Милош Утвић

Информатички практикум 2

Софтвер, 2. део

Системски софтвер: услужни софтвер

Услужни софтвер (енгл. utility or accessory software, accessories)

- Када се данас набави рачунар, постоје две могућности: спољне меморије су или празне или садрже само инсталиран оперативни систем (ОС). Када би ОС чинило само језгро које управља процесором, меморијама (радном и спољном) и периферијским уређајима, рачунарски систем би био функционалан, али за обичног корисника неупотребљив: не би могао да прочита или запише обичан текст, погледа слику, репродукује аудио или видео запис, приступи интернету, а о неким пословним применама и да не говоримо. Дакле, кориснику је неопходан одговарајући апликативни софтвер. Произвођачи ОС-а су решили овај проблем тако што уз ОС испоручују једноставне помоћне програме који пружају неке основне услуге док корисник не набави одговарајући апликативни софтвер. Типични пример програма који чине помоћни или услужни софтвер за Windows, тзв. Windows Accessories, представљају Notepad, Paint, Calculator.

Услужни софтвер данас

- Према томе, услужни софтвер данас једним делом представљају помоћне апликације које долазе уз оперативни систем, тј. програми који су се раније третирали као апликативни софтвер, али су у међувремену постали неопходни свим корисницима (Notepad, Calculator). Управо је то једна линија раздвајања између услужног и апликативног софтвера: услужни софтвер је потребан свима, док апликативни има специфичну намену, па самим тим и ужи круг корисника у односу на услужни софтвер.
- Другу линију раздвајања дефинишу програми који су првобитно развијени као апликативни софтвер, тј. имају специфичну намену, али се те намене више односе на улоге које припадају оперативном систему, те такви програми представљају надградњу или допуну за постојеће могућности оперативног система. Ови програми су више од значаја за администраторе оперативних система (нпр. програми за партиционисање или дефрагментацију HDD-а).
- Према томе, иако се услужни софтвер убраја у системски софтвер, он је заправо на граници између апликативног и системског софтвера.

Типови услужног софтвера

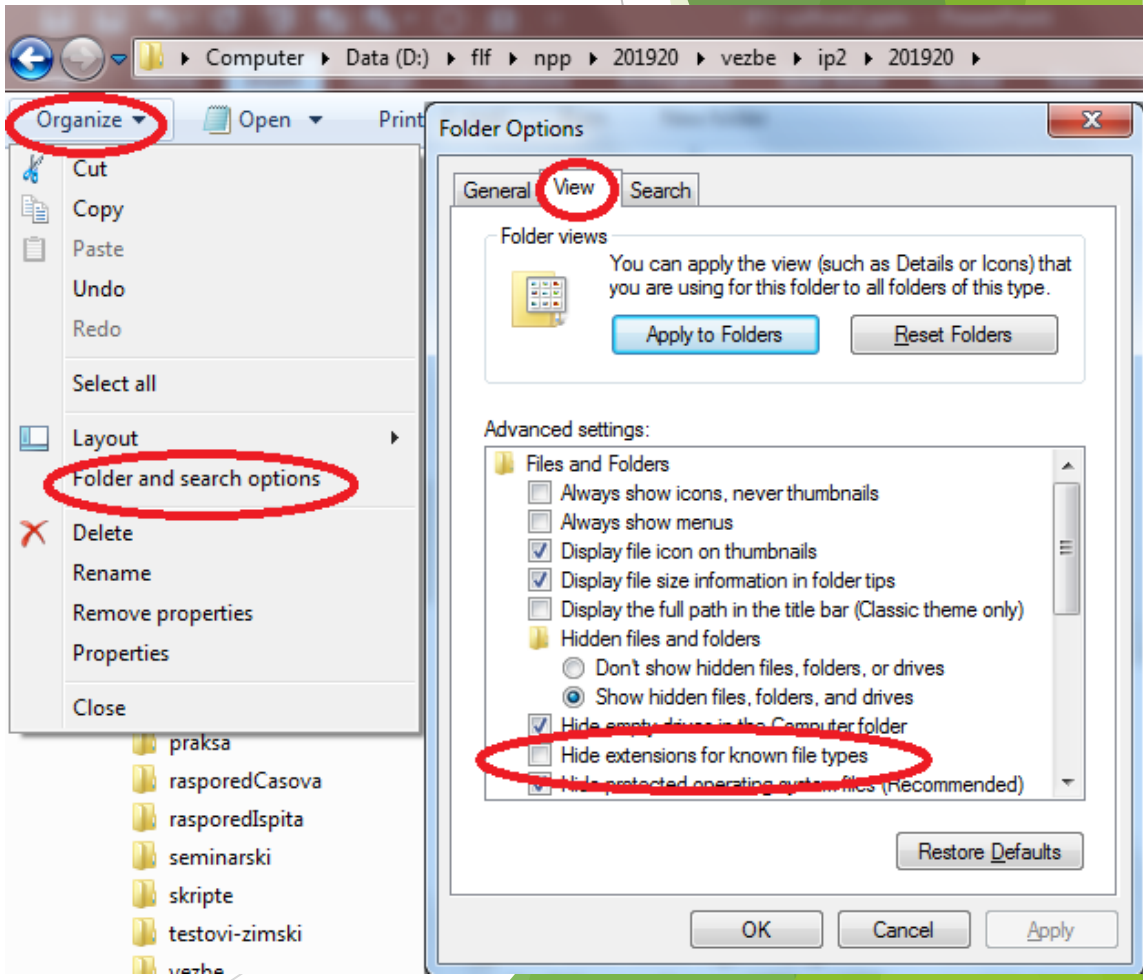
- Програми за управљање системом датотека и каталога, управљачи датотечким системом (енгл. file managers);
- Програми за архивирање, компресију и декомпресију података или архивери/компресори/декомпресори (енгл. file archivers, compressors);
- Програми за уређивање/обраду (обичног) текста, уређивачи текста (енгл. text editors);
- Програми за мрежну комуникацију, комуникациони софтвер;
- Програми за детекцију и онемогућавање злонамерног софтвера, анти-малвер (енгл. anti-malware software).

Управљачи системом датотека и каталога (file managers, скр. FM)

- Типичан пример FM-а је My Computer (некадашњи назив Windows Explorer, одатле пречица **WinKey + E**, тј. ⊞ + **E**). Популарни су и Total Commander, Midnight Commander итд. За детаљнију листу и поређење, видети https://en.wikipedia.org/wiki/Comparison_of_file_managers
- Основна улога: операције са датотекама и каталозима (креирање каталога, копирање, премештање, преименовање датотека и каталога), преглед и измена својстава датотека и каталога (промена власништва, промена дозвола за модификовање и извршавање, сакривање/откривање датотека и каталога у приказу итд.).

Пример: подешавање приказа пуног назива датотеке (My Computer)

- Уобичајено је да је Windows, тачније My Computer, подешен тако да не приказује поједине типове датотека, тј. приказује само део имена пре тачке (нпр, уместо **pera.txt** приказује само **pera**). У том случају, када се покуша операција преименовања, промениће се само део имена испред тачке, не и сам тип. Студенти који нису свесни овога, често преименују датотеку са семинарским радом, којој се не види тип, тако што јој додају још један тип, па тако **pera.txt** постаје **pera.txt.txt** а My Computer приказује само **pera.txt.**
- **Да бисте видели пуно име и избегли овакве грешке, искључите опцију Hide extensions for known file types** (My Computer, мени Organize, ставка Folder and search options, картица View).



Сродни програми (понекад саставни део FM-а)

- Временом се све више улога додаје управљачима датотечким системом. Тако My Computer омогућава дељење садржаја са рачунарима у мрежи, док Total Commander обухвата и неке од следећих програма који можда не могу да се сматрају подтиповима FM-а, али су им свакако сродни по функцији (већина је доступна независно од конкретног FM-а):
  - програми за прављење резервних копија података за случај да дође до губитка оригиналних верзија података (енгл. backup software),
  - програми за упоређивање садржаја датотека и каталога, тј. проналажење идентичних датотека, односно утврђивање разлика између више верзија истог документа (енгл. file comparers),
  - програми за дефрагментацију HDD-а (енгл. disk defragmenters),
  - програми за групно (пакетно) преименовање датотека и каталога који имају одређену структуру (енгл. batch renamers),
  - програми за ослобађање радне меморије (енгл. memory management tools),
  - програми за анализу рада и процену животног века хардверских уређаја (нпр. HDD-а) и софтвера (енгл. benchmark software).

