Математички факултет

Универзитет у Београду

МАСТЕР РАД

Електронске лекције о програмском језику Delphi за ученике средњих школа

Ментор:

проф. др Мирослав Марић

Кандидат:

Милица Јовановић, 1121/2013

Београд, 2016.

Садржај

Увод

У образовању је, услед технолошког развоја, заживео концепт учења које се заснива на информационо-комуникационој технологији - елктронског учења. Електронско учење подразумева прикупљање информација и усвајање садржаја разноразних области посредством информационих и комуникационих уређаја (персоналних и преносних рачунара, дигиталне телевизије, мобилних телефона итд.). Поред наведених уређаја, пренос и прикупљање информација, као и комуникација је умногоме олакшао развој глобалне рачунарске мреже – интернета.

Коришћењем рачунара се може унапредити настава на више начина, а један од њих је креирање електронских лекција и њихово дистибуирање путем интернета. Један од значаја електронских лекција јесте могућност представљања садржаја преко видео и аудио материјала. Поред тога, јавна доступност лекција омогућава стицање знања и вештина не само ученика у формалном процесу учења, већ и свим категоријама корисника који су заинтересовани за одређену област [1], [2].

У овом раду су представљене електронске лекције о програмском језику Delphi и намењене су ученицима средњих школа у циљу олакшавања процеса учења основних концепата овог програмског језика. Електронске лекције се заснивају на програмирању и програмском језику Delphi из више разлога. Учење програмирања у школском узрасту може допринети ефиксаном коришћењу програмских језика за решавање различитих проблема у даљем образовању, професионалном раду и свакодневном животу. Конкретно, учење програмског језика Delphi је значајно због могућности учења објектно-оријентисаног концепта програмирања. Стицањем знања и вештине програмирања коришћењем програмског језика Delphi, лако се може научити неки други програмски језик сличних особина.

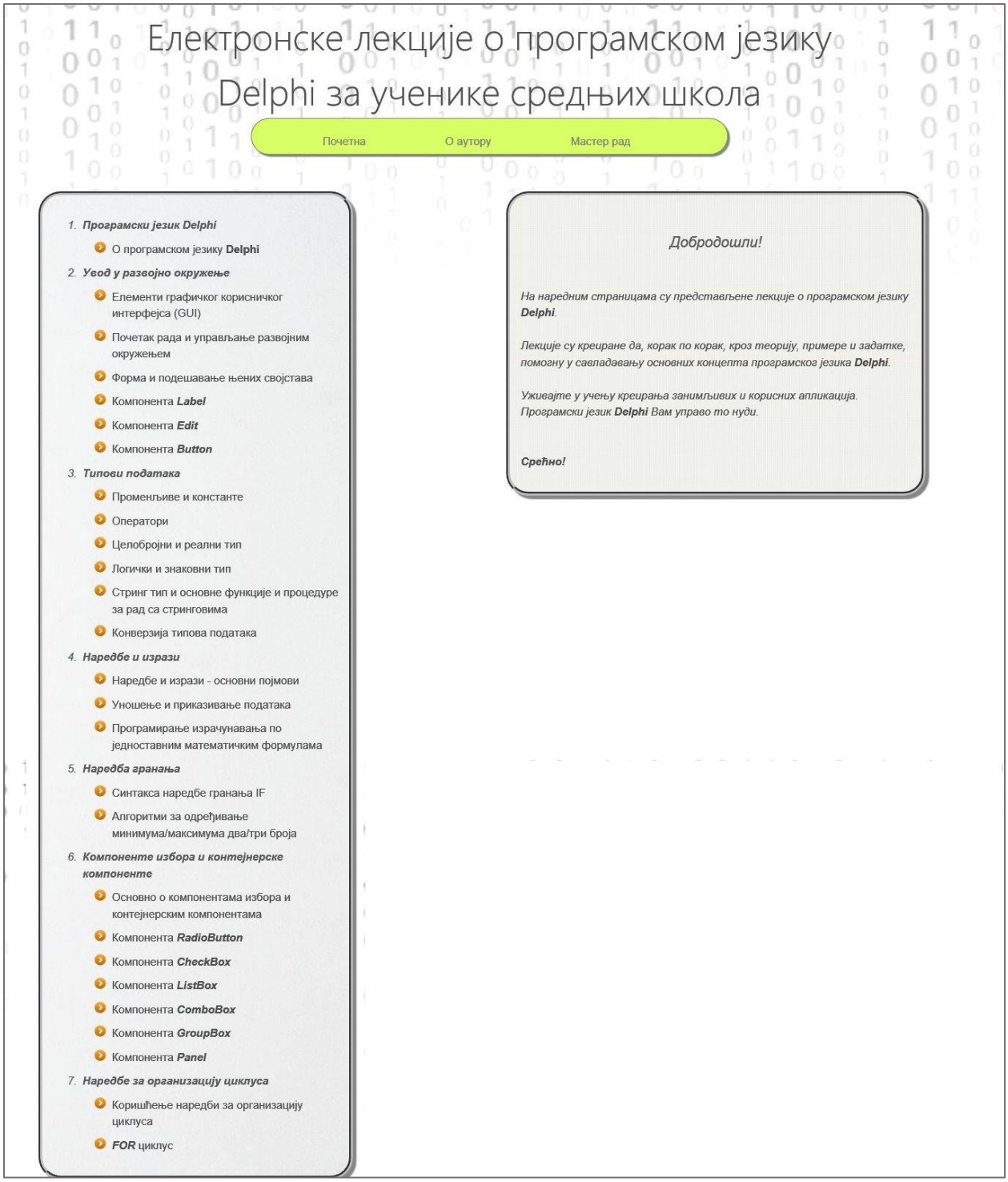
Што се самог одабира програмског језика Delphi тиче, могућа мана је постојање једноставнијих програмских језика, уколико су циљна група почетници у програмирању. Иако су лекције креиране са претпоставком да ученик нема предзнања, усвајање основних појмова и принципа везаних за Delphi би можда било једноставније, уколико би ученик имао знање о основним појмовима неког једноставнијег програмског језика.

1. Електронске лекције о програмском језику Delphi за ученике средњих школа

Електронске лекције о програмском језику Delphi за ученике средњих школа се налазе на адреси:

http://www.alas.matf.bg.ac.rs/~ml09140/Master/

Почетна страна је приказана на слици 1.



Слика 1. Почетна страна

На почетној страни се налазе називи лекција. Лекције су груписане у седам различитих целина:

1. Програмски језик Delphi;
2. Увод у развојно окружење;

3. Типови података;
4. Наредбе и изрази;
5. Наредба гранања;
6. Компоненте избора и контејнерске компоненте;
7. Наредбе за организацију циклуса.

Свака страна електронских лекција садржи: главни наслов и навигациони мени, стрелице за кретање кроз лекције, наслов лекције и садржај лекције.

Главни наслов и навигациони мени су приказани на слици 2.

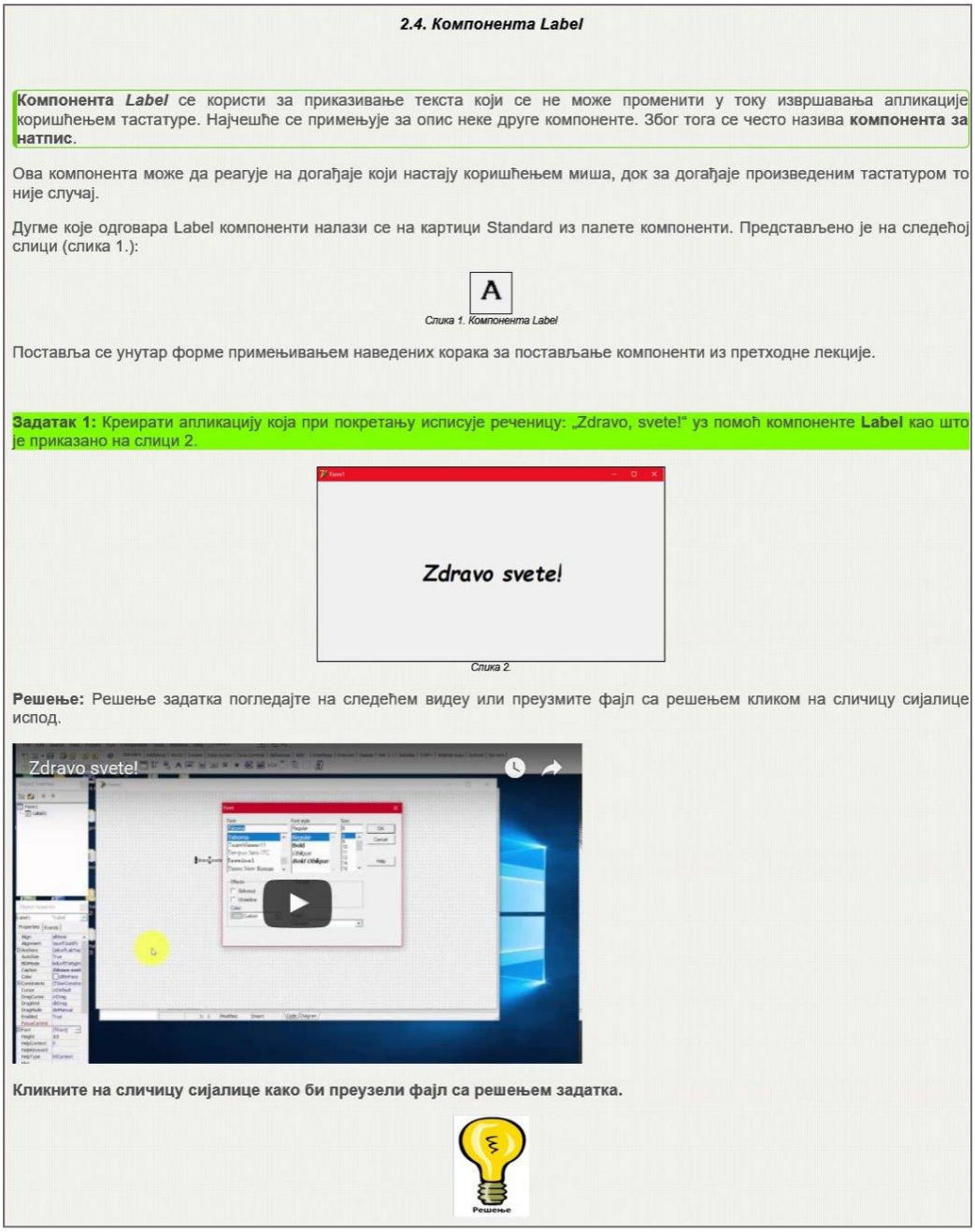
Слика 2. Главни наслов и навигациони мени

На слици 2. се може видети да навигациони мени садржи три текстуална менија. Мени под називом „Почетна“ има функцију да враћа приказ почетне стране. „О аутору“ приказује страну где се наводи аутор лекција, а „Мастер рад“ приказује PDF фајл који садржи мастер рад.

Стрелице за кретање кроз лекције (слика 3.) помажу бржем кретању кроз лекције. Стрелица усмерена налево приказује претходну лекцију, док стрелица усмерена надесно приказује наредну лекцију.

Слика 3. Стрелица за кретање кроз лекције

На слици 4. се може видети наслов и делимичан приказ садржаја лекције под називом „Компонента Label“.



2.4. Компонента Label

Компонента *Label* се користи за приказивање текста који се не може променити у току извршавања апликације коришћењем тастатуре. Најчешће се примењује за опис неке друге компоненте. Због тога се често назива **компонента за натпис**.

Ова компонента може да реагује на догађаје који настају коришћењем миша, док за догађаје произведеним тастатуром то није случај.

Дугме које одговара Label компоненти налази се на картици Standard из палете компоненти. Представљено је на следећој слици (слика 1.):

A

Слика 1. Компонента Label

Поставља се унутар форме примењивањем наведених корака за постављање компоненти из претходне лекције.

Задатак 1: Креирати апликацију која при покретању исписује реченицу: „Zdravo, svete!“ уз помоћ компоненте **Label** као што је приказано на слици 2.

Zdravo svete!

Слика 2.

Решење: Решење задатка погледајте на следећем видеу или преузмите фајл са решењем кликом на сличицу сијалице испод.

Zdravo svete!

Кликните на сличицу сијалице како би преузели фајл са решењем задатка.

Решење

Слика 4. Пример дела лекције „Компонента Label“

Текст који дефинише појам у лекцији се налази унутар оквира зелене боје.
Садржај лекције је пропраћен сликама које допуњују и додатно објашњавају текст.
У великом броју лекција се налазе задаци, који обједињују теоријски садржај. Текстови задатака су визуелно издвојени од осталог текста, јер се налазе у зеленим правоугаоницима. Задаци су пропраћени сликом изгледа апликације која одговара захтеву задатка и решењем, које је у већини случајева представљено у виду видео материјала. Поред тога, решење се може преузети кликом на сличицу сијалице. Уколико задатак захтева писање програмског кода, решење садржи и исечак кода апликације.

2. Програмски језик Delphi

Delphi је програмски језик високог нивоа, који омогућава писање како структурних, тако и објектно-оријентисаних апликација [3].

Delphi се (према [4]) дефинише као комбинација програмског језика и окружења за софтверски развој (Software Development Kit, SDK) која омогућава креирање самосталних, мобилних, конзолних и веб апликација.

Програмски језик Delphi има јасно дефинисану синтаксу и једноставан је за употребу. Програмски кôд писан коришћењем програмски језик Delphi је лак за читање. Поседује интегрисано развојно окружење (Integrated Development Environment, IDE), што омогућава: развијање изгледа апликације методом превуци-и-пусти (drag-and-drop), додавање догађаја итд. Подржава концепт брзог развоја апликација (Rapid Application Development, RAD). Поред развоја апликација, Delphi се може користити за имплементацију рачунарске графике и базе података [4].

2. 1. Историјат и верзије програмског језика Delphi

Компанија Borland је 1995. године направила језик Delphi, наследника програмског језика Pascal.

Pascal је програмски језик који је 1970. године креирао швајцарски информатичар Никлаус Вирт. Језик је дизајниран тако да буде ефикасан и у први план стави добру програмерску праксу тога времена (пре свега структурно програмирање). Иако никада није био доминантан у индустрији, Pascal се интензивно користио у настави програмирања. Године 1983. језик је стандардизован (ISO 7185:1983) и од тада се појавило неколико верзија стандарда. Током 1980-их најпопуларније развојно окружење за Pascal био је Turbo Pascal компаније Borland.

С популаризацијом објектно-оријентисане парадигме развија се нацрт језика Object Pascal, који проширује језик Pascal концептима објектно-оријентисаног програмирања.

На основу нацрта језика Object Pascal, компанија Borland је 1995. године креирала језик и интегрисано развојно окружење Delphi, чиме је од језика Pascal направљен први визуелни програмски језик.

Delphi је 2008. од компаније Borland откупила компанија Embarcadero, па се језик и развојно окружење данас називају Embarcadero Delphi [3], [5].

Верзије програмског језика Delphi (према [3]):

1. Delphi 1 – прва верзија која је настала 1995. године. Омогућила је објектно-оријентисано програмирање, примену форми као интеграцију са Windows окружењем, везу са базама података, преводилац;

2. Delphi 2 – друга верзија, настала 1996. године. Ова верзија у односу на претходну, садржи: пун Win32 преводилац (пуна подршка за Windows 95), подршку за велике стрингове, наслеђивање форми, итд;
3. Delphi 3 – трећа верзија, настала 1997. године;
4. Delphi 4 – четврта верзија, настала 1998. године. У верзији Delphi 4 су уведени: динамички низови, преклапање метода, подршку за Windows 98 итд;
5. Delphi 5 – пета верзија, настала 1999. године. Овој верзији је додата подршка за различита окружења за desktop апликације, концепт оквира (frames), подршка за XML итд;
6. Delphi 6 – шеста верзија, настала 2000. године. Подржава веб сервисе као основе бизнис пословања на интернету. Такође је додата подршка за унакрсну компилацију како за Windows, тако и за Linux;
7. Delphi 7 – седма верзија, настала 2001. године. Ова верзија је донела интеграцију за Microsoft.Net технологије;
8. Delphi 8 – осма верзија, настала 2002. године. Унапређује VCL (Visual Component Library) компоненте и CLX (Component Library for Cross platform) компоненте, преводилац, део за дизајн итд;
9. Delphi 2005;
10. Delphi 2006.

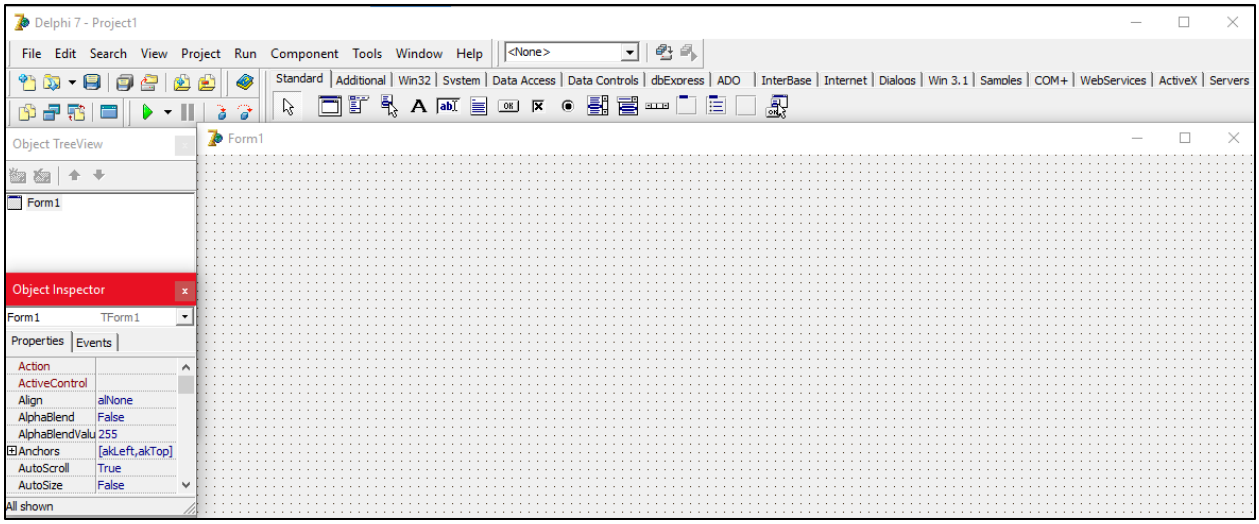
У оквиру електронских лекција о програмском језику Delphi за ученике средњих школа се обрађује верзија Delphi 7.

3. Увод у развојно окружење

У оквиру дела „Увод у развојно окружење“, кроз електронске лекције су представљени елементи интегрисаног окружења за развој програмског језика Delphi и начин на који се тим елементима управља. Затим је обрађен део који се односи на форму и подешавање њених својстава, а након тога су обрађене једноставније визуелне компоненте: *Label*, *Edit* и *Button*.

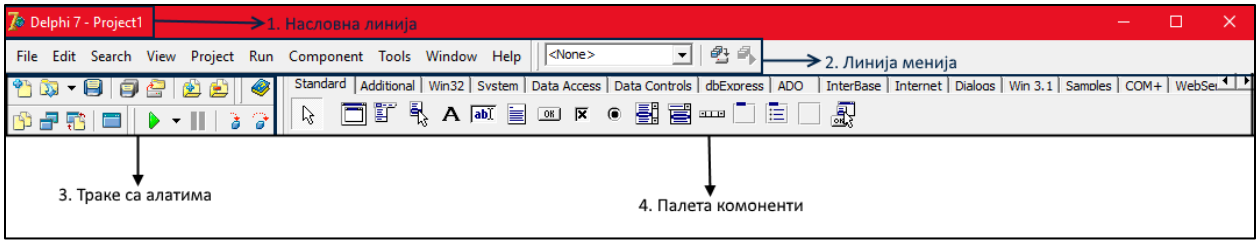
3.1. Елементи графичког корисничког интерфејса (GUI)

Графички кориснички интерфејс (GUI - Graphical User Interface) је део система који служи за комуникацију између корисника и самог система. Представља јако битан део јер је једини видљив спољашњим корисницима.
Графички кориснички интерфејс користи визуелне елементе, као што су: прозори, менији, дугмад, итд.
На слици 5. је приказана празна форма и интегрисано окружење за развој програмског језика Delphi.



