koordinatu tacke 0: 13
Unesite y koordinatu tacke 0: 4
Unesite ime tacke 1: Sombor
Unesite x koordinatu tacke 1: 12
Unesite y koordinatu tacke 1: 9
Unesite ime tacke 2: Becej
Unesite x koordinatu tacke 2: 9
Unesite y koordinatu tacke 2: 7
Unesite ime tacke 3: Sabac
Unesite x koordinatu tacke 3: 7
Unesite y koordinatu tacke 3: 1
Unesite ime tacke 4: Donji Milanovac
Unesite x koordinatu tacke 4: 22
Unesite y koordinatu tacke 4: 2
Unesite ime tacke 5: Bujanovac
Unesite x koordinatu tacke 5: 18
Unesite y koordinatu tacke 5: -7
Unesite x koordinatu tacke c: 18
Unesite y koordinatu tacke c: 3
Unesite maksimalnu Cebisevljevu razdaljinu: 10
tacka Novi Sad = (13, 4)
tacka Sombor = (12, 9)
tacka Becej = (9, 7)
tacka Donji Milanovac = (22, 2)
tacka Bujanovac = (18, -7)
```


Postigli smo da pozivamo funkcije pridružene konkretnom objektu

```
float x;
float y;
string label;
for(unsigned i = 0; i < cnt; ++i)
{
    points[i].get_x = get_x;
    points[i].get_y = get_y;
    points[i].get_label = get_label;
    points[i].set_x = set_x;
    points[i].set_y = set_y;
    points[i].set_label = set_label;

    cout << "Unesite ime tacke " << i << ": ";
    getline(cin >> ws, label);
    points[i].set_label(points[i], label);

    cout << "Unesite x koordinatu tacke " << i << ": ";
    cin >> x;
    points[i].set_x(points[i], x);

    cout << "Unesite y koordinatu tacke " << i << ": ";
    cin >> y;
    points[i].set_y(points[i], y);
}
```

Ali još uvek moramo ručno da pridružujemo funkcije svakom novom objektu

```
float x;
float y;
string label;
for(unsigned i = 0; i < cnt; ++i)
{
    points[i].get_x = get_x;
    points[i].get_y = get_y;
    points[i].get_label = get_label;
    points[i].set_x = set_x;
    points[i].set_y = set_y;
    points[i].set_label = set_label;

    cout << "Unesite ime tacke " << i << ": ";
    getline(cin >> ws, label);
    points[i].set_label(points[i], label);

    cout << "Unesite x koordinatu tacke " << i << ": ";
    cin >> x;
    points[i].set_x(points[i], x);

    cout << "Unesite y koordinatu tacke " << i << ": ";
    cin >> y;
    points[i].set_y(points[i], y);
}
```

Ali još uvek moramo ručno da pridružujemo funkcije svakom novom objektu

To znači da korisnik još uvek mora da zna koje funkcije treba da poziva i mora da vodi računa da to ispravno obavi

Potrebna nam je još jedna funkcija koju piše programer koji je napisao strukturu i pristupne funkcije, tako da korisnik ne mora da poznaje internu implementaciju, a koja će obezbediti automatsko povezivanje funkcija sa svakim novokreiranim objektom

Konstruktori (lagani uvod)

Šta je konstruktor?

Konstruktor je funkcija koja rezerviše potreban prostor za novi objekat i priprema ga za upotrebu

U našem primeru

```
#include <string>
#include <cmath>

typedef struct Point
{
    float r;
    float t;

    std::string label;

    //Getters
    float (* get_x)(const Point& p);
    float (* get_y)(const Point& p);
    const std::string& (* get_label)(const Point& p);

    //Setters
    void (* set_x)(Point& p, float x);
    void (* set_y)(Point& p, float y);
    void (* set_label)(Point& p, const std::string& label);
} Point;

//Constructor
Point* newPoint();
```

U našem primeru

```
#include <algorithm>
#include "point.h"

