ALGEBRA

# STRUKTURE PODATAKA I ALGORITMI

Predavanje 02

Ishod 1

1

## Ponavljanje

- Napisati program koji definira kompleksni broj te omogućuje njegov pravilan ispis, množenje sa skalarom te zbrajanje s drugim kompleksnim brojem. U glavnom programu demonstrirajte rad svih operacija.
  - Pitanja koja si moramo postaviti:
    - Hoćemo li odabrati strukturu ili klasu?
    - Koji članovi će biti privatni, a koji javni?
    - Ako su nam varijable privatne, kako ćemo ih postaviti?



ALGEBRA

3

# Rješenje

```
#pragma once
class KompleksniBroj {
private:
    int x;
    int y;
public:
    void inicijaliziraj(int px, int py);
    void mnozi(int n);
    void zbroji(KompleksniBroj k);
    void ispisi();
};
```

KompleksniBroj.h

```
#include "KompleksniBroj.h"

int main() {
    KompleksniBroj k1;
    k1.inicijaliziraj(5, 5);
    k1.mnozi(2);
    k1.ispisi();

    KompleksniBroj k2;
    k2.inicijaliziraj(5, 2);

    k1.zbroji(k2);
    k1.ispisi();
    return 0;
}
```

Source.cpp

```
#include <iostream>
#include "KompleksniBroj.h"
using namespace std;

void KompleksniBroj::inicijaliziraj(int px, int py) {
    x = px;
    y = py;
}
void KompleksniBroj::mnozi(int n) {
    x *= n;
    y *= n;
}
void KompleksniBroj::zbroji(KompleksniBroj drugi) {
    x += drugi.x;
    y += drugi.y;
}
void KompleksniBroj::ispisi() {
    cout << x << " + " << y << "i" << endl;
}
```

KompleksniBroj.cpp



ALGEBRA

4

# KONSTRUKTORI I DESTRUKTORI



ALGEBRA

5

## Preopterećenje metode (više na OOP)

- Svaka metoda može imati više verzija
  - Može biti preopterećena (engl. *overloaded*)
  - Moraju se razlikovati po broju i/ili tipovima parametara
  - Nazivi parametara i povratna vrijednost nisu važni
- Probajmo u Visual Studiju pozvati sve tri metode – kako kompajler zna koju želimo pozvati?

```
void ispisi(int a) {
      cout << "A: " << a << endl;
}
void ispisi(int a, int b) {
      cout << "B: " << a << " " << b << endl;
}
void ispisi(double a, double b) {
      cout << "C: " << a << " " << b << endl;
}
```






## Problem

- Pokrenimo sljedeći kôd:

```
int main() {
      KompleksniBroj k1;
      k1.ispisi();

      return 0;
}
```

- Zašto dobijemo takav rezultat?
- Ima li smisla postojanje kompleksnog broja kojemu nisu definirani *x* i *y*?
- Možemo li kako spriječiti korištenje takvih neinicijaliziranih kompleksnih brojeva?



## Konstruktor

- Svaka struktura i klasa imaju jedan ili više konstruktora (engl. *constructor*)
  - Konstruktor je metoda koja se automatski poziva prilikom izrade (konstruiranja) objekta
    - Konstruiranje objekta na stogu:

      ```
      KompleksniBroj k;
      ```

    - Konstruiranje objekta na hrpi:

      ```
      KompleksniBroj* k1 = new KompleksniBroj;
      ```

  - Konstruktor se najčešće koristi za postavljanje početnog stanja objekta
    - Npr. primi neke parametre i prekopira ih u članske varijable



ALGEBRA

8

## Dizajn konstruktora

- Naziv konstruktora uvijek mora biti jednak nazivu strukture ili klase
- Nema nikakvu povratnu vrijednost (nema čak niti `void`)
- Možemo ih definirati koliko god želimo
  - Preopterećenje konstruktora (jer i konstruktor je metoda)
- Konstruktor bez parametara se naziva *defaultni* konstruktor
  - Ako mi ne definiramo nijedan konstruktor, automatski se kreira od strane kompajlera
  - Ako mi kreiramo ijedan konstruktor, *defaultni* konstruktor neće biti automatski kreiran!
    - Ako nam treba, moramo ga mi kreirati