Слика 5. Delphi окружење и празна форма

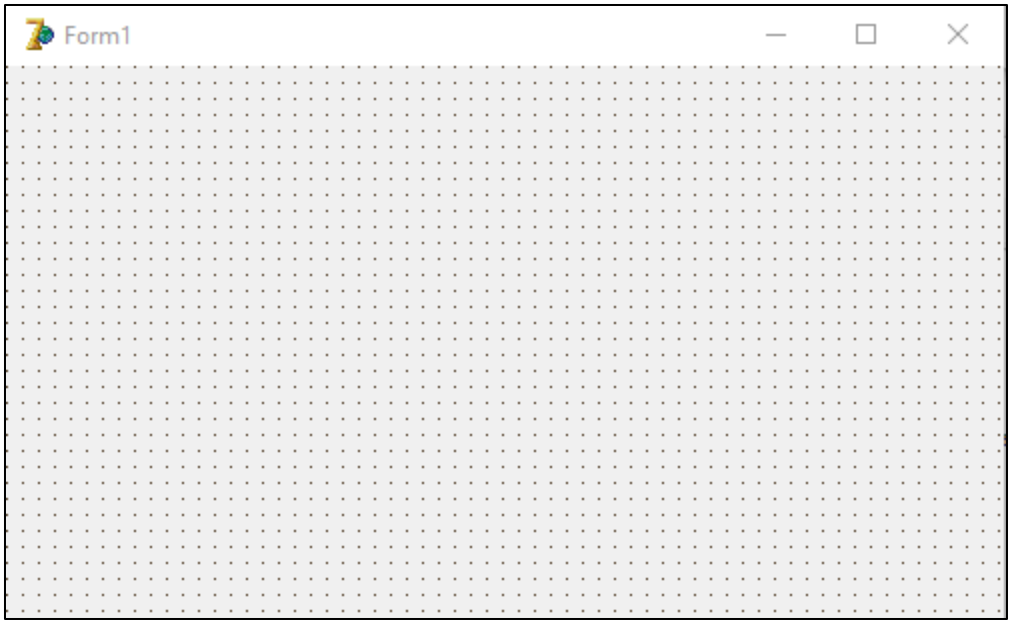
Delphi окружење се састоји из неколико делова, који нису спојени, тј. може им се мењати позиција уз помоћ миша.
Прозор који се налази на врху (слика 6.) и који се може сматрати главним прозором, садржи: насловну линију, линију менија, траке са алатима и палету компоненти.



Слика 6. Главни прозор Delphi окружења

1. **Насловна линија** садржи иконицу, назив програмског језика - Delphi 7 и назив пројекта. Уколико није промењен назив и уколико је први пројекат писаће Project1;
2. **Линија менија** садржи падајуће меније са алатима који служе за управљање развојним окружењем;
3. **Траке са алатима** (Toolbars) омогућавају да се креира, отвори, сачува, покрене апликација, итд...
4. **Палета компоненти** (Component Palette) садржи широк спектар компоненти, које су груписане, што олакшава њихово коришћење. Називи група компоненти налазе се на језичцима дуж горњег дела палете.

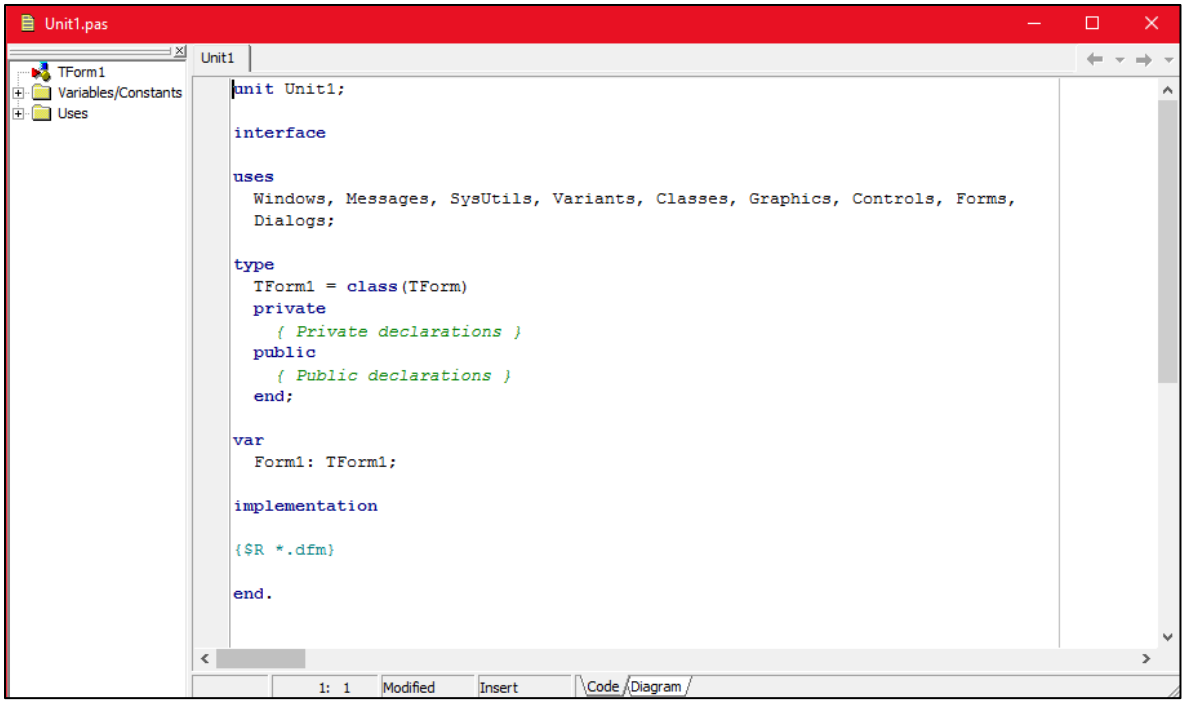
Главни део Delphi окружења заузима радна површина – форма (слика 7.) која служи за дизајнирање апликација. Налази се испод главног прозора, са десне стране.



Слика 7. Празна форма

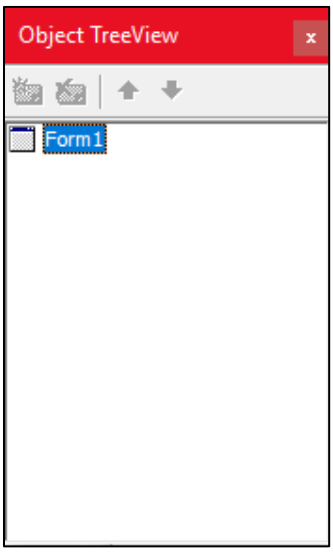
Форма представља прозор у оквиру програма и служи да се на њу поставе компоненте. У оквиру форме се такође мења позиција и величина компоненти. Позиционирање компоненти олакшава истачкана површина саме форме.
Иако је празна, форма поседује особине Windows прозора: може се премештати, могу јој се мењати димензије, може се минимизирати, максимизирати и затворити. Има и своју иконицу која се додељује при минимизацији.

Иза прозора за форму, налази се **едитор програмског кода** (слика 8.) који служи за модификовање кода програма.



Слика 8. Едитор програмског кода

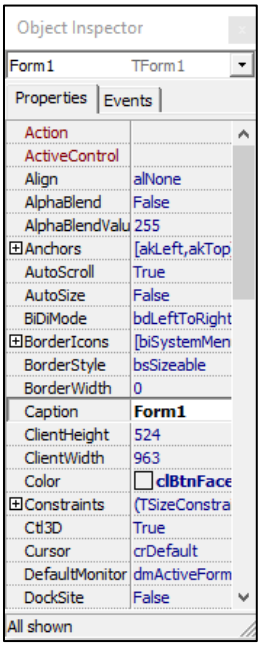
Испод главног прозора, у левом делу, налази се **прозор за хијерархијски приказ компоненти апликације** (Object TreeView) у коме иницијално пише Form1 уколико није промењен назив форме и уколико унутар форме није додата ниједна компонента. Прозор за хијерархијски приказ компоненти апликације је приказан на слици 9.



Слика 9. Прозор за хијерархијски приказ компоненти апликације

У тренутку постављања компоненте унутар форме, у овом делу се уписује њен назив, што омогућава кориснику преглед свих компоненти које је поставио. Сва имена компоненти су иницијално додељена и уколико форма садржи више компоненти истог типа, нумерација се

врши природним бројевима: 1, 2, 3... Промена имена компоненте, одражава се аутоматски у овом прозору.
Врло важан део Delphi окружења представља инспектор објеката (Object Inspector) који је приказан на слици 10.



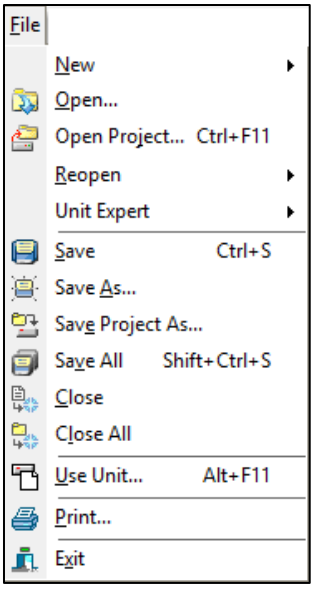
Слика 10. Инспектор објеката

Инспектор објеката служи за подешавање карактеристика и догађаја компоненти.
Инспектор објеката има два језичка: **Properties** и **Events**. Језичак Properties се односи на карактеристике, а Events на догађаје постављених компоненти.
Карактеристике које се могу мењати компонентама су на пример: наслов, висина, ширина, боја, име компоненте итд. Услед модификације карактеристика, резулатати се истовремено приказују, уколико је промена карактеристика могућа. Листа карактеристика које су доступне разликују се од компоненте до компоненте, али и поред тога постоје карактеристике које су заједничке за све компоненте.
Догађај је нешто што се појављује као резултат интеракције компоненте са корисником или са Windows оперативним системом. На пример, може се креирати догађај који се генерише када корисник кликне на неку компоненту. Као и код карактеристика, свака компонента има индивидуално скуп догађаја на који може реаговати.

3. 2. Почетак рада и управљање развојним окружењем

У првом делу електронске лекције „Почетак рада и управљање развојним окружењем“ представљени су менији из главне линије менија Delphi окружења.

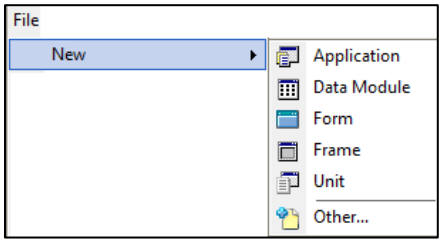
File мени (слика 11.) је први (по редоследу) мени из линије менија.



Слика 11. File мени

У овом менију се најчешће користе следеће опције:

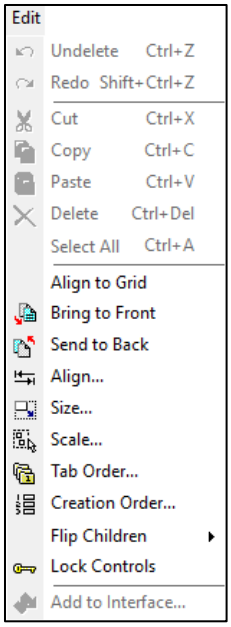
- *New* - служи за креирање нових фајлова. Стрелица усмерена надесно означава да постоји подмени (слика 12.) који нуди креирање: нове апликације (опција *Application*), модула за податке (опција *Data Module*), форме (опција *Form*), оквира (опција *Frame*) итд;



Слика 12. Подмени New

- *Open* и *Open Project* - служе за отварање сачуваних Delphi пројеката или модула;
- *Save*, *Save As*, *Save Project As* и *Save All* - служе за чување фајлова и промена извршених у току рада.

Следећи мени из линија менија јесте *Edit*, који је приказан на слици 13.

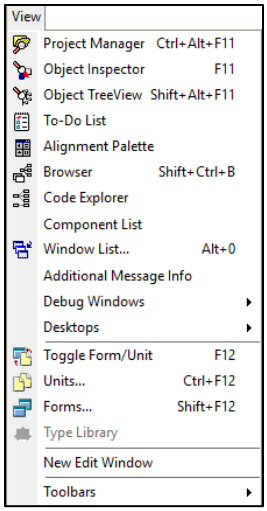


Слика 13. Edit мени

Из *Edit* менија се најчешће користе следеће опције:

- *Undelete* – служи да се грешком обрисана компонента врати назад;
- *Cut* - служи да селектовану компоненту обрише, али истовремено и смести на Clipboard, што омогућава њено коришћење уколико је то потребно;
- *Copy* - слична је претходној опцији *Cut*, са разликом да селектовану компоненту не брише. Уз помоћ ове опције прави се копија и погодна је уколико је потребно да се унутар форме поставе више компоненти истог типа и истих особина. Копије попримају све особине оригинала, једино се име мења, што је природно, будући да две различите компоненте не могу имати исто име;
- *Paste* - служи за „лепљење“, тј. постављање садржаја који су настали услед коришћења опција *Cut* и *Copy*;
- *Delete* - служи за брисање селектоване компоненте. Ова опција се разликује од опције *Cut*, јер се обрисана компонента не може додати уз помоћ *Paste* опције. Може се једино вратити уз помоћ опције *Undelete*;
- *Select All* - служи за селектовање свих компоненти које су постављене унутар форме.

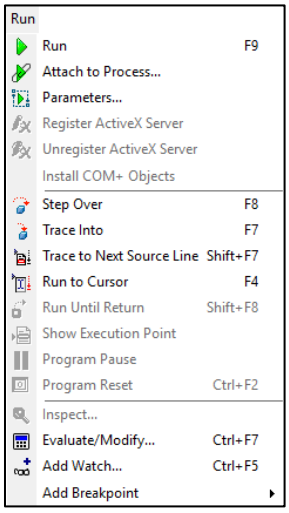
Мени *View* (слика 14.) садржи доста опција, али се најчешће користе следеће:



Слика 14. View мени

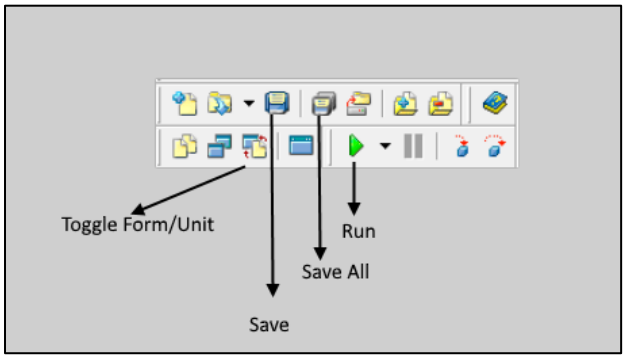
- *Object Inspector* - служи за укључивање или искључивање приказа инспектора објеката. Пожељно је да је овај прозор приказан у току рада јер се често користи;
- *Object Tree View* - служи за укључивање или искључивање прозора за хијерархијски приказ компоненти апликације;
- *Toggle Form/Unit* - служи за мењање приказа радне површине и едитора кода;
- *Toolbars* - служи за укључивање и искључивање приказа елемената развојног окружења који се налазе на траци са алатима или палете компоненти.

Мени *Run* (слика 15.) садржи опцију *Run* којом се покреће програм под условом да не постоје грешке при креирању програма. Уколико програм садржи грешку, преводилац је пријављује у виду поруке која се налази у доњем делу едитора програмског кода.



Слика 15. Run мени

Многе опције из поменутих менија се могу користити уз помоћ одговарајућих дугмади из траке са алатима (слика 16.).



Слика 16. Траке са алатима

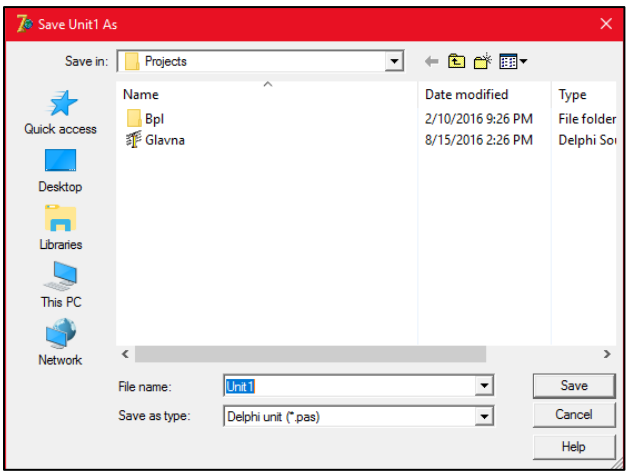
Такође постоје и тастери или комбинација тастера са тастатуре које омогућавају коришћење наведених опција.
Неке од тих пречица су: *Ctrl+S* (идентична је опцији *Save*), *Ctrl+Del* (идентична је опцији *Delete*), *F9* (одговара опцији *Run*) итд. Све пречице које се могу користити при раду су наведене десно од имена опција у одређеним менијима. Поред тога, приликом позиционирања курсора миша на одговарајуће дугме из траке са алатима, поред имена опције пише и одговарајућа скраћеница.

3. 2. 1. Чување, покретање и отварање сачуваних апликација

У оквиру електронске лекције „Почетак рада и управљање развојним окружењем“ је детаљно објашњено како се могу сачувати, покренути и отворити сачуване апликације. Свако објашњење је пропраћено сликама.

Апликације се чувају коришћењем опције *Save All* из менија *File* или кликом на дугме пречице *Save All* из траке са алатима.

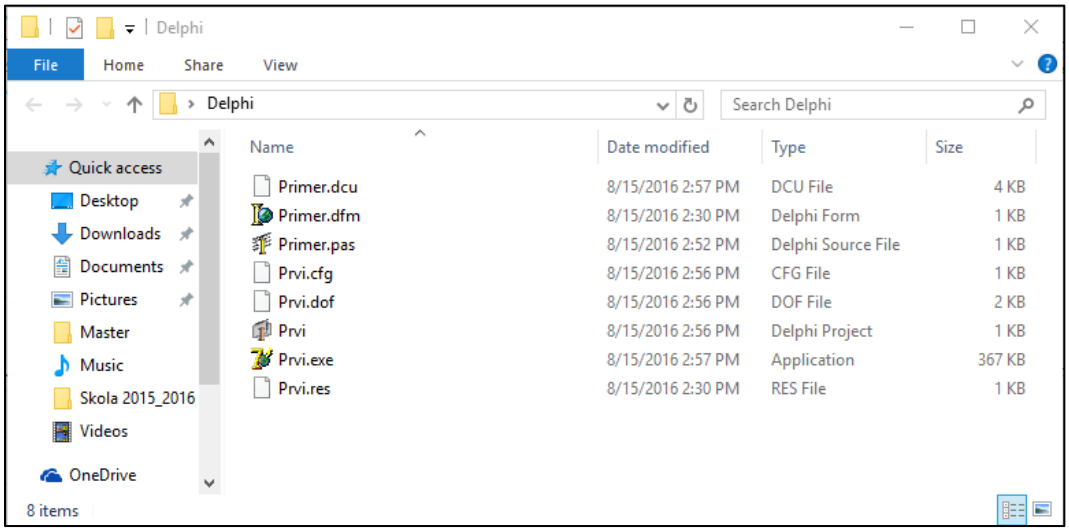
Одабиром ове опције приказује се прозор који се може видети на слици 17.



Слика 17. Прозор који се отвара након одабира опције Save All

У делу *Save in* врши се одабир директоријума у ком ће апликација бити сачувана.
У делу *File name*, који се односи на име фајла модула форме, биће понуђено генеричко име *Unit1*. Битно је напоменути да име фајла модула форме не сме бити исто као име форме, да не би дошло до колизије имена.
Уколико се, на пример, имену фајла модуле форме додели име Primer, а затим одабере опција *Save*, креираће се фајл Primer.pas. Фајл са наставком .pas садржи изворни кôд програма и не сме се брисати.
Након тога, у истом делу, треба креирати име пројекта (генеричко име ће бити *Project1*).
Уколико се, примера ради, имену пројекта додели име Prvi и одабрере опција *Save* аутоматски се додељује наставак .dpr, тако да је име пројектног фајла Prvi.dpr.
Приликом покретања апликације креира се извршни фајл са именом Prvi.exe.

На слици 18. се може видети сви фајлови који се генеришу након чувања и покретања програма:



Слика 18. Пример прозора фолдера у ком је сачувана Delphi апликација

Већ су споменути фајлови: Primer.pas, Prvi.dpr и Prvi.exe. Поред њих, генерисани су:

- фајл са наставком .dcu који представља преведени кôд програмског модула и може се брисати јер га Delphi поново креира кад год се преводи апликација;
- фајл са наставком .dfm који садржи вредности које представљају својства форме, као и својства свих компоненти које су постављене унутар форме. Ови фајлови се не смеју брисати;
- фајл са наставком .res који садржи бинарне ресурсе попут системске иконе програма и остале битмапе, а могу се креирати и мењати помоћу опције *Image Editor* из менија *Tools*.

