STRUKTURE PODATAKA I ALGORITMI

Predavanje 01

Ishod 1

1

INFORMACIJE O KOLEGIJU

2

Nastavnici

- Predavanja:
  - Izv. prof. dr. sc. Goran Đambić
  - goran.dambic@algebra.hr
  - Konzultacije: bilo kada, uz prethodnu najavu e-mailom ili na Teams

O kolegiju

- Ciljevi kolegija:
  - Upoznati osnovne apstraktne tipove podataka i algoritme
  - Naučiti koristiti konkretne implementacije u C++
  - Dostići početnu razinu programiranja u C++
- Kolegij je obavezan dio programa i nosi **6 ECTS** bodova (otprilike 30 sati / bod)
  - 30 sati predavanja (15 tjedana po 2 sata)
  - 45 sati vježbi (15 tjedana po 3 sata)
  - 105 sati samostalnog rada (15 tjedana po 7 sati)
    - **Količina samostalnog rada znatno ovisi o stečenom znanju iz kolegija Programiranje**

Potpis

- Za stjecanje prava na potpis potrebno je prisustvovati nastavi u postotku propisanom Pravilnikom o studijima i studiranju

Dolaznost na predavanja i vježbe	
najmanje 50% fizičke prisutnosti na predavanjima	najmanje 60% fizičke prisutnosti na vježbama

- Tko ne dobije potpis, mora sljedeće godine ponovno upisati kolegij, platiti upis kolegija te nema pravo polaganja ispita

Prikupljanje bodova i ocjene

- Na kolegiju je moguće skupiti najviše **100 bodova** :
  - Domaće zadaće: najviše **6 bodova**
  - Školske zadaće: najviše **6 bodova**
  - Dva međuispita: najviše **88 bodova**
  - Usmenog ispita nema

$$\sum = 100$$

- Ocjene:
  - 92,01 – 100,00 bodova: izvrstan (5)
  - 75,01 – 92,00 bodova: vrlo dobar (4)
  - 58,01 – 75,00 bodova: dobar (3)
  - 50,01 – 58,00 bodova: dovoljan (2)
- Po svakom ishodu morate imate barem 50% bodova

SPA ishodi učenja

Ishod	MINIMALNI ISHODI UČENJA (Po uspješnom završetku predmeta, student će moći)	ŽELJENI ISHODI UČENJA (Uspješan student bi trebao moći)
I1	Odrediti i argumentirati vremensku složenost a priori i a posteriori za zadani algoritam izveden u programskom jeziku.	Kreirati složenija programska rješenja koristeći projekte s više datoteka i korisnički definirane tipove podataka.
I2	Konstruirati rješenje korištenjem linearnih struktura podataka (lista, vezana lista, stog, red) i pripadajućih algoritama.	Konstruirati složenije rješenje korištenjem linearnih struktura podataka (lista, vezana lista, stog, red) i pripadajućih algoritama.
I3	Konstruirati rješenje korištenjem hijerarhijskih struktura podataka (stablo, gomila, prioritetni red) i pripadajućih algoritama.	Konstruirati složenije rješenje korištenjem hijerarhijskih struktura podataka (stablo, gomila, prioritetni red) i pripadajućih algoritama.
I4	Konstruirati rješenje korištenjem rječnika temeljenim na stablima i pripadajućih algoritama	Konstruirati složenije rješenje korištenjem rječnika temeljenim na stablima i pripadajućih algoritama
I5	Opisati algoritme sortiranja i pretraživanja te konstruirati rješenje temeljeno na algoritmima sortiranja i pretraživanja.	Opisati algoritme sortiranja i pretraživanja te konstruirati kompleksnije rješenje temeljeno na algoritmima sortiranja i pretraživanja.
I6	Kreirati rješenje korištenjem tehnika adresiranja te argumentirati njihovu vremensku složenost.	Kreirati složenije rješenje korištenjem tehnika adresiranja te argumentirati njihovu vremensku složenost.

