

УНИВЕРЗИТЕТ У БЕОГРАДУ  
МАТЕМАТИЧКИ ФАКУЛТЕТ



Бојан Барџић

ИМПЛЕМЕНТАЦИЈА ТЕКСТ ЕДИТОРА ЗА  
ПИСАЊЕ КОДА

мастер рад

Београд, 2024.

**Ментор:**

др Весна МАРИНКОВИЋ, доцент  
Универзитет у Београду, Математички факултет

**Чланови комисије:**

др Милан БАНКОВИЋ, доцент  
Универзитет у Београду, Математички факултет

др Иван ЧУКИЋ, доцент  
Универзитет у Београду, Математички факултет

**Датум одбране:** 15. јануар 2016.



## Наслов мастер рада: Имплементација текст едитора за писање кода

**Резиме:** Фијуче ветар у шибљу, леди пасаже и куће иза њих и гунђа у оца-цима. Ницо, чежњиво гледаш фотелју, а Ђура и Мика хоће позицију себи. Људи, јазавац Џеф трчи по шуми глођући неко сухо жбуње. Љубави, Олга, хајде поћи у Фуци и чут ћеш њежну музику срца. Боја ваше хаљине, госпо-ђице Џафић, тражи да за њу кулучим. Хаџи Ђера је заћутао и бацио чежњив поглед на шољу с кафом. Џабе се зец по Хомолу шуња, чувар Јожеф лако ће и ту да га нађе. Очачар Филип шаље осмехе туђој жени, а његова кућа без деце. Бутић Ђуро из Фоче има пун џак идеја о слагању ваших жељица. Џајић одскочи у аут и избеже Ђон халфа Пецеља и његов шамар. Пламте оцаци фабрика а чађаве гује се из њих дижу и шаљу ноћ. Ајшо, лепото и чежњо, за љубав срца мога, дођи у Хаџиће на кафу. Хучи шума, а иза жутог цбуна и пања ђак у цвећу деље сеји фрулу. Гоци и Јаћиму из Бање Ковиљаче, флаша цина и жеђ падаху у исту уру. Џаба што Феђа чупа за косу Миљу, она јури Живу, али њега хоће и Даца. Док је Фехим у ципу журно љубио Загу Чађевић, Циле се ушуњао у ауто. Фијуче кошава над оцацима а Иља у гуњу журећи уђе у суху и топлу избу. Боже, центлмени осећају физичко гађење од прљавих шољица! Дочепаће њега јака шефица, вођена љутом срц-бом злих жена. Пази, гецо, брже однеси шефу тај ђавољи чек: њим плаћа цех. Фине цукце озлеђује бич: одгој их пажњом, стрпљивошћу. Замишљао би кафецију влажних прстића, црнег од чађи. Ђаче, уштеду плаћај жаљењем због циновских цифара. Цикљаће жалфија између тог бусења и пешчаних двораца. Зашто гђа Хаџић лечи живце: њена љубав пред фијаском? Јеж хоће пецкањем да вређа љубичастог цина из флаше. Џеј, љубичаст зец, лаже: гађаће одмах поквашен фењер. Плашљив зец хоће јефтину дињу: грожђе искамчи џабе. Џак је пун жица: чућеш тад свађу због ломљења харфе. Чуј, цукац Флоп без даха с гађењем жваће стршљена. Ох, задњи шраф на ципу слаб: муж гђе Цвијић љут кочи. Шеф џабе звиждуће: млађи хрт јаче кљуца њеног пса. Одбациће кавгација плаштом чађ у жељезни фењер. Дебљи кро-јач: згужвах смеђ филц у тањушни цепић. Џо, згужваћеш тихо смеђ филц најдебље крпењаче. Штеф, бацих сломљен дечји зврк у цеп гђе Жунђић. Де-бљој згужвах смеђ филц — њен шкрт цепчић.

**Кључне речи:**

# Садржај

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Увод</b>                                     | <b>1</b>  |
| 1.1      | Текст едитори . . . . .                         | 1         |
| <b>2</b> | <b>Структуре података у текст едиторима</b>     | <b>2</b>  |
| 2.1      | Бафер са размацима . . . . .                    | 2         |
| 2.2      | Уже . . . . .                                   | 4         |
| 2.3      | Табела делова . . . . .                         | 9         |
| <b>3</b> | <b>Основне функционалности</b>                  | <b>14</b> |
| 3.1      | Обрада уноса . . . . .                          | 14        |
| 3.2      | Курсор . . . . .                                | 16        |
| 3.3      | Операције са појединачним карактерима . . . . . | 18        |
| 3.4      | Селекција текста . . . . .                      | 19        |
| 3.5      | Управљање текстуалним датотекама . . . . .      | 20        |
| <b>4</b> | <b>Закључак</b>                                 | <b>24</b> |
|          | <b>Библиографија</b>                            | <b>26</b> |

# Глава 1

## Увод

### 1.1 Текст едитори

Текст едитор је програм за измену текстуалних датотека. Најчешћи типови датотека који се измењују коришћењем ових програма су једноставне текстуалне датотеке, датотеке које садрже изворни код, код језика за означавање као и конфигурационе датотеке. Неки од најпознатијих оваквих програма су *Visual Studio Code* [1], *Notepad* [4], *Notepad++* [3], *VIM* [5] и *Emacs* [2].

Постоји више врста текст едитора. Постоје едитори једноставног текста, где информација о датотеци представља само текст. Док такође постоје и едитори богатог текста, где информација о датотеци поред текста садржи и неке додатне информације везано за изглед текста (фонт, величина фонта, боја текста, маргине). Овај рад ће се бавити искључиво едиторима једноставног текста.

## Глава 2

# Структуре података у текст едиторима

Текст у датотеци се може посматрати као линеарни низ карактера, тако се и свака измена над тим текстом може посматрати као додавање текста у неки део низа или брисање подниза текста. Када би комплетан садржај текстуалне датотеке која је отворена чувана као јединствен низ карактера, операције уметања и брисања текста биле би временски јако скупе ( $O(n)$ , где је  $n$  дужина текста) и њихово узастопно извршавање над неким великим текстом би имало за последицу изузетну неефикасност текст едитора. Како би се овај проблем превазишао осмишљене су различите структуре података које ове операције врше ефикасније. Најпознатије овакве структуре су бафер са размацама (енг. *gap buffer*), уже (енг. *rope*) и табела делова (енг. *piece table*).