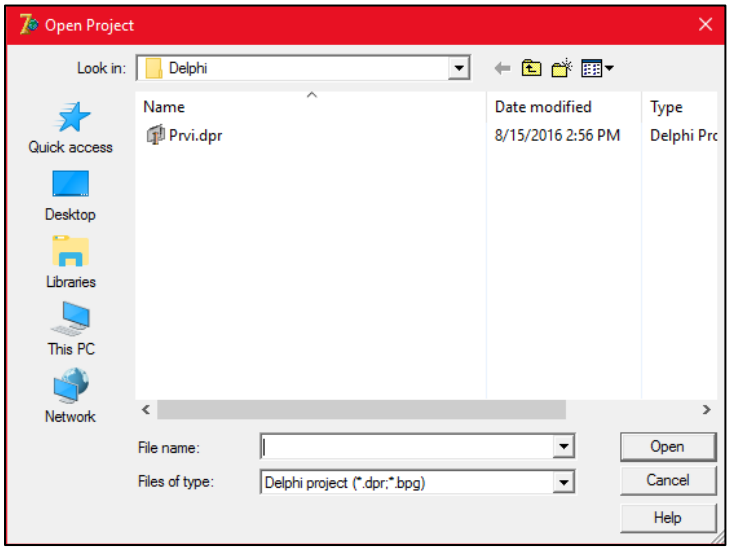
Опције *Save As* и *Save Project As* су погодне за чување активних модула и пројекта под неким другим именом. Коришћење ових опција је погодно уколико предстоји креирање

апликације која је слична некој претходној. Чување пројекта под другим именом и извршавање одређених корекција (због прилагођавања тренутном захтеву) знатно штеди време од рада на пројекту испочетка.

Креирана апликација се може покренути:

- одабиром опције *Run* из менија *Run*;
- притиском на дугме пречицу *Run* из траке са алатима (слика 16.);
- притиском тастера *F9* на тастатури.

Уколико је покренуто Delphi окружење, сачувана апликација се отвара коришћењем опције *Open Project* која се налази у менију *File*. Након одабира ове опције, отвара се прозор који се може видети на слици 19.



Слика 19. Пример прозора Open Project

У делу *Look in* се уписује име директоријума који садржи жељену апликацију. Након тога треба да се (једним кликом левог тастера миша) одабере фајл са наставком .dpr, а потом изабере опција *Open*.
Уколико Delphi окружење није покренуто, отварање сачуване апликације се врши тако што се из директоријума у ком је сачувана, двоструким кликом левог тастера миша отвори фајл са наставком .dpr.

Битно је напоменути да се апликација може покренути, а да се притом не покрене цело Delphi радно окружење. Поред тога, програм може да се пренесе на неки други рачунар, који нема инсталирано радно окружење програмског језика Delphi.
Све ово омогућава извршни фајл са наставком .exe који настаје након успешног превођења програма.

3. 3. Форма и подешавање њених својстава

Форма се спомиње као део Delphi окружења у лекцији „Почетак рада и управљање развојним окружењем“. У тој лекцији је укратко наведено шта форма означава, док су детаљнији опис, као и карактеристике и догађаји који су везани за форму представљени у лекцији под називом „Форма и подешавање њених својстава“. Теоријски део употпуњују слике које прате саджај ове лекције.

Форма је главни део за формирање Delphi апликација. Представља компоненту типа **TForm**.

На почетку, пројекат садржи једну форму, чији је иницијални назив *Form1*. Касније, по потреби, може се додати више форми, које могу бити повезане у апликацији на жељени начин.

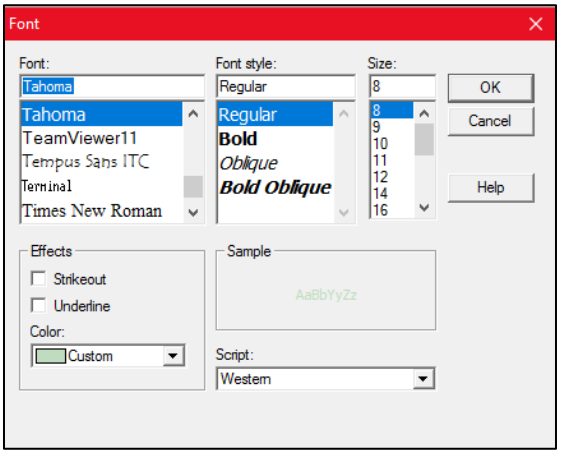
На слици 7. је био приказан изглед празне форме. Такође је наведено да се својства компоненти подешавају у инспектору објеката.

У лекцији је представљен следећи списак својстава форме које се најчешће подешавају:

- *Name* - служи за мењање имена форме. Форми се мења име тако што се десно од опције *Name* уписује жељено име. Измена ове особине ће се извршити притиском тастера *Enter* или притиском било ког тастера миша изван овог дела. Име форме не сме да садржи знак за празнину, као ни специјалне знакове, као што су на пример: интерпункцијски знаци, затим +, *, /, знак за размак итд. Уколико се јави потреба за одвајањем речи, може се користити знак за доњу црту „_“. Од тренутка мењања имена форме, форма се позива новим именом приликом писања кода програма;
- *Caption* - служи за мењање наслова форме. Промена ове карактеристике одразиће се у насловној линији. За разлику од имена форме, наслов форме може да садржи празна места и специјалне карактере;
- *Color* - служи за мењање боје форме. Након притиска левог тастера миша, десно од овог својства, појавиће се стрелица усмерена надоле, чијом се одабиром отвара падајући мени који нуди широку палету боја која форма може да има;
- *Top* - служи за задавање целобројне вредности која се односи на удаљеност форме од горње ивице екрана;
- *Left* - служи за задавање целобројне вредности која се односи на удаљеност форме од леве ивице екрана;
- *Width* - служи за задавање целобројне вредности која се односи на ширину форме;
- *Height* - служи за задавање целобројне вредности која се односи на висину форме;
- *Border Icons* – дефинише која се системска дугмад појављују у форми у току рада програма. Избор укључује системски мени, дугме за минимизирање, односно максимизирање форме и дугме за помоћ;
- *Autoscroll, HorzscrollBar, VertscrollBar* – контролишу траке за померање форми. Уколико је карактеристика *Autoscroll* дефинисана као True, траке за померање се аутоматски појављују уколико је форма сувише мала да би приказала све

компоненте. Карактеристике *HorzscrollBar* и *VertscrollBar* додатно имају неколико сопствених карактеристика које контролишу операције са траком за хоризонтално и вертикално померање;

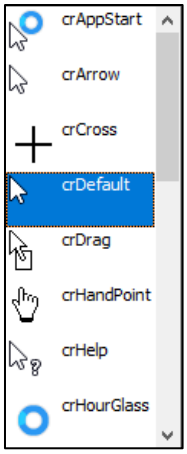
- *Font* – дефинише фонт који ће бити коришћен у форми. Дугме које се налази десно од натписа *Font* отвара прозор следећег изгледа (слика 20.).



Слика 20. Прозор Font

На слици 20. се може видети да се у оквиру прозора *Font* може мењати: фонт коришћењем опције *Font*, стил фонта уз помоћ опције *Font style*, као и величина слова опцијом *Size*. У делу *Effects* су понуђене две опције. Уколико је штиклирана опција под називом *Strikeout*, текст ће бити прецртан, док одабрана опција *Underline* омогућава да текст буде подвучен. Одељак под називом *Color* нуди падајућу листу са бојама, која се отвара притиском стрелице која је усмерена надоле, чиме се може променити боја текста. Све измене везане за текст се могу видети у делу под називом *Sample*. Свака компонента унутар форме наслеђује фонт форме. Дакле, фонт који користе све компоненте се може истовремено мењати променом фонта форме. Свакој компоненти је могуће индивидуално променити особине фонта које су наведене у тексту изнад;

- *Cursor* – служи за промену изгледа курсора миша када се позиционира изнад површине форме. На слици 21. се може видети мени који се приказује када се десно од опције *Cursor*, после притиска левог тастера миша, притисне стрелица која је усмерена надоле.



Слика 21. Мени са различитим изгледима курсора миша

Форма може да реагује на више догађаја, а неки од тих догађаја су:

- *OnCreate* - активира се при покретању апликације;
- *OnClose* - активира се при затварању покренуте апликације;
- *OnClick* - активира се при притиску левог тастера миша када је курсор миша позициониран изнад површине форме;
- *OnDblClick* – активира се након двоструког притиска левог тастера миша када је курсор миша позициониран изнад површине форме.

Догађаји се креирају тако што се двоструким кликом, десно од имена догађаја унутар картице **Events** инспектора објеката, отвара едитор кода и костур процедуре за обраду догађаја, где се пишу одређене наредбе, које се извршавају у тренутку када се догађај деси и оне утичу на ток рада апликације.

Пример 1: Костур процедуре за обраду догађаја *OnCreate* чине следеће линије кода:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
{Unutar ključnih reči begin i end treba napisati naredbe koje
se izvršavaju kada se "pokretač" događaja desi. U slucaju
događaja OnClick "pokretac" je kreiranje forme, tj. pokretanje
aplikacije.}
end;
```

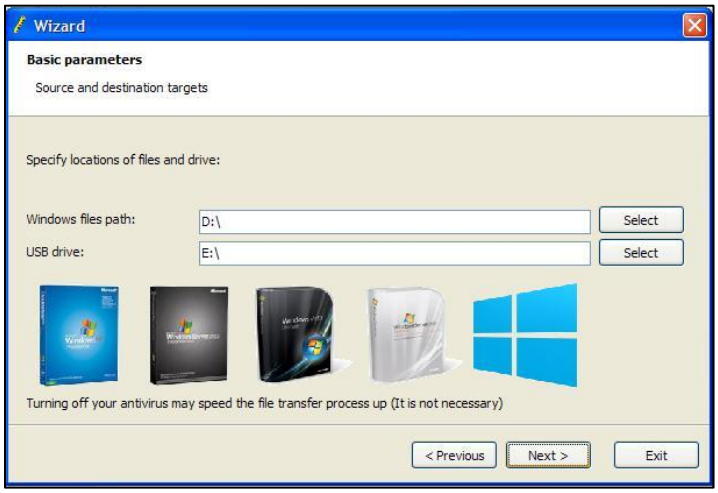
3. 3. 1. Додавање компоненти форми

Развојно окружење програмског језика Delphi нуди широк спектар компоненти различитих функционалности, којима могу бити подешене многе особине које утичу на њихов изглед.

Када је први пут наведен термин „компонента“ у лекцијама које претходе лекцији „Додавање компоненти форми“, укратко је наведено шта он подразумева, јер се лекција односила на развојно окружење, а не на компоненте. У овој лекцији је наведена дефиниција компоненте:

Компонента подразумева одвојену софтверску целину која извршава одређену унапред дефинисану функцију.

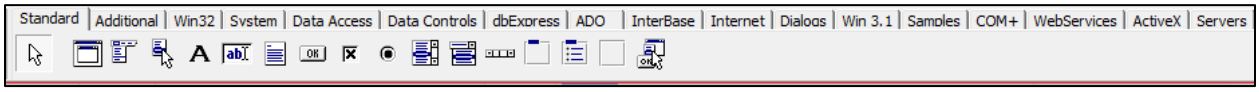
С обзиром да дефиниција не осликава у потпуности компоненту, сликом 22. је у лекцији представљен пример прозора који садржи неке компоненте (нпр. *Label*, *Edit* и *Button*).



Слика 22. Пример прозора који садржи компоненте Label, Edit и Button

Поступак додавања компоненте форми, у лекцији је објашњен на следећи начин:

1. Левим тастером миша врши се одабир одређене компоненте из палете компоненти (слика 23.);



Слика 23. Палета компоненти

2. Курсор миша треба да се позиционира унутар форме, на место где компонента треба да буде смештена;

3. Притиском левог тастера миша одабрана компонента биће постављена унутар форме.

На крају се налази задатак кроз који се обједињује претходни садржај из лекције. На слици 24. је приказан изглед задатка из лекције.

Задатак 1: Креирати апликацију као што је представљено на слици 5. Форма треба да има следеће карактеристике:
- **име**: PrvaForma;
- **натпис**: Prvi program;
- **боја**: плава;
- **удаљеност од горње ивице екрана**: 250;
- **удаљеност од леве ивице екрана**: 200;
- **ширина**: 350;
- **висина**: 450;
- **изглед курсора миша**: crHandPoint;
- додати догађај који се активира након притиска левог тастера миша, а који мења боју форме у зелену;
- додати догађај који се активира након двостуког притиска левог тастера миша на форми, а који мења боју форме у црвену;
- наћи компоненту **Button** (дугме) у палети компоненти и додати је унутар форме.

Слика 5. Изглед покренуте апликације

Слика 24. Задатак 1 из лекције „Додавање компоненти форми“

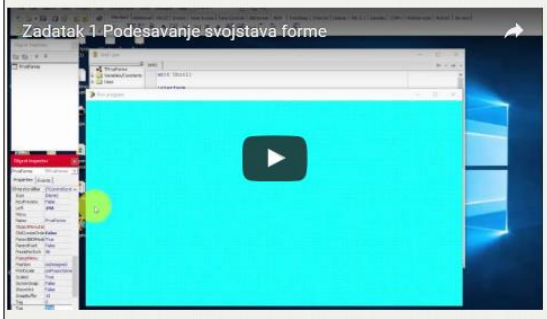
Захтев задатка јесте подешавање основних својстава форме, креирање догађаја и додавање компоненте форми. Решење је приказано у виду видео материјала, који може помоћи ученику да научи на који начин се претходни садржај лекције примењује на креирање апликације. Поред тога, део решења задатка је приказан у виду исечка програмског кода који приказује написане наредбе у оквиру процедура за обраду догађаја *OnClick* (догађај који настаје након једног клика мишем) и *OnDblClick* (догађај који настаје након двоструког клика мишем):

```
procedure TPrvaForma.FormClick(Sender: TObject);
begin
 PrvaForma.Color := clGreen; {Izvršavanjem ove naredbe se menja
boja forme u zelenu.}
end;

procedure TPrvaForma.FormDblClick(Sender: TObject);
begin
 PrvaForma.Color := clRed; {Izvršavanjem ove naredbe dolazi do
promene boje forme u crvenu.}
end;
```

Решење задатка се такође може преузети.

На слици 25. је приказан изглед решења задатка из лекције.

Решење: Решење задатка погледајте на следећем видеу или преузмите фајл са решењем кликом на сличицу сијалице испод.



Следеће линије представљају одсечак програмског кода који се односи на процедуре **OnClick** и **OnDblClick**:

```
procedure TPrvaForma.FormClick(Sender: TObject);
begin
PrvaForma.Color := clGreen; {Ova linija koda predstavlja naredbu. Naredbe ce biti objasnjene u petom poglavlju.}
end;