Point* newPoint()
{
    Point* this_p = (Point*)(malloc(sizeof(Point)));

    //this_p -> r je isto sto i (*this_p).r
    this_p -> r = 1.0;
    this_p -> t = 1.0;

    this_p -> get_x = get_x;
    this_p -> get_y = get_y;
    this_p -> get_label = get_label;
    this_p -> set_x = set_x;
    this_p -> set_y = set_y;
    this_p -> set_label = set_label;

    return this_p;
}

float get_x(const Point& p)
{
    return p.r * cos(p.t);
}
```


U našem primeru

```
float x;  
float y;  
string label;  
for(unsigned i = 0; i < cnt; ++i)  
{
```

```
    points[i].get_x = get_x;  
    points[i].get_y = get_y;  
    points[i].get_label = get_label;  
    points[i].set_x = set_x;  
    points[i].set_y = set_y;  
    points[i].set_label = set_label;
```

```
    cout << "Unesite ime tacke " << i << ": ";  
    getline(cin >> ws, label);  
    points[i].set_label(points[i], label);  
  
    cout << "Unesite x koordinatu tacke " << i << ": ";  
    cin >> x;  
    points[i].set_x(points[i], x);  
  
    cout << "Unesite y koordinatu tacke " << i << ": ";  
    cin >> y;  
    points[i].set_y(points[i], y);  
}
```

Ovo je isto

```
#include <algorithm>  
#include "point.h"
```

```
Point* newPoint()  
{  
    Point* this_p = (Point*)(malloc(sizeof(Point)));  
  
    //this_p -> r je isto sto i (*this_p).r  
    this_p -> r = 1.0;  
    this_p -> t = 1.0;
```

```
    this_p -> get_x = get_x;  
    this_p -> get_y = get_y;  
    this_p -> get_label = get_label;  
    this_p -> set_x = set_x;  
    this_p -> set_y = set_y;  
    this_p -> set_label = set_label;
```

```
    return this_p;  
}
```

```
float get_x(const Point& p)  
{  
    return p.r * cos(p.t);  
}
```

ali se sada automatski
poziva pri kreiranju objekta

U našem primeru

```
#include <iostream>
#include <string>
#include "point.h"
```

```
#define MAX_CNT 100
```

```
using namespace std;
```

```
int main()
{
```

```
    Point* points[MAX_CNT];
    unsigned cnt = 0;
```

newPoint vraća pokazivač
na novi objekat, pa ćemo i mi
čuvati pokazivače

U našem primeru

```
float x;  
float y;  
string label;  
for(unsigned i = 0; i < cnt; ++i)  
{  
    points[i] = newPoint();  
  
    cout << "Unesite ime tacke " << i << ": ";  
    getline(cin >> ws, label);  
    points[i] -> set_label(*points[i], label);  
  
    cout << "Unesite x koordinatu tacke " << i << ": ";  
    cin >> x;  
    points[i] -> set_x(*points[i], x);  
  
    cout << "Unesite y koordinatu tacke " << i << ": ";  
    cin >> y;  
    points[i] -> set_y(*points[i], y);  
}
```

Za svaki novi objekat, samo pozovemo konstruktor i on će sve odraditi za nas

U našem primeru

```
float x;  
float y;  
string label;  
for(unsigned i = 0; i < cnt; ++i)  
{  
    points[i] = newPoint();  
  
    cout << "Unesite ime tacke " << i << ": ";  
    getline(cin >> ws, label);  
    points[i] -> set_label(*points[i], label);  
  
    cout << "Unesite x koordinatu tacke " << i << ": ";  
    cin >> x;  
    points[i] -> set_x(*points[i], x);  
  
    cout << "Unesite y koordinatu tacke " << i << ": ";  
    cin >> y;  
    points[i] -> set_y(*points[i], y);  
}
```

No, moramo ispoštovati to gde se očekuje referenca a gde pokazivač (u ovom slučaju, mi smo pisali te funkcije, pa ih možemo i izmeniti ako želimo; u opštem slučaju ne)

U našem primeru