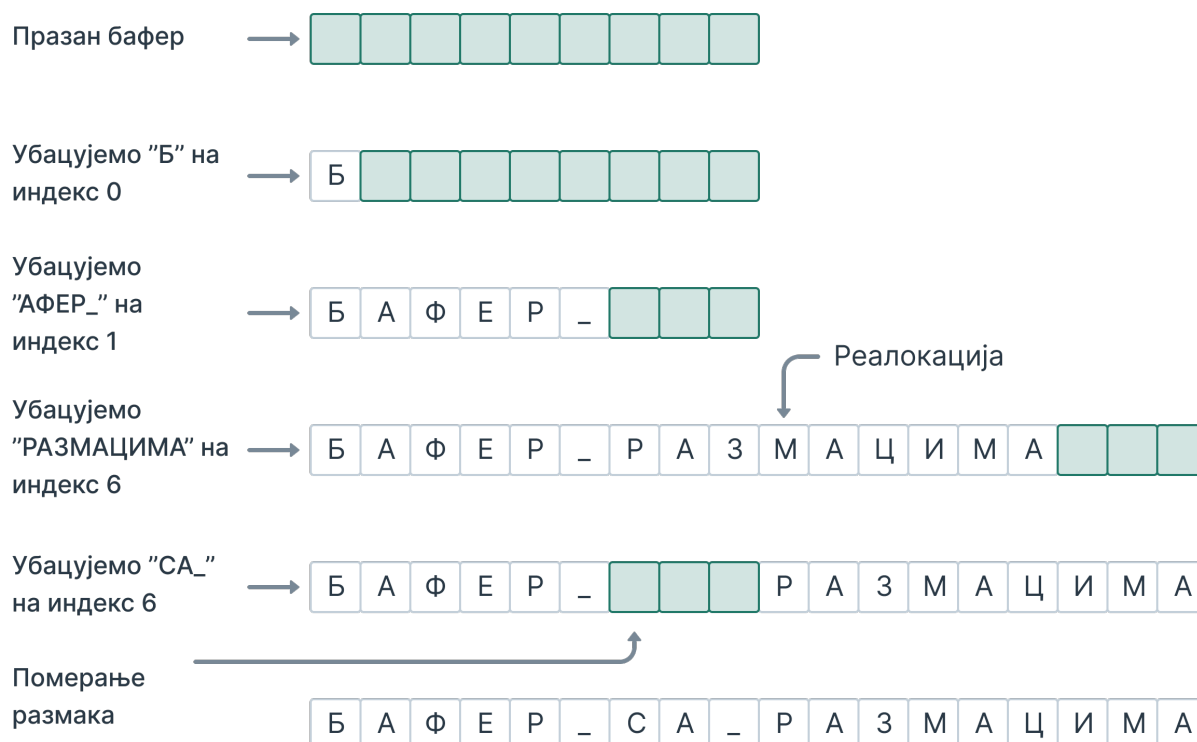
### 2.1 Бафер са размацама

Бафер са размацама (енг. *gap buffer*) је структура података која се заснива на идеји да постоји један линеаран низ чија је средина „празна“, док се са леве и десне стране налази текст. Како се врше измене над неким делом текста тако се средина „помера“ превлачењем последњег елемента леве стране на почетак десне или обрнуто. Када се бафер попуни (размак није довољно велики за нову операцију), тада се алоцира нови бафер већих димензија, најчешће дупло већи, и у њега се копира текст из старог бафера.



У односу на класичан низ карактера, бафер са размацима је доста ефикаснији јер не захтева реалокацију низа при свакој измени. Ефикасност операција уметања и брисања зависи највише од тога колики је размак између индекса на коме се врши измена текста и леве или десне границе размака. Ако је размак већи на потребној позицији, временска сложеност је  $O(1)$ . Међутим, ако је размак на једном крају а потребно је да се мења други крај текста сложеност ће бити  $O(n)$ . У просечном случају временска сложеност наведених операција је константна, јер су карактери најчешће писани један за другим. Треба напоменути и то да је с времена на време потребно вршити реалокацију низа, која је линеарне сложености.

Разматрана структура података је једноставна за имплементацију и често се користи за једноставне текстуалне уносе. Познати текст едитор *Emacs* [2] користи ову структуру у својој имплементацији. На слици 2.1 приказана је илустрација рада бафера.



Слика 2.1: Илустрација бафера са размаком

У првом кораку бафер је празан. У следећем кораку се додаје ниска "g" на почетну позицију. После тога на крај текста се додаје ниска "ap\_", а затим и ниска "buffer". У последњем кораку се додаје ниска "between" на индекс број 6. Пошто је бафер у том тренутку пун, врши се реалокација. Када се текст прекопира у нови низ, помера се почетак празног дела на жељени индекс и додаје се нова ниска.

## 2.2 Уже

Уже (енг. *rope*) је бинарно стабло у коме сви чворови који нису листови садрже број карактера у левом подстаблу тог чвора. У листовима се налазе ниске које садрже делове текста. Када се прође кроз листове од првог листа слева, добија се целокупан текст.

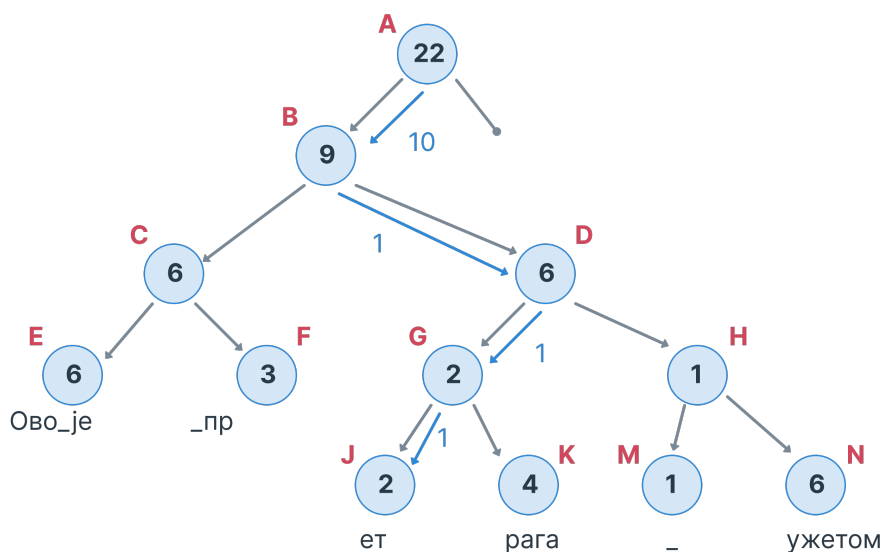
Предност ове структуре у односу на обичну ниску је што операције као што су уметање, брисање и претрага текста у просечном случају захтевају  $O(\log n)$  време (где је  $n$  дужина текста), а не  $O(n)$ . Такође, није потребно чувати текст у непрекидном делу меморије, већ се чворови могу налазити на одвојеним местима.

Мане ужета јесу што је то комплексна структура и чешће су грешке како у раду са њом, тако и при њеној имплементацији. Поред тога, заузима више меморије него обична ниска (због родитељских чворова који повезују листове).

### Претрага

Приступ карактеру на  $i$ -том индексу се врши тако што се рекурзивно пролази кроз стабло почевши од корена. Ако је индекс мањи од вредности у текућем чвору, настављамо претрагу у његовом левом подстаблу. У супротном од  $i$  се одузима вредност у тренутном чвору и наставља се претрага у његовом десном подстаблу. Када се наиђе на лист враћа се  $i$ -ти карактер из ниске која се налази у чвору. Сложеност операције је иста као код претраге бинарног стабла, а то је  $O(\log n)$ . Пример претраге дат је на слици 2.2.