procedure TPrvaForma.FormDblClick(Sender: TObject);
begin
PrvaForma.Color := clRed; {Ovo je takodje naredba.}
end;
```

Кликните на сличицу сијалице како би преузели фајл са решењем задатка.

Слика 25. Приказ решења задатка 1 из лекције „Додавање компоненти форми“

3. 4. Компонента Label

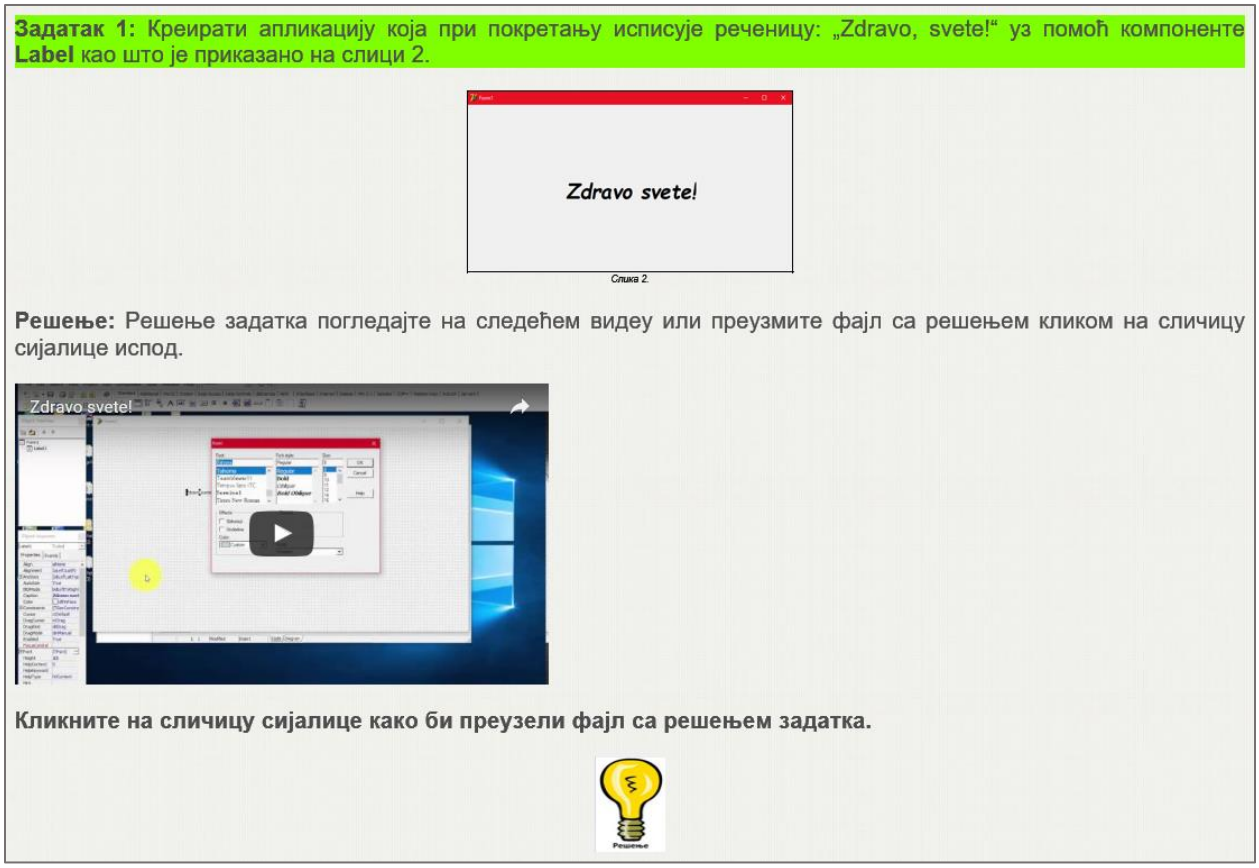
Компонента **Label** се користи за приказивање текста који се не може променити у току извршавања апликације коришћењем тастатуре. Најчешће се примењује за опис неке друге компоненте. Због тога се често назива компонента за натпис.

Компонента **Label** може да реагује на догађаје који настају коришћењем миша, док за догађаје произведеним тастатуром то није случај.

Дугме које одговара **Label** компоненти је приказано је на слици 26. Налази се на картици Standard из палете компоненти.

Слика 26. Изглед дугмета које одговара компоненти Label

У лекцији која се односи на компоненту **Label** је, поред дефинисања компоненте и навођења њене употребе, приказан задатак са решењем. Задатак се односи на устаљен програм за почетнике програмере, који исписује реченицу „Zdravo svete!“. На слици 27. је приказан изглед задатка и решења из лекције.

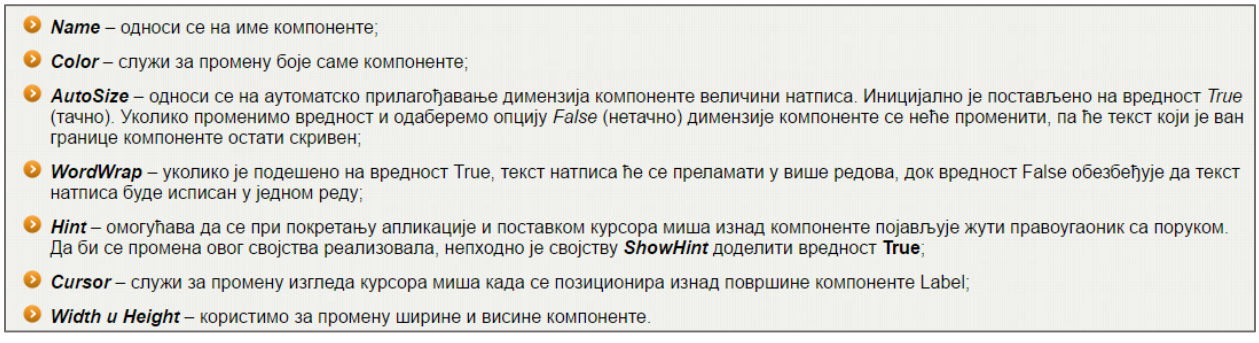


Слика 27. Приказ задатка 1 и решења из лекције „Компонента Label“

Решење задатка је приказано видео материјалом.

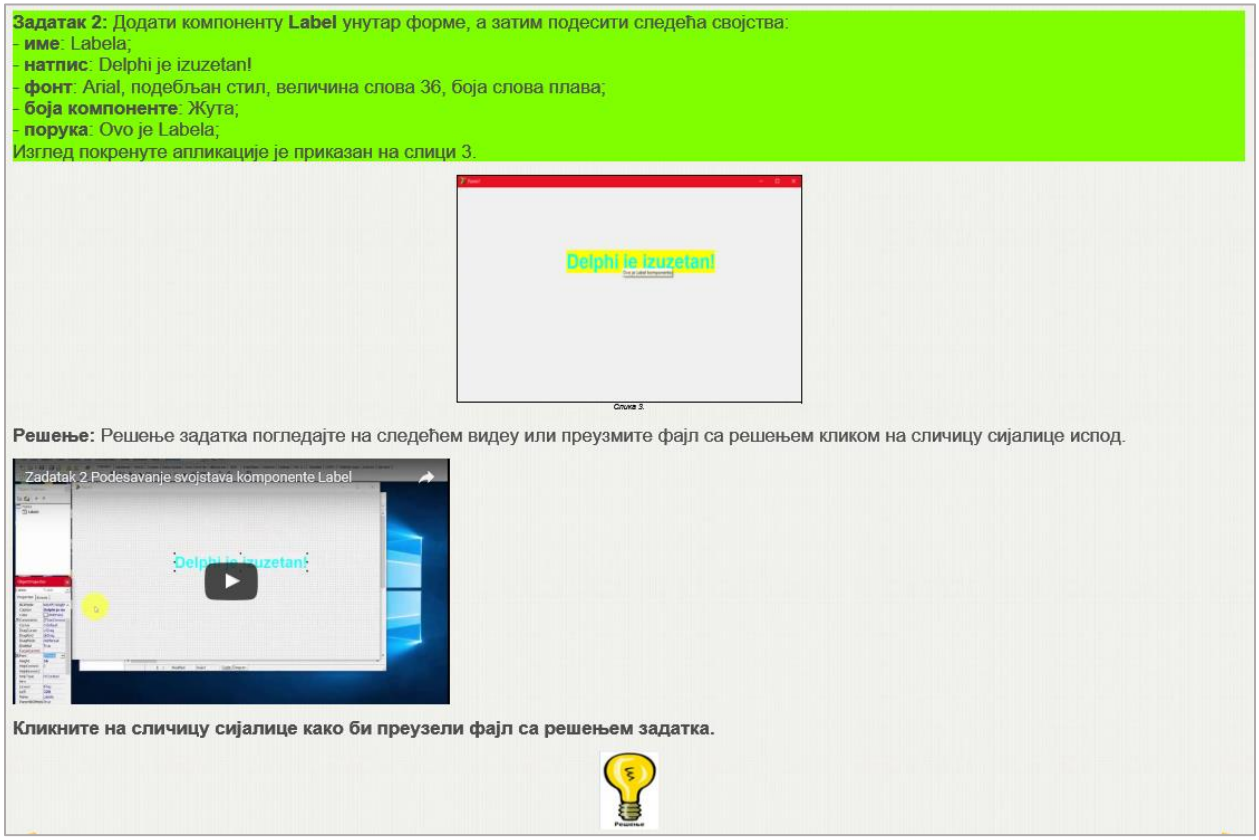
Циљ овог задатка је приказивање постављања компоненте **Label** унутар форме и подешавање њених најбитнијих својстава: *Caption* (натпис) и *Font* (фонт).

Поред наведених својстава, постоје и многа друга која се могу мењати унутар инспектора објеката. Неке од тих карактеристика су детаљније објашњена у лекцији (слика 28.).



Слика 28. Својства компоненте Label

Лекција садржи задатак који се односи на подешавања својстава компоненте **Label**, као што су: *Name* (име), *Caption* (натпис), *Font* (фонт), *Color* (боја) итд. На слици 29. је приказан изглед задатка и решења из лекције.



Слика 29. Задатак 2 из лекције компонента Label

На слици 29. се може видети да је решење представљено уз помоћ видеа и да се такође може преузети.

3. 5. Компонента Edit

Компонента **Edit** служи за унос или исписивање неког текстуалног податка. Често се назива оквир за текст. Текст се може унети само у једном реду и придружује се својству *Text*.

Компонента **Edit** се налази у оквиру картице Standard. Дугме-пречица која одговара овој компоненти је представљено на слици 30.

Слика 30. Изглед дугмета-пречице Edit компоненте

Својства **Edit** компонете која се најчешће мењају су: *Text* (текст), *Name* (име), *Color* (боја), *Font* (фонт), *Border Style* (стил оквира) итд. У лекцији су наведена својства детаљно објашњена, иако су многа од њих иста као за претходне компоненте.

Компонента **Edit** реагује на догађаје *OnClick* и *OnDblClick*, тј. на догађаје који се активирају једним или двоструким притиском левог тастера миша, али се они ретко користе код ове контроле.

За **Edit** компоненту су најчешће везани догађаји: *OnExit*, *OnKeyPress* и *OnChange*.

OnExit догађај врши проверу исправности података, када корисник промени фокус и пређе са ове на неку другу компоненту. Због тога се ова компонента често назива **Edit** контрола – контролише валидност података, јер они могу да буду у погрешном формату, немогући, недозвољени итд.

OnKeyPress догађај настаје када корисник притисне неки тастер на тастатури и често се користи када треба проверити да ли је одређени тастер у питању.

OnChange догађај настаје када корисник промени садржај у **Edit** компоненти.

3. 6. Компонента Button

Компонента **Button** (дугме) се врло често користи при креирању Windows апликација и представља вид окидача за неку одређену команду.

Компонента **Button** се налази на картици Standard из палете компоненти. На слици 31. је приказан изглед дугмета-пречице које се користи како би ова компонента била постављена унутар форме.

Слика 31. Изглед дугмета-пречице Edit компоненте

У лекцији се налази детаљано објашњење својстава компоненте **Button** (слика 32.).

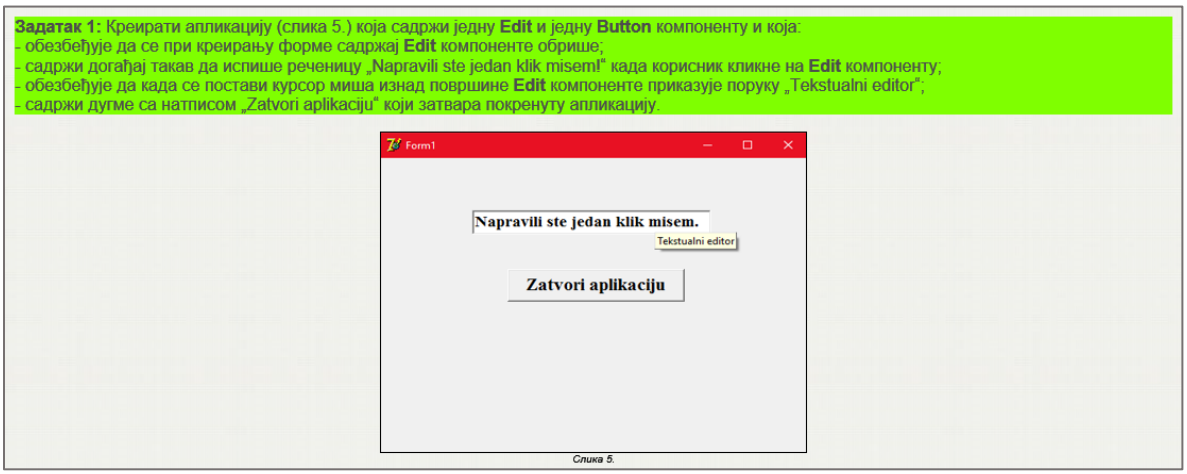
- ***Name*** – односи се на име компоненте;
- ***Caption*** – служи за промену натписа компоненте. Уколико корисник не изврши промену, на овој компоненти ће писати генерички натпис Button1;
- ***Cursor*** – служи за промену изгледа курсора миша када се позиционира изнад површине компоненте Button;
- ***Enabled*** - служи за подешавање доступности коришћења дугмета. Наиме, уколико ово својство има вредност *True* (тачно) омогућено је коришћење дугмета, а уколико има вредност *False* (нетачно) **није** омогућено његово коришћење;
- ***Font*** – служи за мењење имена фонта, стила, величине и боје слова. Такође је могуће подешавање да текст буде прецртан или подвучен;
- ***Width u Height*** – користимо за промену ширине и висине компоненте;
- ***Hint*** – омогућава да се при покретању апликације и поставком курсора миша изнад компоненте појављује жути правоугаоник са поруком. Да би се промена овог својства реализовала, непходно је својству ***ShowHint*** доделити вредност **True**;

Слика 32. Својства компоненте Button из лекције „Компонента Button“

Када се компонента **Button** постави унутар форме, она сама по себи не врши никакву функцију. На особи је да испрограмира шта жели да се деси када се активира дугме.

У ту сврху се најчешће користи догађај *OnClick*.

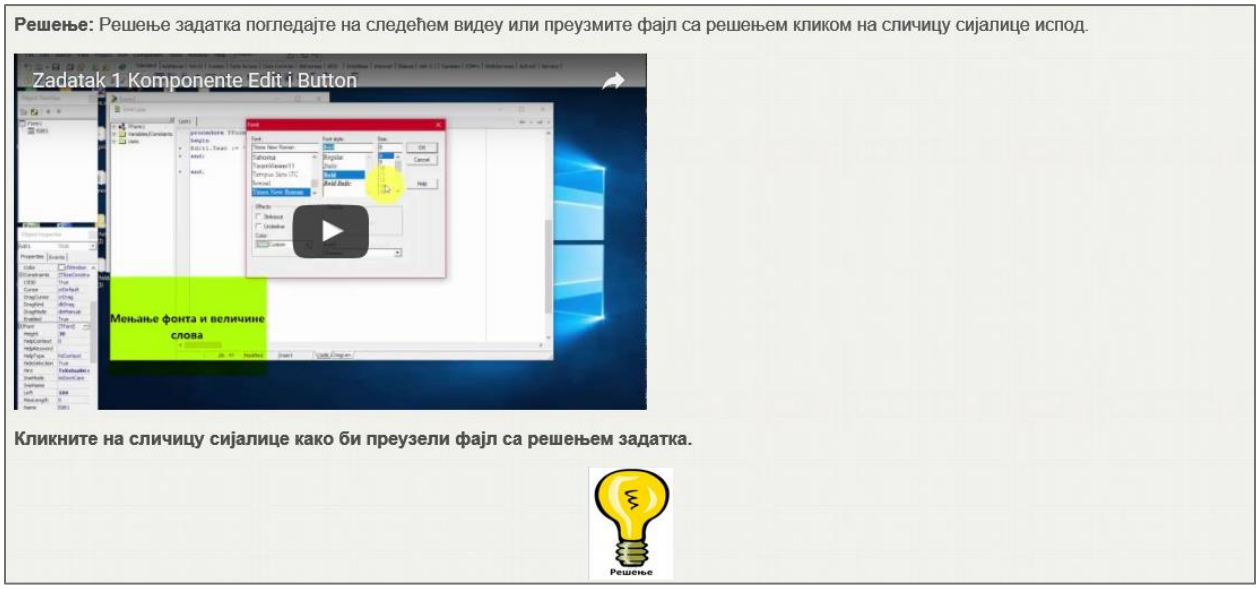
На слици 33. је приказан задатак, из лекције, који обједињује садржаје везане за компоненте **Edit** и **Button**.



Слика 33. Приказ задатка 1 из лекције под називом „Компонента Button"

Циљ задатка је да обједини претходне садржаје везане за компоненте **Edit** и **Button**.

Приказ решења задатка у лекцији се може видети на слици 34. Ученик може да погледа решење уз помоћ видео материјала или да преузме фајл са апликацијом.



Слика 34. Приказ решења задатка 1 из лекције

4. Типови података

Типови података дефинишу начин на који преводилац уписује податке у меморију. Подацима се назива све оно што може бити предмет обраде рачунаром или се може добити као резултат обраде. Подаци могу бити бројеви, слова, скупови и други сложени облици.

У неким програмским језицима постоји могућност додељивања било ког типа вредности променљивој. У програмском језику Delphi се мора декларисати тип променљиве пре него што та променљива користи. Ово омогућава преводиоцу да провери тип променљиве како би био сигуран да ће све бити у реду када програм буде био покренут. Неодговарајуће коришћење типова података резултује грешком преводиоца.

У оквиру дела под називом „Типови података“ кроз лекције су обрађени различити типови података који се користе при писању програма. Иако програмски језик Delphi омогућава коришћење великог броја типова, простих и сложених, лекције обрађују типове података који су предвиђени за ученике средњих школа. Како свака лекција о типовима података користи термин „променљива“, прва лекција обрађује променљиве и константе. У овој лекцији се такође налази и део о декларацији променљиве. Над различитим типовима података се могу користити само одређени оператори. За сваки тип података је индивидуално наведен скуп оператора који се могу користити у раду са њима. Због тога, друга лекција под називом „Оператори“ обрађује појам оператора пре навођења одређених типова података. Након тога следе лекције које обрађују: целобројни, реални, логички, знаковни и стринг тип. Кроз последњу лекцију из дела „Типови података“, ученик може да се упозна на који начин се врши конверзија типова података.

4. 1. Променљиве

Промељива представља вредност коју садржи локација у меморији рачунара. Означава вид структуре која има своје „место“ у меморији рачунара где складишти вредности. Може да чува улазне и излазне вредности, али и помоћне вредности које су некад неопходне за добијање резултата.

Сама реч „променљива“ говори да се вредност коју садржи мења, тј. није константна. Када променљива добије нову вредност, претходна се брише и стара вредност се не може више користити.

Свака променљива има своје име, вредност и тип.

Име (идентификатор) служи да на јединствен начин означи променљиву. Свака променљива мора имати своје име и две различите променљиве не смеју имати исто име.
За име променљиве дозвољени су следећи симболи:

- Сва слова (мала и велика) енглеског алфабета: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z;
- Арапске цифре: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9;
- Доња, тј. подвучена црта: _.

Име промењиве се креира комбиновањем слова и цифри. Први знак мора бити слово.
Није дозвољено одвајати делове имена знаком за празнину. Уколико је неопходно да се неки делови имена одвоје може се користити подвучена црта (_).
Такође није дозвољено да се за име промењиве користе резервисане речи, као што су: **and**, **array**, **begin**, **case**, **const**, **div**, **downtt**, **do**, **else**, **end**, **file**, **for**, **function**, **goto**, **if**, **in**, **label**, **mod**, **nil**, **not**, **of**, **or**, **procedure**, **program**, **record**, **repeat**, **set**, **then**, **to**, **type**, **until**, **var**, **while**, **with**.

Резервисане речи имају увек исто, унапред дефинисано значење и не могу се користити у друге сврхе. Резервисане речи означавају различите елементе језика.

Поред резервисаних речи, такође није дозвољено коришћење имена уграђених функција (sqr, abs, succ, pred, sin, cos, arctan, ln, exp, sqrt, frac) и аритметичких оператора (+, -, *, /).

Пример 2:

```
a3, br_1, Obim_kv {Ovo su primeri ispravno napisanih imena promenljivih.}
c 2, 45P, abs, i+1 {Ovo su primeri neispravno napisanih imena promenljivih.}
```

Програмски језик Delhi разликује велика и мала слова.

Препоручљиво је да име промењиве описује вредност коју она садржи. Није препоручљиво да име промењиве чине велики број карактера због прегледности кода програма и уштеде времена.

Тип промењиве представља тип податка коју она чува. Свака промењива мора имати свој тип података који одређује скуп вредности (домен) које промењива може садржати.
Промењива може бити бројевног, логичког и стринг типа. За сваки тип податка постоје одређене дозвољене вредности као и операције које се могу вршити над тим вредностима.

Вредност промењиве се односи на неку вредност (величину) коју она чува и та вредност се може мењати у току извршавања програма.
Уколико се промењива користи за неку обраду или израчунавање, она мора садржати вредност. У супротном, преводилац ће пријавити грешку.

4. 1. 1. Декларација променљивих

Све променљиве које се користе у току креирања програма морају се претходно декларисати.
Декларација променљиве је попут „најаве“ да је потребно обезбедити променљивој део меморије, како би се њена вредност сачувала за даље коришћење. Уколико се употреби променљива, а не изврши декларација пре тога, преводилац програма ће сигнализирати грешку.

Декларација променљиве се врши уз помоћ резервисане речи **var** иза које се наводи име променљиве и тип променљиве. Између имена и типа променљиве се пише знак за две тачке (:).
Пример 3: Декларација променљиве broj која је целобројног типа и променљиве s која је стринг типа се врши следећим линијама кода:

```
var
broj : integer; {Ovom naredbom se deklarise promenljiva cije je ime broj. Rezervisana rec integer se odnosi na celobrojni tip.}
s : string; {Na ovaj nacin se deklarise promenljiva cije je ime s. Rezervisana rec string se odnosi na string tip. }
```

Уколико постоје више променљивих истог типа, декларација може да се напише уз помоћ једне наредбе. Имена променљивих се у том случају раздвајају зарезом.

Пример 4: Декларација променљивих br1, br2 и br3 реалног типа се врши следећим линијама кода:

```
var
br1, br2, br3 : real; {Ovom naredbom su deklarisane tri promenljive realnog tipa. }
```

При писању програма, у зависности од задатка и проблема, не може се рећи да је број променљивих унапред одређен. Од особе која креира програм зависи колико ће променљивих бити укључено, јер се неки делови програма могу урадити и без њиховог коришћења. Променљиве не треба избегавати, јер без њиховог употребе, кôд програма би био преоптерећен, компликован и преобиман.

4. 2. Константе

Именована константа је идентификатор са додељеном вредношћу која се не мења.
Дефинише се коришћењем кључне речи **const**.
Пример 5: Дефиниција константи се реализује на следећи начин:

```
const
pi = 3.14; {Ovom naredbom je definisana konstanta pi i dodeljena
joj je vrednost 3.14.}
email = 'pedja@yahoo.com'; {Ovom naredbom je definisana konstanta
email i dodelje joj je konstantan string.}
```

Као и променљива, и константе имају своје име и вредност, док се тип података при дефиницији не наводи. Преводилац одређује тип података константе на основу вредности која јој је додељена. Додела вредности константи врши се уз помоћ оператора једнакости (=). Битно је напоменути да се оператори за доделу вредности променљивама и константама разликују.

Поред именованих константи, програмски језик Delphi омогућава коришћење и **неименованих константи**. Неименоване константе се користе приликом писања програмског кода навођењем њихових вредности.

4. 3. Оператори

Оператори дефинишу операције које је потребно извршити над одређеним вредностима да би се добио резултат. **Операција** пресликава коначан скуп података **(операнде)** у коначан скуп података **(резултате)**. Променљиве и константе имају улогу операнада.

У програмском језику Delphi постоји велики број оператора - **унарних** и **бинарних**.

Унарни оператори делују на један аргумент и то су: + (представља предзнак позитивног броја) и - (представља предзнак негативног броја).

Бинарни оператори делују на два (или више) аргумената. У бинарне операторе спадају **аритметички** и **релацијски** оператори. Аритметички оператори служе за дефинисање аритметичких операција (сабирање, одузимање, итд.) над бројевним вредностима, док релацијски оператори служе за поређење вредности.

Аритметички оператори су представљени у табели 1.

Табела 1. Аритметички оператори

Оператор	Операција
+	сабирање
-	одузимање

*	множење
/	дељење
div	целобројно дељење
mod	остатак при целобројном дељењу

Релацијски оператори су представљени у табели 2.

Табела 2. Релацијски оператори

Оператор	Релација
=	једнако
<>	различито
<	мање
>	веће
<=	мање или једнако
>=	веће или једнако

Релацијски оператори омогућавају поређење два операнда истог типа, а добијени резултат је логичког типа. Уколико је поређење тачно резултат ће имати вредност True, а уколико је нетачно имаће вредност False.

4. 3. 1. Приоритет оператора

Уколико се у изразу јаве више оператора, преводилац одређује хронолошки след извршавања операција поштујући скуп правила по којим оператори већег приоритета имају предност и извршавају се пре оператора нижег приоритета. Уколико оператори имају исти приоритет, извршавање операција се врши слева надесно.

Приоритет оператора је одређен следећим правилима:

1. Унарни оператори су већег приоритета од бинарних;
2. Оператори div, mod, * и / су истог приоритета, те се њихово извршавање врши слева надесно;
3. Оператори div, mod, * и / имају виши приоритет у односу на операторе + и -;
4. Аритметички оператори имају већи приоритет од релацијских оператора.

Пример 6: Овај пример представља примену правила приоритета оператора на израчунавање вредности израза.

