

# Извештај

---

## I Информације о аутору

Име: Стефан Кандић

Број индекса: SW 73/2017

Предмет: Објектно орјентисано програмирање 2

Тема: визуализација различитих алгоритама сортирања у језику Це++ уз подршку ФЛТК библиотеке

## II Рад У/И подсистема

Компонента програма задужена за руковање улазно-излазним системом је класа ИО, која се налази у тесној сарадњи са графичком компонентом, јер су једна од друге зависне.

Први задатак јој је да анализира аргументе командне линије, и да визуелној компоненти проследи задате параметре којима програм касније рукује.

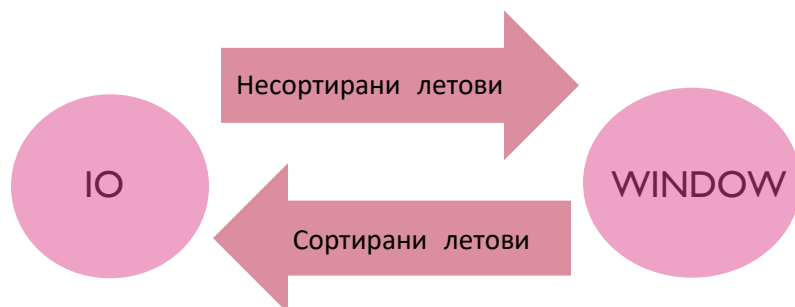
При самом покретању програма, или након корисничке команде из графичког дела апликације, учитава летове из изабраног текстуалног фајла.

Ако фајл не задовољава унапред задату структуру, она главни прозор обавештава о грешци којом он даље рукује без икаквих ометњи у самом раду.

Структура улазне датотеке:

```
Destination Departure Flight No. Gate No.  
dest1; departure1; fnum1; gate1;  
...
```

Када од главног прозора добије сигнал, у излазни фајл који садржи као обежење уписује прослеђени тест који садржи излазне резултате и сортиране летове.



### III Списак класа,структура и изузетака

#### 1. Структуре података

1.1. vector<?>

1.2. list<?>

1.3. < iterator >

#### 2. Енумерације

2.1. Tip

#### 3. Изузеци

3.1. InvalidData

#### 4. Struct

4.1. CheckBox

4.2. Slider

#### 5. Класе

5.1. Flight

5.2. VisualizedFlight

5.3. Instruction

5.3.1. ColorChange

5.3.2. ColorSwap

5.3.3. WidgetSwap

5.3.4. RevertColor

5.4. Sort

5.4.1. Selection Sort

5.4.2. Quick Sort

5.5. IO

5.6. MyWindow

## IV Објашњење коришћених структура

### Структуре података:

- `vector<?>` - вектори су секвенца контејнера који представљају низове променљиве величине

Као и низови, вектори користе суседне меморијске локације за складиштење елемената, али за разлику од низова њихова величина се може мењати динамички. Ово може изазвати да низ мора бити премештен на другу меморијску локацију када се додају нови елементи.

Због тога, у поређењу са низовима, вектори користе више меморије али пружају ефикасно решење за низове којима нећемо унапред знати број елемената.

- `list<?>` - листе су секвенце контејнера који дозвољавају операције брисања и додавања нових елемената на било којој позицији у константном времену

Двоструко су спрегнуте, и могу чувати своје елементе на логички неповезаним меморијским локацијама помоћу показивача.

У поређењу са другим контејнерима из стандардне библиотеке, листе имају боље перформансе при додавању, брисању и померању елемената са/на било коју позицију

Њихова мана је немогућност директног приступа својим елементима на основу њихове позиције у листи.

- `<iterator>` - итератор је објекат који показујући на одређени елемент у неком контејнеру елемената, има способност да итерира кроз остале елементе тог контејнера

## Енумерације:

- `enum Tip {flightNum, departure, destination, gate}` - служи за одабир критеријума сортирања летова

## Изузеци :

- `InvalidData` – позива се када улазни фајл нема предвиђену структуру и одбацује изабрани фајл, програм наставља нормално са радом и кориснику се нуди опција да види како улазни фајл треба да изгледа

## Struct - ови:

- `struct CheckBox` – структура направљена по угледу на `Button` из `graphlib.h` и служи за избор активних поља у програму
- `struct Slider` - структура направљена по угледу на `Button` из `graphlib.h` и служи за избор вредности из одређеног опсега

## Класе :

- `Класа Flight` са следећим члановима

[I] приватни чланови(атрибути)

- `string f_number` – број лета (јединствени идентификатор)
- `string destination` - дестинација
- `string departure` – време поласка
- `string gate`– ознака капије

## [2] методе

- Гетери и сетери за приватне чланове
- `bool operator == (Flight &f2)` – пореди два лета на основу њиховог идентификатора
- `int compare(Flight f2, Tip crit)` – пореди два лета на основу енума
- `int compareDeparture (Flight &f2)` – пореди два лета на основу времена њиховог поласка

## [3] преклопљени оператори

- `istream& operator>> (istream &in, Flight &f)`; – оператор за учитавање лета из произвољног тока
- `ostream& operator<< (ostream &out, Flight &f)` - оператор за уписивање у произвољни излазни ток

- **Класа VisualizedFlight** : `public Fl_Widget` – представља графичку репрезентацију једног лета у програму

## [1] атрибути

- `Flight flight` - лет
- `int unsortedIndex` – позиција лета у фајлу
- `int sortedIndex` – позиција лета након сортирања
- `int numOfFlightsTotal` – укупан број летова
- `Fl_Color boja` – боја компоненте