Тражи се карактер на индексу број 10. Прво се обилази корен који садржи дужину укупног текста, пошто је тражени индекс мањи од вредности у тре-



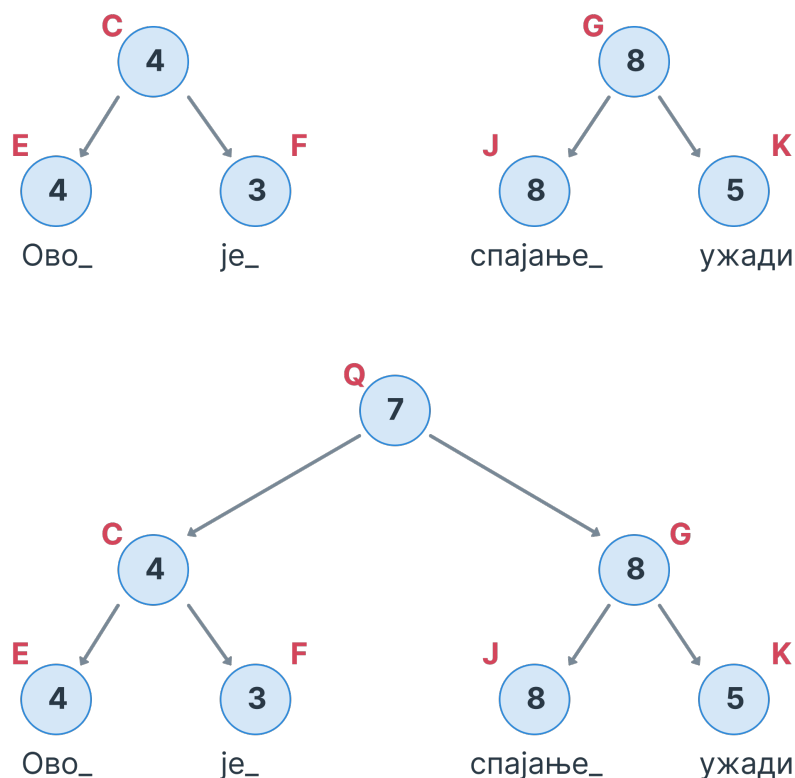
Слика 2.2: Претрага помоћу ужета

нутном чвору иде се у његово лево подстабло. Следећи чвор садржи вредност 9, што значи да је тражени индекс већи од вредности. Одузима се та вредност од траженог индекса и претрага се наставља у десном подстаблу тренутног чвора. Сада је тражени индекс 1 и он се поново упоређује са тренутним чвором. Пошто је 1 мање од 6 прелази се у лево подстабло. У следећем чвору вредност је 2 па се поново иде у лево подстабло. На крају се наилази на лист и у нисци коју он садржи се дохвата карактер на индексу 1 и прослеђује се као повратна вредност.

Пре него што буде обрађена операција брисања, прво ће бити уведене две нове операције које ће бити потребне за њену реализацију. Прва операција је надовезивање једног ужета на друго, док је друга операција дељење једног ужета на два нова ужета по неком карактеру.

## Конкатенација

Конкатенација или надовезивање је операција којом се на уже, које садржи текст  $t_1$ , надовезује друго уже које садржи текст  $t_2$  и добија се ново уже које садржи текст  $t_1t_2$ . Конкатенација два ужета,  $r_1$  и  $r_2$ , се врши тако што се направи нови родитељски чвор чије ће лево дете бити корен од  $r_1$ , а десно корен од  $r_2$ . Вредност новог чвора ће бити сума дужина ниски у



Слика 2.3: Конкатенација два ужета

свим листовима ужета  $r_1$ . Да би израчунали вредност новог корена потребно је израчунати суму дужина ниски свих листова који се налазе у  $r_1$ . То се постиже рекурзивним обиласком  $r_1$ , где се на коначну суму додаје вредност тренутног чвора и затим се рекурзивно обилази десно подстабло тренутног чвора. Сложеност ове операције је  $O(\log n)$  за балансирано стабло. Пример конкатенације два ужета се може видети на слици 2.3

## Дељење

Дељење ниске  $s$  по индексу  $i$  на две ниске,  $s_1$  и  $s_2$ , где  $s_1$  садржи карактере од почетка ниске  $s$  до  $i$ -тог индекса (укључујући и карактер на  $i$ -том индексу), а  $s_2$  садржи карактере десно од  $i$ -тог индекса па до краја ниске, се врши на следећи начин.

Најпре се налази  $i$ -ти карактер, ако је то последњи карактер ниске у листу онда се не ради ништа. У супротном се дели лист на два нова листа, где је  $i$ -ти карактер последњи карактер ниске левог листа. Нека је лист чији је  $i$ -ти карактер последњи у нисци означен са  $d$ . Сада се листови деле у две групе  $l_1$  и  $l_2$ . У  $l_1$  се налазе листови лево од  $d$  као и сам  $d$ , док се у  $l_2$  налазе листови десно од  $d$ . Листови из се  $l_2$  одвајају од главног ужета и затим се спајају заједно. Алгоритам се завршава тако што се балансирају два новодобијена ужета.