7

Ishodi učenja i provjere znanja

	M1	M2	Domaća zadaća	Školska zadaća	MAX
I1	20		2	2	24
I2	20		2	2	24
I3		12		1	13
I4		12	2		14
I5		12		1	13
I6		12			12
Ukupno	40	48	6	6	100

8

Organizacija gradiva

Tjedan	Tema
1-4	Uvod, STL – Standard Template Library, Složenost algoritama
5-8	Vektor, Povezana lista, Stog i red
9	Stabla
10	Prioritetni red
11	Rječnici implementirani binarnim stablima traženja
12-13	Sortiranje i pretraživanje
14-15	Rječnici implementirani *hash* tablicama

9

Općenito o ispitima

- Na svakom kolegiju vrijedi **pravilo 3 + 1,** što znači da student mora položiti ispit iz najviše 4 izlaska
  - 3 redovna izlaska – Uključena u cijenu školarine
  - 1 izvanredni izlazak – Odlukom o naknadi troškova 4. prijava ispita se naplaćuje
  - Vremenski rok za položiti ispit je **12 mjeseci** od dana upisa
  - Ako student u 12 mjeseci ne položi kolegij, **mora ponovno upisati kolegij te ponovno polagati sve ishoda učenja kako je definirano kolegijem**
- Vodite računa o rokovima prijave i odjave ispita na Infoeduci
  - Ako niste prijavili ispit na vrijeme, ne možete pristupiti ni pismenom, niti usmenom dijelu, ni obrani projekta.
  - Ako je student prijavio više ispitnih rokova iz istog kolegija, pri dobivanju ocjene kojom je zadovoljan, dužan je odjaviti svaki sljedeći rok koji je iz tog kolegija prijavio. U suprotnom, studentu se u Infoeduku unosi nedovoljan (1).

10

Akademski standard ponašanja

- U komunikaciji (pisanoj i usmenoj) pridržavati se pravila poslovne komunikacije primjerene akademskoj razini.
- Potrebno je držati se jasno definirah rokova za predaju zadataka (zadaća, seminarskih radova, projekata i sl.).
  - Svaki zadatak, domaća zadaća, projekt itd., poslani nakon definiranog roka neće se ocjenjivati.
- Samo oni studenti koji mogu potvrditi svoje pohađanje, smatrat će se prisutnima.
  - Potpisivanje drugih studenata ili registracija njihovom karticom nije dopušteno i može biti predmet stegovnog postupka. Nastavnik će obrisati prisustvo ako utvrdi da je student prijavljen, a da nije prisutan na nastavi.

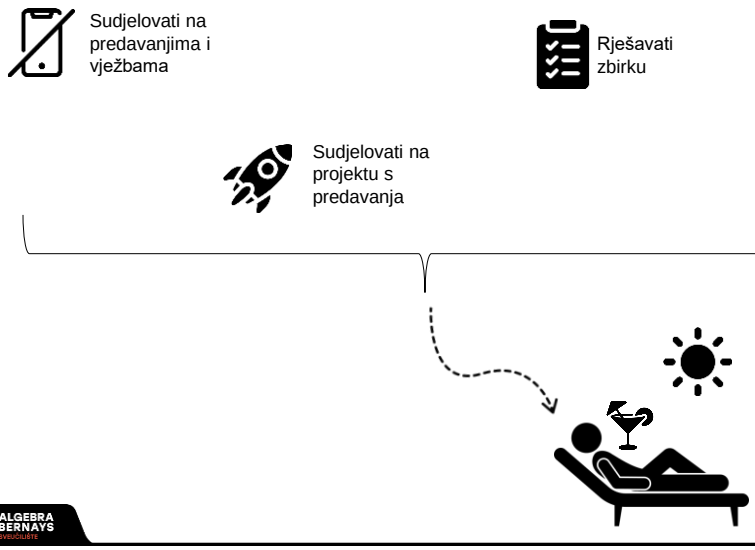
Pravila ponašanja na nastavi – fizička prisutnost