Пример: Advanced Renamer (1)

- Претпоставимо да имате колекцију датотека у формату mp3 које се зову
  - 01_Vuce_bubo_lenja.mp3 02_Ivin_voz.mp3
  - 03_Pozdravite_moga_tatu.mp3 04_Lako_je_prutu.mp3
  - 05_Zakleo_se_Bumbar.mp3 06_Nema_zemlje_Dembelije.mp3
  - 07_Kad_je_bio_mrak.mp3 08_Suma_blista_suma_blista.mp3

  и још десетак песама u извођењу Драгана Лаковића и хора Колибри које желите да преименујете тако да нова имена имају структуру попут

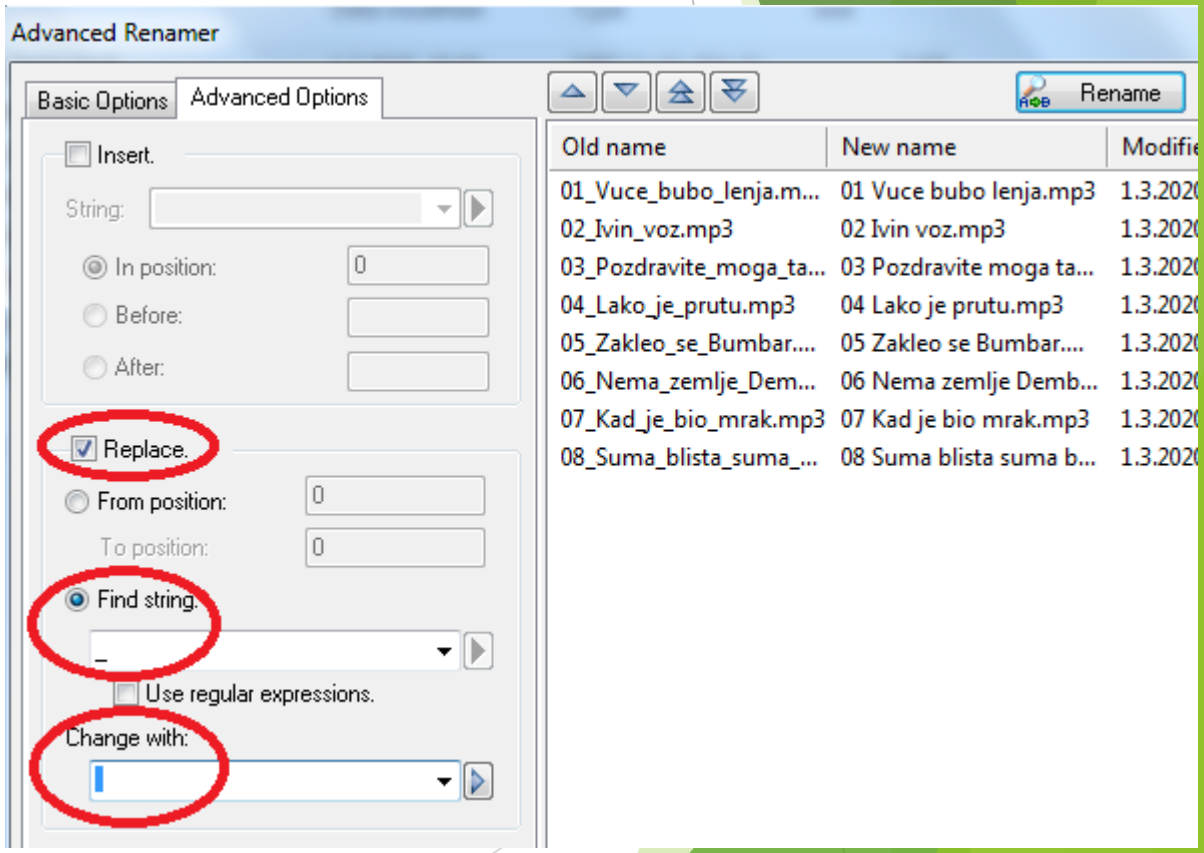
  Dragan Lakovic i Kolibri – 01 Vuce bubo lenja.mp3

  Dragan Lakovic i Kolibri – 02 Ivin voz.mp3 итд.

  тј. не желите подвлаке већ размаке и желите да се на почетку имена наведу извођачи (одвојени од назива песме размаком, цртицом и још једним размаком. Уместо појединачног преименовања датотека, можете искористити програм Advanced Renamer за групно (пакетно) преименовање.

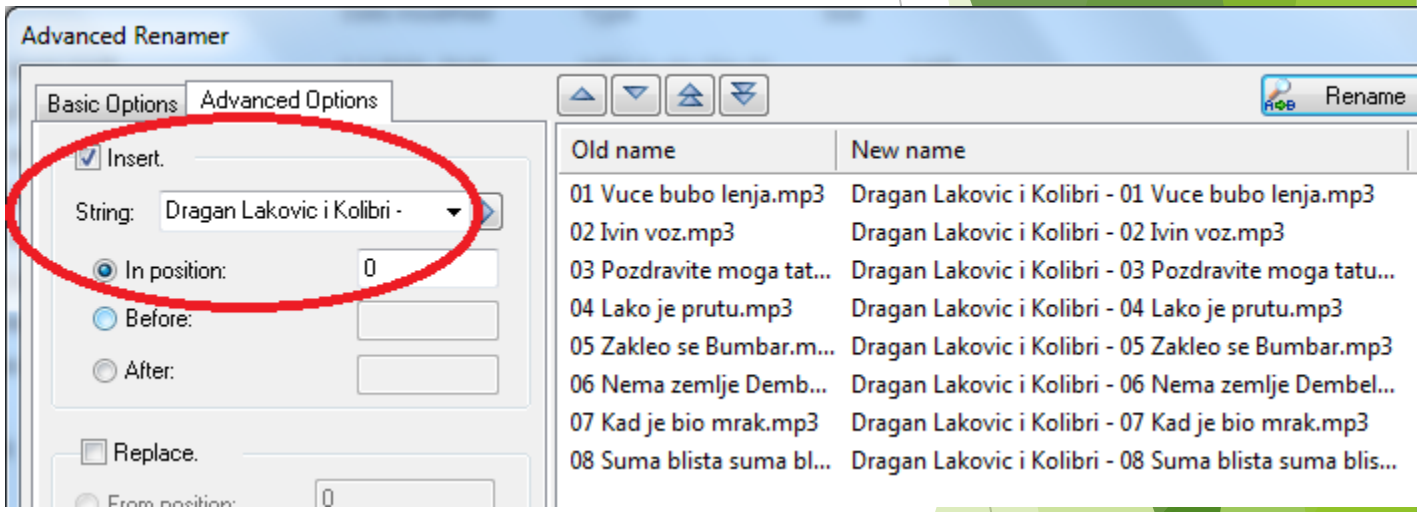
Пример: Advanced Renamer (2)

- У првом кораку обележимо све датотеке и превучемо их у програм. Потом бирамо картицу Advanced options и опцију Replace како бисмо све подвлаке (_) заменили размаком (попунимо поља Find string и Change with). Програм увек приказује како ће изгледати нова имена пре него што обави преименовање, тако да када будемо сигурни да је све подешено како треба, активирамо дугме Rename да бисмо преименовали све датотеке.



Пример: Advanced Renamer (3)

- У другом кораку бирамо опцију Insert како бисмо испред назива сваке песме убацили извођача „Dragan Lakovic i Kolibri – “ (текст у пољу String) на сам почетак назива датотеке (In position: 0).



Поново видимо како ће изгледати нова имена пре самог преименовања и кад је све спремно, активирамо дугме Rename да обавимо преименовање свих датотека.