```
 4 + 7 * 8 div 6 = 13 {Kako su operacije množenja i celobrojnog
deljenja većeg prioriteta u odnosu na sabiranje, prvo se
izvršava njihovo izračunavanje (sleva nadesno, jer su međusobno
istog prioriteta).
                  7 * 8 = 56, 56 div 6 = 9
Naposletku se vrši sabiranje brojeva 4 i 9, te je vrednost ovog
izraza 13.}
```

Уколико се јави потреба за утицањем на ток извршавања операција у изразу, операнди и оператори се могу сместити унутар заграда и тиме се обезбеђује највиши приоритет.

4. 3. 2. Оператор доделе

Оператор доделе служи за доделу вредности објектима који имају својство да попримају нове и мењају старе вредности. Означава се са „:=“.

Иако се лекција о наредби доделе налази у наредном делу под називом „Наредбе и изрази“, у овој лекцији је наведена њена синтакса, због навођења примера који садрже наредбу доделе.

Синтакса наредбе доделе у најосновнијем случају:

objekat_koji_prihvata_vrednost := vrednost;

Битно је напоменути да се вредност са десне стране увек додељује левој страни.

Пример 7: Наредба доделе се може написати у оквиру следећег исечка програмског кода:

```
var
a : integer; // Deklaracija celobrojne promenljive a
begin
 a := 150; {Nakon deklaracije promenljive a, može se izvršiti
naredba dodele. Ovom naredbom dodele je promenljivoj a
dodeljena vrednost broja 150.}
end;
```

4. 4. Целобројни тип

Целобројни тип података представља подскуп скупа целих бројева.

У табели 3. су представљени основни целобројни типови података у језику Delphi, меморијски простор који заузимају и опсег вредности у ком се може наћи сваки тип података.

Табела 3. Целобројни типови података, меморијски простор и опсег

Тип податка	Меморијски простор	Могући опсег вредности
Byte	1 бајт	0 до 255
Word	2 бајта	0 до 65 535
ShortInt	1 бајт	-128 до 127
Integer	4 бајта	-2 147 483 648 до 2 147 483 647
LongInt	4 бајта	-2 147 483 648 до 2 147 483 647
SmallInt	2 бајта	-32 768 до 32 767

При решавању задатака који су предвиђени за ученике средњих школа, најчешће се користи тип Integer, јер опсег вредности у већини случајева задовољава захтеву задатака.

Пример 8: Декларација променљиве mojBroj целобројног типа врши се на следећи начин:

```
var
mojBroj : integer;
```

Над операндима целобројног типа се користе следећи оператори:

- Унарни оператори: + и -;
- Аритметички оператори: +, -, *, /, div и mod;
- Релацијски оператори: =, <>, <, >, <=, >=.

Над целобројним типом могу да се примене и уграђене функције које су представљене у табели 4.

Табела 4. Функције које се могу користити над целобројним типом

Функција	Операција
sqr(x)	квадрат броја x
abs(x)	апсолутна вредност броја x

succ(x)	следбеник броја x
pred(x)	претходник броја x

4. 5. Реални тип

Реални тип података представља подскуп скупа реалних бројева.

Као што је сличај у математици, тако и у програмирању постоје величине које се не могу изразити целим бројевима (као на пример, обим и површина круга).

Бројеви који поред целобројног садрже и децимални део називају се реални бројеви. Приликом писања реалних бројева, целобројни и децимални део се раздвајају децималном тачком.

Пример 9: Примери реалних вредности могу бити:

```
8.1, -3.56, 0.097, -11.0 итд.
```

Поред представљеног начина писања реалних бројева (у примеру 9), програмски језик Delphi омогућава и писање реалних бројева са покретном тачком. Овакав начин писања је нарочито прикладан за врло велике и врло мале бројеве.

Пример 10: Примери неких записа реалних бројева са покретном тачком су:

```
53.4E+6
6.42E-4
```

Овакво писање реалних бројева се назива још и **експоненцијална нотација**. Експонент (број наведен иза слова „E“) означава за колико места треба померити децималну тачку. Позитиван експонент означава да децималну тачку треба померити удесно, док негативан експонент означава да децималну тачку треба померити улево. На празним местима која настају померањем децималне тачке се уписују нуле. Дакле, експонент представља степен броја 10 којим треба помножити наведени реалан број.

У табели 5. су представљени основни реални типови података, меморијски простор који заузимају и опсег вредности у ком се може наћи сваки тип података.

Табела 5. Реални типови података, меморијски простор и опсег

Тип података	Меморијски простор	Могући опсег вредности
Real	8 бајтова	5.0 E10-324 до 1.7 E 10308

Extended	10 бајтова	3.4 E 10-4932 до 1.1 E 104932
Single	4 бајта	1.5 E10-45 до 3.4 E 1038

При решавању задатака који су предвиђени за ученике средњих школа, најчешће се користи Real тип, јер опсег вредности у већини случајева задовољава захтеву задатака.

Пример 11: Декларација промењиве mojRealniBroj реалног типа врши се на следећи начин:

```
var
mojRealniBroj : real;
```

Над операндима реалног типа могу се применити следећи оператори:

- Унарни оператори: + и -;
- Аритметички оператори: +, -, *, /;
- Релацијски оператори: =, <>, <, >, <=, >=.

У табели 6. наведене су функције које се могу користити над реалним типом. Повратна вредност тих функција је реалног типа.

Табела 6. Функције које се могу користити над реалним типом

Функција	Операција
abs(x)	апсолутна вредност броја x
sqr(x)	квадрат броја x
sqrt(x)	квадратни корен броја x
sin(x)	синус броја x
cos(x)	косинус броја x
arctan(x)	аркус тангес броја x
ln(x)	логаритам броја x
exp(x)	степен x за основу e
frac(x)	децимални део реалног броја x

Поред функција чији је кодомен подскуп скупа реалних бројева, примењују се функције чије су повратне вредности целобројног типа:

– **Trunc(r)** – делује на реални аргумента **r** тако што одстрањује део иза децималне тачке (децимални део) и као повратну вредност има целобројни део аргумента;

Пример 12:

```
var
r : real;
i : integer;
begin
 r := 12.2;
 i:= trunc(r);  {Funkcija trunc deluje na argument r tako što odstranjuje deo iza decimalne tačke, a zatim novonastalu celobrojnu vrednost dodeljuje promenljivoj i. Na taj način je promenljivoj i dodeljena vrednost 12.}
end;
```

– **Round(r)** - заокружује реалан аргумента **r**. Заокруживање се врши на следећи начин: Уколико је прва децимала мања или једнака броју 5, децимални део се одстрањује, а део лево од децималне тачке остаје непромењен. Уколико је прва децимала једнака броју 5 а иза ње постоје још неке децимале или уколико је прва децимала већа од 5, децимални део се одтрањује, а целобројни део се увећава за 1.

Пример 13:

```
var
r1, r2 : real;
i1, i2 : integer;
begin
 r1 := 2.3;
 i1 := round(r1);  {S obzirom da je prva decimala argumenta r1 3, odnosno manja od 5, decimalni deo se odstranjuje, tako da se promenljivoj i1 dodeljuje vrednost 2.}
 r2 := 14.51;
 i2 := round(r2);  {Kako je prva decimala argumenta r2 5, a iza nje postoji druga decimala, decimalni deo se odstranjuje, a celobrojni deo se uvećava za 1. Promenljivoj i2 se ovom naredbom dodeljuje vrednost 15.}
end;
```

4. 6. Логички тип

Логички тип података се користи за означавање истинитости неког исказа и може имати две вредности: *True* (тачно) и *False* (нетачно).

У програмском језику Delphi, логички тип података се означава резервисаном речју **boolean**. Промењљиве логичког типа заузимају 1 бајт.

Пример 14: Декларација променљиве logPromenljiva логичког типа врши се на следећи начин:

```
var
logPromenljiva : boolean;
```

Над вредностима логичког типа података могу се примењивати операције чији је резултат такође вредност логичког типа.
Операције које се примењују над овим типом су представљене у табели 7.

Табела 7. Оператори који се могу користити над логичким типом

Оператор	Операција
not	негација
and	конјукција (логичко „и“)
or	дисјункција (логичко „или“)

Операција not је вишег приоритета у односу на операције and и or.

4. 7. Знаковни тип

Знаковни тип података је уређен скуп знакова који могу бити:

- Слова енглеског алфабета;
- Знакови арапских цифара од 0 до 9;
- Знакови интерпункције;
- Специјални знакови, као и знак за празнину.

Знаковни тип података се декларише коришћењем резервисане речи **char**. Промењљиве овог типа заузимају 1 бајт.

Пример 15: Декларација променљиве znak koja je знаковног типа врши се на следећи начин:

```
var
znak : char;
```

У програмском језику Delphi знаковне вредности се записују као знак између једноструких наводника.

Пример 16: Исправно написане знаковне вредности су:

'A', '8', '*', итд.

Коришћење једноструких наводника при означавању знаковног типа је веома важно.

Пример 17: Постоји разлика у следећим записима:

7 означава број.
'7' представља запис броја 7 и нема бројевну тежину.

Сваком знаку у скупу придружен је цео број који се назива **кôд**. Најчешће коришћена табела за кодирање карактера је **ASCII** (*American Standard Code for Information Interchange*).

Пример 18: По ASCII распореду важи:

Низ битова 00001100 је ознака за крај стране приликом штампања и редни број 12 представља његов кôд.
Низ битова 01000001 је ознака карактера A и редни број 65 представља његов кôд.

Поређење вредности знаковног типа уз помоћ релацијских оператора (<, <=, >, >=, =, <>) врши на основу поређења ASCII кодова.

Пример 19: Приликом поређења карактера A и B важи следећа неједнакост:

'A' < 'B' јер је број 65 који представља кôд карактера A мањи од броја који представља кôд карактера B, а то је 66.

Постоје две функције које знак повезују са редним бројем у уређеном скупу знакова:

- **Ord(x) –** као повратну вредност враћа редни број (кôд) прослеђеног карактера x;

Пример 20:

ord('A') = 65
ord('B') = 66

- **Chr(n)** – као повратну вредност враћа знак чији је редни број (код) прослеђен као аргумент функције n.

Пример 21:

chr(65) = 'A'
chr(66) = 'B'

4. 8. Стринг тип

Стринг представља секвенцу, тј. низ карактера (знакова) стандардног типа Char. Сваки стринг има своју дужину која заправо представља број карактера унутар стринга.

Низ знакова се у програмирању назива **нискама**.

Стрингови се веома често користе у програмирању. Програмски језик Delphi дефинише три различита типа: *long string* (дуги стринг), *short string* (кратки стринг) и *wide string* (широки стринг). Ово је наведено само ради информисаности ученика, јер се најчешће користи дуги тип стрингова, због динамичког резервисања меморије које подразумева да особа која креира програм не мора да води рачуна о меморијском простору који се додељује објекту овог типа.

За декларисање променљиве стринг типа користи се резервисана реч **`string`**.

Пример 22: Декларација променљиве s1 типа стринг реализује се на следећи начин:

```
var
s1 : string;
```

Постоји још један начин за декларисање променљиве стринг типа. Уколико је унапред позната дужина променљиве овог типа, тај број се уписује унутар угластих заграда ([и]).

Пример 23: Декларација променљиве s2 стринг типа дужине 30 врши се на следећи начин:

```
var
s2 : string[30];
```

У току рада, често се користе стринг константе.

Стринг константе представљају непроменљиве вредности стринг типа и могу имати произвољан број карактера, укључујући и стринг без карактера (празан стринг). Стринг константе се наводе између полунаводника (‘ и ‘). Уколико је празан стринг у питању између једноструких наводника се ништа не уписује. Празан стринг има дужину једнаку броју 0.

4. 8. 1. Основне функције и процедуре за рад са стринговима

Програмски језик Delphi садржи велики број функција и процедура за рад са стринговима. Функције и процедуре које се најчешће користе су:

1. Функција **Length**

Функција **Length** даје целобројну дужину текуће вредности промењиве или константе типа стринг;
Позив функције се врши на следећи начин:

length(ime_promenljive) или **length('Neki string')**

Пример 24:

```
var
d1, d2 : integer;
s : string;
begin
 d1 := length('Programiranje'); {Promenljivoj d1 je ovom naredbom dodeljena dužina stringa 'Programiranje'. Kako je dužina ovog stringa 13, d1 je dodeljena vrednost 13.}
 s := 'Delphi je zabavan';
 d2 := length(s); {Promenljivoj d2 je dodeljena vrednost 17 (15 slova i 2 razmaka - ukupno 17 karaktera).}
end;
```

2. Функција **Copy**

Функција **Copy** даје нови стринг, који заправо представља део прослеђеног стринга.
Позив функције се врши на следећи начин:

copy(ime_promenljive, p, d) или **copy('Neki string', p, d)**

при чему су вредности **p** и **d** целобројне.

Позивањем ове функције настаје нови стринг који је део прослеђеног (као први аргумент) формиран копирањем **d** знакова, почев од позиције **p**.

Пример 25:

```
var
s : string;
begin
 s := copy('Programiranje', 1, 7); {Promenljivoj s se ovom naredbom dodeljuje string 'Program' koji je nastao kopiranjem prvih 7 karaktera pocev od prve pozicije.}
end;
```

3. Функција **Concat**

Функција **Concat** формира нови стринг надовезивањем више стрингова који се прослеђују као аргументи. Устаљен израз у програмирању за надовезивање стрингова је **конкатенација**.
Позив функције се врши на следећи начин:

concat(ime_string_promenljive1, ime_string_promenljive2,…)

Као аргумент може бити прослеђен и стринг константа, као и комбинација променљивих и константи.

Пример 26:

```
var
s, sn : string;
begin
 s := 'Petar';
 sn := concat(s,' ','Petrovic'); {Promenljivoj sn je dodeljen string 'Petar Petrovic' koji je nastao nadovezivanjem 3 stringa: 'Petar', prazan string i 'Petrovic'.}
end;
```

4. Функција **Pos**

Позив функције **Pos** врши се на следећи начин:

pos(s1, s2)

где се **s1** и **s2** замењују конкретним именима променљивих типа стринг (или константама типа стринг).

Функција **Pos** испитује да ли се **s1** садржи у стрингу **s2** и ако се садржи, ова функција враћа број који означава позицију првог појављивања стринга **s1** идући слева надесно. Уколико се **s1** не садржи у стрингу **s2** функција ***Pos*** враћа 0.

Пример 27:

```
var
p1, p2 : integer;
begin
 p1   :=   pos('gram','programiranje');   {Promenljivoj   p1   je dodeljena vrednost 4, jer se od cetvrte pozicije string'gram' nalazi u stringu 'programiranje'.}
```

```
 p2 := pos('lopta','logaritam'); {Promenljivoj p2 je dodeljena vrednost 0, jer se string 'lopta' ne sadrzi u stringu 'logaritam'.}
end;
```

5. Процедура **Delete**

Процедура **Delete** врши брисање дела стринга.
Позив процедуре:

delete(ime_promenljive_tipa_string, p, d)

где су вредности **p** и **d** целобројне.

Процедура **Delete** врши брисање стринга, почев од позиције **p** за **d** карактера.

Пример 28:

```
var
s : string;
begin
 s := 'Moreplovac';
 delete(s, 5, 6);  { String promenljiva s dobija novu vrednost 'More' koja je nastala brisanjem 6 karaktera 'plovac' pocev od pete pozicije stringa 'Moreplovac'.}
end;
```

У наведеним примерима резултат рада функција је додељена некој променљивој док резултат рада процедуре није. У томе лежи суштинска разлика функција и процедура. Функције имају повратне вредности, док процедуре не. Повратне вредности функција се морају доделити неком објекту, док се процедура самостално позива.

6. Процедура **Insert**

Процедура **Insert** врши модификацију једног стринга тако што унутар њега умеће неки други стринг.
Позив процедуре:

insert(string1, string2, p)

Ова процедура умеће стринг **string1** у стринг **string2**, почев од позиције **p** (**p** је целобројна вредност).

Пример 29:

```
var
s1, s2 : string;
begin
 s1 := 'ana';
 s2 := 'ans';
insert(s1, s2, 3); {String promenljivoj s2 se dodeljuje
vrednost 'ananas' koji je nastao umetanjem stringa 'ana' u
string 'ans' počev od pozicije 3, odnosno iza karaktera 'n'.}
end;
```

7. Процедура **Str**

Позив процедуре:

str(broj, s)

У конкретном примеру **broj** се мења бројем или променљивом целобројног или реалног типа, а **s** именом променљиве типа стринг.

Процедура **Str** обезбеђује да се нумеричка вредност првог аргумента промени у стринг тип и да се додели променљивој чије се име прослеђује као други аргумент. Промена типова података се још назива конверзијом типова. Детаљније објашњење се налази у лекцији „Конверзија типова података“.

Пример 30:

```
var
s : string;
begin
 str(125, s); {String promenljivoj s se ovom naredbom dodeljuje
vrednost '125'.}
end;
```

8. Процедура **Val**

Процедура **Val** врши проверу да ли прослеђени стринг садржи коректан бројни запис.

Позив процедуре се реализује на следећи начин:

val(s, broj, kod_greske)

при чему се у конкретном примеру **s** мења константом или променљивом типа стринг, а **broj** променљивом целобројног или реалног типа. Трећи аргумент представља променљиву целобројног типа.

Ако стринг садржи коректан бројни запис (целобројни, реални у фиксном или покретном зарезу) конверзија је успешна и **kod_greske** се поставља на 0. Ако стринг садржи некоректан бројни запис, **kod_greske** добија вредност различиту од 0 (позицију на којој се зауставила процедура превођења).

Пример 31:

```
var
s1, s2: string;
kod1, kod2 : integer;
br1, br2 : real;
begin
 s1 := '123';
 s2 := '1a23';
 val(s1, br1, kod1); {Promenljiva br1 dobija vrednost 123, a kod1 vrednost 0, jer se u stringu s1 nalazi korektan brojni zapis.}
 val(s2, br2, kod2); {Ovom naredbom br2 ne dobija vrednost, a kod2 dobija vrednost 2, jer se na drugoj poziciji nalazi karakter 'a'.}
end;
```

4. 8. 2. Операције над стринговима

Поред функција и процедура, над стринговима могу да се извршавају операције посредством следећих оператора:

1. Оператор **+**

   Оператор **+** служи за конкатенацију (надовезивање) стрингова. Дакле, има исти резултат као функција **concat**.

Пример 32:

```
var
s1, s2: string;
begin
 s1 := 'Danas' + 'je' + 'lep' + 'dan'; {Promenljiva s1 dobija vrednost 'Danasjelepdan'. Dakle, fale razmaci.}
 s2 := 'Danas' + ' ' + 'je' + ' ' + 'lep' + ' ' + 'dan';
{Promenljiva s2 dobija vrednost 'Danas je lep dan', jer je su nadovezani stringovi koji sadrze razmak.}
```

```
end;
```

2. Релациони оператори

Стрингови се могу поредити коришћењем релационих оператора = , <>, <, >, <=, >=. Стрингови се пореде знак по знак, идућу слева надесно, тако што се упоређују вредности нумеричких кодова знакова (ASCII кодирање).

Пример 33: При поређењу стрингова важи:

```
'ABCD' < 'ABCE' {Ovaj izraz ima vrednost True jer je 'D' < 'E'.
Kôd karaktera 'D' (68) ima manju vrednost od koda karaktera
'E'.}
'1111' < '99' {Ovaj izraz ima vrednost False. Kôd karaktera '1'
ima manju vrednost od koda karaktera '9'.}
```

Од релационих оператора, најчешће се користе оператори = и <> који утврђују једнакост, односно неједнакост стрингова.

3. Оператор индекса

У тексту је раније наведено да је стринг низ карактера. Сваки члан тог низа има свој јединствени индекс. Индекс припада скупу природних бројева и додељује се на следећи начин: први карактер стринга има индекс 1, други карактер стринга има индекс 2, итд. Нулти елемент промењиве типа стринг садржи дужину стринга, тј. ASCII вредност која је једнака дужини стринга.
Оператор индекса служи за директан приступ карактеру стринга који се налази на одређеној позицији. Често се користи за претраживање стринга карактер по карактер. Означава се угластим заградама ([и]) између којих се уписује индекс одређеног члана низа.
При писању наредби користи се на следећи начин:

ime_string_promenljive[i], i – индекс

Уколико се оваква вредност додељује некој промењивој, она мора бити знаковног или стринг типа.

Пример 34:

```
var
s : string;
```

```
c : char; {Promenljiva c se može deklarisati i kao string promenljiva.}
begin
s := 'Hello!';
c := s[1]; {Znakovnoj promenljivoj c je dodeljen karakter 'H'.}
end;
```

4. 9. Конверзија типова података

У комуникацији апликације са корисником, корисник најчешће уноси улазне податке користећи тастатуру. Како су сви тастери на тастатури представљени знацима, неопходно је податке који не представљају текст (као што су нпр. бројеви) превести у одговарајући тип података како би се искористила сва функционалност тог типа. Такође, врло често се дешава ситуација да се аритметичке операције изводе над подацима различитог типа (целобројног и реалног) па је неопходно податке превести у одговарајући облик.

Процес превођења података из једног у други тип се назива конверзија података.

Као што је наведено у лекцији под називом „Промењиве“, додела вредности промењивој може се извршити само ако је вредност истог типа као промењива. У противном, мора се превести у одговарајући тип.

Конверзија података се може реализовати на два начина: аутоматски (имплицитном конверзијом) или под контролом програмера (експлицитном конверзијом).

4. 9. 1. Имплицитна конверзија

Имплицитна конверзија подразумева да преводилац аутоматски усклађује типове података уколико је то могуће. Усклађивање се врши тако што се „ужи“ тип конвертује у „шири“ тип.

На пример, целобројни тип података (Integer) може да се конвертује у реалан (Real) тип, што је и логично с обзиром да је скуп целих бројева подскуп скупа реалних бројева, па је самим тим „ужи“.

Пример 35: Имплицитна конверзија може да се изврши на следећи начин:

```
var
i : integer;
r, k : real;
begin
 i := 5;
```

```
 r := 14.8;
 k := i + r; {Kako bi operacija sabiranja bila izvršena, vrši se
implicitna konverzija celobrojne vrednosti promenljive i u realnu
vrednost. Promenljiva k mora biti deklarisana kao realna
promenljiva, jer joj se ovom naredbom dodeljuje zbir dva realna
broja.}
end;
```

4. 9. 2. Експлицитна конверзија

Експлицитном конверзијом програмер, додатним кодом, од преводиоца захтева тражену конверзију. Експлицитна конверзија се остварује коришћењем следећих уграђених функција:

- **IntToStr(x)** – конвертује целобројну вредност прослеђеног аргумента x у стринг тип;
- **StrToInt(x)** – конвертује вредност стринг типа прослеђеног аргумента x у целобројну вредност;
- **FloatToStr(x)** - конвертује реалну вредност прослеђеног аргумента x у стринг тип;
- **StrToFloat(x)** - конвертује вредност стринг типа прослеђеног аргумента x у реалну вредност.

Наведене функције су врло важне јер су улазне вредности апликација у већини случајева стринг типа. Поред тога, многе апликације захтевају манипулисање бројевима те је неопходно конвертовати улазне вредности у бројевне вредности уз помоћ функција **StrToInt** или **StrToFloat**. Са друге стране, резултати (који имају бројевну вредност) представљају излазне вредности, које је неопходно претходно конвертовати у стринг тип уз помоћ функција **IntToStr** и **FloatToStr**.

Поред наведених функција, функције које врше конверзију реалног у целобројни тип су: **Trunc** и **Round**.

Процедура **Str** се такође користи за конверзију целобројног у стринг тип.

5. Наредбе и изрази

У делу „Наредбе и изрази“ кроз лекције су прво обрађене наредбе и изрази иако је ученик имао прилике да се са њима сусретне у лекцијама у претходном поглављу. Наредбе и изрази су се често наводили кроз примере, јер су ове конструкције незаобилазне при писању кода програма. Након лекције о наредбама и изразима, следи лекција која обрађује поступак уношења и приказивања података. Последња лекција из овог поглавља „Програмирање израчунавања по једноставним математичким формулама“ обједињује садржаје из

претходних лекција и кроз задатке омогућава ученику да научи да креира апликације, попут калкулатора итд.

5. 1. Наредбе и изрази – основни појмови

Наредбе представљају основне конструкције програмског језика и може се тумачити као инструкција која својим извршавањем диктира рачунару да „уради“ одређену акцију.

При писању наредби, мора се водити рачуна о синтакси. Свака наредба се завршава тачка-зарезом (;).

Наредбе се могу поделити на неизвршне и извршне.

У **неизвршне наредбе** спадају наредбе за дефинисање и декларисање објеката који се користе у програму.

Извршне наредбе чине „радни“ део програма и у њих спадају: аритметичко-логичке наредбе, управљачке, улазне, излазне наредбе итд.

У наредбе спада и структура која омогућава доделу вредности одређеном објекту – наредба доделе.

Синтакса наредбе доделе у најосновнијем случају:

Ime_promenljive := vrednost;

Пример 36:

```
ime := 'Marija'; {Primer naredbe dodele kojom se promenljivoj ime dodeljuje string 'Marija'.}
broj := 13.76; {Primer naredbe dodele kojom se promenljivoj broj realna vrednost 13.76.}
```

Вредност са десне стране оператора доделе се увек додељује објекту који се налази на левој стани.

Поред наведеног најосновнијег примера наредбе доделе, променљивој се може доделити и вредност израза, под условом да су типови променљиве и вредности израза усклађени.

Синтакса наредбе доделе када се променљивој додељује вредност израза:

Ime_promenljive := izraz;

Овакав тип наредбе доделе се извршава тако што се прво израчуна вредност израза са десне стране оператора доделе, а потом се та вредност додељује променљивој.

Пример 37:

```
a := 17 + 8; {Nakon izračunavanja zbira brojeva 17 i 8, vrednost 25 se dodeljuje promenljivoj a.}
```

```
zbir := zbir + broj; {U ovoj naredbi se i na levoj i na desnoj strani nalazi promenljiva zbir. Ova naredba se izvršava tako što se vrednosti koju čuvaju promenljive zbir i broj sabiraju i taj zbir se dodeljuje promenljivoj zbir čime se njena stara vrednost zamenjuje novom vrednošću.}
osoba := 'Vuk' + ' ' + 'Miletic';
radijani := stepeni*PI/180.0;
```

Израз је конструкција програмског језика којом је представљена нека вредност, односно начин њеног израчунавања.

Изрази се формирају комбиновањем операнада, оператора и облих заграда. Оператори дефинишу које је операције потребно извршити над операндима, а заграде дефинишу редослед вршења тих операција. У случају када редослед вршења операција није одређен заградама, примењују се правила о приоритету оператора. Операнди могу бити константе и променљиве.

У истом изразу се не смеју мешати вредности различитог типа података. Вредности различитих типова могу да се укључе у исти израз једино уколико се искористе фунције за трансформацију или конверзију типова и тиме се различити типови ускладе. Такође треба водити рачуна о томе да променљива која се јавља у изразу мора имати вредност, иначе ће вредност читавог израза бити недефинисана. Треба напоменути и да два оператора у изразу не могу бити један поред другог, нити се израз може завршити оператором. Свака два операнда морају бити раздвојени оператором.

Пример 38:

```
P := aa; {Pogresno napisan izraz, jer se između imena promenljivih mora napisati operator.}
P := a * a; // Ispravno napisan izraz
K := 7 * -3; {Pogresno napisan izraz, jer dva operatora ne smeju biti jedan pored drugog.}
K := 7 * (-3); // Ispravno napisan izraz
```

Изрази могу бити:

- Аритметички;
- Логички.

Аритметички изрази настају комбиновањем аритметичких оператора, уграђених функција које делују на бројевне вредности и операнада који имају бројевну вредност. Вредност аритметичких израза је такође бројевна.

Логички изрази представљају комбинацију логичких оператора и операнада (било ког типа) и вредност таквих израза може бити једино True или False.

Пример 39:

```
O := 2 * r * PI; {Na desnoj strani naredbe dodele se nalazi primer aritmetičkog izraza.}
b := (a < 0) and (b < 0); {Primer logičkog izraza se nalazi na desnoj strani naredbe dodele.}
```

5. 2. Уношење и приказивање података

Уношење и приказивање података се може извршити на више начина, у зависности од њиховог типа и функције саме апликације. У свим примерима и задацима који су наведени у електронским лекцијама је наглашено које компоненте треба да се поставе за унос и приказивање података, тако да ученик кроз ову лекцију може да научи које наредбе то омогућавају.

За унос података се најчешће користи компонента **Edit**. Улазни подаци се обрађују посредством наредби програмског кода и исписују се у већини случајева унутар друге **Edit** компоненте. **Edit** компонента је погодна уколико су вредности на пример бројевне. Уколико је потребно исписати одговор (који је резултат рада програма) у виду реченице, погодна компонента је **Label**.

Треба напоменути да ово није формално правило и уколико се искористе неке друге компоненте за унос или приказивање података, преводилац неће пријавити грешку, мада ће у том случају апликација понекад радити на неочекивани начин. На пример, неочекивано би било да апликација исписује неки резултат као натпис компоненте **Button**.

Унос и приказивање података уз помоћ **Edit** и **Label** компоненте је у лекцији објашњено на примерима. Примери садрже коментаре који додатно објашњавају наредбе.

Пример 40: Наредба која променљивој ceo_br целобројног типа додељује број који је уписан у **Edit** компоненту (чије је име нпр. *Edit1*) је облика:

```
ceo_br := StrToInt(Edit1.Text); {Edit1.Text sadrži zapis broja - tekst tipa string, te je neophodno konvertovati string tip u celobrojni (integer) tip korišćenjem funkcije StrToInt. Promenljiva ceo_br se pri pisanju koda programa mora prethodno deklarisati.}
```

Најбитније својство **Edit** компоненте је својство *Text* које је типа String. Уколико је име **Edit** компоненте, на пример *Edit1*, својству *Text* се приступа са *Edit1.Text*.

Пример 41: Наредба која променљивој **realan_br** реалног типа додељује број који је уписан у **Edit** компоненту (чије је име нпр. *Edit1*) је облика:

```
realan_br := StrToFloat(Edit1.Text); {Edit1.Text sadrži zapis broja - tekst tipa string, te je neophodno konvertovati string tip u realan (real) tip korišćenjem funkcije StrToFloat. Promenljiva realan_br se pri pisanju koda programa mora prethodno deklarisati.}
```

Пример 42: Наредба која променљивој s стринг типа додељује текст из **Edit** компоненте (чије је име нпр. *Edit1*) је облика:

```
s := Edit1.Text; {U ovoj naredbi konverzija tipova nije potrebna, ukoliko se s deklariše kao string promenljiva.}
```

Пример 43: Наредба која омогућава да се изврши сабирање вредности из две **Edit** контроле (*Edit1* и *Edit2*) и да се та вредност додели променљивој zbir има следећи облика:

1. начин: Писање наредбе без увођења помоћних променљивих:

```
zbir := StrToFloat(Edit1.Text) + StrToFloat(Edit2.Text); {Ovo je kraći način jer se ne uvode promenljive koje će predstavljati sabirke. U ovoj naredbi sabirci su konvertovani tekstovi Edit komponenata Edit1 i Edit2.}
```

2. начин: Писање наредбе коришћењем помоћних променљивих:

```
sabirak1 := StrToFloat(Edit1.Text); {Tekst iz prve Edit komponente se konvertuje iz string tipa u realan tip, a zatim dodeljuje promenljivoj sabirak1.}

sabirak2 := StrToFloat(Edit2.Text); {Tekst iz druge Edit komponente se konvertuje iz string tipa u realan tip, a zatim dodeljuje promenljivoj sabirak2.}

zbir := sabirak1 + sabirak2; {Vrednosti promenljivih sabirak1 i sabirak2 se sabiraju i taj rezultat se dodeljuje promenljivoj zbir.}
```

Иако је 1. начин краћи, 2. начин је ефикаснији, јер штеди време уколико, на пример, више пута треба користити вредности из **Edit** контрола. Такође, коришћењем помоћних променљивих кôд програма биће читљивији, прегледнији и мањег обима.

Претходни примери се односе на унос података коришћењем **Edit** компоненте. Наредни примери се односе на приказивање података.

Пример 44: Наредбе које служе за исписивање вредности промењљивих унутар поља **Edit** компоненте су облика:

```
Edit1.Text := IntToStr(prvi_br); {Ova naredba služi za ispisivanje vrednosti promenljive prvi_br koja je celobrojnog tipa unutar polja Edit1 kontrole.}
Edit2.Text := FloatToStr(drugi_br); {Ova naredba služi za ispisivanje vrednosti promenljive drugi_br koja je realnog tipa unutar polja Edit2 kontrole.}
Edit3.Text := naziv; {Ova naredba služi za ispisivanje vrednosti promenljive naziv koja je string tipa unutar polja Edit3 kontrole.}
```

Пример 45: Наредба која исписује у текстуалном пољу **Edit** компоненте вредност обима једнакостарничног троугла (уколико је са a означено име промењљиве која садржи вредност реалног типа и која одговара дужини странице троугла) је облика:

1. начин: Писање наредбе без коришћења промењљиве која садржи вредност обима троугла:

```
Edit1.Text := FloatToStr(3 * a);  {Prvo se vrši izračunavanje vrednosti izraza 3 * a. Zatim se ta vrednost konvertuje u string tip i dodeljuje svojstvu Text Edit komponente.}
```

2. начин: Писање наредбе са коришћењем промењљиве O која садржи вредност обима троугла:

```
O := 3 * a; {Ovom naredbom se vrši izračunavanje obima, a potom se ta vrednost dodeljuje promenljivoj O.}
Edit1.Text := FloatToStr(O); {Naredba za ispisivanje vrednosti promenljive O unutar polja Edit komponente.}
```

Примери за исписивање података уз помоћ **Label** компоненте нису наведени у лекцији, јер су врло слични примерима за исписивање података коришћењем **Edit** контроле.
Једина разлика је у томе што је за компоненту **Label** везано својство *Caption*. Уколико је име компоненте, на пример, *Label1*, својству за натпис се приступа са *Label1.Caption*.

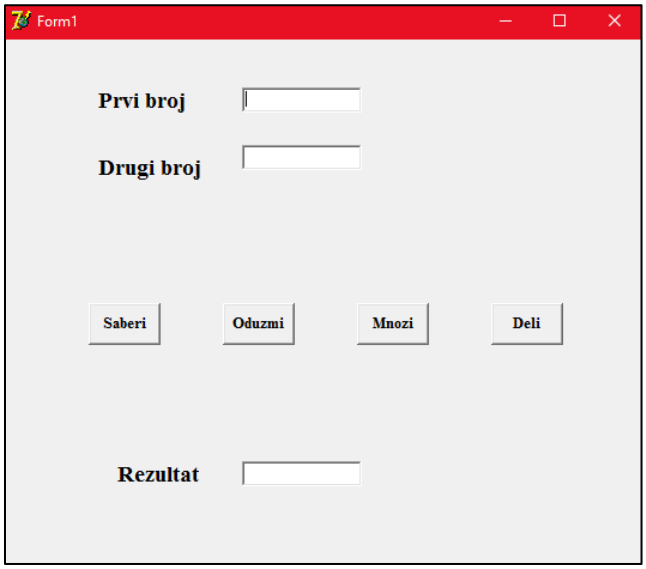
5. 3. Програмирање израчунавања по једноставним математичким формулама

Лекција „Програмирање израчунавања по једноставним математичким формулама“ садржи два задатка. Уз помоћ решења ученик може да сазна како се креира апликација која

обезбеђује основне функције калкулатора и апликација која рачуна и исписује обим троугла на основу унетих координата темена.

Задаци осликавају градиво из претходних лекција у чему се огледа њихов значај. Уз добро разумевање решења задатака ученик може да примени знање и самостално креира апликације које се односе на израчунавање неких других величина из области математике.

На слици 35. је приказан изглед апликације из првог задатка:



Слика 35. Пример изгледа апликације

Апликација треба да садржи: две **Edit** компоненте за унос реалних бројева и једну за приказ резултата, **Label** компоненте које их описују и четири дугмета за операције сабирања, одузимања, множења и дељења. Апликација треба да садржи четири **Button** компоненте, које ће омогућити, након клика, исписивање резултата: сабирања, одузимања, множења или дељења унетих бројева.

У решењу нису наведене карактеристике компоненти. Ученик може самостално да их подеси и тиме утиче на изглед апликације или да погледа видео или преузме фајл са решењем и на тај начин сазна конкретне вредности карактеристика.

Након постављања визуелних компоненти следи креирање догађаја за сва четири дугмета. С обзиром да се испис врши након клика на одређену **Button** компоненту, погодан догађај је *OnClick*. У решењу су променљиве br1, br2 и rezultat (које се односе на први и други операнд и резултат) декларисане као глобалне, како би биле видљиве у свим процедурама за обраду догађаја *OnClick***.**

Исечак програмског кода који омогућава, између осталог, да дугмад функционишу је следећег облика:

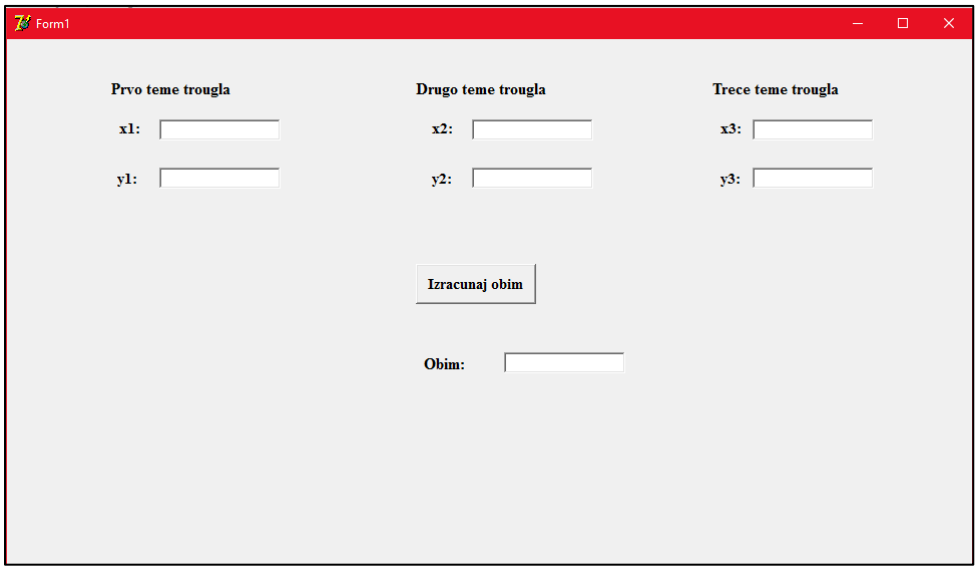
```
var
  Form1: TForm1;
```

```
  var br1, br2, rezultat : real; {Naredba za deklaraciju
globalnih promenljivih.}


implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);{Procedura
OnClick dugmeta za sabiranje.}
begin
 br1 := StrToFloat(Edit1.Text); {Tekst iz prve Edit kontrole se
preuzima, konvertuje u realan tip a zatim dodeljuje promenljivoj
br1.}
 br2 := StrToFloat(Edit2.Text); {Tekst iz druge Edit kontrole se
preuzima, konvertuje u realan tip a zatim dodeljuje promenljivoj
br2.}

 rezultat := br1 + br2; {Ovom naredbom se sabiraju vrednosti
promenljivih br1 i br2, a zatim se zbir dodeljuje promenljivoj
rezultat.}

 Edit3.Text := FloatToStr(rezultat); {Ova naredba konvertuje
realan u string tip promenljive rezultat, a zatim dodeljuje
svojstvu Text Edit komponente, što omogućava ispis rezultata.}

end;

procedure TForm1.Button2Click(Sender: TObject); {Procedura
OnClick dugmeta za oduzimanje.}
begin

br1 := StrToFloat(Edit1.Text);
br2 := StrToFloat(Edit2.Text);

rezultat := br1 - br2;
Edit3.Text := FloatToStr(rezultat);


end;

procedure TForm1.Button3Click(Sender: TObject); {Procedura
OnClick dugmeta za množenje.}
begin

br1 := StrToFloat(Edit1.Text);
br2 := StrToFloat(Edit2.Text);
```

```
rezultat := br1 * br2;
Edit3.Text := FloatToStr(rezultat);

end;

procedure TForm1.Button4Click(Sender: TObject); {Procedura
OnClick dugmeta za deljenje.}
begin

br1 := StrToFloat(Edit1.Text);
br2 := StrToFloat(Edit2.Text);

rezultat := br1 / br2;
Edit3.Text := FloatToStr(rezultat);


end;
```

Други задатак из лекције „Програмирање израчунавања по једноставним математичким формулама“ се односи на апликацију која врши израчунавање и исписивање обима троугла на основу координата темена које се уносе у текстуална поља едит компоненти.

Изглед апликације се може видети на слици 36.



Слика 36. Пример изгледа апликације

Када корисник кликне на компоненту **Button** са натписом „Izracunaj obim“, у текстуалном пољу **Edit** компоненте коју описује компонента **Label** „Obim“, треба да се испише резултат.

Решавање задатка се започиње постављањем компоненти унутар форме и подешавањем својстава за фонт и величину компоненти. Решење задатка садржи фајл са апликацијом, где

ученик може да сазна које су вредности карактеристика коришћене да би апликација изгледала као на слици 36. или може самостално да креира изглед апликације.

Како се испис врши када корисник кликне на компоненту **Button**, треба да се напишу следеће линије кода у костуру процедуре *OnClick*:

```
procedure TForm1.Button1Click(Sender: TObject);
var
x1, x2, x3, y1, y2, y3 : real; {Deklaracija promenljivih koje će čuvati vrednosti koordinata temena.}
a, b, c : real; {Deklaracija promenljivih koje će sadržati vrednosti dužina stranica trougla.}
O : real; {Deklaracija promenljive koja će sadržati vrednost obima trougla.}
{S obzirom da programski kôd sadrži samo jednu proceduru, promenljive se mogu deklarisati kao lokalne.}
begin
  x1 := StrToFloat(Edit1.Text);
  y1 := StrToFloat(Edit2.Text);
  x2 := StrToFloat(Edit3.Text);
  y2 := StrToFloat(Edit4.Text);
  x3 := StrToFloat(Edit5.Text);
  y3 := StrToFloat(Edit6.Text);

  a := sqrt(sqr(x1-x2) + sqr(y1-y2)); {Pri pisanju naredbi za izračunavanje stranica trougla, korišćen je obrazac koji omogućava izračunavanje rastojanja između dve tačke korišćenjem njihovih koordina.}
  b :=  sqrt(sqr(x1-x3) + sqr(y1-y3));
  c := sqrt(sqr(x3-x2) + sqr(y3-y2));

  {Nakon pisanja naredbi za izračunavanje dužina stranica, može se napisati naredba za izračunavanje obima.}