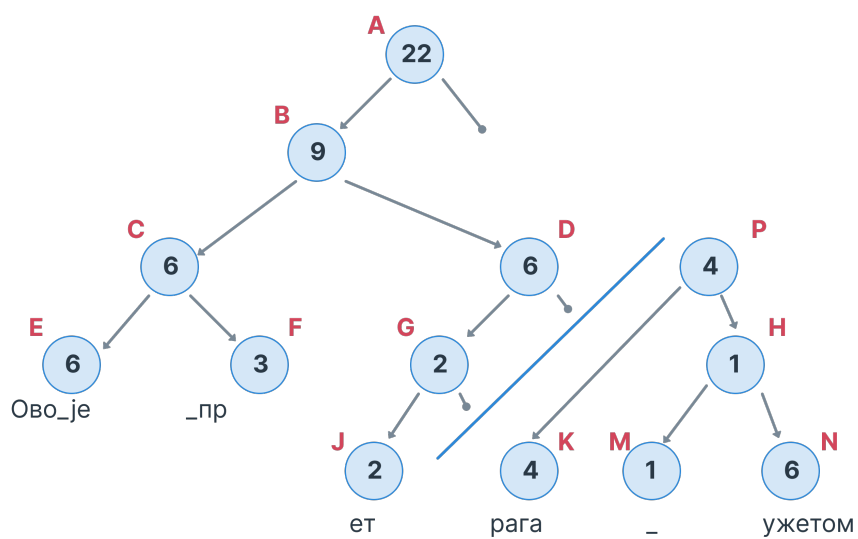
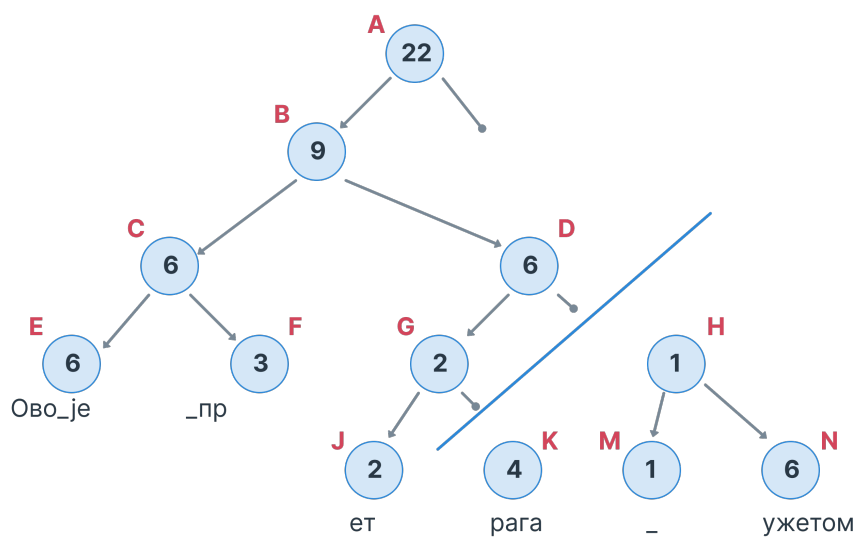
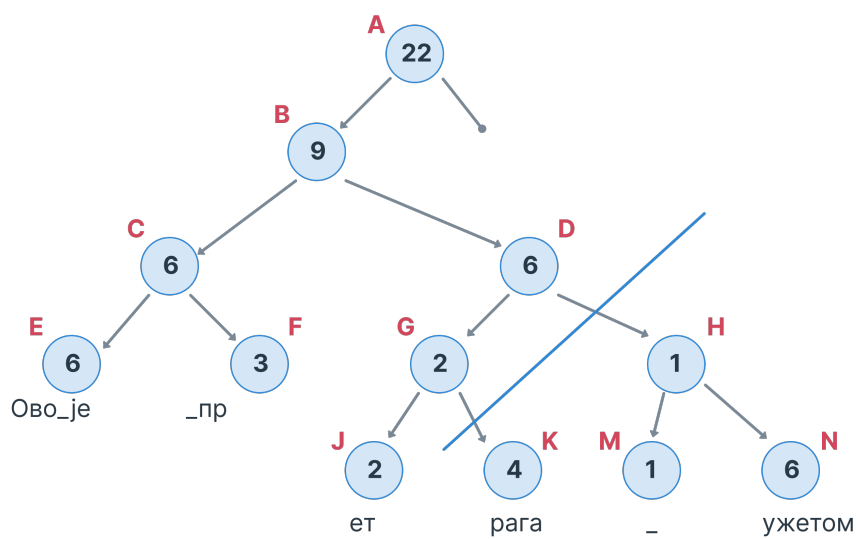
Сложеност је иста као код претходних операција. На слици 2.4 се може видети пример дељења. Дељење се врши по индексу 10. Прво се проналази карактер на датом индексу на начин описан у делу о претрази. Затим се проверава да ли је тај карактер последњи у листу, пошто јесте, врши се подела на  $l_1$  и  $l_2$  по том листу.

### Уметање

Да би се нека ниска  $s$  уметнула у уже  $r$  на индексу  $i$ , довољно је да се искористе претходно дефинисане операције дељења и конкатенације. Прво се уже  $r$  подели по индексу  $i$  и добију се два ужета  $r_1$  и  $r_2$ , затим се на  $r_1$  надовеже ниска  $s$  и добија се ново уже  $r_3$ . На крају се на уже  $r_3$  надовеже  $r_2$ . Ова операција се састоји од три операције сложености  $O(\log n)$ , тако да је њена укупна сложеност  $O(\log n)$ .

### Брисање

Уколико је потребно обрисати сегмент ниске  $s$  смештене у ужету  $r$  која почиње на  $i$ -том карактеру, а завршава се на  $(i + l)$ -том карактеру, онда се то може урадити у три корака. Прво се изврши дељење ужета  $r$  по индексу  $i$  на два ужета  $r_1$  и  $r_2$ , затим се  $r_2$  подели по  $l$ -том индексу на  $r_3$  и  $r_4$ . Последњи корак је надовезивање  $r_1$  и  $r_4$ . Из истих разлога као код уметања, сложеност операције је  $O(\log n)$ .



Слика 2.4: Делѐње ужета

### 2.3 Табела делова

Табела делова (енг. *piece table*) је структура података која се састоји од два бафера у којима се налази текст и повезане листе чворова који показују на текст у баферу. Текст едитор који је имплементиран током рада на овој тези користи ову структуру података, тако да ће бити детаљније описана него претходне две структуре података.

У први бафер се учитава текст који се већ налазио у датотеци коју смо отворили и тај бафер се назива оригинални бафер (енг. *original buffer*). Од тренутка после учитавања текста из датотеке па надаље он остаје непромењен.

У други бафер, који се назива бафер за додавање (енг. *add buffer*), се додаје текст који се током времена уписује у текст едитор. Сваки пут када се додаје текст, без обзира где се додаје, он се додаје на његов крај. Важно је напоменути да када се брише текст, он се не брише из бафера за додавање, већ остаје у њему. На овај начин никада нема потребе за померањем текста који се већ налази унутар бафера.

Поставља се питање како је онда могуће исписати текст у правилном редоследу. То се постиже коришћењем двоструко повезане листе чији су елементи тзв. дескриптори делова (енг. *piece descriptors*). Сваки дескриптор садржи информацију о томе на који од два бафера показује, на којем индексу бафера почиње текст и колика је дужина тог текста. Следећи код приказује чланске променљиве класе *PieceDescriptor* у имплементацији рада.

```
class PieceDescriptor {  
    // ...  
private:  
    SourceType m_source;  
    size_t m_start;  
    size_t m_length;  
};
```

Предност ове структуре је што је једноставним операцијама, уметањем и брисањем чворова из повезане листе, могуће ефикасно вршити измене текста.

Мана је потенцијално велико меморијско заузеће бафера за додавање као и фрагментација на доста веома малих „делова”, што чини претрагу кроз листу мање ефикасном.

У наставку текста ће бити описане основе операције над овом структуром података. Најпре ће бити уведене неке ознаке које ће бити употребљаване за све наведене операције:

- $n$  - дужина укупног текста,
- $m$  - број елемената повезане листе,
- $I_n \in \{0, 1, \dots, n - 1\}$  - скуп валидних индекса текста,
- $I_m \in \{0, 1, \dots, m - 1\}$  - скуп валидних индекса повезане листе,
- $p_i$  - почетак  $i$ -тог дескриптора, где је  $i \in I_m$ ,
- $d_i$  - дужина  $i$ -тог дескриптора, где је  $i \in I_m$ .

### Уношење текста

Ако се додаје неки текст дужине  $d$  у текући текст почев од позиције  $i \in \{0, 1, \dots, n\}$ , прво ће бити размотрена два специјалана случаја за индекс  $i$ :

1.  $i = 0$ : У овом случају се додаје нови дескриптор на почетак листе.
2.  $i = n$ : У овом случају се додаје нови дескриптор на крај листе.