ALGEBRA

9

## Korištenje konstruktora

- Konstruktor se automatski poziva pri kreiranju objekta
- Ako struktura ili klasa ima definirano više konstruktora, parametri koje proslijedimo određuju koji će biti pozvan
  - Kao i kod poziva svake funkcije, parametri idu u zagradu
  - Iznimka: ako želimo koristiti *defaultni* konstruktor, ne smijemo pisati zagrade
- Koji konstruktor će biti pozvan:

```
KompleksniBroj k1;
KompleksniBroj k2(4);
KompleksniBroj k3(6, 8);
KompleksniBroj k4[5];
KompleksniBroj* k5 = new KompleksniBroj(4, 6);
```



ALGEBRA



## Pokazivač `this`

- U metodama strukture ili klase je uvijek dostupan poseban pokazivač `this`
  - Pokazuje na onaj objekt na kojem se metoda poziva
  - Više detalja: OOP
- Što bi se dogodilo da ne koristimo `this`:

```
class Kvadrat {
public:
      Kvadrat(int n) {
            this->n = n;
      }
      void ispisi() { cout << n << endl; }
private:
      int n;
};
```



ALGEBRA

## Dva kratka pitanja

- Što ne valja sa sljedeći kôdom:

```
class Pravokutnik {
public:
      Pravokutnik(int a) {}
};
int main() {
      Pravokutnik p;
      return 0;
}
```

- Omogućite kompajliranje sljedećeg kôda iz `main`-a:

```
Pravokutnik p1[10];
Pravokutnik p2(10, 3);
Pravokutnik* p3 = new Pravokutnik(12);
```






## Rješenje problema neinicijaliziranog objekta

- Možemo li spriječiti korištenje neinicijaliziranih brojeva?
  - Da, pravilnim definiranjem konstruktora

```
KompleksniBroj::KompleksniBroj(int px, int py) {
    x = px;
    y = py;
}
…

int main() {
    KompleksniBroj k1(5, 5);
    k1.mnozi(2);
    k1.ispisi();
    …
    return 0;
}
```

Jedini način da netko napravi objekt tipa `KompleksniBroj` je da mu definira i x i y



## Destruktor

- Svaka struktura i klasa može imati destruktor
  - Metoda koja se poziva prilikom uništenja objekta
    - Završetkom funkcije (za objekte na stogu)
    - Pozivom `delete` (za objekte na hrpi)
  - Naziv jednak nazivu strukture ili klase s tildom ispred, nema povratne vrijednosti, nema parametara

```
class Pravokutnik {
public:
    ~Pravokutnik() {
        cout << "Destruktor je pozvan" << endl;
    }
};
```

  - Koristi se za otpuštanje resursa (primjerice, ako u konstruktor ide `new`, onda u destruktor ide `delete`)



ALGEBRA

14

## Pitanje

- Što će ispisati sljedeći kôd:

```
class Kvadrat {
public:
      Kvadrat(int n) {
            cout << "Konstruktor, n=" << n << endl;
            this->n = n;
      }
      ~Kvadrat() { cout << "Destruktor, n=" << n << endl; }
private:
      int n;
};
int main() {
      Kvadrat k1(4);
      Kvadrat* k2 = new Kvadrat(5);
      delete k2;
      Kvadrat k3[2] { Kvadrat(6), Kvadrat(7) };
      return 0;
}
```



ALGEBRA

15

# *OPCIONALNI NAČIN KORIŠTENJA KONSTRUKTORA



ALGEBRA

16

## Inicijalizacija polja

- Dva alternativna načina zapisivanja:

```
int a[] = { 10, 20, 30, 40, 50 };
int b[] { 10, 20, 30, 40, 50 };
```



ALGEBRA

17

## Inicijalizacija varijabli

- Standardni način:

```
int a = 10;
char b = 'M';
float c = 2.2f;
```

- C++11 specifični način
  - Preuzeto s inicijalizacije polja
  - Cilj: ujednačiti inicijalizacije varijabli, polja i objekata

```
int a{ 10 };
char b{ 'M' };
float c{ 2.2f };
```




18

## Inicijalizacija objekata

```
class Tocka {
private:
    int x;
    int y;
public:
    Tocka(); // Inicijalizira x i y na 0
    Tocka(int x, int y);
};
```