Архивери/компресори

- Приликом копирања већег броја датотека и каталога са једне спољне меморије рачунарског система на другу или са једног рачунара у мрежи на други јавља се потреба да се пренос података убрза. Један од начина за убрзање јесте комбиновање и компресија података пре копирања, тј. спајање (паковање) датотека и каталога у једну нову датотеку (архиву) са другачијим записом података тако да величина добијене архиве буде мања од збирне величине полазних датотека и каталога. Када архива стигне на одредиште, примењује се распакивање и декомпресија којом се архива разлаже на полазне датотеке и каталоге. Наведени пример се односи на случај компресије и декомпресије без губитка информације.
- Програми за комбиновање (архивирање, паковање) података се још називају **архивери** (енгл. file archivers), док се програми за архивирање, компресију и декомпресију зову **компресори** (енгл. compressors).
- Постоји више различитих формата архива који се разликују по алгоритму који се користи за архивирање, компресију, односно и архивирање и компресију података: **.zip, .rar, .7z, .tar, .gz, .tgz, .iso, .jar, .bz2,…**
- Комерцијални архивери, компресори: WinZip, WinRAR ,…
- Слободни или само бесплатни архивери и компресори: 7z, lzArc, tar, gzip, gunzip, unrar (декомпресује архиве у формату .rar),…

Остали примери компресије

- Разни формати дигиталних слика, аудио и видео датотека такође користе компресију како би одговарајуће датотеке заузимале мање места на спољној меморији. Софтвер који репродукује компримоване слике, аудио и видео датотеке заправо врши декомпресију пре саме репродукције. За ове формате је карактеристично да могу бити резултат и компресије без губитака (нпр. слике у формату TIFF, PNG, аудио у формату FLAC) и компресије са губитком информација (слике у формату JPEG, аудио у формату MP3, видео у формату DivX) јер би у противном величине датотека биле знатно веће.

Смисао компресије

- Из наведеног следи да нема много сврхе користити компресоре да би се смањила величина дигиталних слика, аудио и видео датотека.
- Најбољи резултати се постижу приликом компримовања текста.
- С друге стране, архивери су увек корисни за комбиновање многобројног садржаја у једну датотеку чиме се олакшава манипулација, укључујући и убрзање копирања јер је оперативном систему лакше да направи копију само једне (спојене) датотеке уместо да за сваку од многобројних датотека или каталога прави посебну копију.
- Нажалост, у случају многобројног садржаја увек се одређено време губи на паковање и распакивање, тако да цела ствар има смисла само је укупно потрошено време за паковање, копирање и распакивање мање од времена потребног за директно копирање без паковања.

Пример: стари и нови стандардни формати датотека пакета MS Office

- Нови стандардни формати датотека програма MS Word (.docx) и MS Excel (.xlsx) су заправо архиве у формату .zip. (Због популарности формата .zip, често се у жаргону користи израз „зиповање“, било за архивирање, било за компресију датотека и каталога).
- Ако на пример, документ у старом формату .doc, величине око 47 KiB и креиран програмом MS Word, конвертујемо у формат .docx, добићемо датотеку величине око 22 KiB. Ако добијеној датотеци променимо тип у .zip и распакујемо архиву, добићемо више датотека и каталога (у конкретном примеру 15 датотека и 5 каталога) чија је укупна величина око 104 KiB.

Уређивачи текста

- Омогућавају уређивање обичног текста на нивоу карактера и линија (редова), у неким случајевима и на нивоу колона. Посебни карактери чувају информацију о крају реда.
- Делови текста (низ суседних карактера) се могу уносити (у режиму уметања или брисања, енгл. Insert/Overwrite), копирати, премештати, брисати.
- Приказ текста се може форматирати на нивоу целог текста, али је ограничен само на програм у коме се текст уређује и формат се не чува у датотеци са текстом, а ни било где другде. Чувају се само карактери текста.
- Текст се може претраживати и пронађени делови текста се могу заменити. Могућности варирају од програма до програма (може се тражити и замењивати искључиво наведени текст или програм дозвољава и формуле за опис скупа речи одређене структуре као што су џокер-знаци или регуларни изрази).
- Примери: Notepad, Notepad++, PSPad и многи други. За детаљнију листу видети https://en.wikipedia.org/wiki/Comparison_of_text_editors

MS Word није (само) уређивач текста

- MS Word, иако има опције за уређивање, не спада у уређиваче текста и услужни софтвер већ у
  - апликативни софтвер за форматирање текста (текст-процесори, енгл. word processors) и
  - софтвер за припрему за штампу (енгл. desktop publishing software).
- Основна разлика између уређивача текста и текст-процесора: ако у програму Notepad промените величину фонта или фамилију фонта, промена наступа за **целокупан** текст и форматирање се не чува се у датотеци са текстом, чувају се само карактери текста. MS Word, с друге стране, не само што може да различито форматира делове текста, већ се форматирање чува у истој датотеци у којој је и текст.

Пример корисних опција програма Notepad++

- Брзо пребацивање делова текста из великих слова у мала и обрнуто (десни клик на обележени текст, опције контекстног менија **lowercase**, **UPPERCASE**);
- Елиминација празних линија, укључујући или искључујући оне које садрже само белине: размаке, табулаторе и знак за нови ред (мени Edit > Line operations > **Remove Empty Lines**, односно **Remove Empty Lines (Containing Blank Characters)**);
- Сортирање линија (мени Edit > Line operations > више опција чији назив почиње са **Sort Lines…**);
- Брисање белина на крају или почетку линије или на оба места. (мени Edit> Blank operations, све опције);
- Промена кодног распореда текста (мени Encoding, обично бирамо **Encode in UTF-8, без BOM-a**);
- Претрага и замена применом регуларних израза. При томе може да се обради текући документ, сви отворени документи или све датотеке задатог типа на одређеној адреси (мени Search, опције, **Find**, **Replace** и **Find in files…**).

Регуларни изрази (скр. RE)

- Коначан низ карактера текста називаћемо **ниска**.
- Регуларни изрази (RE) су специјалне ниске, тј. обрасци/формуле којима се означава некакав скуп ниски (коначан или бесконачан). Најпростији случај је када RE као ниска означава саму себе, на пример, RE **као** означава саму ту ниску, тј. једночлани скуп {**као**}.
- Оно што омогућава да RE означава више ниски јесу метакарактери. **Метакарактери су** поједини карактери који се не користе у свом уобичајеном значењу, тј. не представљају сами себе, већ имају специјално значење. На пример, у регуларним изразима које користи Notepad++ се користе метакарактери:
  - ^ (да означи почетак линије),
  - $ (као ознака краја линије),
  - . (тачка као ознака за џокер, тј. замењује било који карактер) итд.
- Следећи слајдови дају преглед метакарактера које користе RE у програму Notepad++ и примере њихове употребе.

Метакарактер	Чита се	Значење	Пример	Препознаје
|	или	Унија (алтернација) ниски	као|ко|к’о	трочлани скуп ниски **{као, ко, к’о}**
()	заграде	Груписање	к(ао|о|’о)	трочлани скуп ниски **{као, ко, к’о}**
^	капица	Почетак линије	^Б	све линије које почињу словом **Б**
$	долар	Крај линије	сти$	све линије које се завршавају на **сти**
.	тачка	Џокер, ма који карактер	^..$	све линије са тачно 2 карактера
[*карактери*]	класа	Карактерска класа (унија појединачних карактера)	[аеиоу]	један произвољан вокал, тј. елемент петочланог скупа {а, е, и, о, у}
[^*карактери*]	негација класе	Негација карактерске класе	[^аеиоу]	један произвољан карактер који није вокал, тј. не припада скупу {а, е, и, о, у}
*	звездица	0 или више појављивања	мјауу*	{мјау, мјауу, мјаууу, мјауууу,...}
+	плус	1 или више појављивања	мјау+	{мјау, мјауу, мјаууу, мјауууу,...}
?	упитник	Необавезно појављивање (једном или ниједном)	ка?о	двочлани скуп ниски {као, ко}
{*m*, *n*}	понав-љање	Понављање најмање *m* пута, а највише *n* пута	(0|1){1,3}	скуп највише троцифрених бинарних бројева {0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111}
{*m*, *n*}	понав-љање	Понављање тачно *n* пута	(0|1){3}	скуп троцифрених бинарних бројева {000, 001, 010, 011, 100, 101, 110, 111}