За све остале случајеве пролази се кроз листу слева на десно. Нека је сума дужина свих дескриптора пре  $j$ -тог, при чему је  $j \in I_m$ , једнака  $s_j$ . Зауостављамо се када буде важио услов  $s_j + d_j \geq i$ , где је  $j$  индекс елемента листе на ком се налазимо. Разликују се два случаја:

1.  $s_j + d_j = i$ : У овом случају убацује се нови дескриптор испред  $j$ -тог елемента повезане листе дескриптора.
2.  $s_j + d_j > i$ : У овом случају се дели тренутни дескриптор на два тако да леви садржи  $i - s_j$  почетних карактера оригиналног дескриптора, а десни остале. Ово се постиже тако што се дужина  $j$ -тог дескриптора



поставља на  $i - s_j$ , затим се прави нови дескриптор чији је почетак  $p_j + (i - s_j)$ , а дужина  $d_j - (i - s_j)$  и умеће се испред  $j$ -тог. На крају се додаје дескриптор са новим текстом између ова два.

Сложеност ове операције је  $O(m + d)$ , јер се кроз листу пролази највише  $m$  пута. Додавање новог дескриптора и дељење претходног на два су операције сложености  $O(1)$ , док додавање новог текста дужине  $d$  у бафер има сложеност  $O(d)$ .

## Брисање

Брисање неког дела текста чији се индекси налазе у целобројном интервалу  $[p, k)$ , где су  $p \in I_n$ ,  $k \in I_n \cup \{n\}$ , се врши на аналоган начин као код додавања. Пролази се кроз повезану листу све док не буде важио услов  $s_i + d_i \geq p$  за неко  $i \in I_m$ . Затим се редом пролази кроз све дескрипторе који садрже текст чији су индекси из опсега  $[p_j, p_j + d_j)$  за  $j \geq i$ ,  $j \in I_m$  и имају пресек са  $[p, k)$  и ажурирамо их по следећим правилима:

1. Ако је пресек суфикс опсега  $[p_j, p_j + d_j)$  дужине  $d_p$ , онда се одузима суфикс од дескриптора тако што му се смањује дужина за  $d_p$ .
2. Ако је пресек префикс опсега  $[p_j, p_j + d_j)$  дужине  $d_p$ , онда се одузима префикс од дескриптора тако што му се помера почетак у десно за  $d_p$  и смањује дужина за исто толико.
3. Ако је пресек цео опсег  $[p_j, p_j + d_j)$ , онда се брише цео дескриптор.
4. Ако је  $[p, k)$  садржан у  $[p_j, p_j + d_j)$  и није ни префикс ни суфикс опсега, онда се дескриптор дели на два нова, где први садржи опсег  $[p_j, p)$ , а други  $[k, p_j + d_j)$ .

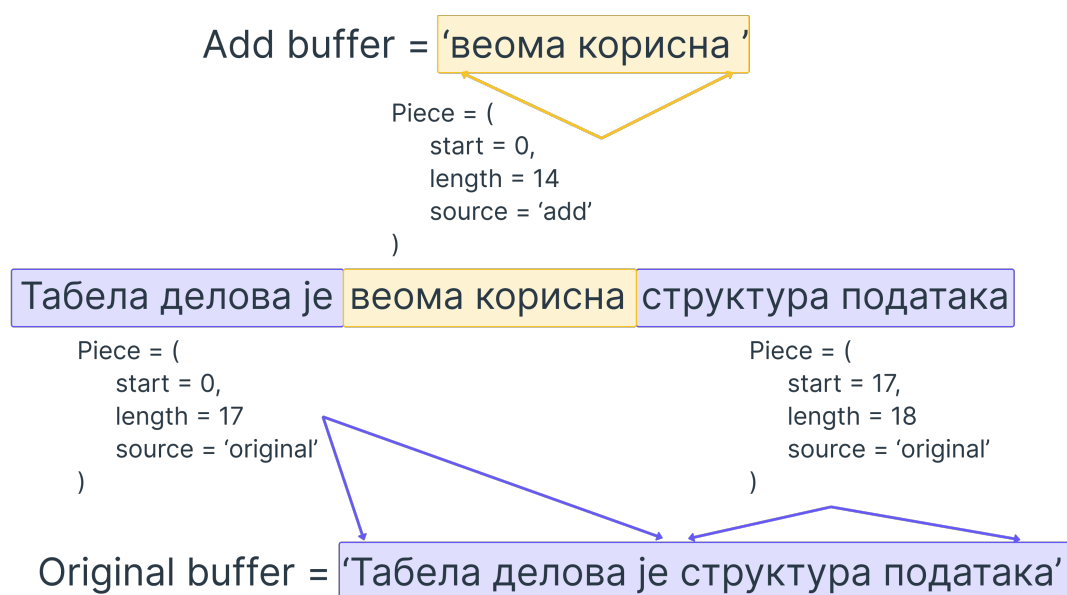
Пошто се брисањем мења један или више дескриптора у повезаној листи, треба приметити нека правила везана за горе наведене случајеве. Ако се мења више дескриптора у једном брисању, онда се они морају налазити на узастопним позицијама у повезаној листи. Случај 1. може важити само за последњи дескриптор који ће бити промењен у брисању. Аналогно, случај 2. може важити само за први који ће бити промењен, док случај 3. може важити

за све које ће се променити. Уколико за неки дескриптор важи случај 4, онда је он једини дескриптор који ће се ажурирати.

Временска сложеност је  $O(m)$ , јер се пролази највише  $m$  пута кроз листу и свако ажурирање дескриптора је сложености  $O(1)$ .

## Исписивање

Уколико је потребно да се испише целокупан текст на стандардни излаз или у неку датотеку, то се постиже помоћу следеће процедуре. Пролази се кроз листу дескриптора од почетка до краја и за сваки дескриптор се исписује подниска одговарајућег бафера која починје на индексу  $p_j$  и има дужину  $d_j$ , где је  $j \in I_m$ . На слици 2.5 се може видети како изгледа приказ једног текста помоћу табеле делова.



Слика 2.5: Приказ рада табеле делова

По овом механизму где се целокупан текст добија тако што надовезујемо „делове” бафера је сама структура података добила име. Један од познатијих текст едитора који од 2018. користи табелу делова у својој имплементацији је *Visual Studio Code* [1].

Сложеност ове операције је  $O(\sum_{i=0}^{m-1} d_i)$  тј једнака је суми дужина свих дескриптора. Пошто је сума свих дужина једнака дужини укупног текста, онда се сложеност може једноставније записати као  $O(n)$ .

## Глава 3

# Основне функционалности

### 3.1 Обрада уноса

Када се користе текст едитори, корисник већину команди задаје преко тастатуре, док је мањина задата преко миша. Један од проблема при имплементацији текст едитора јесте како исправно обрадити све овакве уносе. Пре описа обраде, прво ће бити речи о самим карактерима и њиховим специфичностима.

#### Карактери

За писање текста у програмским језицима се најчешће користи *ASCII* табела. Она кодира један бајт у одговарајући карактер у табели, тј. постоји 128 различитих карактера.