- Na nastavu se dolazi na vrijeme.
- Pri ulasku u učionicu student prilazi do stola i prijavljuje se na nastavu karticom te sjeda na dostupno mjesto za rad.
- Ometanje nastave i neaktivno sudjelovanje na nastavi nije dozvoljeno.
  - Repetitivno kršenje ovog pravila sankcionira se prijavom Stegovnom povjerenstvu

Literatura

- Obvezna literatura:
  - Demistificirani C++
- Preporučena literatura:
  - O'Dwyer, A. (2017) Mastering the C++17 STL: Make full use of the standard library components in C++17. Birmingham: Packt Publishing
- Dodatna literatura
  - Cormen, T. (2009) Introduction to Algorithms. 3rd edn. Cambridge: MIT Press.

Kako ostvariti ishode učenja



Programiranje vs SPA

15

JAVNI I PRIVATNI ČLANOVI

16

Strukture i klase (1/2)

- Tipovi podataka se dijele na:
  - Ugrađene (`int`, `void`, `double`, `char`, `wchar_t`, `long`, ...)
  - Korisnički definirane (strukture i klase)
- Članovi strukture su *defaultno* javni (engl. `public`), a klase privatni (engl. `private`)
  - Članovi su najčešće varijable (engl. *fields*) i funkcije (koje se onda zovu članske funkcije ili metode)

```
struct Pravokutnik {
    int a;
    int b;
};
```

```
class Pravokutnik {
    int a;
    int b;
};
```


Strukture i klase (2/2)

- I strukturi i klasi možemo eksplicitno definirati koji članovi su javni, a koji su privatni

```
struct Pravokutnik {
public:
    int a;
private:
    int b;
};
```

```
class Pravokutnik {
public:
    int a;
private:
    int b;
};
```

- Privatnim članovima možemo pristupati samo iz funkcija koje se nalaze u toj strukturi/klasi
- Javnim članovima možemo pristupati i iz `main`-a (i svih ostalih funkcija)

Kad koristiti strukturu, a kad klasu? (1/3)

- Osim *defaultne* razlike u vidljivosti članova, strukture i klase nemaju dodatnih razlika što se tiče kompajlera
- Razlika je u semantici (onome što tip nama znači):
  - Ako imamo potrebu za tipom čija je glavna namjena čuvanje podataka bez puno operacija na njima => struktura
  - Za sve ostalo => klasa

Kad koristiti strukturu, a kad klasu? (2/3)

- Google C++ Style Guide:
  - google.github.io/styleguide/cppguide.html
  - „structs should be used for passive objects that carry data, and … lack any functionality other than access/setting the data members. The accessing/setting of fields is done by directly accessing the fields rather than through method invocations. Methods should not provide behavior but should only be used to set up the data members, e.g., constructor, destructor, Initialize(), Reset(), Validate().”
  - „If more functionality is required, a class is more appropriate.”
  - „If in doubt, make it a class.”

Kad koristiti strukturu, a kad klasu? (3/3)

- Primjeri:
  - Ako želimo čuvati podatke o širini i visini pravokutnika?
    - Struktura
  - Ako želimo čuvati podatke o širini i visini pravokutnika te uz njih pružiti 10 operacija s pravokutnikom (iscrtavanje, izračun površine, opsega, množenje skalarom, …)?
    - Klasa
- Savjet:
  - Neka vam prvi izbor bude klasa
  - Strukturu koristite samo kao kontejner podataka
- Na ovom kolegiju nije važno znati pravilno odabrati
  - krivi odabir neće biti negativno vrednovan

Javni i privatni članovi (1/3)

- Primjer tipa podataka (koliko članova ima tip):

```
class Pravokutnik {
public:
    void inicijaliziraj(int s, int v) {
        sirina = s;
        visina = v;
    }
    int povrsina() {
        return sirina * visina;
    }

private:
    int sirina;
    int visina;
};
```