Примена регуларних израза у програму Notepad++

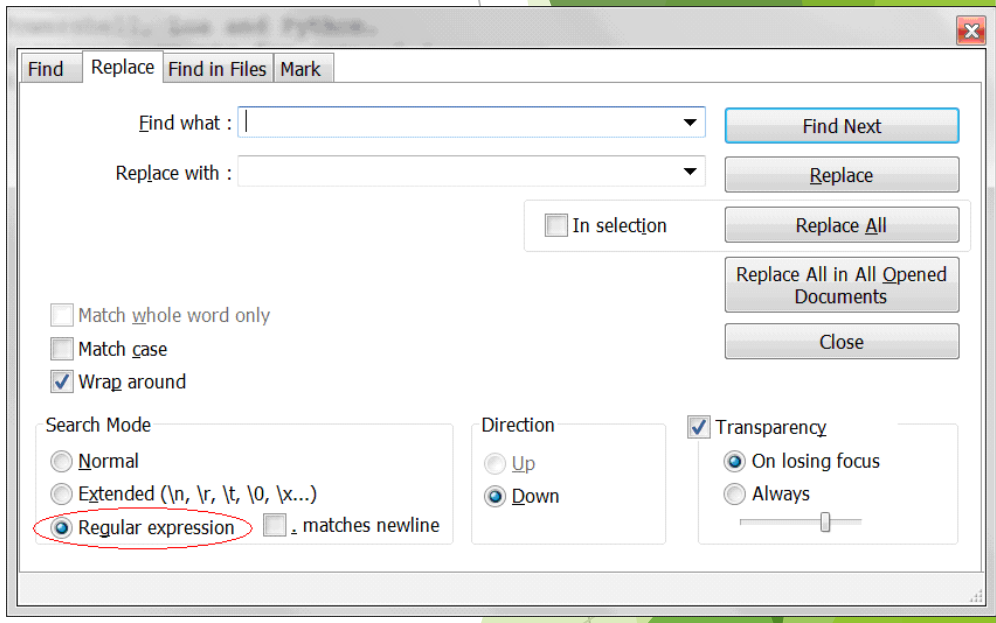
- Пре сваке претраге или замене потребно је проверити да ли је у дијалогу *Find/Replace* изабрана опција *Regular expressions*.
- Надаље ће ознаке

  Find: *тражениТекст*

  Replace: *текстЗамене*

означавати да у поље *Find* дијалога треба унети *тражениТекст*, а у поље *Replace*: дијалога — *текстЗамене*.

Да не би било неспоразума да ли нека белина представља део регуларног израза или не, регуларни израз ће увек бити обојен и то црвено, ако се користи за претрагу, односно плаво, ако се користи за замену.



Примери употребе регуларних израза (1)

- Пронађимо у тексту све верзије појављивања речи **дошао** (дошао, дошо, дош’о, došao, došo, doš’o).
- I начин: користимо унију (алтернацију) ниски

  Find: дошао|дошо|дош’о|došao|došo|doš’o
- II начин: користимо унију и могућност скраћеног записа:

  Find: дош(ао|о|’о)|doš(ao|o|’o)
- Приметимо да се дописивање и унија „слажу“ као множење и сабирање. Наиме, као што важи $2 \cdot (3 + 4) = 2 \cdot 3 + 2 \cdot 4$, тако важи да је дош(ао|о|’о) исто што и дошао|дошо|дош’о. Скраћени запис дош(ао|о|’о) можемо да прочитамо и као: низ карактера дош на које се дописује или ао или о или ’о.

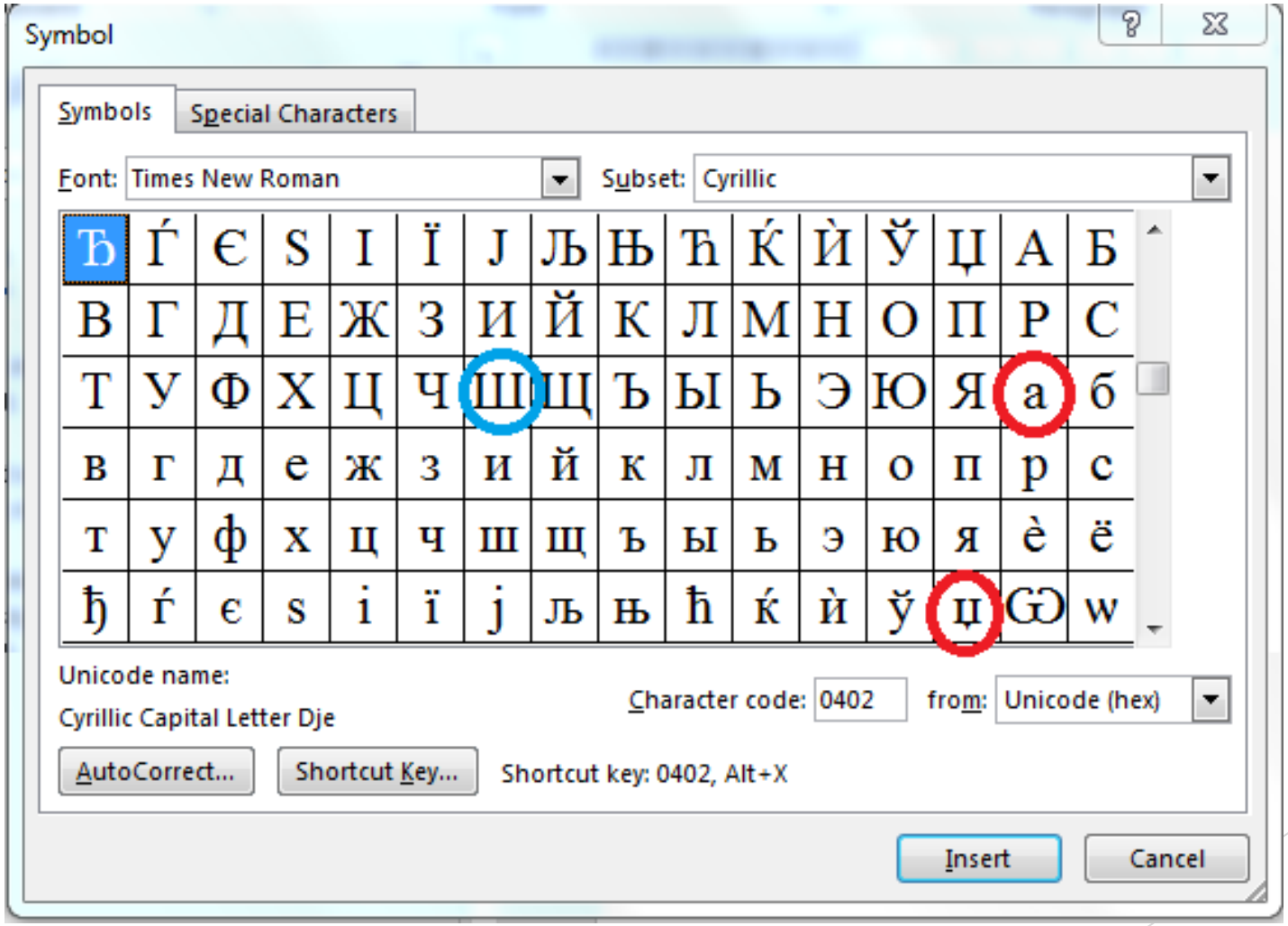
Карактерске класе

- Једна од погодности код коришћења карактерских класа је могућност да се кратким изразом обухвати подскуп свих карактера Unicode-а који се налазе између два задата карактера. Тако се уместо набрајања свих слова енглеске абецеде (великих, односно малих) могу навести само прво и последње слово и цртица између њих као ознака да је у питању опсег (ово функционише само зато што су енглеска слова представљена у Unicode-у на суседним позицијама и у истом поретку као у абецеди). Слично важи и за декадне цифре.