Немају сви карактери из *ASCII* табеле визуелну репрезентацију, тј. не могу сви да се штампају на излаз. Карактери који имају вредност у табели од 0 до 31 су тзв. контролни карактери (енг. *control characters*) који служе за слање сигнала за комуникацију са периферним уређајима. Остале карактере, који имају вредност од 32 до 127, се могу писати на излаз и њих називамо штампаним карактерима (енг. *printable characters*). На табели 3.1 се може видети њен садржај.

|    | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9  | A   | B   | C  | D  | E  | F   |
|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|-----|-----|----|----|----|-----|
| 0x | nul | soh | stx | etx | eot | enq | ack | bel | bs  | ht | lf  | vt  | ff | cr | so | si  |
| 1x | dle | dc1 | dc2 | dc3 | dc4 | nak | syn | etb | can | em | sub | esc | fs | gs | rs | us  |
| 2x | sp  | !   | ”   | #   | \$  | %   | &   | '   | (   | )  | *   | +   | ,  | -  | .  | /   |
| 3x | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9  | :   | ;   | <  | =  | >  | ?   |
| 4x | @   | A   | B   | C   | D   | E   | F   | G   | H   | I  | J   | K   | L  | M  | N  | O   |
| 5x | P   | Q   | R   | S   | T   | U   | V   | W   | X   | Y  | Z   | [   | \  | ]  | ^  | _   |
| 6x | ‘   | a   | b   | c   | d   | e   | f   | g   | h   | i  | j   | k   | l  | m  | n  | o   |
| 7x | p   | q   | r   | s   | t   | u   | v   | w   | x   | y  | z   | {   |    | }  | ~  | del |

Табела 3.1: ASCII табела.

## Унос помоћу тастатуре

Уколико корисник притисне неки тастер, текст едитор ће га исписати само ако тастер кодира штампани или карактер за форматирање текста (као што су *Tab* и *Enter*). Сви остали тастери су игнорисани и то омогућава да се неки од њих кодирају за друге команде у едитору. Поред тога, могуће је комбиновање контролног и штампаног карактера да би се кодирале додатне команде. Ово се постиже притиском на два или три тастера истовремено. Најчешће су комбинације два тастера, где је први обично *Ctrl*.

Имплементацију обраде ових уноса врши метода *handleKeyboardInput()* класе *TextEditor*. тако што се стално проверава да ли је притиснут један или више тастера у датом тренутку и да ли се тај унос мапира у неку специјалну команду. Уколико јесте, извршава се одговарајућа команда. Ако није притиснута ниједна, гледа се да ли је притиснут неки штампани карактер, уколико јесте, прослеђује се методи за унос.

## Унос помоћу миша

Унос команди помоћу миша је мањи и своди се на померање курсора и селекцију текста. Такође се помоћу менија могу активирати додатне команде и алати. О курсору и селекцији, као и додатним командама ће бити више речи

у наставку рада. Сигнали који се обрађују су тастер миша који је притиснут (леви, десни и средњи), позиција курсора миша као и то да ли корисник користи један клик, дупли клик или држи тастер миша.

Имплементација је сложенија него код обраде уноса тастатуре. Метода *handleMouseEvent()* класе *TextEditor* обрађује све улазе са миша. Она се стално позива и проверава се да ли је корисник притиснуо леви тастер на мишу, уколико јесте активира се одговарајућа команда и прослеђује се позиција миша. Такође се проверава да ли је корисник држао леви тастер и уколико јесте, активира одговарајућу методу и шаље две позиције миша, на почетку и крају држања.

### 3.2 Курсор

Једна од основних ствари у сваком текст едитору јесте курсор. Он служи да прикаже на ком делу текста ће бити извршена наредна промена. Углавном је представљен као пар који чине ред и колона на којој се курсор налази. Иако су у овом раду до сада измене представљане помоћу индекса неког низа карактера, оваква репрезентација је интуитивнија и прегледнија за кориснике. Пример овакве репрезентације се може видети у класи *TextCoordinates* текст едитора.

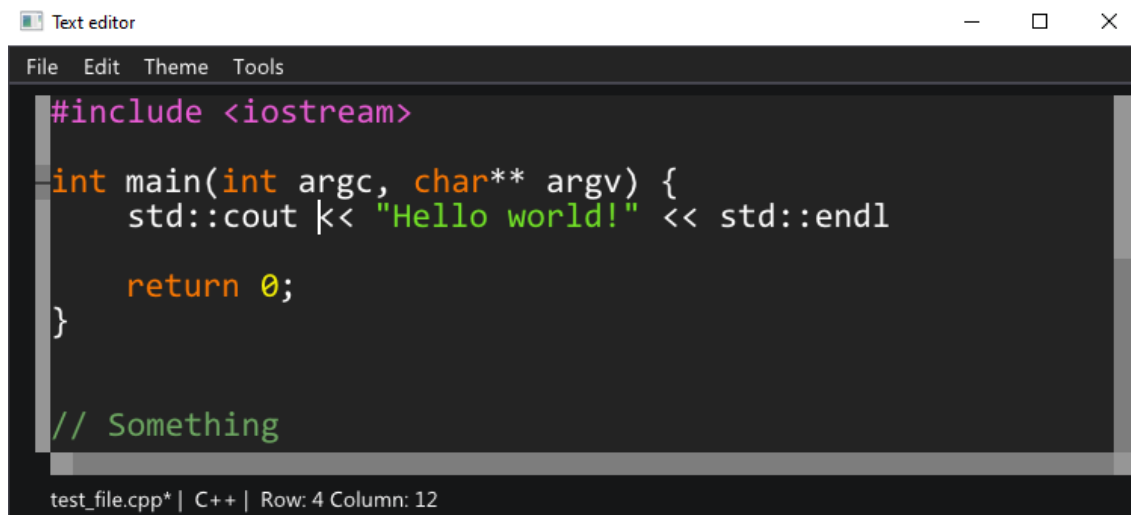
```
class TextCoordinates {  
    // ...  
public:  
    size_t m_row;  
    size_t m_col;  
};
```

Такође треба напоменути да се у овој репрезентацији обе координате почињу од један, а не од нула као код индекса, као и да се вредност колоне може кретати у целобројном интервалу  $[1, d + 1]$ , где је  $d$  дужина тренутног реда на ком се курсор налази.

Курсор се на већини данашњих рачунара помера позиционирањем миша на жељено место курсора и притиском на леви тастер или помоћу стрелица

на тастатури. Такође постоје тастери као што су *Home* и *End* који померају курсор на почетак, односно на крај тренутног реда. На следећем примеру кода могу се видети неке од метода за померање курсора, док се на слици 3.1 може видети курсор у текст едитору представљен белом усправном цртом.

```
class Cursor {
public:
    //...
    void moveRight(size_t times = 1);
    void moveLeft(size_t times = 1);
    void moveUp(size_t times = 1);
    void moveDown(size_t times = 1);
    void moveToBeginning();
    void moveToEnd();
    void moveToEndOfFile();
    //...
};
```



Слика 3.1: Приказ курсора у текст едитору

### 3.3 Операције са појединачним карактерима

#### Унос

Једна од најосновнијих функционалности текст едитора је уношење текста карактер по карактер. Тај унос се врши тако што корисник притиска тастер за одговарајући штампани карактер и он се уноси на тренутну позицију курсора, тј. испред карактера на ком се налази курсор, уколико није на крају реда.