## [2] методе

- `void draw ()` – исцртава компоненту на прозору
- `void write (const char* s)` – исписује текстуални садржај преко компоненте
- `void move (int jean, int dva)` – помера компоненту
- `bool operator = (VisualizedFlight &vf)` – пореди две компоненте на основу њиховог лета

- **Класа `Instruction`** – представља једну недељиву надердбу која се мора извршити како си се сортирање визуализовало

[1] атрибути

- **`VisualizedFlight *vf`** – показивач на графичку компоненту над којом инструкција треба да се изврши

[2] методе

- **`virtual void do_it () = 0`** – виртуелна метода која представља имплементацију саме инструкције

- **Класа `ColorChange` :`public Instruction`** – промена боје једне компоненте приказане на главном прозору

[1] атрибути

- Атрибути родитељске класе
- **`Fl_Color color`** – боја у коју компонента треба да се обоји

[2] методе

- **`void do_it ()`** – имплементације своје функционалности

- Класа `ColorSwap` : `public Instruction` - замена боје две компоненте приказане на главном прозору

[1] атрибути

- Атрибути родитељске класе
- `VisualizedFlight *vf2` – показивач на другу графичку компоненту над којом инструкција треба да се изврши

[2] методе

- `void do_it ()` – имплементације своје функционалности

- Класа `WidgetSwap` - замена места двема компонентама на главном прозору

[1] атрибути

- Атрибути родитељске класе
- `VisualizedFlight *vf2` – показивач на графичку компоненту над којом инструкција треба да се изврши
- `Fl_Color color1` – боја којом треба да се обоји друга компонента
- `Fl_Color color2` – боја којом треба да се обоји друга компонента

[2] методе

- `void do_it ()` – имплементације своје функционалности



- `class RevertColor: public Instruction` – бојење колекције компоненти у њихову почетну боју

[1] атрибути

- `vector<VisualizedFlight*>` `vector` – колекција показивача на компоненте које треба да се обоје
- `Fl_Color` `color` – боја у коју треба да се обоје

[2] методе

- `void do_it ()` – имплементације своје функционалности

- `Класа Sort`

[1] приватни атрибути

- `unsigned long` `num_cmps` – број поређења приликом сортирања
- `unsigned long` `num_swaps` - број замена елемената

[2] методе

- Гетери и сетери
- `virtual void sort(vector<VisualizedFlight*> &data, Tip, list<Instruction*> &instructions) = 0` - виртуелна метода коју треба класе наследнице да имплементирају

- `class SelectionSort : public Sort`

[1] методе

- `void sort(vector<VisualizedFlight*> &data, Tip, list<Instruction*> &instructions)` – преклопљена метода свог родитеља која омогућава сортирање вектора елемената Selection sort алгоритмом

➤ Класа QuickSort: public Sort

[1] методе

- void sort(vector<VisualizedFlight\*> &data, Tip, list<Instruction\*> &instructions) – преклопљена метода свог родитеља која омогућава сортирање вектора елемената Quick sort алгоритмом
- void quickSort(vector<VisualizedFlight\*> &data, list<Instruction\*> &instructions, int low, int high, Tip en)
- int partition(std::vector<VisualizedFlight\*> &a, list<Instruction\*> &instructions, int i, int j, Tip en) – враћа позицију на коју треба да се уметне пивот

➤ Класа IO – класа задужена за улазно излазни систем програма

[1] атрибути

- string inPath – улазни фајл
- string outPath – излазни фајл
- string zaglavlje – излазни резултат сортирања који се уписује у излазну датотеку
- string crit – критеријум сортирања
- string type – тип алгоритма за сортирање
- vector<Flight> letovi
- vector<VisualizedFlight> viz

[2] методе

- void check\_arguments (int argc, char\* argv[]) – проверка аргумената командне линије
- void loadFlights () – учитавање летова из улазног фајла
- void writeFlights(vector<VisualizedFlight\*> &v, string header) – уписивање сортираних летова у излазни фајл

➤ Класа `MyWindow` : `Window`

[1] приватни атрибути

- `bool` `sorting` - да ли је сортирање у току
- `bool` `sorted` – да ли је сортирање завршено
- `vector<VisualizedFlight>` `mainVector`
- `IO` `inOut` – излазно улазна компонента

[2] јавни атрибути

- `CheckBox` компоненте помоћу којих се контролише критеријум и начин сортирања
- `Slider` `slider` – избор брзине визуелизације сортирања
- `float` `plotSpeed` – вредност горенаведене компоненте
- `Button` компоненте којима се контролише сама визуелизација сортирања
- `list<Instruction*>` `instrukcije` – листа инструкција извршених приликом сортирања елемената
- `vector<VisualizedFlight*>` вектори који садрже графичке компоненте у зависности од изабраног критеријума сортирања
- `Tip` `criteria` – тренутно изабрани критеријум сортирања

[3] методе

- Callback методе за графичке компоненте прозора
- `bool` `wait_for_button()` – позива петљу графичке компоненте
- `void` `fill()` – добављање информација о летовима из УИ компоненте
- `bool` `criteriaCheck()` – провера да ли су сви критеријуми задовољени за почетак сортирања
- `bool` `initializeSort()` – иницијализација сортирања
- `void` `displayStat()` – повратна информација о завршеном сортирању
- `void` `draw()` – исртавање прозора

## V Структуре аргумената командне линије и пример коришћења програма

Аргументи командне линије треба да имају следећу структуру

```
naziv.exe putanja1.txt putanja2.txt nacin_sortiranja algoritam
```

где је:

1. Први аргумент име извршног фајла
2. Други аргумент је путања до фајла из ког учитавамо податке о летовима
3. Трећи аргумент је путања до фајла у који уписујемо податке о сортираним летовима
4. Четврти критеријум сортирања летова ( “**fnum**” за сортирање по броју лета, “**dep**” за сортирање по времену одласка, “**dest**” по дестинацији, “**gate**” по капији)
5. И пети алгоритам сортирања ( “**ss**” за Selection Sort , и “**qs**” за Quick Sort )

Програм може да ради нормално са било којим бројем унесених аргумената (једино је извршни фајл обавезан аргумент)

Ако је неки аргумент неправилно унесен он ће бити игнорисан

Позиционирањем у директоријум где се налази извршни фајл, и покретањем команде :

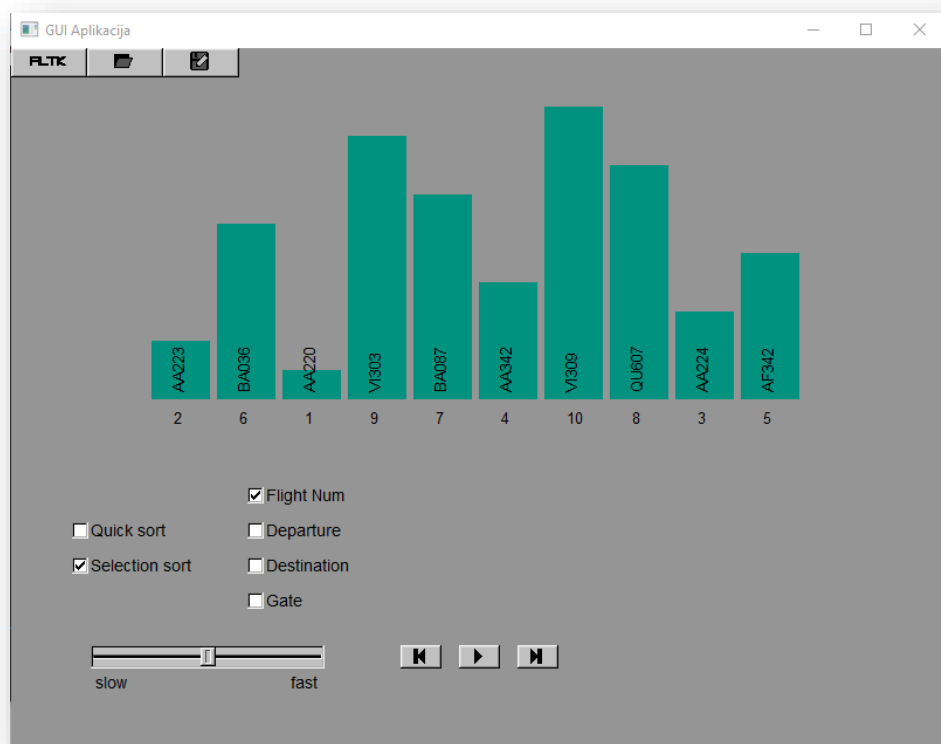
```
start SortingFlights.exe Test01.txt out.txt fnum ss
```

```
Command Prompt
Microsoft Windows [Version 10.0.17134.472]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Stefan>cd C:\Users\Stefan\Desktop\Vizuelizacija\Kodovi\VS15\SortingFlights\Debug
C:\Users\Stefan\Desktop\Vizuelizacija\Kodovi\VS15\SortingFlights\Debug>start SortingFlights.exe Test01.txt
out.txt fnum ss
```

Доћи ће до самог покретања програма где ће :

- Као улазни фајл бити изабран *Test01.txt*
- А као излазни *out.txt*
- Критеријум сортирања ће бити број лета
- А алгоритам сортирања **Selection sort**



## Структура излазног фајла:

- Коришћен алгоритам : X
- Критеријум сортирања : X
- Очекивани број поређења за алгоритам: X
- Број поређења: X
- Броја замена елемената: X
- сортирани лет1
- сортирани лет 2
- ...

## VI Опис алгоритама сортирања

### ➤ Selection Sort

Као први алгоритам сортирања коришћен је **Selection sort**. За сваку колекцију независно од распореда елемената он сортирање врши квадратном сложености.

Прецизније, број поређења ће сваки пут износити  $\frac{n^2-n}{2}$ , где је  $n$  број елемената колекције.

У свакој итерацији овог алгоритма крећемо од  $i$ -тог елемента ( $0 \leq i \leq n$ ), и идемо до краја колекције тражећи најмањи елемент.

Када дођемо до краја колекције, елемент који смо нашли заменимо са  $i$ - тим елементом.

Пошто имамо  $n$  елемената, сваки пут ћемо имати  $n$  итерација.

## ➤ Quick Sort

Као други алгоритам сортирања коришћен је **Quick sort**, пре свега због своје in-place особине, односно не креирања никаквих помоћних структура ради сортирања колекције. Његов најгори случај сортирања  $O(n^2)$ , али у просеку он ради боље од алгоритама чије је време  $O(n * \log n)$

Алгоритам чине три једноставна корака који се одвијају рекурзивно:

1. Одређивање пивота, тј елемента са којим ћемо поредити све остале елементе из колекције
2. Следеће, користећи пивот, колекцију делимо на два дела, лево од пивота су сви елементи мањи од њега, а десно сви елементи већи од њега
3. Понављамо алгоритам над овим партицијама док колекција није потпуно сортирана