Често коришћене елементарне карактерске класе

- Нажалост, наша слова (ћирилична и латинична) се не могу обухватити формулама типа а-ш, односно а-ž, али постоје заобилазна решења.
- Произвољна декадна цифра **[0-9]** **или** **\d**
- Произвољно велико слово енглеске абецеде **[A-Z]**
- Произвољно мало слово енглеске абецеде **[a-z]**
- Произвољно слово енглеске абецеде **[A-Za-z]**
- Произвољно велико слово српске латинице (и енглеског алфабета, укључујући Q, Y, W) **[A-ZĆČĐŠŽ]**
- Произвољно мало слово српске латинице (и енглеског алфабета, укључујући q, y, w) **[a-zćčđšž]**
- Произвољно слово српске латинице (и енглеског алфабета, укључујући Q, q, Y, y, W, w) **[A-Za-zĆćČčĐđŠšŽž]**

Распоред ћириличних слова (Unicode, MS Word, дијалог Insert Symbol)



Често коришћене елементарне карактерске класе (2)

- Произвољно велико ћирилично слово српске азбуке (и понеко које није, дакле, апроксимација) [Ђ-Ш]
- Произвољно мало ћирилично слово српске азбуке (и понеко које није, апроксимација) [а-џ]
- Произвољно ћирилично слово српске азбуке (и понеко које није, апроксимација) [Ђ-Ша-џ]
- Опрез!! Цртица за опсег има специјално значење **само унутар** средњих заграда, па се разликују рег. изрази:
  - 0-9 препознаје скуп који садржи само једну ниску: 0-9 (нула, цртица девет) и
  - [0-9] препознаје скуп ниски $\{0,1,2,3,4,5,6,7,8,9\}$

Примери употребе регуларних израза (2)

- Пронађимо у тексту све датуме структуре гггг-мм-дд (д = дан, м = месец, г = година), на пример 2019-10-31.
- Најгрубља апроксимација оваквог датума је „четири цифре, цртица, две цифре, цртица, две цифре“, односно

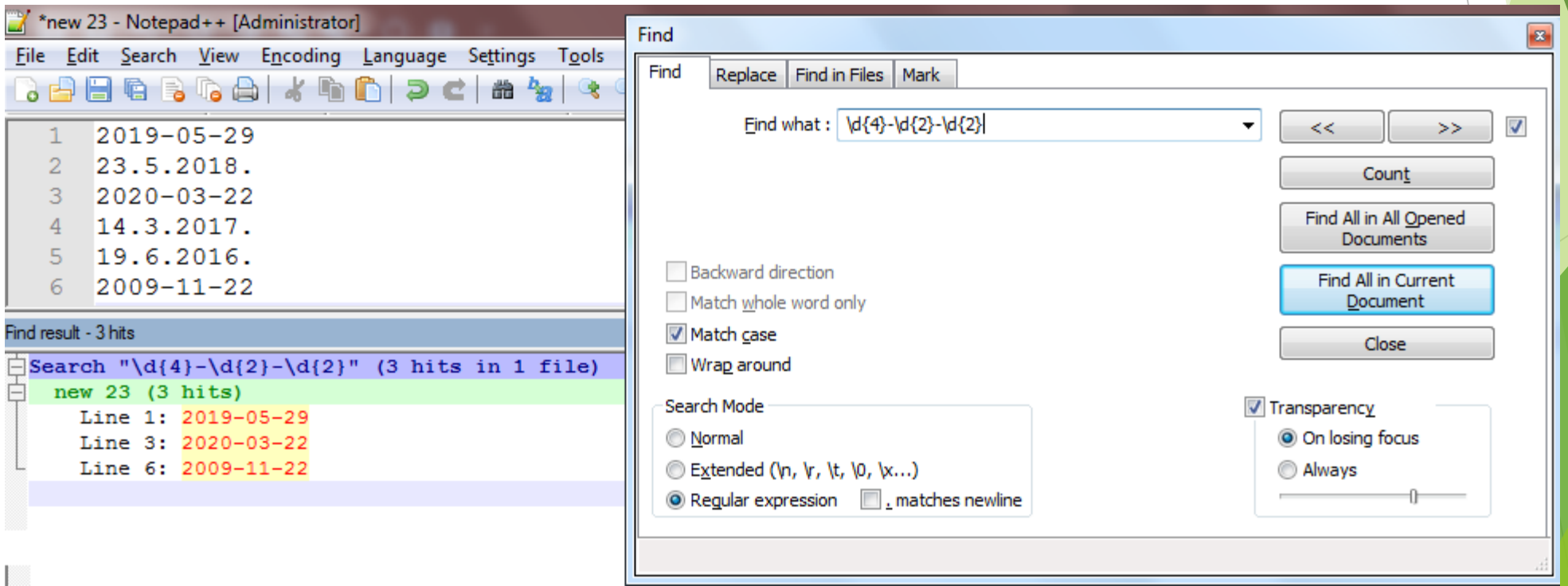
  Find: [0-9]{4}-[0-9]{2}-[0-9]{2}

или

  Find: \d{4}-\d{2}-\d{2}

Анализа резултата претраге

- Понекад је потребно пажљиво анализирати колико је употребљени RE прецизан, погтово када се изводе операције замене. У ту сврху је корисно дугме Find All in Currrent Document која у посебном прозору приказује све линије (са редним бројем линије) које садрже тражени образац, као и укупан број резултата. Ако нас занима само укупан број резултата које би образац пронашао, довољно је искористити дугме Count. Обе опције се могу користити и случају када се користи „нормална“ претрага без рег. израза.



Пример: конверзија датума у тексту помоћу регуларних израза

- Пример: заменити све датуме структуре гггг-мм-дд (д = дан, м = месец, г = година) одговарајућим датумима структуре дд.мм.гггг у једном потезу. На пример, на овај начин се у тексту датум облика 2019-10-31 замењује датумом облика 31.10.2019.

  Find: ([0-9]{4})-([0-9]{2})-([0-9]{2}) или Find: (\d{4})-(\d{2})-(\d{2})

  Replace: \3.\2.\1.

- Коришћење обичних заграда у регуларним изразима служи не само за избегавање, односно наметање приоритета, већ и за „памћење“ делова препознатог обрасца. Сваком пару заграда у регуларном изразу одговара једна промењива (највише њих 9) чији бројеви у ознакама \1, \2,… \9 представљају редни број одговарајућег пара заграда посматрано слева надесно. Део структуре који је препознат у оквиру неког пара заграда, меморише се у одговарајућој промењивој чија вредност може да се користи приликом замене текста (у овом примеру \1 = година, \2 = месец, \3 = дан)

Анотација регуларним изразима

Регуларне изразе сте имали прилике да користите при изради семинарског рада, тј. при обележавању пагинације (претпоставка је да је редни број странице у књизи у посебној линији):

Find: ^([0-9]+)$ или Find: ^(\d+)$

Replace: <pb n='\1' />

као и приликом обележавања пасуса (у случају да свака линија текста у датотеци увек представља један пасус, тј. крај линије је и крај пасуса):

Find: ^(.+)$

Replace: \1

- Претходни израз може још да се прецизира, али о томе други пут. Више о регуларним изразима учићете у наредним курсевима.