- Standardni način:

```
Tocka k1;
Tocka k2(5, 3);
```

- C++11 specifični način:

```
Tocka k1{};
Tocka k2{ 5, 3 };
```




19

# JOŠ NEKE C++ SITNICE



ALGEBRA



## Verzije C++-a

- C++ je programski jezik koji se kroz povijest proširivao novim funkcionalnostima
- Svaki kompajler zna raditi s određenim verzijama jezika
- Neke od verzija:
  - C++98
  - C++11
  - C++14
  - C++17
- Primjerice, na vektoru je metoda `push_back` prisutna od C++98, dok je metoda `shrink_to_fit` uvedena tek u C++11



ALGEBRA

## Pogreške

- Ponekad moramo iz metode javiti pozivatelju da se dogodila pogreška
  - Primjerice, nema datoteke koju želite otvoriti
- Postoje dva načina za to napraviti:
  - Tradicionalni, vraćanjem `true` za uspjeh ili `false` za neuspjeh
    - Alternativno, korištenjem cijelog broja
  - Moderni, korištenjem `try`/`catch` bloka
    - Metoda baci iznimku ako se dogodi pogreška
    - Pozivatelj uhvati iznimku i obradi je
    - Više detalja: OOP
- Na SPA neće biti naglasak na ovoj temi
  - Ako vam zatreba, možete koristiti bilo koji način






## Primjer tradicionalnog načina

```
bool podijeli(int a, int b, int& rezultat) {
    if (b == 0) {
        return false;
    }
    rezultat = a / b;
    return true;
}

int main() {
    int rezultat;
    bool uspjeh = podijeli(17, 0, rezultat);

    if (!uspjeh) { cout << "Dijeljenje s nulom" << endl; }
    else { cout << rezultat << endl; }

    return 0;
}
```



## Primjer modernog načina

```
int podijeli(int a, int b) {
    if (b == 0) {
        throw exception("Dijeljenje s nulom");
    }
    return a / b;
}

int main() {
    try {
        cout << podijeli(17, 0) << endl;
    }
    catch (const exception& err) {
        cout << err.what() << endl;
    }

    return 0;
}
```






## Klasa `stringstream`

- Do sada smo koristili sljedeće tijekove:
  - `cin`, `cout`, `ifstream`, `ofstream`
- Klasa `stringstream` predstavlja ulazno/izlazni tijek prema međuspremniku znakova u memoriji:
  - Uključimo zaglavlje: `#include <sstream>`
  - Napravimo objekt: `stringstream sstr;`
  - Upisujemo: `sstr << "XY" << 22 << endl;`
  - Čitamo:
    ```
    sstr >> broj;
    getline(sstr, ime);
    sstr.str();
    ```
  - Čistimo sadržaj:
    ```
    sstr.str("");
    sstr.clear();
    ```



## Primjene klase `stringstream`

- Klasa `stringstream` za nas ima dvije osnovne primjene:
  1. Spajanje više stringova u jedan veći string (konkatenacija)
  2. Pretvaranje tipova podataka (najčešće string => nešto)
     - Alternativa C-olikim funkcijama atoi, atof, …
- Riješimo sljedeće zadatke:

1. Napišimo funkciju koja prima tri stringa i vraća jedan spojeni string.
2. Napišimo funkciju koja prima string sa tri broja odvojena zarezima i vraća njihov zbroj.



ALGEBRA



## Rješenja

```
string spoji(string s1, string s2, string s3) {
    stringstream sstr;
    sstr << s1 << " " << s2 << " " << s3;
    return sstr.str();
}
int zbroji(string s) {
    stringstream sstr;
    sstr << s;

    int suma = 0;
    int n;
    while (sstr >> n) {
        suma += n;
    }
    return suma;
}
int main() {
    cout << spoji("ovo", "je", "test") << endl;
    cout << zbroji("49 1 10") << endl;
    return 0;
}
```



ALGEBRA

# UVOD U C++ STANDARD TEMPLATE LIBRARY (STL)



ALGEBRA

28

## Uvod

- STL biblioteka (engl. *Standard Template Library*) je skup:
  - Kontejnera (predložaka klasa)
    - Kontejner u sebi čuva više podataka nekog tipa
  - Algoritama/funkcija
  - Iteratora (predavanje 04)
- Svrha STL-a je pružiti programeru funkcionalnosti koje su mu svakodnevno potrebne
- Poznavanje STL-a je prvi ozbiljan korak u razvoju svakog C++ programera
  - cplusplus.com/reference/stl/