Metode (tj. funkcije na klasi) mogu koristiti privatne članove iste klase

Javni i privatni članovi (2/3)

- Primjer korištenja:

```
int main() {
    Pravokutnik p;
    p.inicijaliziraj(10, 5);
    cout << p.povrsina() << endl;
    cout << p.sirina << endl;
    cout << p.visina << endl;

    return 0;
}
```

`main` može koristiti javne članove

`main` ne može koristiti privatne članove

- Niti `main` niti ostale funkcije koje nisu unutar te klase ne mogu koristiti privatne članove te klase
  - Članovi su privatni za tu klasu

Javni i privatni članovi (3/3)

- Javni i privatni članovi služe za ostvarivanje OOP koncepta zvanog enkapsulacija:
  - Javni članovi definiraju sučelje kojeg smiju koristiti korisnici objekta
  - Privatni članovi služe za čuvanje stanja objekta
  - Puno detaljnije na: OOP
- Mi ćemo pravilo enkapsulacije koristiti na sljedeći način:
  - Javni članovi će biti: sve funkcije koje planiramo koristiti iz `main`-a
  - Privatni članovi će biti: sve varijable i one funkcije koje nećemo koristiti iz `main`-a

Primjer

```
class Osoba {
    string ime;
    void inicijaliziraj(string i, string p) {
        ime = i;
        prezime = p;
    }
    void ispisi() {
        cout << ime << " " << prezime << endl;
    }
    string prezime;
};
```

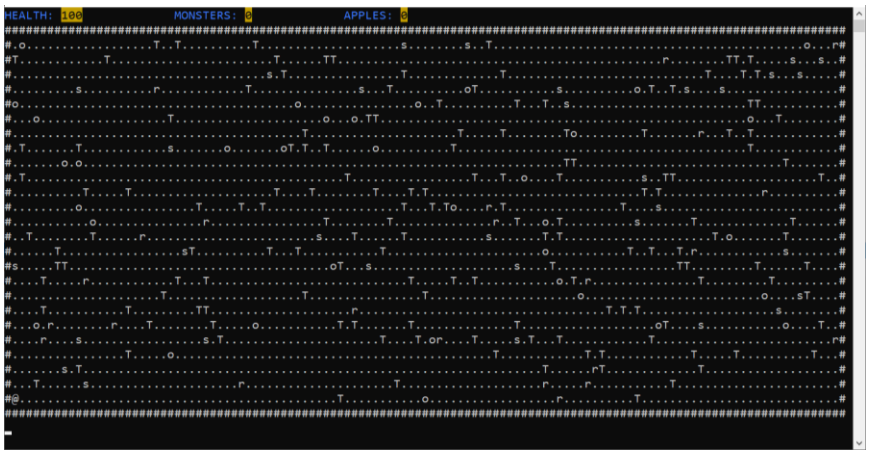
1. Koje članove trenutno možemo koristiti iz `main`-a?
2. Koji članovi bi trebali biti privatni, a koji javni?
3. Preuredimo…

Rješenje

```
class Osoba {
public:
    void inicijaliziraj(string i, string p) {
        ime = i;
        prezime = p;
    }
    void ispisi() {
        cout << ime << " " << prezime << endl;
    }
private:
    string ime;
    string prezime;
};
```

Projekt: The Beginning

- Napišimo konzolnu igru koja se sastoji od mape po kojoj se igrač može kretati, skupljati hranu i boriti se s protivnicima.