Унос се имплементира тако што се дохвати вредност курсора и преведе се у индекс непрекидног дела текста. Затим се на том индексу у табели делова уноси притиснути карактер и курсор се помера за једно место у десно. На примеру функције *insertCharToPieceTable* која припада класи *TextBox* се може видети поступак убацивања карактера у табелу делова.

```
bool TextBox::insertCharToPieceTable(char c) {  
    auto coords = m_cursor->getCoords();  
    size_t index = m_lineBuffer->textCoordinatesToBufferIndex(coords);  
    return m_pieceTableInstance->getInstance().insertChar(c, index);  
}
```

#### Брисање

Брисање појединачних карактера се врши тако што корисник позиционира курсор на жељено место и притисне тастер за брисање карактера. Треба напоменути да постоје два таква тастера, то су *Backspace* и *Delete*. *Backspace* брише карактер лево од курсора, док *Delete* брише онај на позицији курсора. После брисања се у првом случају курсор помера за једно место у лево, док у другом остаје на истој позицији. Брисање је имплементирано аналогно уносу.

Текст едитор који је имплементиран током рада на овој тези користи две одвојене методе за обе врсте брисања. То су *backspace()* и *deleteChar()* класе *TextBox*.



## 3.4 Селекција текста

Селекција текста је функционалност која кориснику пружа могућност означавања неког непрекидног сегмента текста ради брисања или копирања. Копирање ће бити објашњено у наставку рада. Она се врши помоћу миша, тако што корисник превлачи курсор миша преко дела екрана где се налази текст који жели да изабере и унесе команду за брисање или дељење. Такође се може вршити преко специјализованих команди које ће бити објашњене у наставку одељка.

Сама селекција је у имплементацији при раду представљена помоћу две текстуалне координате које представљају почетак и крај, с тим да почетак мора бити пре краја. Ове врсте координата су већ спомињане у одељку о курсору. Такође треба чувати информацију о томе да ли је селекција активна. У следећем примеру кода се може видети пример чланских променљивих класе *Selection*.

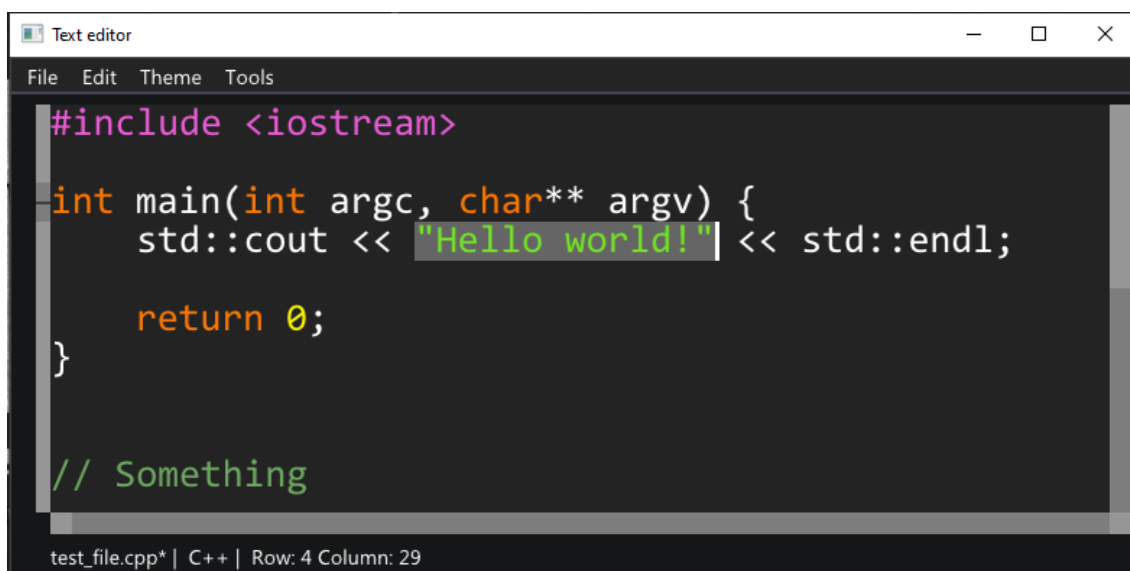
```
class Selection {  
    //...  
private:  
    bool m_active;  
    bool m_rectangular;  
    TextPosition m_start;  
    TextPosition m_end;  
};
```

Када метода *handleMouseInput()* детектује држање левог тастера, она прослеђује две позиције миша методи *setMouseSelection()* класе *TextBox*. Онда она на основу те две позиције рачуна на које текстуалне координате оне показују и додељује мању од тих координати за почетак селекције, а већу за крај. На крају се селекција означава као активна.

Уколико је селекција активна и корисник кликне било где на текстуално поље едитора, онда се она деактивира. Такође се може деактивирати притиском на стрелице за померање курсора. При притиску на тастере за брисање док постоји активна селекција, сав текст унутар ње се брише. Исто важи и

за операције сечења. (енг. *cut*) и налепљивања (енг. *paste*). Деактивирање селекције се имплементира тако што се чланска променљива *m\_active* постави на вредност *false*.

Притиском на комбинацију тастера *Shift* и *Home* се почетак селекције поставља на почетак реда, а крај на тренутну позицију курсора. Док се притиском на тастере *Shift* и *End* почетак селекције поставља на позицију курсора, а крај на последњу колону у реду. Тастери *Ctrl* и *A* селекују читав текст. На слици 3.2 се може видети пример селекције, где је означени део текста обојен другачијом бојом.



Слика 3.2: Приказ селекције у текст едитору

### 3.5 Управљање текстуалним датотекама

Сврха текст едитора јесте мењање текстуалних датотека. Да би то било могуће, потребно је обезбедити кориснику могућности креирања нових датотека, као и отварање постојећих. Такође је потребно омогућити чување измена које корисник прави као и могућност избора локације где ће датотека бити сачувана. У наредним поглављима ћемо прећи кроз те функционалности.

## Чување датотека

Корисник може сачувати тренутни садржај едитора у датотеку притиском на дугме *Save* у менију или комбинацијом тастера *Ctrl* и *S*. Тренутни садржај се чува на жељену локацију и уколико није дефинисана, тражи се од корисника да је унесе. Такође је дозвољено и чување тренутне датотеке на нову локацију притиском на дугме *Save as* у менију, где је поступак исти као са недефинисаном путањом.