Комуникациони софтвер

- О комуникационом софтверу ће више бити речи у оквиру одељка о рачунарским мрежама. У питању је софтвер који омогућава различите врсте комуникација између два или више рачунара, односно размену информација.
- Пар популарних примера ради стицања представе о чему се ради:
  - Skype, Viber, WhatsApp, Facebook Messenger…
  - Google Chrome, Mozilla Firefox, …
  - MS Outlook, Thunderbird…
  - WinSCP, програми који користе протокол BitTorrent…
  - putty, Bitvise SSH Client …
  - TeamViewer, Chrome Remote Desktop, …

Анти-малвер софтвер

- Као што само име каже, у питању је софтвер за детекцију и онемогућавање злонамерног софтвера. Поједини примери су специјализовани само за борбу против одређеног типа злонамерног софтвера (на пример, анти-вирусни софтвер), али су често и обједињени у један софтверски пакет.
- Анти-малвер софтвер ради тако што с времена на време анализира садржај спољних меморија, посебно датотеке ОС-а, а надгледа и комуникацију програма са рачунарском мрежом. Наиме, употреба комуникационог софтвера и рад у мрежи може да представља ризик: на пример, софтвер за шпијунирање шаље податке са нашег рачунара некоме у мрежи или неки рачунар у мрежи напада наш рачунар непрекидним слањем пакета података (углавном бесмислених) чиме се „гуши“ комуникација са мрежом. Подврста анти-малвер софтвера која надгледа и по потреби ограничава мрежни саобраћај, односно размену података са осталим рачунарима у мрежи назива се **ватробрани** (енгл. firewalls). Примери ватробрана: Windows Firewall, Zone Alarm, …
- Примери анти-малвер софтвера (у питању су или програми или фирме које имају више специјализованих програма): ESET NOD32, Kaspersky, AVG, Avast, Avira, Windows Defender, …

Заштита рачунара и себе

- Анти-малвер софтвер комбинује коришћење база информација о раније регистрованом малверу са разним статистичким методама које упозоравају на потенцијални малвер. Редовно ажурирање база информација које користи анти-малвер софтвер, непрекидна анализа мрежног саобраћаја и повремена анализа спољних меморија које користимо може донекле да спречи напад малвера. Међутим, као што илуструје следећа, нажалост, истинита прича, човек као корисник рачунара остаје најслабија карика у ланцу сигурности: Кевин Митник и изворни кôд телефонске компаније Моторола.
- Преузимање материјала са интернета је посебан ризик. Ако вам неко нуди да преузмете музику или видео и кренете то да урадите, а приметите да се у дијалогу Save As име датотеке завршава на .exe, дефинитивно је у питању малвер и само је потребно одустати од преузимања.
- Посебно је опасно преузимати програме са непоузданих сајтова, поготово пиратске верзије. Никад не можете сигурни да ли је у питању заиста само користан програм који не захтева плаћену регистрацију или неки малвер који ће вам одмах или током времена направити штету, било тако што ће покварити хардвер, софтвер, обрисати податке без могућности да их повратите ако нема резервне копије, било тако што ће искористити ваш рачунар или, још горе, ваш идентитет за нелегалне активности због којих можете да будете кажњени било новчаном, било затворском казном.

Системски софтвер: језички процесори

Програмски језици

- У историји је забележено више примера уређаја који су могли да се програмирају, укључујући музичке кутије, Жакаров ткачки разбој (програм на бушеним картицама) и планове машина које никад нису направљене (аналитичка машина Чарлса Бебиџа за коју постоје описи програма и алгоритама које је 1843. године написала Ејда Бајрон, грофица од Лавлејса).
- Овде ће бити речи само о програмским језицима коришћеним на дигиталним електронским рачунарима почев од средине XX века.
- Вештачки језици који се користе за писање рачунарских програма називају се **програмски језици.**

Лексика, синтакса и семантика програмских језика

- Као што природни језици имају своју азбуку (коначан скуп симбола), лексику (поједностављено: „све речи језика“), синтаксу (поједностављено: „правила исправног комбиновања речи у реченице, граматичка правила“) и семантику (поједностављено: „значења речи и реченица“), такав је случај и са вештачким језицима какви су програмски језици. Дисциплине рачунарства које се баве програмским језицима користе другачију терминологију у односу на природне језику, па тако говоре о токенима (поједностављено: уместо о „речима, интерпункцијским знацима и осталим симболима азбуке“), наредбама (поједностављено: уместо о „реченицама“), а појављују се и специфични појмови попут константи (литерала), променљивих, типова података, израза, блокова, опсега, потпрограма (функција и процедура), итд.

Подела програмских језика

- Програмски језици се деле на
  - машински зависне (или ниже програмске) језике у које спадају:
    - машински језици,
    - симболички (асемблерски) језици и
    - макроасемблерски језици
  - и машински независне (или више програмске) језике.
- Као што њихови назив говоре, програми на машинским језицима су везани за конкретну архитектуру рачунара (тј. скуп инструкција процесора) и практично су непреносиви на друге архитектуре рачунара без значајног преправљања.
- С друге стране, машински независни језици су конципирани тако да користе синтаксу која је довољно апстрактна и блиска природном језику да опише решење проблема без ослањања на конкретну архитектуру рачунара.

Машински језици

- Првобитни програмски језици за дигиталне електронске рачунаре били су уско везани за начин на који је рачунар (у ужем смислу: процесор и радна меморија) конструисан, па су по томе и добили назив **машински језици**. Програм на машинском језику је заправо бинарни запис инструкција које је процесор у стању да изврши без икакве прераде програма. Као такав, програм на машинском језику је тешко читљив чак и онима коју добро познају синтаксу тог језика и архитектуру рачунара за који је програм написан.
- Пример: машински језик процесора **Intel® 64**

Пример програма на машинском језику

- Овде је наведен програм у радној меморији (RAM) као низ инструкција једног фиктивног једноадресног процесора.
- Црвеном бојом је означен бинарни број који представља „шифру“, тј. операциони кôд (скр. опкод) инструкције, док су плавом бојом наведене адресе у радној меморији (RAM) које се користе у инструкцијама.
- Три инструкције су специфичне (црвена боја адресе) јер њихове адресе нису праве адресе већ део опкода (лош дизајн, али тако је направљен процесор).

адреса у RAM-у	програм (машински језик)
0000	1001 0001
0001	0011 1010
0010	1001 0001
0011	0011 1011
0100	0101 1010
0101	0001 1011
0110	0011 1100
0111	1001 0010
1000	0000 0000

Симболички (асемблерски) језици

- С обзиром на тешку читљивост програма на машинском језику, дошло се на идеју да се приликом писања програма уместо бинарних бројева користе симболи за означавање опкодова инструкције (нпр, SAB за 0001, MUA за 0101, AUM за 0011, итд.). Такође, адресе меморијских локација које се наводе у инструкцији се замењују одговарајућим бројевима у систему са већом основом, најчешће хексадекадном.
- У примеру који ћемо навести користићемо декадни систем ради једноставности. Поново користимо црвену боју за операциони кôд (опкод) инструкције, а плаву боју за адресу у радној меморији (RAM) која се користи у инструкцији.

бинарно	бинарно	декадно	декадно	декадно	декадно
Адреса у RAM-у	**Машинска инструк-ција програма**	**Адреса у RAM-у**	**Машинска инструк-ција програма**	**Инструкција програма (асемблерски језик)**	**Објашњење (све адресе су декадне)**
0000	1001 0001	0	9 01	ULAZ	Улаз са тастатуре у акумулатор
0001	0011 1010	1	3 10	AUM 10	Сачувај садржај акумулатора на адресу 10
0010	1001 0001	2	9 01	ULAZ	Улаз са тастатуре у акумулатор
0011	0011 1011	3	3 11	AUM 11	Сачувај садржај акумулатора на адресу 11
0100	0101 1010	4	5 10	MUA 10	Учитај садржај са адресе 10 у акумулатор
0101	0001 1011	5	1 11	SAB 11	Сабери садржај акумулатора са садржајем адресе 11 и резултат смести у акумулатор
0110	0011 1100	6	3 12	AUM 12	Сачувај садржај акумулатора на адресу 12
0111	1001 0010	7	9 02	IZLAZ	Испиши садржај акумулатора на екран
1000	0000 0000	8	0 00	KRAJ	Крај програма, престанак рада

Асемблер

- Запис програма на асемблерском језику је свакако читљивији од записа на машинском језику (макар ономе ко познаје архитектуру рачунара на коме ће се програм извршавати, пре свега, скуп инструкција процесора). Међутим, тако записан програм процесор више „не разуме“, односно није у стању да изврши.
- Решење је развој посебне врсте програма, асемблера, који ће аутоматски превести запис програма са асемблерског језика на машински језик. Асемблер је једноставан програм који преводи заменом симболичких опкодова и адреса одговарајућим бинарним бројевима (нпр, SAB 11 се замењује са 0001 1011).
- Асемблер је уједно и најједноставнији пример **језичког процесора**.