27

Projekt: okvir rješenja (deklaracija klase)

```
class Game {
private:
    char map[rows][columns];
    double health;
    int apples_collected;
    int monsters_defeated;
    int player_row;
    int player_col;
    bool can_move(int target_row, int target_col);
    bool is_on_enemy();
    bool is_on_food();
    void defeat_monster();
    void eat_food();
public:
    void initialize();
    void generate(unsigned int seed);
    void draw();
    void move(char action);
    bool process();
};
```

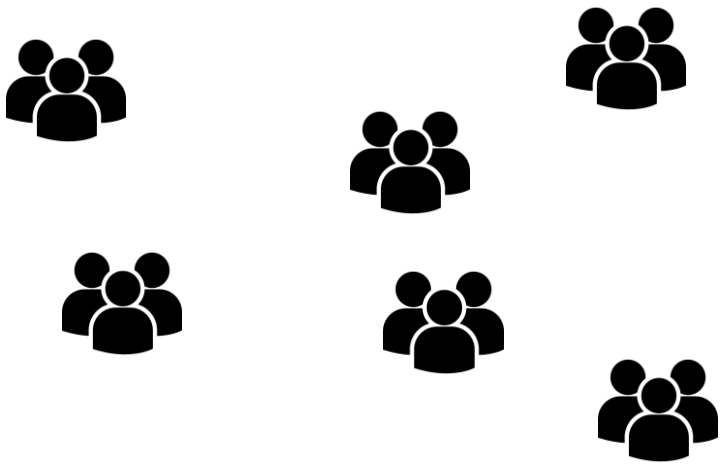

28

Projekt: okvir rješenja (main)

```
int main() {
    Game game;
    game.initialize();
    game.generate(42);
    game.draw();

    while (true) { // game loop
        char action = (char)_getch();
        if (action == (char)27) { // escape
            break;
        }

        game.move(action);
        if (!game.process()) {
            system("cls");
            cout << endl << endl << endl << endl;
            cout << fg_red << "\t\t\t\tGAME OVER" << fg_white << endl;
            cout << endl << endl << endl << endl;
            break;
        }

        game.draw();
    }
```


29

Projekt: podjela u grupe



30

Projekt: grupni *brainstorming* (10 min)

- Organizirajte se kako ćete surađivati unutar grupe jer **sljedeći put prezentirate**
- Grupa 1: istražite kako dodati pustinju i drveće grupirati u šumarke.
- Grupa 2: osmislite bolji način borbe.
- Grupa 3: osmislite načine upravljanja zdravljem i štitom.
- Grupa 4: osmislite kako više mapa uključiti u igru i posložiti boje.
- Grupa 5: osmislite RPG atribute i njihov utjecaj na igru.
- Grupa 6: osmislite sustav *inventoryja*.
- Grupa 7: osmislite predmete koje igrač može skupiti i njihov utjecaj.
- Grupa 8: osmislite *lore* igre i kako ga prezentirati igraču.
- Grupa 9: osmislite jednog NPC-a i interakciju s njim.
- Grupa 10: osmisliti spremanje i učitavanje pozicije.

31

ORGANIZACIJA C++ PROJEKTA

32

Kompajliranje i linkanje

- U C++ se kompajliraju samo .cpp datoteke
- Postupak kompajliranja je sljedeći:
  - Na mjesto #include se iskopira kompletan sadržaj iz .h datoteke (pretprocesiranje)
  - Svaki .cpp se kompajlira neovisno o ostalima (.cpp => .obj)
  - Na kraju se sve .obj datoteke linkaju i nastaje .exe

Problemi #1

- Prethodno rješenje radi, ali imamo dva problema:
  - Prvi problem: jedna velika datoteka nije jednostavna za izmjene jer često imamo na stotine klasa i struktura pa datoteka može postati ogromna
  - Drugi problem: ako nam je sve u jednoj.cpp datoteci, i najmanja promjena traži ponovno kompajliranje svega => može trajati duuuuugo
- Bolje rješenje je podijeliti C++ projekt na više datoteka:
  - .h datoteke (zaglavlja) sadrže deklaracije (klasa, funkcija, …)
  - .cpp datoteke sadrže implementacije

Rješenje problema #1 (1/3)