```
Point* c = newPoint();  
  
cout << "Unesite x koordinatu tacke c: ";  
cin >> x;  
c -> set_x(*c, x);  
  
cout << "Unesite y koordinatu tacke c: ";  
cin >> y;  
c -> set_y(*c, y);
```

I za c sve
isto radimo

U našem primeru

```
Unesite broj tacaka u P: 6
Unesite ime tacke 0: Novi Sad
Unesite x koordinatu tacke 0: 13
Unesite y koordinatu tacke 0: 4
Unesite ime tacke 1: Sombor
Unesite x koordinatu tacke 1: 12
Unesite y koordinatu tacke 1: 9
Unesite ime tacke 2: Becej
Unesite x koordinatu tacke 2: 9
Unesite y koordinatu tacke 2: 7
Unesite ime tacke 3: Sabac
Unesite x koordinatu tacke 3: 7
Unesite y koordinatu tacke 3: 1
Unesite ime tacke 4: Donji Milanovac
Unesite x koordinatu tacke 4: 22
Unesite y koordinatu tacke 4: 2
Unesite ime tacke 5: Bujanovac
Unesite x koordinatu tacke 5: 18
Unesite y koordinatu tacke 5: -7
Unesite x koordinatu tacke c: 18
Unesite y koordinatu tacke c: 3
Unesite maksimalnu Cebisevljevu razdaljinu: 10
tacka Novi Sad = (13, 4)
tacka Sombor = (12, 9)
tacka Becej = (9, 7)
tacka Donji Milanovac = (22, 2)
tacka Bujanovac = (18, -7)
```

Naravno, konstruktor može i da prima parametre

Te da kreirani objekat inicijalizuje na drugačiji način pomoću njih

O tome više sledeće nedelje

Šta smo zaboravili?

Da vidimo šta valgrind kaže o upotrebi memorije

```
Unesite maksimalnu Cebisevljevu razdaljinu: 10
tacka Novi Sad = (13, 4)
tacka Sombor = (12, 9)
tacka Becej = (9, 7)
tacka Donji Milanovac = (22, 2)
tacka Bujanovac = (18, -7)
==24490==
==24490== HEAP SUMMARY:
==24490==      in use at exit: 670 bytes in 13 blocks
Press ENTER or type command to continue
```

Negde imamo curenje...

```

#include <algorithm>
#include "point.h"

Point* newPoint()
{
    Point* this_p = (Point*)(malloc(sizeof(Point)));

    //this_p -> r je isto sto i (*this_p).r
    this_p -> r = 1.0;
    this_p -> t = 1.0;

    this_p -> get_x = get_x;
    this_p -> get_y = get_y;
    this_p -> get_label = get_label;
    this_p -> set_x = set_x;
    this_p -> set_y = set_y;
    this_p -> set_label = set_label;

    return this_p;
}

float get_x(const Point& p)
{
    return p.r * cos(p.t);
}

```

Pri svakom pozivu konstruktora, memorija je rezervisana, ali nikada nije oslobođena

Destruktor

Destruktor je funkcija inverzna konstruktoru

Njen zadatak je da oslobodi svu memoriju rezervisanu za neki objekat, kada ga više ne koristimo

U našem primeru

```
#ifndef POINT_H
#define POINT_H

#include <string>
#include <cmath>

typedef struct Point
{
    float r;
    float t;

    std::string label;

    //Getters
    float (* get_x)(const Point& p);
    float (* get_y)(const Point& p);
    const std::string& (* get_label)(const Point& p);

    //Setters
    void (* set_x)(Point& p, float x);
    void (* set_y)(Point& p, float y);
    void (* set_label)(Point& p, const std::string& label);
} Point;

//Constructor
Point* newPoint();

//Destructor
void delPoint(Point* this_p);
```

U našem primeru

```
#include <algorithm>
#include "point.h"