ALGEBRA

29

## Primjer STL kontejnera: array<T,N> (1/3)

- array<T,N> je samo lagani omotač oko običnog polja na stogu
- array<T,N> je generička klasa i pri izradi objekta te klase programer mora dati dva podatka:
  - T: tip podataka koji će se čuvati u arrayu
  - N: veličina polja koju treba rezervirati na stogu
- Primjerice, što će napraviti svaka od linija:

```
array<string, 50> p2;
array<Pravokutnik, 7> p3;
array<int, 5> p1 = { 11, 22, 33, 44, 55 };
```



ALGEBRA



## Primjer STL kontejnera: array<T,N> (2/3)

- Primjer korištenja:

```
array<int, 5> p = { 11, 22, 33, 44, 55 };
for (unsigned i = 0; i < p.size(); i++) {
    cout << p[i] << endl;
}
```

- Nije li jednostavnije koristiti obično polje?
  - Jest ☺
  - Međutim, klasa array nudi sučelje slično ostalim kontejnerskim klasama tako da je izmjena kontejnera lagana:

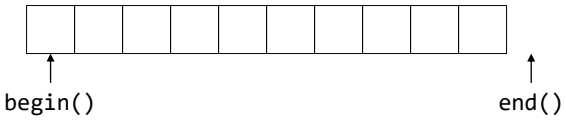
```
vector<int> p = { 11, 22, 33, 44, 55 };
for (unsigned i = 0; i < p.size(); i++) {
    cout << p[i] << endl;
}
```



ALGEBRA

## Primjer STL kontejnera: array<T,N> (3/3)

- array<T,N> nudi i dvije metode koje vraćaju „pokazivače”:
  - begin() vraća „pokazivač” na prvi element
  - end() vraća „pokazivač” na prvi element iza kraja




  - Korist: većina ostalih kontejnera nude jednake metode



ALGEBRA



## Primjer STL algoritama (1/3)

- Velik broj algoritama definiranih u zaglavlju <algorithm>
  - cplusplus.com/reference/algorithm/
- Pogledat ćemo sljedeće implementacije algoritama:
  - reverse(od, do) – preslaguje sve elemente u rasponu [od, do) od kraja prema početku

```
array<int, 5> p = { 11, 22, 33, 44, 55 };

reverse(p.begin(), p.end());

for (unsigned i = 0; i < p.size(); i++) {
    cout << p[i] << endl;
}
```



ALGEBRA

## Primjer STL algoritama (2/3)

- `count(od, do, val)` – vraća ukupan broj elemenata u rasponu [od, do) koji su jednaki `val`
  - Koristi `operator==` za provjeru jednakosti

```
array<int, 7> p = { 11, 22, 33, 11, 44, 55, 11 };

int n = count(p.begin(), p.end(), 11);

cout << n << endl;
```

- Koji broj bi bio ispisan ako bismo srednju liniju zamijenili s:

```
int n = count(p.begin(), p.begin() + 3, 11);
```





## Primjer STL algoritama (3/3)

- `for_each(od, do, funkcija)` – primjenjuje zadanu funkciju za svaki element u rasponu [od, do)

```
void mnozi(int& broj) {
    broj *= 2;
}
void ispisi(int& bla) {
    cout << bla << endl;
}
int main() {
    array<int, 5> p = { 11, 22, 33, 44, 55 };
    for_each(p.begin(), p.end(), mnozi);
    for_each(p.begin(), p.end(), ispisi);

    return 0;
}
```

# PARSIRANJE DATOTEKA






## Parsiranje datoteka

- Parsiranje datoteka se često sastoji od sljedećih koraka:
  - Pročitaj string do zadanog znaka
    - Prema potrebi, string pretvori u drugi tip podataka
    - Ponavljaj za sve podatke u tom retku
  - Od pročitanih podataka konstruiraj objekt i stavi ga u kontejner (polje, vektor, povezanu listu, … )
  - Ponavljaj sve dok ima redaka u datoteci
- Moguć pristup:

```
while (true) {
    if (!getline(dat, str, ',')) {
        break; // Nisam uspio pročitati, kraj datoteke.
    }
    …
}
```



## Primjer: parsiranje datoteke

- Zadatak: ispišimo koje godine je vodostaj rijeke Huron (Michigan) bio najviši
  - Datoteka: LakeHuron.csv
  - Izvor: vincentarelbundock.github.io/Rdatasets/datasets.html
- Primjer prvih 5 redaka:

```
"","time","LakeHuron"
"1",1875,580.38
"2",1876,581.86
"3",1877,580.97
"4",1878,580.8
```