Чување у имплементацији при раду извршава метода *save()* класе *TextEditor* и може се видети у наредном примеру кода.

```
void TextEditor::save() {
    if (m_activeTextBox->getFile() == nullptr)
        saveAs();
    else {
        m_activeTextBox->save();
        m_inactiveTextBox->save();
    }
}
```

За недефинисану путању позива се метода *saveAs()*, док се у супротном зове метода *save()* класе *TextBox* која уписује садржај едитора преко старог садржаја датотеке на задатој путањи. На следећем примеру се може видети метода *saveAs()*.

```
void TextEditor::saveAs() {
    auto path = saveFileDialog();

    if (!path.empty()) {
        if (path.find_last_of('.') == std::string::npos)
            path += ".txt";
        m_activeTextBox->saveAs(path);
        m_inactiveTextBox->saveAs(path);
    }
}
```

Она отвара дијалог где корисник треба да изабере локацију где ће бити сачувана датотека и враћа путању. Уколико није дефинисана екстензија датотеке биће постављена на обичну текстуалну датотеку. Затим се позива истоимена метода класе *TextBox* која поставља нову путању и уписује садржај едитора на дату локацију.

Да би се водила евиденција о изменама у тренутној датотеци користи се индикатор који се назива прљави бит (енг *dirty bit*). Он се активира уколико имамо неке измене које нисмо сачували. Када корисник отвара датотеку, прљави бит је неактиван. Сваком променом текста активира се прљави бит, док се сваким чувањем деактивира. Он је обично представљен карактером звездеце поред имена датотеке која је отворена.

### Отварање нове датотеке

Корисник може отворити нову датотеку на дугме *New file* у менију или комбинацијом тастера *Ctrl* и *N*. Када корисник покрене ову акцију, тренутни фајл се затвара и уколико је активан прљави бит, едитор отвара дијалог где корисника пита да ли жели да сачува измене. Затим се садржај текстуалног поља као и табеле делова се ресетује и курсор се поставља на почетак.

У имплементацији при раду за ову функционалност задужена је метода *newFile()* класе *TextEditor*. Она проверава да ли је фајл сачуван и позива методу *handleFileNotSaved()* уколико није. Затим позива истоимену методу класе *TextBox* која ресетује свој садржај. У наредном примеру кода се може видети описана метода.

```
void TextEditor::newFile() {
    auto id = handleFileNotSaved();
    if (id == IDCANCEL)
        return;

    m_activeTextBox->newFile();
    if (m_splitScreen)
        m_inactiveTextBox->setFile(m_activeTextBox->getFile());
}
```

### Отварање постојеће датотеке

Уколико корисник жели да отвори постојећу датотеку он може то учинити притиском на дугме *Open* у менију или комбинацијом тастера *Ctrl* и *O*. Када покрене ову команду, корисник бира датотеку коју жели, и она се отвара у текст едитору. Ову функционалност имплементира метода *open()* класе *TextEditor*. Она је приказана на следећем примеру кода.

```
void TextEditor::open() {
    auto id = handleFileNotSaved();
    if (id == IDCANCEL)
        return;

    auto path = openFileDialog();

    if (!path.empty()) {
        m_activeTextBox->open(path);
        m_inactiveTextBox->open(path);
    }
}
```

Прво се проверава прљави бит, и уколико је активан отвара дијалог за чување помоћу функције *handleFileNotSaved()*. Затим се отвара дијалог за избор датотеке помоћу функције *openFileDialog()* и враћа се изабрана путања. На крају се позива истоимена меотда класе *TextBox* и отвара се датотека и њен садржај се учитава у едитор.

## Глава 4

### Закључак

Фијуче ветар у шибљу, леди пасаже и куће иза њих и гунђа у оџацима. Ниџо, чежњиво гледаш фотелју, а Ђура и Мика хоће позицију себи. Људи, јазавац Џеф трчи по шуми глођући неко сухо жбуње. Љубави, Олга, хајде пођи у Фуџи и чут ћеш њежну музику срца. Боја ваше хаљине, госпођице Џафић, тражи да за њу кулучим. Хаџи Ђера је заћутао и бацио чежњив поглед на шољу с кафом. Џабе се зец по Хомољу шуња, чувар Јожеф лако ће и ту да га нађе. Оџачар Филип шаље осмехе туђој жени, а његова кућа без деце. Бутић Ђуро из Фоче има пун џак идеја о слагању ваших жељица. Џајић одскочи у аут и избеже Ђон халфа Пецеља и његов шамар. Пламте оџаци фабрика а чађаве гује се из њих дижу и шаљу ноћ. Ајшо, лепото и чежњо, за љубав срца мога, дођи у Хаџиће на кафу. Хучи шума, а иза жутог џбуна и пања ђак у цвећу деље сеји фрулу. Гоџи и Јаћиму из Бање Ковиљаче, флаша џина и жеђ падаху у исту уру. Џаба што Феђа чупа за косу Миљу, она јури Живу, али њега хоће и Даџа. Док је Фехим у џипу журно љубио Загу Чађевић, Џиле се ушуњао у ауто. Фијуче кошава над оџацима а Иља у гуњу журећи уђе у суху и топлу избу. Боже, џентлмени осећају физичко гађење од прљавих шољица! Дочепаће њега јака шефица, вођена љутом срџбом злих жена. Пази, геџо, брже однеси шефу тај ђавољи чек: њим плаћа цех. Фине џукце озлеђује бич: одгој их пажњом, стрпљивошћу. Замишљао би кафеџију влажних прстића, прњег од чађи. Ђаче, уштеду плаћај жаљењем због џиновских џифара. Џикљаће жалфија између тог бусења и пешчаних двораџа. Зашто гђа Хаџић лечи живце: њена љубав пред фијаском? Јеж хоће пецкањем да вређа љубичастог џина из флаше. Џеј, љубичаст зец, лаже: гађаће одмах поквашен фењер. Плашљив зец хоће јефтину дињу: грожђе

искамчи цабе. Цак је пун жица: чућеш тад свађу због ломљења харфе. Чуј, цукац Флоп без даха с гађењем жваће стршљена. Ох, задњи шраф на ципу слаб: муж гђе Цвијић љут кочи. Шеф цабе звиждуће: млађи хрт јаче кљуца њеног пса. Одбациће кавгација плаштом чађ у жељезни фењер. Дебљи кројач: згужвах смеђ филц у тањушни цепић. Цо, згужваћеш тихо смеђ филц најдебље крпењаче. Штеф, бацих сломљен дечји зврк у цеп гђе Жуњић. Дебљој згужвах смеђ филц — њен шкрт цепчић.

# Библиографија

- [1] Visual Studio Code. <https://code.visualstudio.com/>.
- [2] Open Source Community. Emacs. <https://www.gnu.org/software/emacs/>.
- [3] Open Source Community. Notepad++. <https://notepad-plus-plus.org/>.
- [4] Microsoft. Windows Notepad. [https://en.wikipedia.org/wiki/Windows\\_Notepad](https://en.wikipedia.org/wiki/Windows_Notepad).
- [5] Bram Moolenaar. VIM. <https://www.vim.org/>.



# Биографија аутора

**Бојан Барџић** је рођен 5. априла 1999. године у Призрену. Основне студије је завршио на Математичком факултету у Београду, на смеру информатика 2022. године са просечном оценом 8.64. Тренутно је студент мастер студија на истом факултету.