симболички опкод	бинарни опкод
ULAZ	1001
AUM	0011
MUA	0101
SAB	0010
IZLAZ	1001
KRAJ	0000

Виши програмски језици

- Кад се једном утврдило да програм не мора да се пише на машинском језику, поставило се питање да ли мора да се пише и на асемблерском језику, тј. да ли програмер мора да познаје скуп инструкција процесора или може програм да пише на језику који је ближи уобичајеним нотацијама за решавање проблема. Другим речима, да ли низ симболичких инструкција за сабирање два броја може да се замени формулом на коју смо навикли:

MUA 10
SAB 11
AUM 12

Z = X + Y

Историја виших програмских језика

- Још током четрдесетих година XX века појавили су се први виши програмски језици, али први који је стекао популарност је FORTRAN (аутор је Џон Бекус, 1954. године). Сам назив је спој скраћеница од енглеских речи **FORmula и TRANslation** (формула, превод) који сугерише да су наредбе језика математичке формуле које треба превести на машински језик. Једна од једноставних наредби FORTRAN-а је управо **Z = X + Y** („сабери вредности променљивих X и Y и добијени збир додели као вредност променљивој Z“) коју сада треба превести на асемблерски, а онда на машински језик.

Z = X + Y

MUA 10
SAB 11
AUM 12

0101 1010
0010 1011
0011 1100

Развој виших програмских језика

- После FORTRAN-а (који је у међувремену постао Fortran) па све до данас развијено је на хиљаде програмских језика. У почетку је постојала тенденција да се направи један универзални програмски језик који би био погодан да реши све алгоритамски-решиве проблеме. Међутим, то се није показало као право решење. Различити приступи су створили различите програмске парадигме (процедуралну, функционалну, логичку, објектно-оријентисану,…) и показало се да је свака од њих погоднија за решавање само одређене потклase проблема, али ниједна од њих није успела да изнедри општеприхваћени универзални програмски језик.
- Следеће адресе су покушаји да се хронолошки наброје неки од најутицајнијих програмских језика и да се стекне утисак о њиховој бројности и разноврсности:
- https://www.levenez.com/lang/lang_a4.pdf
- https://dev.vcdevhub.com/technolush/uploads/2019-04-04/files/programming-languages-evolution-2-small.jpg
- https://3t7bol18ef963l8x6yzv7ja1-wpengine.netdna-ssl.com/wp-content/uploads/2015/07/Programming-Languages-Infographic.jpg
- http://rigaux.org/language-study/diagram-light.png
- (датум приступа: 8.4.2020)

Језички процесори

- **Језички процесори** су програми који преводе на машински језик запис програма на неком другом програмском језику. Уопштење језичких процесора су **језички преводиоци** који преводе запис са једног програмског језика на други или чак са једног **језика за означавање** (енгл. markup language) на други.
- Као што смо управо видели, асемблер је најједноставнији језички процесор који преводи запис програма на асемблерском језику у запис програма на машинском језику. Поједини језички процесори преводе програм на асемблерски језик, а потом се добијени запис даље преводи асемблером на машински језик.
- Основна мотивација за развој језичких процесора је једноставније и брже писање програма, али ништа мање важно је и једноставније одржавање програма.
- Наравно, одређена цена се плаћа јер се троши време на превођење програма. Међутим, кад се све узме у обзир, значајно је већа уштеда времена потребног за писање и одржавање програма од времена изгубљеног на рад језичких процесора, тј. на превођење програма на машински језик.

Грешке у програму

- Приликом писања програма појављују се и грешке које можемо поделити на следеће групе:
  - синтаксичке грешке (енгл. syntax errors) и
  - семантичке грешке (енгл. semantic errors).
- **Синтаксичке грешке** обухватају све грешке настале непоштовањем граматичких правила програмског језика, укључујући грешке које бисмо можда могли да посматрамо одвојено — лексичке грешке попут употребе недозвољених симбола којих нема у азбуци језика, некоректно записаних бројева (запета уместо децималне тачке), датума итд.
- Један од задатака језичких процесора је откривање и указивање на синтаксичке грешке у програму јер се део програма који садржи синтаксичку грешку не може превести.

Семантичке грешке у програму

- Семантичке грешке се могу јавити у програму који је синтаксички исправан, преведен и ради, али не ради у складу са оним што је постављено као циљ пре писања програма, тј. производи некоректан или нежељени резултат.
- Баналне семантичке грешке су последица грешака у куцању (на пример, уместо да неки резултат подели са 100, програм дели са 10 јер нисмо откуцали још једну нулу) или превида (на пример, ако се не узму у обзир сви случајеви који могу да наступе приликом неког рачунања).
- Програм са семантичким грешкама у појединим случајевима може да произведе огромне материјалне штете (на пример милијарду долара због експлозије ракета Аријана-5 1996. године после само 40 секунди од лансирања), па чак и да угрози људске животе.
- Семантичке грешке се тешко откривају, али постоје и алатке за то. По популарном енглеском називу за семантичку грешку, **bug** (код нас преведено као *баг* или као *буба*), програм за њихово откривање се на енглеском назива **debugger** (код нас превођено као требљач, требилица, па чак и дибагер, дебагер,…)

Врсте језичких процесора

- Основна подела језичких процесора за програмске језике је на **компилаторе** (у ужем смислу) и **интерпретаторе**. Понекад се чак појам „компилатор“ (у ширем смислу) изједначава са појмом „језички процесор“, па се интерпретатор сматра специјалним случајем компилатора у ширем смислу.
- Могуће је развити за исти програмски језик и компилатор (у ужем смислу) и интерпретатор, па је погрешно рећи да је неки језик компилаторски или интерпретаторски, већ само да је нека његова имплементација компилаторска или интерпретаторска.
- Дефинисаћемо и упоредићемо компилаторе и интерпретаторе на примеру два језичка процесора за програмске језике C и Python који раде на рачунару са оперативним системом Windows.

Изворни кôд (C)

- Почетни запис програма на неком од машински независних (виших програмских) језика се назива **изворни кôд.**
- У прозору са белом позадином наводимо пример једноставног изворног кода на програмском језику C у датотеци **zbir.c** који од корисника очекује да унесе два цела броја, а потом исписује њихов збир. У црном прозору је резултат тестирања програма из командне линије.

```
#include <stdio.h>

void main()
{
    int a;
    int b;
    int c;

    printf("Unesi dva cela broja i pritisni Enter:\n");
    scanf("%d%d", &a, &b);
    c = a + b;
    printf("%d\n", c);
}
```

```
D:\c-examples>zbir.exe
Unesite dva broja i pritisnite Enter:
3
6
Zbir brojeva 3 i 6 je 9

D:\c-examples>
```