## VII Напредни ООП Концепти

### ➤ Наслеђивање

- Класа `MyWindow` наслеђује класу `Window` која наслеђује класу `Fl_Window` из FLTK библиотеке
- Структуре `CheckBox` и `Slider` које наслеђују класу `Widget` дефинисану у `gui.h`
- Класа `VisualizedFlight` наслеђује класу `Widget`
- Класе `ColorChange`, `ColorSwap`, `WidgetSwap`, `RevertColor` наслеђују апстрактну класу `Instruction`

### ➤ Преклапање оператора

- Класе `Flight` и `VisualizedFlight` имају преклопљен оператор `=` који служи за поређење две инстанце исте класе
- Класа има преклопљене оператора за читање и писање у произвољни ток података

### ➤ Полиморфизам

- При визуализацији сортирања, графичка компонента има листу `list<Instruction*>` класе `Instruction` и позивањем њене методе `do_it()` се реализује визуелизација, али пошто је ова класа апстрактна, овај вектор чине њене класе наследнице и позивањем методе `do_it()` се позива метода коју је свака од класа наследница преклопила тако да обавља различите функционалности

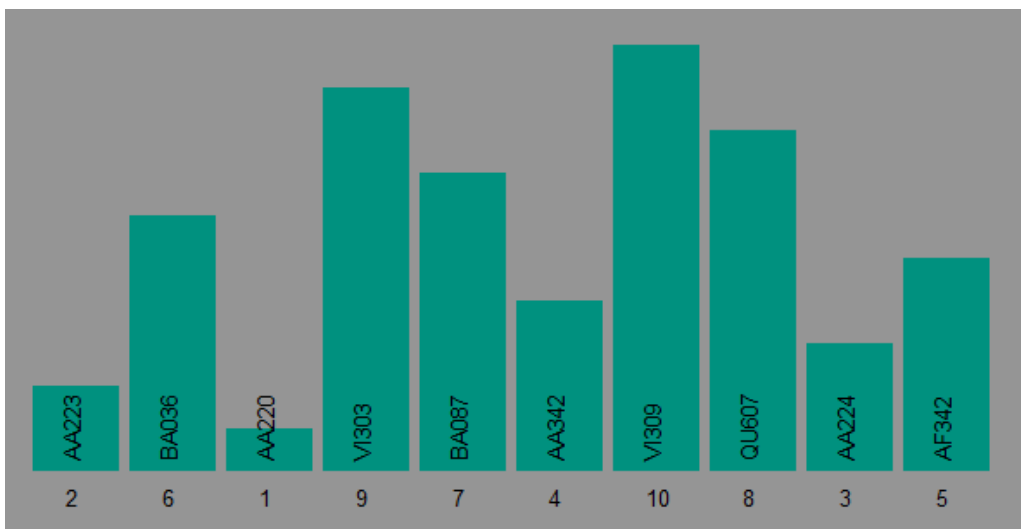
### ➤ Енкапсулација

- При сортирању летова, класа `Sort` не зна како се елементи пореде, већ о томе брине `enum Tip` и сама класа `Flight`

## VIII Графичка спрега

Графичку корисничку спрегу чини прозор са следећим компонентама :

- I. На централном делу прозора се налазе компоненте које представљају летове, то су правоугаони елементи са одређеним атрибутом лета и позицијом коју ће тај елемент заузимати након сортирања.



Ако летови нису учитани овај део прозора ће бити празан

### 2. Мени бар

На самом врху прозора се налази мени бар који се састоји од 3 дугмета:

- i. Дугме за приказ основних информација о аутору
- ii. Дугме за избор улазног фајла из кога се летови читају
- iii. Дугме за избор излазног фајла, када је улазни фајл изабран  
ово дугме служи за упис резултата сортирање у тај фајл

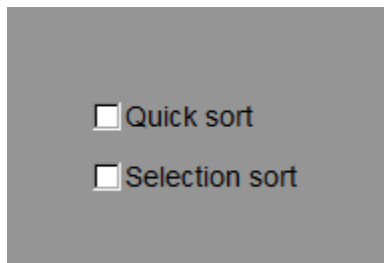


### 3. Поља за избор алгоритама сортирања

Испод самих летова налази се прва група поља, тзв поља за избор алгорита сортирања.

Састоји се од два поља :

- Поље за избор Quick Sort алгорита
- Поље за избор Selection Sort алгорита



☐ Quick sort

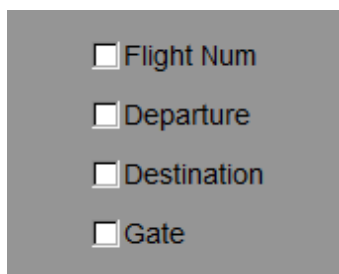
☐ Selection sort

### 4. Поља за избор критеријума сортирања

Друга група поља испод летова су поља којима бирамо у односу на које обележје ћемо сортирати летове

Састоји се од 3 поља :

- Поље за сортирање у односу на број лета
- Поље за сортирање у односу на време полетања
- Поље за сортирање у односу на дестинацију лета
- Поље за сортирање у односу на број капије



☐ Flight Num

☐ Departure

☐ Destination

☐ Gate

## 5. Слајдер за подешавање брзине

Испод наведених поља се налази слајдер чијом манипулацијом можемо мењати саму брзину визуализације сортирања



## 6. Дугмад за контролисање саме визуализације

На самом дну прозора се налазе 3 дугмета којима се цео процес визуализације контролише.