Point* newPoint()
{
    Point* this_p = (Point*)(malloc(sizeof(Point)));

    //this_p -> r je isto sto i (*this_p).r
    this_p -> r = 1.0;
    this_p -> t = 1.0;

    this_p -> get_x = get_x;
    this_p -> get_y = get_y;
    this_p -> get_label = get_label;
    this_p -> set_x = set_x;
    this_p -> set_y = set_y;
    this_p -> set_label = set_label;

    return this_p;
}

void delPoint(Point* this_p)
{
    free(this_p);
}
```

U našem primeru

```
for(unsigned i = 0; i < cnt; ++i)
{
    if(cheby_dist(*points[i], *c) <= max_cheby)
        cout << "tacka " << points[i] -> get_label(*points[i])
            << " = (" << points[i] -> get_x(*points[i]) << ", " << points[i] -> get_y(*points[i]) << ")\n";

    delPoint(points[i]);
    points[i] = NULL;
}

delPoint(c);

return 0;
}
```

U našem primeru

```
==25193== LEAK SUMMARY:
==25193==    definitely lost: 54 bytes in 6 blocks
==25193==    indirectly lost: 0 bytes in 0 blocks
==25193==    possibly lost: 0 bytes in 0 blocks
==25193==    still reachable: 0 bytes in 0 blocks
==25193==    suppressed: 0 bytes in 0 blocks
==25193==
==25193== Use --track-origins=yes to see where uninitialised values come from
==25193== For lists of detected and suppressed errors, rerun with: -s
==25193== ERROR SUMMARY: 31 errors from 6 contexts (suppressed: 0 from 0)
```

U našem primeru

free neće pozvati destruktorklase `std::string` nad alociranim stringovima pa ćemo ih izgubiti

Trebalo bi da pozovemo operator **delete** umesto free-a, koji će to uraditi

Međutim, delete ne ide uz malloc, već uz operator **new**

Kadgod koristimo instance klasa, koristimo new i delete umesto malloc i free

U našem primeru

```
#include <algorithm>
#include "point.h"

Point* newPoint()
{
    Point* this_p = new Point;

    //this_p -> r je isto sto i (*this_p).r
    this_p -> r = 1.0;
    this_p -> t = 1.0;

    this_p -> get_x = get_x;
    this_p -> get_y = get_y;
    this_p -> get_label = get_label;
    this_p -> set_x = set_x;
    this_p -> set_y = set_y;
    this_p -> set_label = set_label;

    return this_p;
}

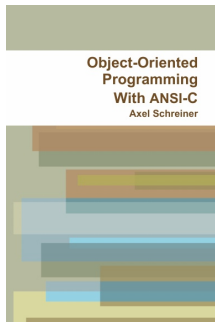
void delPoint(Point* this_p)
{
    delete this_p;
}
```

U našem primeru

```
==25650==  
==25650== HEAP SUMMARY:  
==25650==      in use at exit: 0 bytes in 0 blocks  
==25650==    total heap usage: 24 allocs, 24 frees, 76,166 bytes allocated  
==25650==  
==25650== All heap blocks were freed -- no leaks are possible  
==25650==  
==25650== For lists of detected and suppressed errors, rerun with: -s  
==25650== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)  
  
real    0m42.434s  
user    0m0.596s  
sys     0m0.032s
```

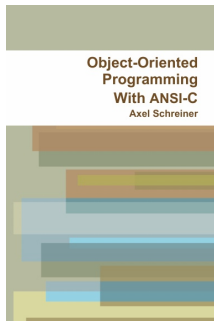
I tako smo stigli do ključnih koncepata objektno-orijentisanog programiranja

Čekaj, a gde su te čuvene klase?



Ljudi su godinama pre rasta popularnosti C++-a koristili principe objektno-orijentisanog programiranja (OOP-a) u klasičnom C-u

Čekaj, a gde su te čuvene klase?



Knjiga iznad opisuje jedan odličan pristup tome. Linux kernel i danas na mnogim mestima koristi C sa principima OOP-a

Principi su dakelo bitniji od jezika

Ako nam je jasno zašto je nekad bitno da naš kod organizujemo na određeni način, često ćemo naći rešenje koje će nam omogućiti da to uradimo upotrebom jezika koji već poznajemo

Ako ne poznajemo principe i nije nam jasno koja je korist od određene organizacije koda u određenoj situaciji, nikada nećemo moći da ispravno iskoristimo mogućnosti koje nam dati jezik pruža

Naravno, neki elementi jezika nam značajno olakšavaju razvoj i održavanje koda

Klase sa 10.000 stopa

Da vidimo kako nam to klase olakšavaju život