- Kako bismo bolje organizirali naš kôd, dodat ćemo tri nove datoteke:
  - Game.h će sadržavati samo deklaraciju klase
  - Game.cpp će sadržavati implementaciju metoda klase
  - Constants.h će sadržavati konstante
- Dodajmo nove datoteke u projekt

Rješenje problema #1 (2/3)

- U .cpp implementiramo sve metode
  - Obavezno uključimo .h datoteku
  - Ispred naziva metode moramo pisati kojoj klasi pripada
  - Uključimo dodatna zaglavlja prema potrebi

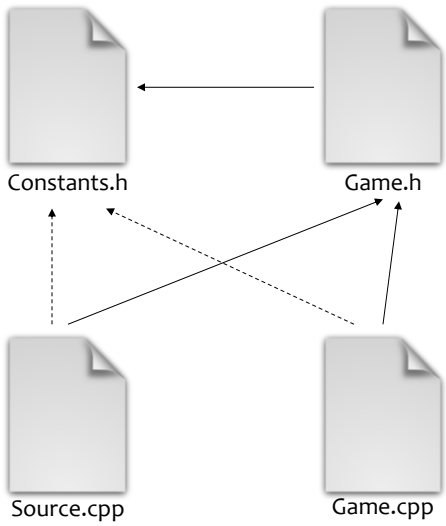
```
#include <iostream>
#include "Game.h"
using namespace std;

void Game::eat_food() {
   map[player_row][player_col] = '.';
   apples_collected++;
}
```

Rješenje problema #1 (3/3)

- Nakon reorganizacije, moramo na ispravan način povezati datoteke u cjelinu:
  - U Game.h ćemo uključiti Constants.h
  - U Game.cpp ćemo uključiti Game.h (posredno i Constants.h)
  - U Source.cpp ćemo uključiti Game.h (posredno i Constants.h)
  - Nikad ne uključujemo .cpp datoteke
- Pravilo:
  - Kad uključujemo standardna zaglavlje, koristimo razlomljene zagrade, primjerice: `#include <string>`
  - Kad uključujemo naša zaglavlja, koristimo dvostruke navodnike, primjerice: `#include "pravokutnik.h"`

37

Primjer povezivanja datoteka



38

Problem #2

- Promijenimo početak datoteke Source.cpp tako da glasi:

```
#include <string>
#include <string>
#include "Game.h"
```

- Možemo li ovo kompajlirati?
- Promijenimo početak datoteke program.cpp tako da glasi:

```
#include <string>
#include "Game.h"
#include "Game.h"
```

- Možemo li ovo kompajlirati?

Rješenje problema #2 – *include guard* (1/2)

- Zbog kopiranja .h datoteka u .cpp datoteke će nastati problemi ako se ista .h datoteka više puta uključi u isti .cpp
  - Izravno ili posredno preko drugih datoteka
- Problem se rješava korištenjem *include guard*-ova

```
#ifndef __GAME_H__ // Ako simbol nije definiran
#define __GAME_H__ // Definiramo ga sad
class Game { … };
#endif
```

- *Include guard* dopušta umetanje datoteke samo jednom
- Svaka .h datoteka mora imati *include guard*

Rješenje problema broj dva – *include guard* (2/2)

- Alternativno, možemo koristiti:

```
#pragma once
class Game { … };
```

- Nije dio C++ standarda
  - Verzije GCC kompajlera manje od 3.4 ne prepoznaju ovakav *include guard*
- Za ovaj kolegij je svejedno koji *include guard* ćete koristiti – ali nešto morate koristiti!

Dodatni materijali

- Dodatni materijali su dostupni na:
  - Public vs Private
    - https://youtu.be/nw-XS8bUgHQ
  - Project organization
    - https://youtu.be/OyUKuUX9F7I

Za sljedeće predavanje

1. Napravite PowerPoint prezentaciju za svoju grupu i u njoj opišite kako ćete implementirati nove funkcionalnosti u igru (grupni rad)
   - Max 3 minute
2. Naučite sve što smo radili danas