Састоји се од три дугмета, које са лева на десно обављају следеће функционисности:

- i. Дугме за враћање сортираних летова у њихов несортиран облик учитан из фајла
- ii. Дугме за приказ следеће инструкције сортирања
- iii. Дугме за приказ свих инструкција до краја сортирања



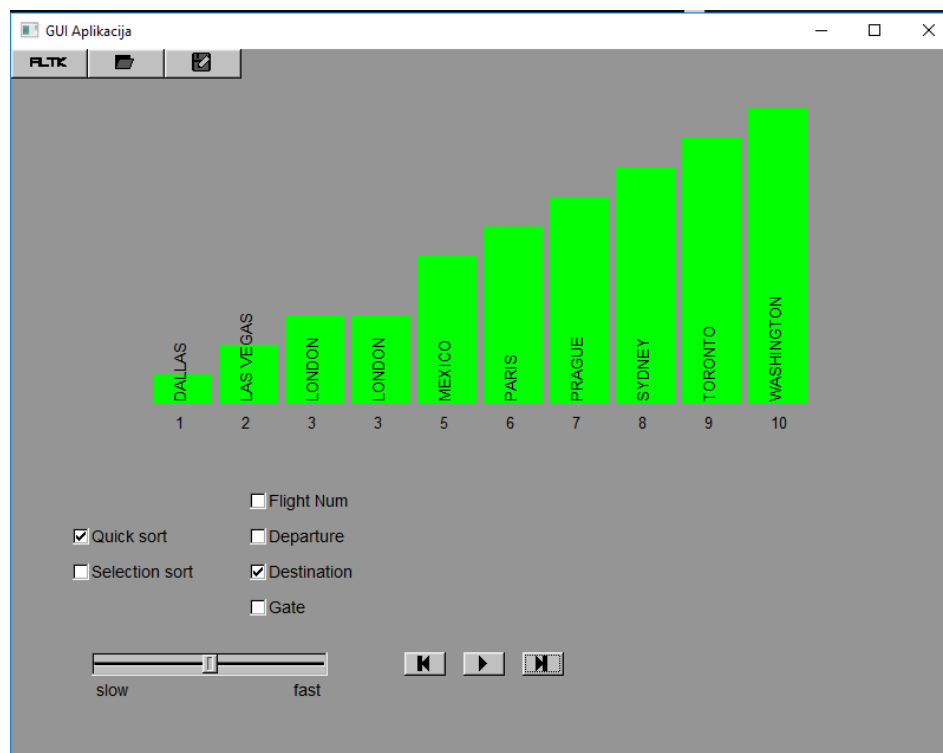
## IX Тестови

- Тест 01 - унапред задати тест случај

Структура улазне датотеке :

Flight number	Departure	Destination	Gate
AA223	21:15	Las Vegas	A3
BA036	21:00	Dallas	A3
AA220	20:30	London	B4
VI303	19:00	Mexico	B4
BA087	17:45	London	B4
AA342	16:00	Paris	A7
VI309	13:20	Prague	F2
QU607	08:30	Toronto	F2
AA224	08:20	Sydney	A7
AF342	07:45	Washington	A3

Изглед програма након сортирања летова у односу на дестинацију:

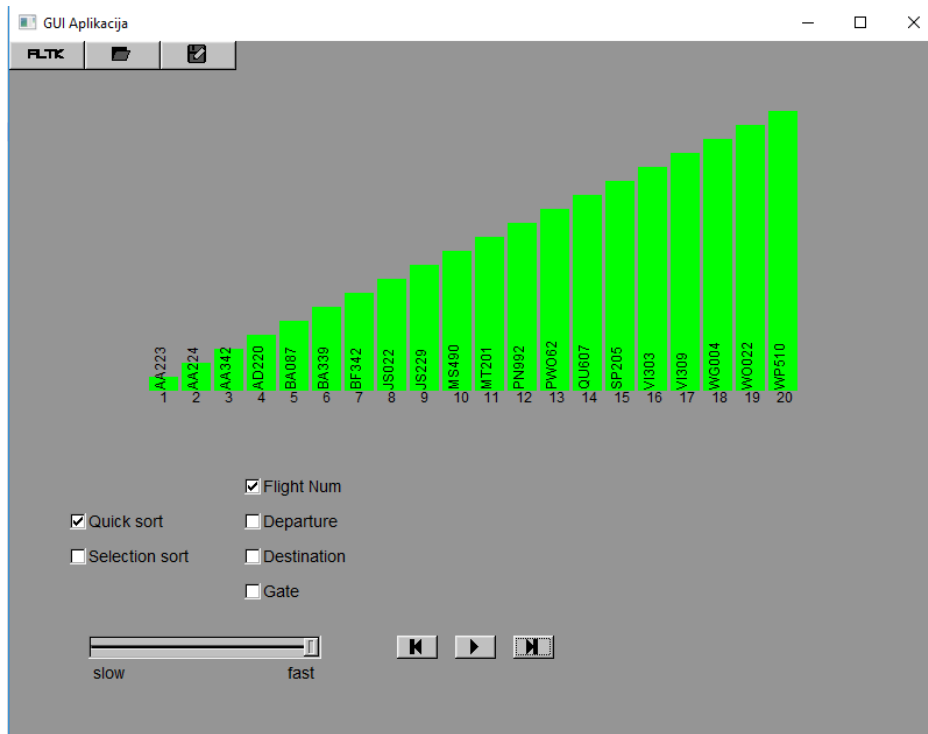


➤ Тест 02 – датотека са двадесет летова летова

Структура улазне датотеке :