```
#ifndef POINT_H
#define POINT_H

#include <string>

class Point
{
    private:
        float x;
        float y;

        std::string label;

    public:
        //Getters
        float get_x() const;
        float get_y() const;
        const std::string& get_label() const;

        //Setters
        void set_x(float x);
        void set_y(float y);
        void set_label(const std::string& label);

        //Other members
        float cheby_dist(const Point& center);
};

#endif
```

Privatnom segmentu mogu pristupiti samo funkcije koje su deo klase

Tu su obično sadržani svi podaci (setimo se apstrakcije podataka)

Da vidimo kako nam to klase olakšavaju život

```
#ifndef POINT_H
#define POINT_H

#include <string>

class Point
{
    private:
        float x;
        float y;

        std::string label;

    public:
        //Getters
        float get_x() const;
        float get_y() const;
        const std::string& get_label() const;

        //Setters
        void set_x(float x);
        void set_y(float y);
        void set_label(const std::string& label);

        //Other members
        float cheby_dist(const Point& center);
};

#endif
```

U javnom segmentu su polja kojima je moguće pristupiti i izvan klase

Tu se najmanje sadrže get i set metode, ali i sve druge koje predstavljaju interfejs klase i korisnika

Da vidimo kako nam to klase olakšavaju život

```
#include <algorithm>
#include "point.h"

float Point::get_x() const
{
    return this -> x;
}

float Point::get_y() const
{
    return this -> y;
}

const std::string& Point::get_label() const
{
    return label;
}

void Point::set_x(float x)
{
    this -> x = x;
}

void Point::set_y(float y)
{
    this -> y = y;
}
```

Definicije funkcija članova
pridružujemo klasi pomoću
identifikator klase::identifikator funkcije

Da vidimo kako nam to klase olakšavaju život

```
#include <algorithm>
#include "point.h"

float Point::get_x() const
{
    return this -> x;
}

float Point::get_y() const
{
    return this -> y;
}

const std::string& Point::get_label() const
{
    return label;
}

void Point::set_x(float x)
{
    this -> x = x;
}

void Point::set_y(float y)
{
    this -> y = y;
}
```

this je pokazivač na objekat iz kog se funkcija poziva i uvek je dostupan

sličan je kao **self** u Python-u samo u C++-u ne navodimo objekat kom je funkcija pridružena kao prvi parametar funkcija

Da vidimo kako nam to klase olakšavaju život

```
#include <algorithm>
#include "point.h"

float Point::get_x() const
{
    return this -> x;
}

float Point::get_y() const
{
    return this -> y;
}

const std::string& Point::get_label() const
{
    return label;
}

void Point::set_x(float x)
{
    this -> x = x;
}
```

Ovde smo mogli da koristimo
i direktno polje x

(return x;)

Da vidimo kako nam to klase olakšavaju život

```
#include <algorithm>
#include "point.h"

float Point::get_x() const
{
    return this -> x;
}

float Point::get_y() const
{
    return this -> y;
}

const std::string& Point::get_label() const
{
    return label;
}

void Point::set_x(float x)
{
    this -> x = x;
}
```

Ali ovde nismo mogli
da koristimo $x = x$;

Za domaći proučiti zašto i šta
su moguća druga rešenja

Da vidimo kako nam to klase olakšavaju život