ALGEBRA



## Razmišljanje i odluke

- Preduvjet: razumjeti što znači koji podatak u datoteci
- Koji nam stupci trebaju za odraditi zadatak?
  - time i LakeHuron
- Koji su tipovi podataka stupaca koji nam trebaju
  - time je int, LakeHuron je double
- Što ćemo s prvom linijom i s prvim stupcem
  - Odbaciti jer nam ne trebaju
- Koji ćemo kontejner koristiti, array ili vektor?
  - Vektor, jer broj linija može varirati



ALGEBRA

## Algoritam (1/3)

- Kako ćemo pročitati i odbaciti prvi redak?

```
"","time","LakeHuron"
"1",1875,580.38
"2",1876,581.86
"3",1877,580.97
"4",1878,580.8
```

- Onda ćemo pročitati i odbaciti sve do prvog zareza:

```
"","time","LakeHuron"
"1",1875,580.38
"2",1876,581.86
"3",1877,580.97
"4",1878,580.8
```

Strana ▪ 40

ALGEBRA

40

## Algoritam (2/3)

- Kako ćemo pročitati tekst do zareza i pretvoriti ga u `int`?

```
"","time","LakeHuron"
"1",1875,580.38
"2",1876,581.86
"3",1877,580.97
"4",1878,580.8
```

- Kako ćemo pročitati sve do kraja retka i pretvoriti u `double`?

```
"","time","LakeHuron"
"1",1875,580.38
"2",1876,581.86
"3",1877,580.97
"4",1878,580.8
```

Strana ▪ 41

ALGEBRA

41

## Algoritam (3/3)

- Od pročitanog `int`-a i `double`-a napravi objekt i stavi ga u vektor
- Ponavljaj dok ima redaka
  - Kako znamo da više nema redaka?
  - Čitanje do prvog sljedećeg zareza će vratiti `false`






## Rješenje (1/3)

- Struktura:

```
struct water_level {
    int year;
    double level;
};
```

- Funkcija main:

```
ifstream dat("LakeHuron.csv");
if (!dat) {
    cout << "Greska pri otvaranju datototeke" << endl;
}

// parsiranje…

dat.close();
```



# Rješenje (2/3)

- Parsiranje:

```
string temp;
getline(dat, temp);

stringstream sstr;
vector<water_level> zapisi;
water_level obj;
while (true) {
      if (!getline(dat, temp, ',')) {
            break;
      }

      getline(dat, temp, ',');
      sstr << temp;
      sstr >> obj.year;
      sstr.str("");
      sstr.clear();
```



ALGEBRA



# Rješenje (3/3)

```
      getline(dat, temp);
      sstr << temp;
      sstr >> obj.level;
      sstr.str("");
      sstr.clear();

      zapisi.push_back(obj);
}
```



ALGEBRA

## Bolje rješenje (1/2)

- Koji dio kôda možemo izvući u funkciju?

```
int izvadi_int(ifstream& dat, stringstream& sstr, string& temp, char s) {
    int rezultat;
    getline(dat, temp, s);
    sstr << temp;
    sstr >> rezultat;
    sstr.str("");
    sstr.clear();
    return rezultat;
}
double izvadi_double(ifstream& dat, stringstream& sstr, string& temp,
                                                              char s) {
    double rezultat;
    getline(dat, temp, s);
    sstr << temp;
    sstr >> rezultat;
    sstr.str("");
    sstr.clear();
    return rezultat;
}
```






## Bolje rješenje (2/2)

- Parsiranje u funkciji `main` sad postaje:

```
while (true) {
      if (!getline(dat, temp, ',')) {
            break;
      }

      obj.year = izvadi_int(dat, sstr, temp, ',');
      obj.level = izvadi_double(dat, sstr, temp, '\n');

      zapisi.push_back(obj);
}
```



## Dodatni materijali

- Dodatni materijali su dostupni na:
  - Constructors and destructors
    - https://youtu.be/LD2iW8fQD1Y
  - Stringstream
    - https://youtu.be/svjUBurBIuA
  - Introduction to STL
    - https://youtu.be/M8scJh4X-XI
  - Parsing files
    - https://youtu.be/IEJI2YWy34s



ALGEBRA