Flight number	Departure	Destination	Gate
MS490	21:00	Montana	A3
VI303	19:00	Mexico	B4
PWO62	22:20	Charlotte	C1
AA223	21:15	Las Vegas	A3
AD220	20:30	London	B4
JS229	13:30	New York	C1
AA342	16:00	Paris	A7
VI309	13:20	Prague	F2
QU607	8:30	Toronto	F2
BA339	8:20	Sydney	A7
BF342	7:45	Washington	A3
AA224	12:45	Los Angeles	A1
JS022	13:25	Ontario	A7
BA087	17:45	London	B4
WO022	19:30	Memphis	C2
WP510	21:00	Boston	D7
SP205	17:15	New Hampshire	B3
PN992	7:15	Seattle	C2
WG004	12:30	Denver	A3
MT201	21:45	Los Angeles	A1

Изглед програма након сортирања ових летова у односу на број лета:

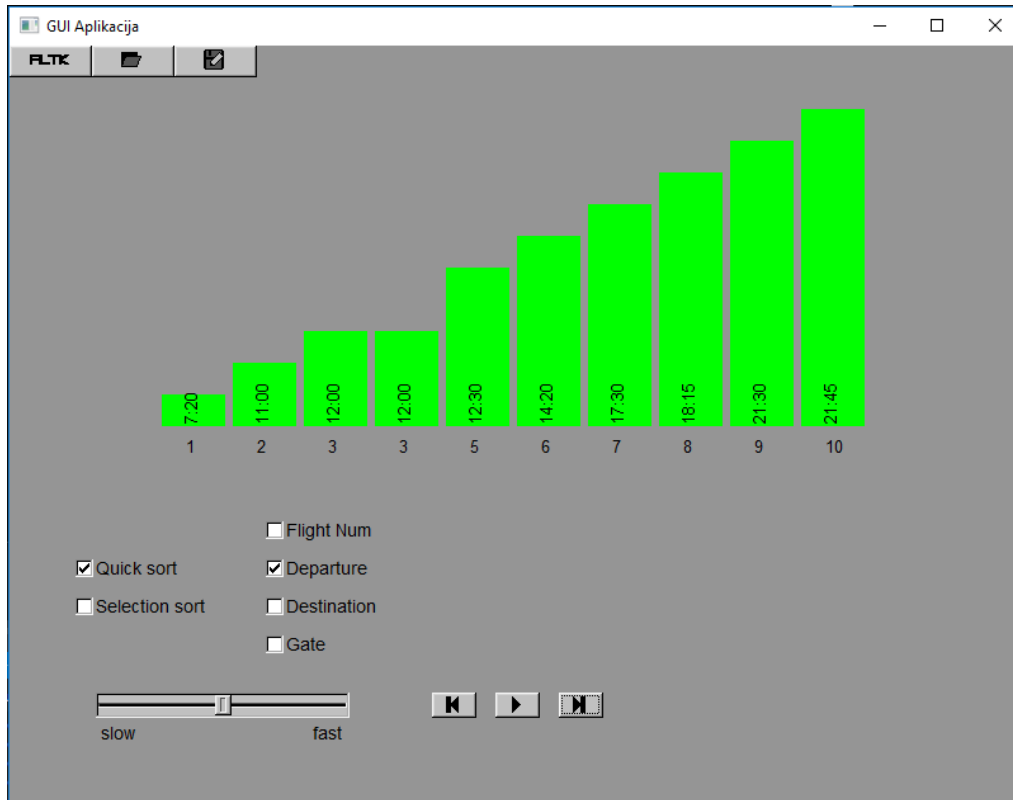


- Тест 03 – пивот елемент у првом рекурзивном позиву је средишњи елемент

Структура улазне датотеке :

Flight number	Departure	Destination	Gate
BD001	12:30	Frankfurt	B1
AA220	7:20	Barbados	B2
AA130	11:00	Abu Dhabi	A5
BC325	12:00	Dubai	C2
AA150	12:00	Copenhagen	A1
CD425	14:20	Jerusalem	C1
CC322	18:15	Rome	A7
BD215	21:45	Genoa	B3
DM321	17:30	Helsinki	C2
DM400	21:30	Stara Pazova	A7

Изглед програма након сортирања ових летова у односу на време поласка:

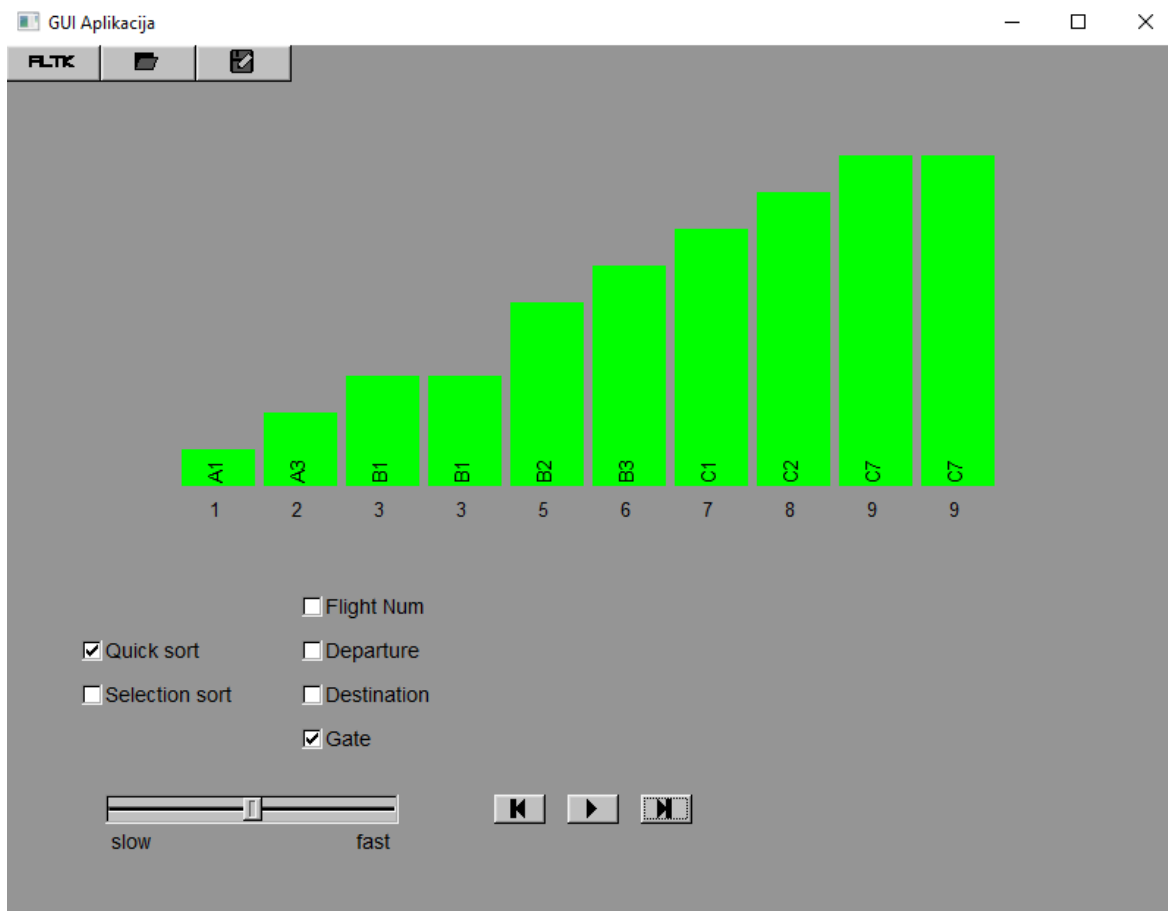


➤ Тест 04 – скоро сортирана колекција летова

Структура улазне датотеке :

Flight number	Departure	Destination	Gate
AA130	11:00	Abu Dhabi	A1
AA220	7:20	Barbados	B1
AA150	12:00	Copenhagen	A3
BC325	12:00	Dubai	B2
BD001	12:30	Frankfurt	B1
BD215	21:45	Genoa	B3
CD425	14:20	Jerusalem	C1
DM321	17:30	Helsinki	C2
CC322	18:15	Rome	C7
DM400	21:30	Stara Pazova	C7

Изглед програма након сортирања ових летова у односу на број капије:



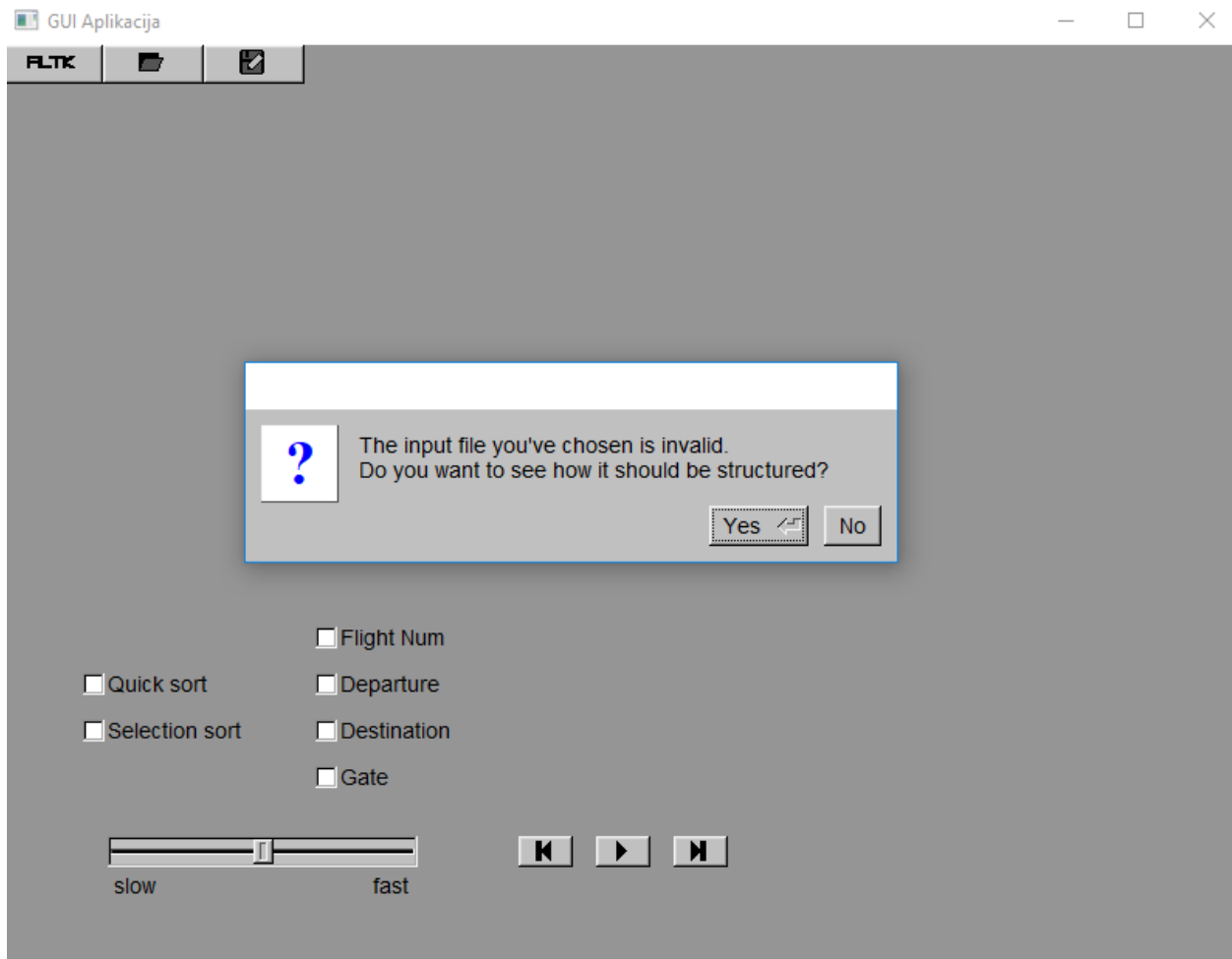


➤ Тест 05 – празна улазна датотека

Структура улазне датотеке :

Flight number	Departure	Destination	Gate

Изглед програма након избора ове датотеке:



Избором “**Yes**” опције се отвара овај фајл и корисник може прочитати како да структурира улазну датотеку.

## X Излазни резултати

### ➤ Тест01 :

Selection sort	Број поређења	Број замена
Број лета	45	7
Време поласка	45	5
Дестинација	45	3
Капија	45	5
Просек	45	5

Quick sort	Број поређења	Број замена
Број лета	25	9
Време поласка	45	5
Дестинација	31	3
Капија	31	7
Просек	33	6

Као што можемо приметити, **Quick sort** има знатно мање поређења у односу на **Selection sort** уз занемарљиво већи број замена елемената.

Али при сортирању у односу на време поласка, оба алгоритма имају исте перформансе, из разлога зато што ће пивотски елемент у сваком рекурзивном позиву бити највећи или најмањи елемент и **Quick sort** неће моћи да искористи своју предност што бирањем добром пивот елемента дели колекцију на два дела која су скоро сортирана и тим убрза процес сортирања.

➤ Тест02 :

Selection sort	Број поређења	Број замена
Број лета	190	18
Време поласка	190	18
Дестинација	190	15
Капија	190	15
Просек	190	16.5

Quick sort	Број поређења	Број замена
Број лета	58	24
Време поласка	70	31
Дестинација	59	26
Капија	87	18
Просек	68.5	24.75

Тек сад са више од 10 елемената можемо видети колико је уствари **Quick sort** потребно мање поређења да би колекцију сортирао.

Опет можемо приметити да **Quick sort** има већи број замена елемената, али у односу на побољшање при броју поређења можемо их заменарити.

➤ Тест03

Selection sort	Број поређења	Број замена
Број лета	45	4
Време поласка	45	7
Дестинација	45	4
Капија	45	6
Просек	45	5.25

Quick sort	Број поређења	Број замена
Број лета	19	4
Време поласка	21	6
Дестинација	19	6
Капија	19	9
Просек	19.5	6.25

Број поређења за **Selection sort** је стандардано  $\frac{n^2-n}{2}$ , али перформансе **Quick sort**-а су сада боље чак и од  $n * \log n$ .

То смо постигли бирањем средишњег елемента за пивот у првом рекурзивном позиву.

➤ Тест04

Selection sort	Број поређења	Број замена
Број лета	45	3
Време поласка	45	5
Дестинација	45	1
Капија	45	2
Просек	45	2.75

Quick sort	Број поређења	Број замена
Број лета	36	3
Време поласка	37	5
Дестинација	43	1
Капија	33	2
Просек	37.25	2.75

Још у првом тест случају смо видели да **Quick sort** има проблем са већ сортираним колекцијама.

Сада када имамо скоро сортирану колекцију ситуација је слична, те можемо закључити да што је колекција “сортиранија” то **Quick sort** све више почиње да личи на **Selection sort**, и када је колекција потпуно сортирана алгоритми раде идентично, с тим што се **Quick sort** позива рекурзивно што је за рачунар захтевније од итеративног приступа **Selection sort**-а.

## XI Уочени проблеми и ограничења

Први проблем је било усклађивање звучних ефеката како би визуализација осим графичке имала и звучну компоненту.

Наиме, пошто је брзина визуализације кориснички задата променљива вредност, проблем је био ускладити звук тако да он подједнако прецизно ради на оба спектрума брзине.

Оптимално решење би била и променљива дужина звучног сигнала у зависности од брзине визуализације, али за сада то није имплементирано.

Звук је подешен тако да најбоље ради за неке средње вредности брзине визуализације, те да на крајевима спектрума буде подједнако лош.

Други проблем је представљао губљење фрејмова од стране главног прозора при брзој визуализацији.

Приликом саме визуализације се понекад може приметити, како се прозор исцртава услед неколико узастопних позива `Fl::flush()` методе.

Овај проблем би се могао решити употребом `multithreading` -а али услед недовољних знања о тој теми као и временског ограничења то такође још увек није имплементирано.

## Ограничења:

1. Коришћење ФЛТК библиотеке за графички део
2. Улазна датотека мора задовољавати структуру иначе летови неће бити учитани (заглавље датотеке је опционо)
3. Пошто се ширина widget-а рачуна као број летова подељен са 500, ако се у датотеци налази више од 500 летова, видгети ће бити ширине мање од 1 (неће бити видљиви на прозору)
4. Иако ће програм подржати фајлове са до 500 летова, већ приликом бројке од око отприлике 100 долази до проблема да су widget-i прилично мали и није лако видети који лет они представљају
5. Алгоритми су само **Selection sort** и **Quick sort**
6. Имамо само 4 податка о појединачном лету те и 4 критеријума сортирања