```
void Point::set_label(const std::string& label)
{
    this -> label = label;
}

float Point::cheby_dist(const Point& center)
{
    return std::max(abs(x - center.get_x()), abs((y - center.get_y())));
}
```


Da vidimo kako nam to klase olakšavaju život

```
#include <iostream>
#include "point.h"

#define MAX_CNT 100

using namespace std;

int main()
{
    Point* points[MAX_CNT];
    unsigned cnt = 0;
```

Da vidimo kako nam to klase olakšavaju život

```
float x;  
float y;  
string label;  
for(unsigned i = 0; i < cnt; ++i)  
{  
    points[i] = new Point;  
  
    cout << "Unesite naziv tacke: ";  
    getline(cin >> ws, label);  
    points[i] -> set_label(label);  
  
    cout << "Unesite x koordinatu tacke: ";  
    cin >> x;  
    points[i] -> set_x(x);  
  
    cout << "Unesite y koordinatu tacke: ";  
    cin >> y;  
    points[i] -> set_y(y);  
}
```

Da vidimo kako nam to klase olakšavaju život

```
Point center;  
  
cout << "Unesite x koordinatu centra: ";  
cin >> x;  
center.set_x(x);  
  
cout << "Unesite y koordinatu centra: ";  
cin >> y;  
center.set_y(y);  
  
cout << "Unesite maksimalnu Cebisevljevu razdaljinu: ";  
unsigned max_cheby;  
cin >> max_cheby;
```

Da vidimo kako nam to klase olakšavaju život

```
cout << "Unesite maksimalnu Cebisevljevu razdaljinu: ";
unsigned max_cheby;
cin >> max_cheby;

for(unsigned i = 0; i < cnt; ++i)
{
    if(points[i] -> cheby_dist(center) <= max_cheby)
        cout << points[i] -> get_label() << " = (" << points[i] -> get_x() << ", " << points[i] -> get_y() << ")\n";

    delete points[i];
}

return 0;
```

Da vidimo kako nam to klase olakšavaju život

```
Unesite broj tacaka u P: 6
Unesite naziv tacke: Novi Sad
Unesite x koordinatu tacke: 13
Unesite y koordinatu tacke: 4
Unesite naziv tacke: Sombor
Unesite x koordinatu tacke: 12
Unesite y koordinatu tacke: 9
Unesite naziv tacke: Becej
Unesite x koordinatu tacke: 9
Unesite y koordinatu tacke: 7
Unesite naziv tacke: Sabac
Unesite x koordinatu tacke: 7
Unesite y koordinatu tacke: 1
Unesite naziv tacke: Donji Milanovac
Unesite x koordinatu tacke: 22
Unesite y koordinatu tacke: 2
Unesite naziv tacke: Bujanovac
Unesite x koordinatu tacke: 18
Unesite y koordinatu tacke: -7
Unesite x koordinatu centra: 18
Unesite y koordinatu centra: 3
Unesite maksimalnu Cebisevljevu razdaljinu: 10
Novi Sad = (13, 4)
Sombor = (12, 9)
Becej = (9, 7)
Donji Milanovac = (22, 2)
Bujanovac = (18, -7)
==29555==
==29555== HEAP SUMMARY:
==29555==      in use at exit: 0 bytes in 0 blocks
==29555==    total heap usage: 23 allocs, 23 frees, 75,790 bytes allocated
==29555==
==29555== All heap blocks were freed -- no leaks are possible
```

To bi bilo sve za danas

Sledeće nedelje zalazimo dublje u konstruktore, destruktore, kompoziciju klasa, nasleđivanje...

Strukture nećemo više eksplicitno obrađivati

No, to ne znači da smo rekli sve o njima što je bitno

Za domaći proučiti šta su *povezane liste* (eng. *linked lists*), čemu služe, kako se implementiraju i kako se koriste

Takođe, videti šta su *unije* i *enumeracije* (eng. *union* i *enumeration*)

Hvala na pažnji