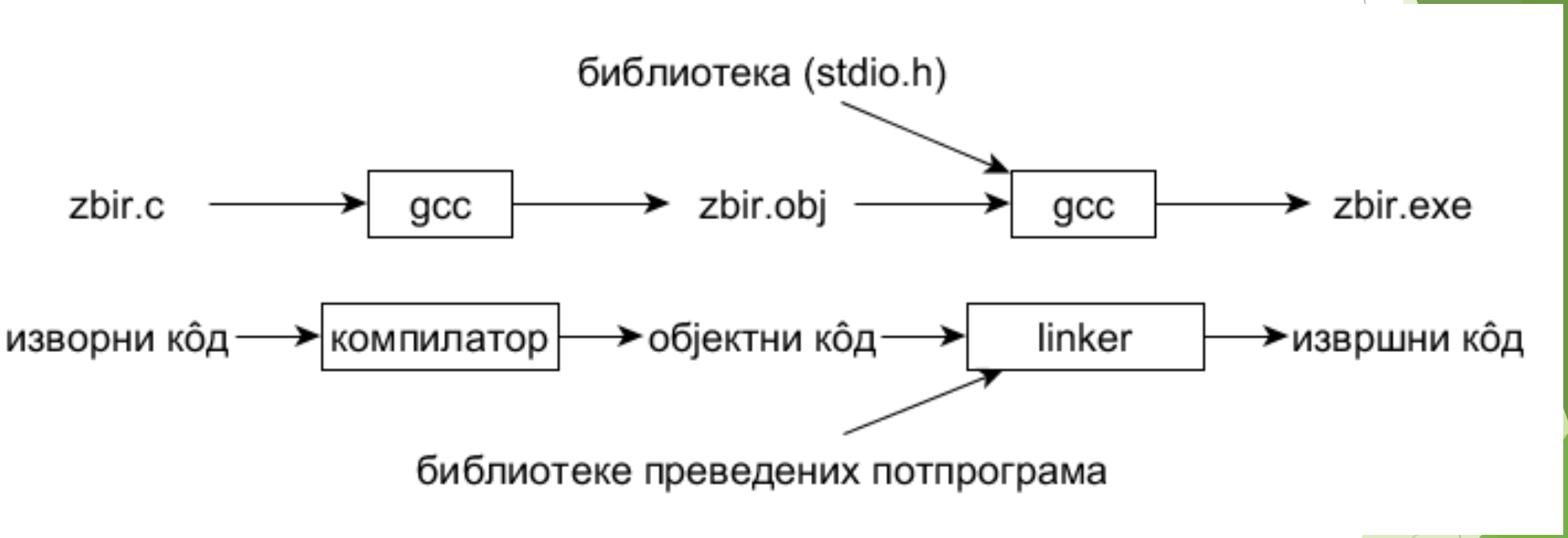
Библиотеке потпрограма

- Свака имплементација језичког процесора за неки програмски језик испоручује се са колекцијом (библиотеком) већ преведених потпрограма (на пример функције или процедуре којима се регулише улаз са тастатуре или исписивање на екран, функције за рад са нискама, целим бројевима, датумима итд.).
- Приликом превођења записа програма, језички процесор или самостално или уз помоћ посебног софтвера (енгл. linker) повезује резултат свог превођења са библиотекама већ преведених потпрограма чији су позиви наведени у програму.
- У претходно наведеном програму (**zbir.c**) **scanf** и **printf** су примери потпрограма из већ преведене библиотеке **stdio** за стандардни улаз (**scanf**) и излаз (**printf**).

gcc: компилатор за C

- gcc је пример слободног софтвера отвореног кода који се користи као компилатор (у ужем смислу) за C. Као и сваки компилатор, gcc :
  - најпре преводи наредбу по наредбу програма и на основу изворног кода у датотеци zbir.c генерише тзв. објектни (циљни) кôд програма у посебној датотеци (на Windows-у такве датотеке имају тип .obj)
  - после повезивања (које ради сам gcc) са већ преведеном библиотеком програмског језика C (у овом примеру stdio), gcc прави извршни кôд, тј. датотеку коју корисник може да изврши (на Windows-у такве датотеке имају тип .exe).
- Процес компилације програма написаног на C-у на примеру датотеке **zbir.c** описан је дијаграмом на следећем слајду.

Како ради компилатор за за C



Рад компилатора (закључак)

- Компилатори преводе комплетан изворни кôд и генеришу програм који се може извршити, тј. извршни кôд или извршни програм (на пример, датотеке типа .EXE на оперативном систему Windows).
- Довољно је једном успешно превести програм компилатором, а онда се извршни кôд може користити произвољан број пута. Међутим, докле год у програму постоји бар једна синтаксичка грешка, компилатор не може да генерише извршни програм.
- Изворни кôд програма није потребан за коришћење извршног кода, што олакшава дистрибуцију власничког извршног кода.
- Као што је већ речено, рад на језику FORTRAN је започео 1954. године, а дистрибуција првог компилатора за FORTRAN започела је 1957. године. Од тада су направљени компилатори (у ужем смислу) за већину програмских језика (COBOL, Pascal, C, C++, Java, C# итд.)

Изворни кôд (Python) и интерпретатор (python)

- Доле лево је пример једноставног изворног кôда на програмском језику Python у датотеци **zbir3.py** који (као и **zbir.c**) очекује од корисника да унесе два цела броја, а потом исписује њихов збир. Доле десно је резултат извршавања програма из командне линије позивом интерпретатора који се зове као и језик (python).

zbir3.py - Notepad

File Edit Format View Help

```
a = int(input("Unesite ceo broj a\n"))
b = int(input("Unesite ceo broj b\n"))

c = a + b

print(f"Zbir brojeva {a} i {b} je {c}\n")
```

C:\Windows\System32\cmd.exe

```
D:\python-examples>py zbir3.py
Unesite ceo broj a
5
Unesite ceo broj b
12
Zbir brojeva 5 i 12 je 17
```

Како ради интерпретатор

- Интерпретатор преводи и одмах извршава једну по једну наредбу изворног кода програма (интерпретира наредбу, по чему је добио име) без генерисања извршног кода (дакле, python не генерише **zbir2.exe**).
- Кориснику је неопходан изворни кôд програма да би могао да га изврши и сваки пут када програм треба поново да се изврши, изворни кôд мора поново да се преводи и интерпретира.
- Први виши програмски језик за који је направљен интерпретатор је LISP (језик је настао 1958. године, а током наредне године и први његов интерпретатор).

Поређење компилатора и интерпретатора

- Власнички софтвер се лакше дистрибуира у случају компилације јер је довољно испоручити извршни кôд без изворног кода и компилатора, док у случају интерпретације корисник мора да има и изворни кôд (читљив или не) и сам интерпретатор да би могао да изврши програм.
- Интерпретирање је спорије у односу на компилацију јер се кôд не само преводи, већ и извршава.
- У случају синтаксичких грешака у програму, компилатор не може да генерише извршну датотеку, док интепретатор извршава једну по једну наредбу програма док не дође до синтаксичке грешке. У том смислу, интерпретатор прецизније открива позицију грешке у програму, а поједини интерпретатори „памте“ већ преведени исправни део програма, па га после исправке грешке не преводе поново, већ га само интерпретирају, тј. настављају са превођењем и интерпретирањем од места исправке.

Комбиновање компилације и интерпретирања

- Поједини језички процесори, на пример за програмске језике Java, .NET-језике (C# итд.), Python и друге, комбинују компилацију и интерпретацију. Прецизније, изворни кôд се најпре преводи у посебну репрезентацију која се назива бајт-кôд. Бајт-кôд је заправо запис програма на некој врсти машинског или асемблерског језика за фиктивни процесор чији се рад симулира софтверски (такав процесор се назива виртуелна машина, енгл. virtual machine). Софтвер за симулацију виртуелне машине је уједно и интерпретатор који анализира и извршава инструкције виртуелне машине.
- Иако се употребом виртуелне машине губи на брзини извршавања програма, добија се на сигурности. Наиме, идеја је да програм који се извршава на виртуелној машини може да изазове само њен крах, а не и правог рачунара на коме се извршава виртуелна машина.

Још једна подела програмских језика (на генерације)

- Постоји и подела програмских језика на генерације:

1. генерација: машински језици
2. генерација: остали машински зависни језици (асемблерски језици и њихова надградња, нпр, макроасемблерски језици)
3. генерација: процедурални (FORTRAN, PASCAL, C,…), функционални (LISP, Haskell, F#,…), логички (PROLOG), објектно-оријентисани програмски језици (C++, Java, C#, Python,…)…
4. генерација: непроцедурални описни језици (нпр. SQL, R,…)
5. генерација: неуспешни покушај да се развије програмски језик који решава проблем без учешћа програмера, корисник само описује проблем и ограничења.
6. генерација:…

Интегрисана развојна окружења

- Како би се програмерима олакшао рад, постало је уобичајено да се у једном програмском пакету обједине сви потребни програми и то:
  - програм за уређивање текста (енгл. text editor) који подржава додатне могућности попут посебног означавања синтаксичких конструкција језика (на пример, различите боје за наредбе, ниске карактера, итд.), контролу отворених и затворених заграда и наводника, памћење претходних позиција у програму које се преправљане, аутоматско допуњавање наредби или убацивање шаблона, аутоматско преименовање имена у целом програму, итд;
  - компилатор или интерпретатор;
  - програм за откривање семантичких грешака (раније споменути debugger, односно требљач, требилица, дибагер,…)
  - документацију програмског језика;
  - разне помоћне алатке за писање тестова да ли програм ради коректно, тестирање регуларних израза, дизајнирање графичког корисничког сучеља (енгл. Graphical User Interface, скр. GUI) итд.
- Такви програмски пакети се називају интегрисана развојна окружења (енгл. Integrated Development Environment, скр. IDE).
- Примери: Visual Studio, SharpDevelop, Eclipse, BlueJ, NetBeans,…