  O := a + b + c;   {Promenljivoj O je ovom naredbom dodeljena vrednost obima trougla, te se rezultat može ispisati. }

  Edit7.Text := FloatToStr(O);
end;
```

6. Наредба гранања

Наредба гранања омогућава да се изврши један део програма (тј. једна или више наредба) у зависности од испуњења одређених услова. Представља врло важну наредбу и незаобилазну конструкцију у апликацијама које захтевају испитивање одређеног услова како би се одређена наредба (или више њих) извршила.

У лекцијама у оквиру дела „Наредба гранања“ представљена је наредба гранања IF навођењем њене синтаксе и задатака за чије решавање је неопходно коришћење ове наредбе. Друга лекција из овог дела обрађује алгоритме за налажење минимума и максимума бројева. Лекција под називом „Алгоритми за одређивање минимума и максимума“ је корисна јер се идеје алгоритама могу користити за одређивање минимума и максимума неких других величина, а не само унетих бројева.

6. 1. Синтакса наредбе гранања IF

Наредба гранања IF се користи за проверу услова и извршавање дела кода у зависности од тога да ли је услов тачан или нетачан.

Наредба гранања IF се може исказати уз помоћ два облика: непотпуног и потпуног.

Синтакса непотпуне наредбе гранања IF:

```
if uslov then
naredba
```

У конкретном примеру део **uslov** се замењује логичким изразом који се испитује, а део **naredba** наредбом која треба да се изврши уколико логички израз има вредност True. Уколико логички израз има вредност False, врши се излаз из ове наредбе и ништа се неће десити.

Задаци наведени у овој лекцији су корисни јер показују коришћење IF наредбе.

Први задатак (слика 37.) се односи на креирање апликације којом се проверава да ли је унет број паран. Унос броја се врши у оквиру **Edit** контроле. Када корисник кликне на компоненту **Button**, у оквиру **Label** компоненте треба да се испише „Broj je paran!“ уколико је унет паран број.

Слика 37. Приказ задатка 1 из лекције

Решење задатка је представљено исечком програмског кода апликације:

```
procedure TForm1.Button1Click(Sender: TObject);
var broj, ostatak : integer;
begin
 broj  := StrToInt(Edit1.Text);  {Ovom naredbom se promenljivoj broj dodeljuje vrednost unetog broja u okviru Edit komponente.}
 ostatak := broj mod 2

 if  ostatak = 0 then
   ispisLabela.Caption := 'Broj je paran!'; {Ukoliko je ostatak 0 ispisaće se rečenica 'Broj je paran!' u Label komponenti čije je ime ispisLabela.
NAPOMENA: Ukoliko je unet neparan broj, izraz ostatak = 0 imaće vrednost False, tako da će nastati izlaz iz IF naredbe i ništa neće biti ispisano.}


end;
```

Циљ овог задатка је коришћење наредбе IF. Из тог разлога, вредности карактеристика компоненти нису наведене. Уколико ученик жели да сазна које су вредности коришћене, може преузети фајл са апликацијом.

Наредба гранања IF пружа могућност извршавања више наредби уколико је условни исказ тачан. У том случају се користе кључне речи **begin** и **end** како би те наредбе биле смештене у блок.

Синтакса непотпуне наредбе гранања IF која омогућава извршавање вишеструких наредби:

```
if uslov then
  begin
    naredba 1;
    naredba 2;
    .
    .
    .
    naredba n;
   end;
```

У неким случајевима може се наметнути захтев да се изврши нека радња када је условни исказ тачан, а да се изврши нека друга радња уколико је условни исказ нетачан.
У том случају се IF наредби придружује исказ **else**.
Исказ else се користи у спрези са исказом IF и означава део кода који ће бити извршен уколико је условни исказ нетачан, тј има вредност False.
Такав облик представља потпун облик IF наредбе.

Синтакса потпуног облика наредбе гранања IF са придруженим исказом else:

```
if uslov then
   naredba1
else
  naredba2;
```

Треба нагласити да се наредба, која се извршава уколико је условни исказ тачан, не завршава знаком тачка-зарез (;).
Ово правило не важи уколико се извршава блок наредби када је условни исказ тачан.

Синтакса потпуног облика наредбе гранања IF са придруженим исказом else која омогућава извршавање блока наредби:

```
if USLOV then
 begin
   naredba1;
   naredba2;
  .
  .
  .
   end
else
  begin
  naredban1;
   naredban2;
```

```
  .
  .
  .
end;
```

Након увођења исказа else први задатак из лекције се може надоградити захтевом да, апликација исписује поруку уколико је унет број непаран. Због тога је у лекцији представљен задатак 2. Изглед апликације треба да буде као у задатку 1 (слика 37.) али да омогући исписивање поруке „Broj je neparan!“ уколико корисник унесе непаран број унутар поља **Edit** компоненте.

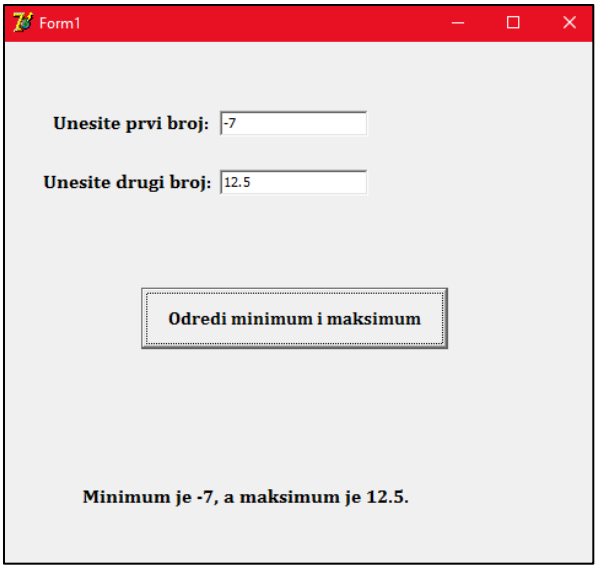
Решење овог задатка (у односу на претходно решење) се разликује једино у следећем исечку програмског кода:

```
procedure TForm1.Button1Click(Sender: TObject);
var broj, ostatak : integer;
begin
 broj := StrToInt(Edit1.Text);
 ostatak := broj mod 2;

 if  ostatak = 0  then
  ispisLabela.Caption := 'Broj je paran!'
 else {Ovde spadaju svi slučajevi za vrednosti ostatka, izuzev
0. Kako je jedina mogućnost da se pri deljenju sa 2 dobije
ostatak 0 ili 1, ukoliko ostatak ima vrednost 1, izvršiće se
naredba nakon iskaza else.}
  ispisLabela.Caption := 'Broj je neparan!';
end;
```

6. 2. Алгоритми за одређивање минимума и максимума бројева

Примена наредбе IF је представљена у електронској лекцији под називом „Алгоритми за одређивање минимума/максимума два/три броја“. Алгоритми за одређивање минимума и максимума унетих бројева су обрађени кроз два задатка. Први задатак се односи на приказивање алгоритма за одређивање минимума и максимума два броја, а у другом задатку три броја. Уколико ученик усвоји идеје алгоритама, представљених у решењу примера из лекције, то знање може да примени на решавање сличних проблема, као и на одређивање минимума и максимума више од три броја.

Захтев првог задатка јесте креирање апликације (слика 38.) која за унета два броја одређује и приказује минимум и максимум. Уколико корисник унесе једнаке бројеве у **Label** компоненти треба да се испише порука: „Uneti brojevi su jednaki.“.



Слика 38. Пример изгледа покренуте апликације

Решење задатка је представљено одсечком програмског кода апликације:

```
procedure TForm1.Button1Click(Sender: TObject);
var  prvi_br,  drugi_br  :  real;  {Deklaracija  promenljivih  za
čuvanje vrednosti brojeva iz Edit komponente.}
begin

prvi_br  :=  StrToFloat(Edit1.Text);  {Ove  naredbe  preuzimaju
upisane  brojeve,  konvertuju  ih  u  realan  tip  i  dodeljuju
promenljivama prvi_br i drugi_br.}
drugi_br := StrToFloat(Edit2.Text);

if prvi_br > drugi_br then    {Ukoliko je prvi_br veći od drugog,
on je maksimum, a drugi je minimum.}
  Label3.Caption := 'Minimum je ' + FloatToStr(drugi_br) + ', a
maksimum je ' + FloatToStr(prvi_br) + '.'; {Ovo je naredba za
ispis rešenja, ukoliko je prvi broj veći od drugog, tj. ukoliko
je prvi broj maksimum a drugi minimum.}


if prvi_br < drugi_br then  { Ukoliko je prvi_br manji od drugog,
on je minimum, a drugi je maksimum. }
  Label3.Caption := 'Minimum je ' + FloatToStr(prvi_br) + ', a
maksimum je ' + FloatToStr(drugi_br) + '.';

if  prvi_br  =  drugi_br  then   {Ukoliko  su  prvi_br  i  drugi_br
jednaki, izvrsice se naredba ispod. }
  Label3.Caption := 'Uneti brojevi su jednaki.';
```

```
end;
```

Треба напоменути да ово није једини начин за решавање, па је у лекцији препоручено да ученик за вежбу креира апликацију на други начин, коришћењем else исказа.

Одређивање минимума и максимума три броја је објашњено кроз други задатак у оквиру лекције.
Изглед апликације је представљен на слици 39. која за унета три реална броја одређује и исписује минимум и максимум.

Слика 39. Пример изгледа покренуте апликације

Решење примера је представљено деловима програмског кода, где ученик може да погледа које наредбе треба написати у оквиру процедура за обраду догађаја *OnClick* постављених компоненти **Button**.

```
procedure TForm1.Button1Click(Sender: TObject);
var a, b, c, min : real; {Promenljive a, b i c će sadržati unete vrednosti brojeva. Promenljiva a će sadržati vrednost prvog unetog broja, b drugog i c trećeg. Promenljiva min ce imati vrednost trazenog minimuma. Promenljive sum ogle biti deklarisane i kao globalne.}
begin
  a := StrToFloat(Edit1.Text);
  b := StrToFloat(Edit2.Text);
  c := StrToFloat(Edit3.Text);

  min := a; {Pretpostavlja se da je prvi broj minimalan. Zatim se vrši ispitivanje da li su ostali brojevi manji od njega. Ukoliko neki od njih jeste, njegova vrednost se dodeljuje promenljivoj min. Ukoliko nije, prvi broj a je minimum.}
```

```
  if b < a then
    min := b;    {Ukoliko je drugi broj manji od prvog, njegova
vrednost se dodeljuje promenljivoj min.}

  if c < min then   {Pri izlasku iz prethodne naredbe if,
promenljiva min sadrzi manji od prva dva uneta broja. Ukoliko je
c manji od trenutnog minimuma, njegova vrednost je zapravo
minimalna te se dodeljuje promenljivoj min.}
    min := c;

 Edit4.Text := FloatToStr(min); {Edit4 komponenta sluzi za ispis
minimuma.}
end;
procedure TForm1.Button2Click(Sender: TObject);
var a, b, c, max : real;
begin

  a := StrToFloat(Edit1.Text);
  b := StrToFloat(Edit2.Text);
  c := StrToFloat(Edit3.Text);

  max := a; {Slično je kao za traženje minimuma. Jedino se sad
traži da li postoji veći broj od pretpostavljenog maksimuma.}


  if b > a then
    max := b; {Ukoliko je drugi broj veći od prvog, njegova
vrednost se dodeljuje promenljivoj max.}

  if c > max then {Pri izlasku iz prethodne naredbe if,
promenljiva max sadrzi veći od prva dva uneta broja. Ukoliko je
c veći od trenutnog maksimuma, njegova vrednost je zapravo
maksimalna te se dodeljuje promenljivoj max.}
    max := c;

 Edit5.Text := FloatToStr(max); {Edit5 komponenta služi za ispis
maksimuma.}
end;
```

7. Компоненте избора и контејнерске компоненте

Компоненте избора омогућавају избор једне, између више понуђених могућности (у случају RadioButton компоненте нпр.) или избор ниједне, једне или више могућности (у случају, нпр. CheckBox компоненте).

Компоненте које међусобно повезане реализују један логички сегмент апликације, могу се сместити на компоненту која се назива контејнерска компонента. Оваквим груписањем се остварује визуелна прегледност логичких целина апликације и самим тим, лакше сналажење корисника. GroupBox и Panel су примери контејнерских компоненти.

У лекцијама су прво представљене компоненте избора: RadioButton (радио дугме), CheckBox (оквир за потврду), ListBox (оквир са листом), ComboBox (комбиновани оквир), а затим су представљене две контејнерске компоненте: GroupBox (оквир за групу) и Panel (плоча).

7. 1. Компонента RadioButton

Компонента **RadioButton** (радио дугме) представља компоненту избора којом се нека опција одабира или не, те је погодна за креирање упитника и тестова. Ова компонента се често користи са више **RadioButton** компонената, с тим да само једна може бити одабрана. Састоји се из два дела: поља за потврду и натписа који ближе објашњава његову намену.

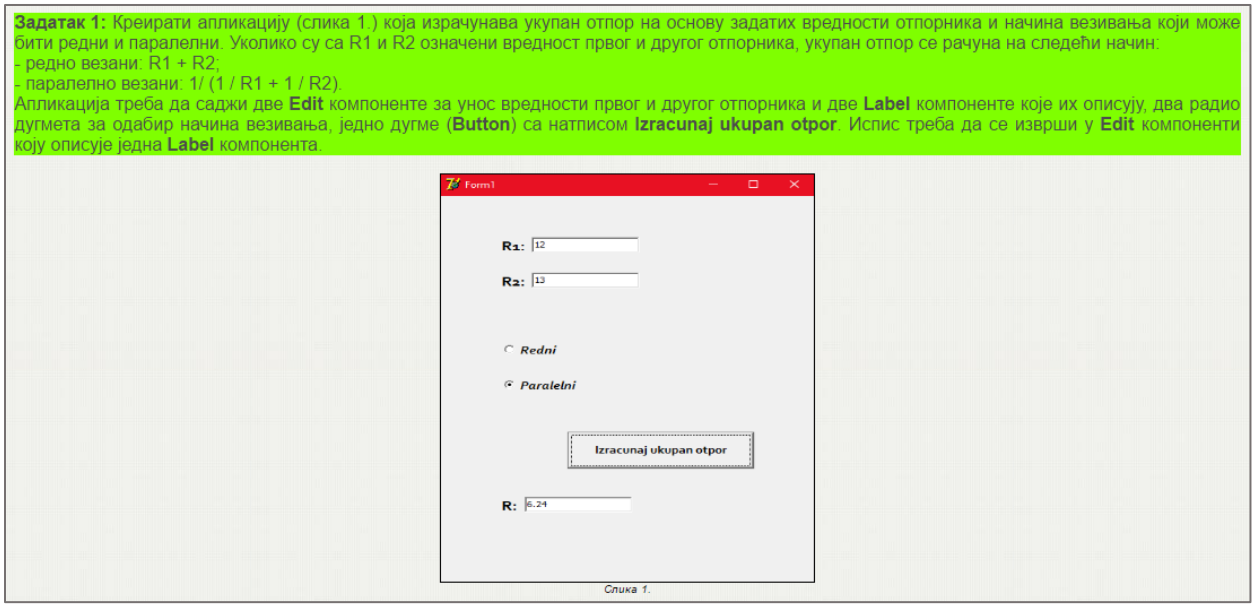
Налази се на картици Standard (слика 40.) у оквиру палете компоненти и поставља се унутар форме као стандардна компонента.

Слика 40. Компонента RadioButton

Врло битно својство **RadioButton** компоненте је *Checked* које служи за проверу стања (да ли је потврђена или не). Уколико има вредност True компонента **RadioButton** је потврђена, а уколико има вредност False значи да није потврђена.

Својство *Checked* за компоненту **RadioButton** се најчешће користи као условни исказ наредбе гранања.

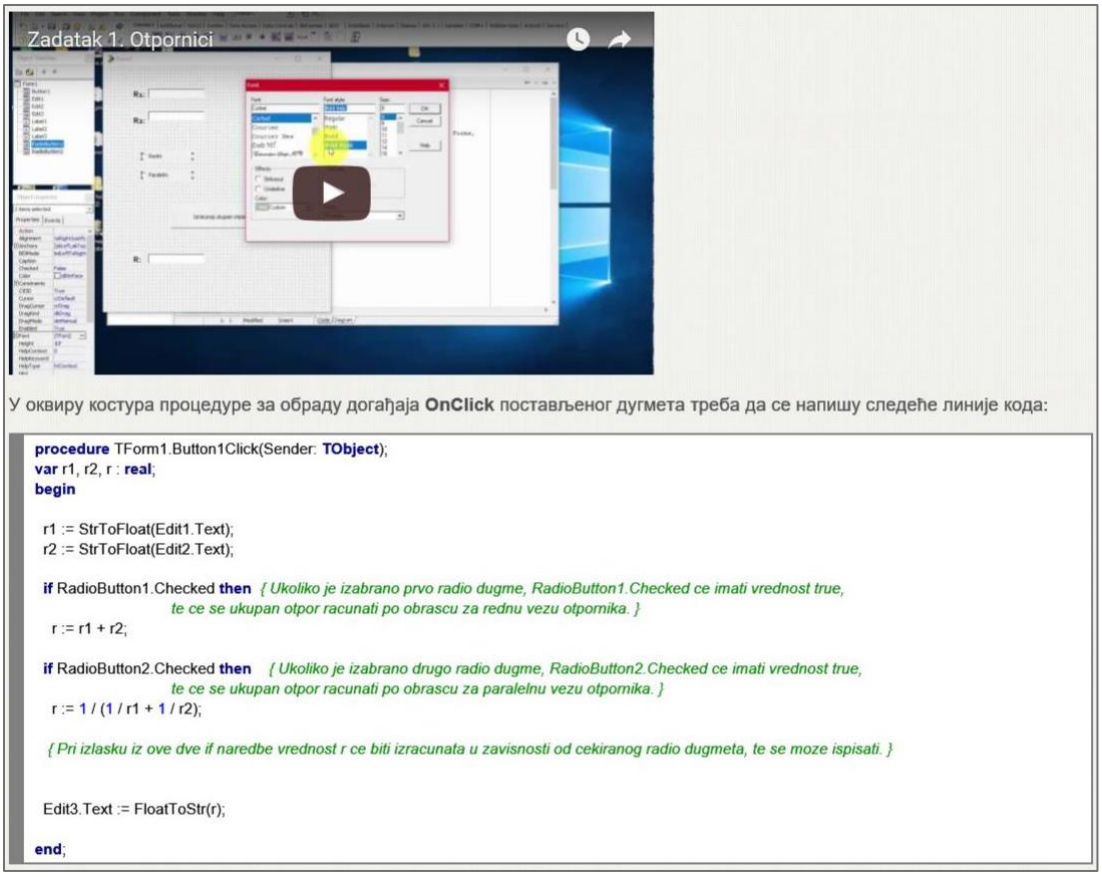
Коришћење компоненте **RadioButton** је у лекцији представљено задатком који се може видети на слици 41.



Слика 41. Приказ првог задатка из лекције „Компонента RadioButton“

На слици 41. се може видети да је захтев задатка креирање апликације која садржи две **RadioButton** компоненте које служе за одабир везе отпорника. Апликација треба да израчуна укупан отпор на основу унетих вредности отпорника у оквиру поља **Edit** компоненти. Испис се такође врши унутар **Edit** контроле, након клика на компоненту **Button** са натписом „Izracunaj ukupan otpor“.

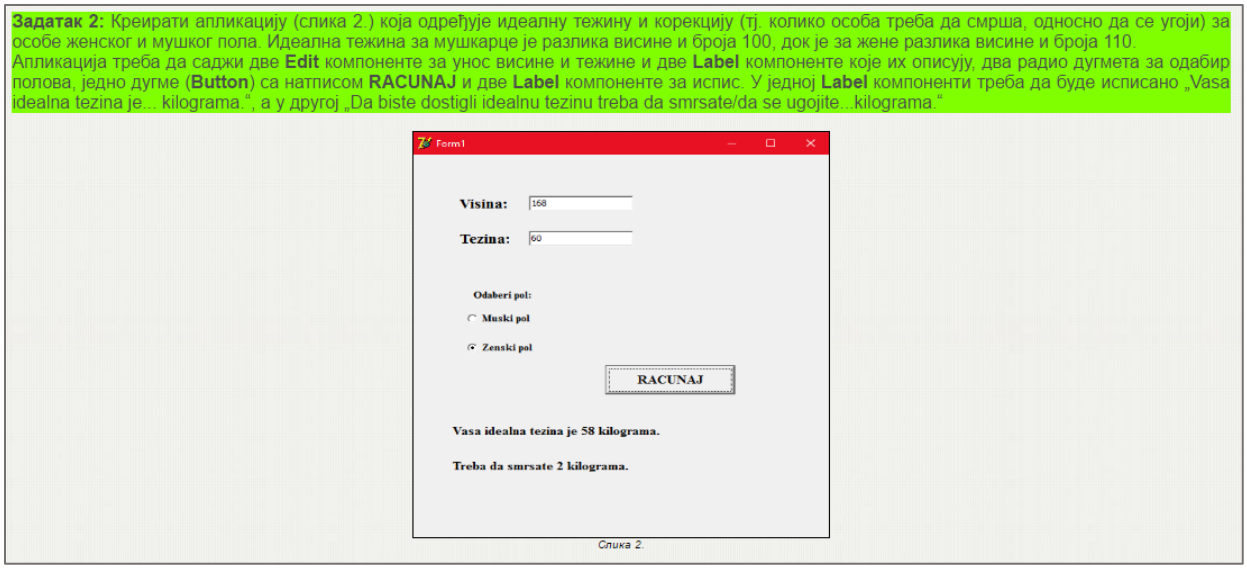
Решење задатка је представљено видео материјалом и исечком програмског кода апликације (слика 42.).



Слика 42. Приказ решења задатка 1 из лекције „Компонента RadioButton“

Решење задатка се такође може преузети.

Лекција садржи још један задатак са решењем, који приказују како се креира апликација за израчунавање идеалне тежине особе у зависности од пола. На слици 43. је приказан изглед задатка.



Слика 43. Приказ другог задатка из лекције „Компонента RadioButton“

Решење задатка је приказано исечком програмског кода:

```
procedure TForm1.Button1Click(Sender: TObject);
var v, t, idealna_t, korekcija : real;
begin
 v := StrToFloat(Edit1.Text);
 t := StrToFloat(Edit2.Text);

 if RadioButton1.Checked then   {Ukoliko je odabrana komponenta
RadioButton1, tj. prvo radio dugme, RadioButton1.Checked, imaće
vrednost True, te će se izvršiti blok naredbi unutar ove if
naredbe.}
 begin
  idealna_t := v - 100;
  Label4.Caption := 'Vasa idealna tezina je ' +
  FloatToStr(idealna_t) + ' kilograma.';
  korekcija := t - idealna_t;

 if korekcija > 0 then
  Label5.Caption := 'Treba da smrsate ' + FloatToStr(korekcija)
+ ' kilograma.'
 else
  Label5.Caption := 'Treba da se ugojite ' +
  FloatToStr(abs(korekcija)) + ' kilograma.'
end;
```

```
 if RadioButton2.Checked then {Ukoliko je drugo radio dugme
odabrano, RadioButton2.Checked, imaće vrednost True, te će se
izvršiti blok naredbi unutar ove if naredbe.}
 begin
  idealna_t := v - 110;
  Label4.Caption   :=   'Vasa   idealna   tezina   je   '   +
FloatToStr(idealna_t) + ' kilograma.';

  korekcija := t - idealna_t;

 if korekcija > 0 then
   Label5.Caption := 'Treba da smrsate ' + FloatToStr(korekcija)
+ ' kilograma.'
 else
   Label5.Caption   :=   'Treba   da   se   ugojite   '   +
FloatToStr(abs(korekcija)) + ' kilograma.'

 end

end;
```

7. 2. Компонента CheckBox

Компонента CheckBox (оквир за потврду) служи за укључивање или искључивање неке опције, те је погодна за креирање упитника и тестова. Слично компоненти **RadioButton**, ова компонента такође има два дела: поље за потврду и натпис који ближе објашњава његову намену. За разлику од **RadioButton** компоненте, уколико апликација саджи више **CheckBox** компоненти, две (или више) могу истовремено бити чекиране.

Налази се на картици Standard (слика 44.) у оквиру палете компоненти и поставља се унутар форме као стандардна компонента.

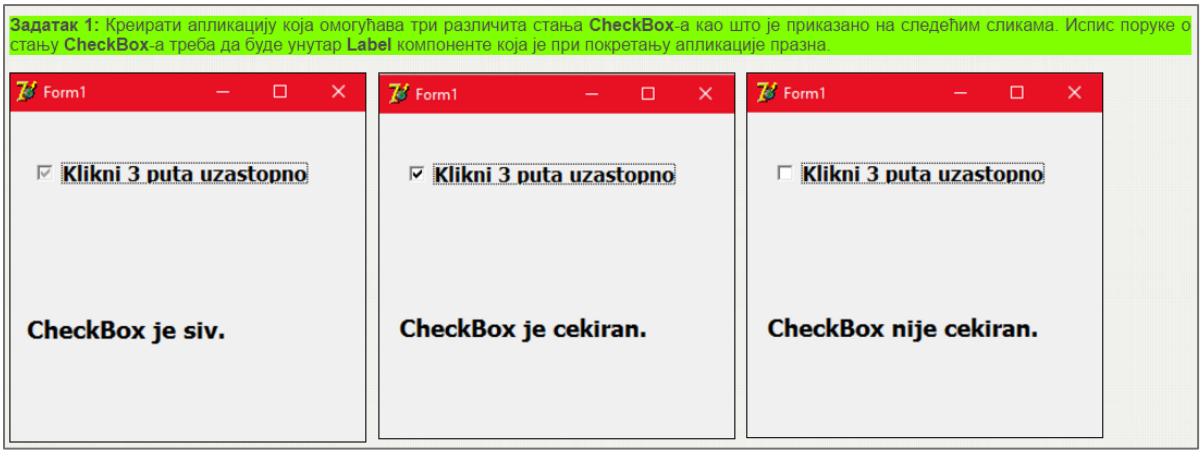
Слика 44. Компонента CheckBox

Провера да ли је **CheckBox** укључен или није се може реализовати провером вредности својства *Checked*, која може бити True (уколико је **CheckBox** укључен), односно False (уколико није укључен).

Компонента **CheckBox** може имати и треће стање које се визуелно приказује као сиво.

Да би се омогућило треће стање **CheckBox**-а потребно је својству *AlowGrayed* поставити вредност на True. У овом случају се провера стања **CheckBox**-а реализује коришћењем својства *State* које може имати три вредности: cbChecked, cbUnchecked и cbGrayed.

Ова карактеристика компоненте **CheckBox** је обрађена и кроз задатак (слика 45.) у лекцији.



Слика 45. Приказ задатка из лекције „Компонента CheckBox“

Решење задатка ученик може да погледа у видео материјалу који се налази у лекцији или преузимањем фајла са решењем.

Поред тога, у лекцији је приказан и исечак програмског кода:

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin

  if CheckBox1.State = cbGrayed
    then Label1.Caption := 'CheckBox je siv.'
  else
    if CheckBox1.State = cbChecked
      then Label1.Caption := 'CheckBox je cekiran.'
    else Label1.Caption := 'CheckBox nije cekiran.';
end;
```

7. 3. Компонента ListBox

Компонента ListBox (оквир са листом) је чест елемент у Windows апликацијама. Ова компонента садржи низ ставки (текстуалних података) и омогућава кориснику да изабере једну или више њих.

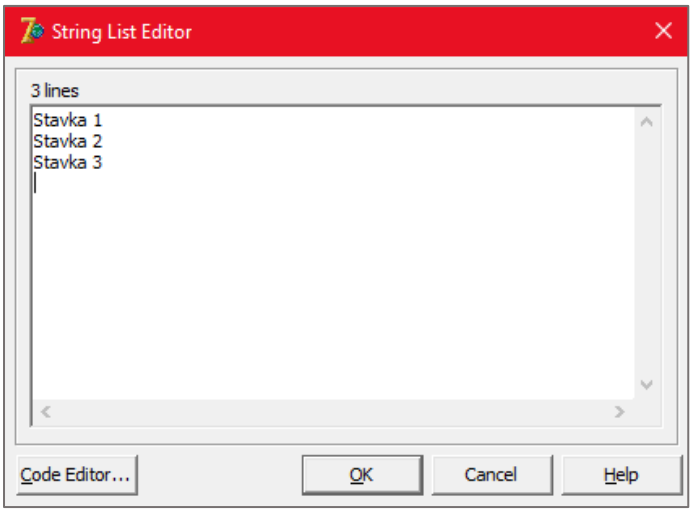
Налази се на картици Standard (слика 46.) у оквиру палете компоненти и поставља се унутар форме као стандардна компонента.

Слика 46. Компонента ListBox

Битно својство код ове компоненте је *Items*, које служи за постављање ставки у току пројектовања или извршавања програма.

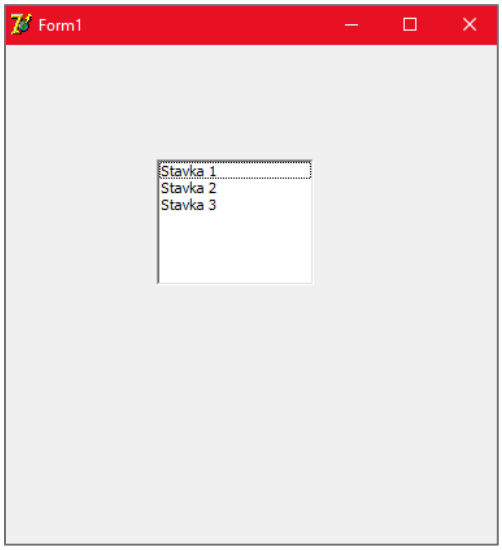
Да би се у току пројектовања поставиле вредности својства *Items*, потребно је:

1. Поставити компоненту **ListBox** унутар форме;
2. У инспектору објеката, десно од опције *Items*, двоструким кликом отвара се едитор стрингова у коме се врши унос ставки (слика 47.);



Слика 47. Едитор стингова за унос ставки

Након клика на дугме са натписом OK, **ListBox** ће изгледати као што је представљено на слици 48.



Слика 48. Изглед апликације након постављања компоненте ListBox са ставкама

У току извршавања је могуће у својство *Items* учитати листу ставки из текстуалног фајла методом **LoadFromFile**.

Пример 46: Наредба која омогућава учитавање листе ставки из текстуалног фајла stavke је облика:

```
ListBox1.Items.LoadFromFile('c:\data\stavke.txt');
```

Садржај својства *Items* се може копирати у фајл методом **SaveToFile**.

Пример 47: Наредба која омогућава копирање ставки компоненте **ListBox1** у фајл Snimi је облика:

```
ListBox1.Items.SaveToFile('C:\Snimi.txt');
```

Уколико је својство *MultiSelect* постављено на вредност False (што је подразумевана вредност) могуће је одабрати само једну ставку. Уколико се јави потреба за одабирањем више ставки из **ListBox**-а, својству *MultiSelect* треба доделити вредност True.

Још једно битно својство компоненте **ListBox** јесте својство *ItemIndex* типа Integer, у коме је садржан редни број (индекс) одабране ставке.

Битно је напоменути да су ставке индексиране у границама од 0 до Items.Count - 1.

Count је својство чија је вредност једнака укупном броју ставки.

Конкретној ставци из **ListBox**-а се приступа са *Items* и угластим заградама, унутар којих се наводи индекс ставке.

Пример 48: Приступање ставкама компоненте **ListBox** се врши следећим наредбама:

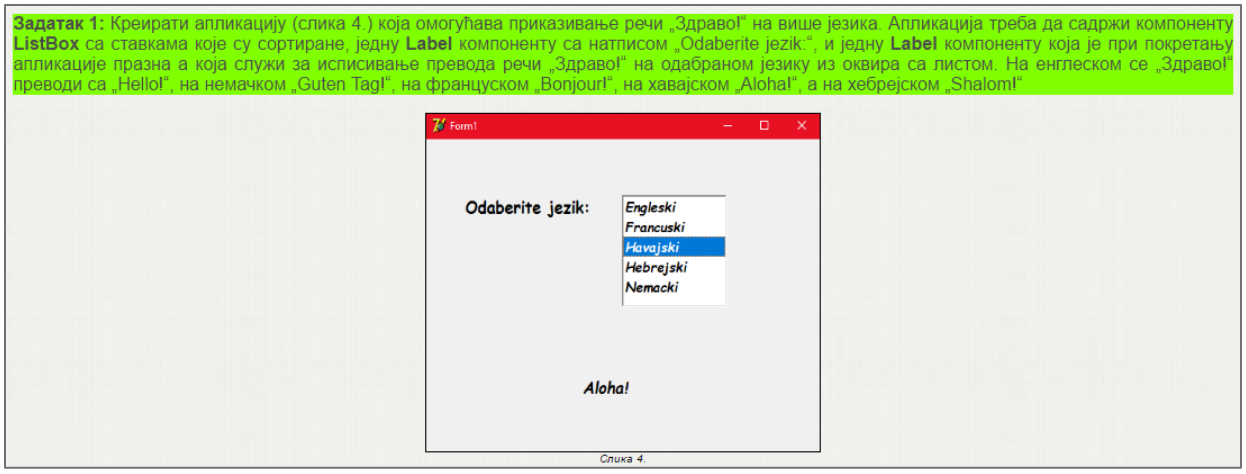
```
ListBox1.Items[0]; {Ovo je identično prvoj stavci komponente
ListBox1. Napomena: Sve stavke su tipa string! }
ListBox1.Items[1]; {Druga stavka komponente ListBox1.}
  .
  .
  .
ListBox1.Items[Count-1]; {Poslednja stavka komponente
ListBox1.}
```

Селектованој ставци се приступа са: **ListBox1.Items[ListBox1.ItemIndex]**, при чему **ListBox1.ItemIndex** садржи целобројну вредност – индекс селектоване ставке.

Компонента **ListBox** у току извршавања располаже својством *Selected* типа Boolean, које информише које ставке су изабране из оквира са листом. С обзиром на тип Boolean, својство *Selected* је погодно за условни исказ наредбе IF.

Треба напоменути да компонента **ListBox** не поседује својство *Caption*.

У лекцији је кроз задатак и решење приказано коришћење компоненте **ListBox**.
На слици 49. је представљен приказ задатка из лекције.



Слика 49. Приказ задатка из лекције компонента ListBox

Захтев задатка је креирање апликације која исписује реч „Здраво!“ на различитим језицима. Одабир језика се врши одабиром одређене ставке компоненте **ListBox**. Када корисник одабере језик, у оквиру компоненте за натпис треба да се испише превод поздрава.
Решење задатка, ученик може видети на постављеном видеу у лекцији.
Такође, решење задатка садржи одсечак програмског кода апликације:

```
procedure TForm1.ListBox1Click(Sender: TObject);
begin
if ListBox1.Selected[0] then
  Label2.Caption := 'Hello!'; {Ukoliko je odabrana prva stavka, izvršiće se ova naredba, tj. ispisaće se u Label komponenti "Hello!".}

if ListBox1.Selected[1] then
  Label2.Caption := 'Bonjour!'; {Ukoliko je odabrana druga stavka, izvršiće se ova naredba, tj. ispisaće se u Label komponenti "Bonjour!".}

if ListBox1.Selected[2] then
  Label2.Caption := 'Aloha!'; {Ukoliko je odabrana treća stavka, izvršiće se ova naredba, tj. ispisaće se u Label komponenti "Aloha!".}

if ListBox1.Selected[3] then
```

```
  Label2.Caption := 'Shalom!';   {Ukoliko je odabrana četvrta
stavka, izvršiće se ova naredba, tj. ispisaće se u Label
komponenti "Shalom!" }

if ListBox1.Selected[4] then
  Label2.Caption := 'Guten Tag!'; {Ukoliko je odabrana peta
stavka, izvršiće se ova naredba, tj. ispisaće se u Label
komponenti "Guten Tag!" }
end;
```

7. 4. Компонента ComboBox

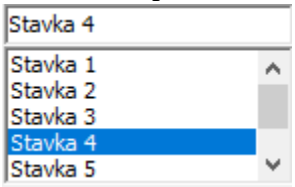
Компонента **ComboBox** (комбиновани оквир) комбинује **Edit** контролу са компонентом **ListBox**. Ова компонента омогућава избор једне од ставки листе, или уношење линије текста преко тастатуре као код **Edit** контрола. Изабрана ставка из листе се поставља у **Edit** контролу.

Налази се на картици Standard (слика 50.) у оквиру палете компоненти и поставља се унутар форме као стандардна компонента.

Слика 50. Компонента ComboBox

Постоје три врсте компоненти типа **ComboBox** које се дефинишу својством *Style*:

1. *Simple* - изгледа као листа са **Edit** контролом у заглављу (слика 51.). Корисник може да одабере елемент са листе (који ће се приказати у заглављу) или да унесе нови текст у поље за текст (**Edit** контролу). **Simple ComboBox** се поставља унутар форме тако што се прво постави компонента **ComboBox**, а затим се у инспектору објеката својство *Style* подешава на вредност csSimple.



Слика 51. Simple ComboBox

2. *Drop-Down* - држи листу „уролану“ и скривену од погледа, све док се не кликне на дугме са стрлицом надоле које се налази са десне стране **Edit** контроле (слика 52.). Тада се појављује падајућа листа и корисник може да изабере ставку са листе која ће се приказати у пољу за текст. **Drop-Down ComboBox** се поставља унутар форме тако што се прво постави компонента **ComboBox**, а затим у инспектору објеката се својство *Style* подешава на вредност csDropDown.

Слика 52. Drop-Down ComboBox

3. *Drop-Down-List* - држи листу „уролану“, која се отвара на клик на стрелицу надоле. Кликом на неку од ставки, она се приказује у пољу за текст. Овај стил **ComboBox**-а је по изгледу исти као **Drop-Down ComboBox** (слика 52.). Једина разлика је у томе што стил *Drop-Down-List* не дозвољава да се било шта унесе у поље за текст (**Edit** контролу). *Drop-Down-List* стил компоненте **ComboBox** се поставља уз помоћ опције csDropDownList својства *Style*.

Постављање ставки унутар **ComboBox**-а је идентично као код **ListBox**-а. У инспектору објеката, десно од опције *Items*, двоструким кликом отвара се едитор стрингова у коме треба унети ставке.

Ставке могу да се додају и приликом извршавања апликације методом **Add** својства *Items*.

Пример 49: Наредба која омогућава додавање ставке Nova stavka компоненти **ComboBox** у току извршавања апликације је облика:

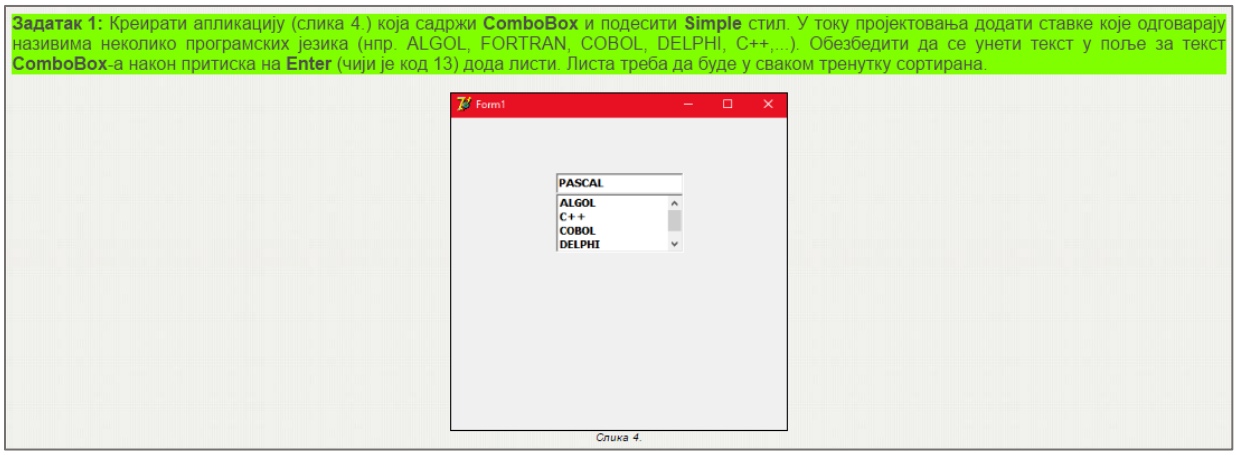
```
ComboBox1.Items.Add('Nova stavka');
```

Такође је могуће као ставку додати и текст из исте контроле.

Пример 50: Наредба која омогућава додавање ставке из **Edit** компоненте је облика:

```
ComboBox1.Items.Add(ComboBox1.Text);
```

У лекцији се налази, поред наведеног описивање компоненте **ComboBox**, задатак (слика 53.) који се односи на коришћење ове компоненте у апликацијама.



Слика 53. Приказ задатка 1 из лекције „Компонента ComboBox“

Ученик креирање апликације може да погледа преко видеа. Такође је у лекцији представљен и одсечак кода који се односи на процедуру *OnKeyPress* компоненте **ComboBox**:

```
procedure TForm1.ComboBox1KeyPress(Sender: TObject; var Key:
Char);
begin
 if Key = #13 then  {Ako je pritisnut taster Enter (kod je 13),
izvršiće se blok naredbi unutar ove IF naredbe.}
 begin
  ComboBox1.Items.Add(ComboBox1.Text);  {Naredba za dodavanje
unetog teksta listi ComboBox-a.}

  ComboBox1.Text := '';   {Naredba koja će nakon izvršavanja
prethodne naredbe izbrisati sadržaj polja za tekst.}
  Key := #0;    {Ova naredba onesposobljava bilo kakvu akciju,
posto znak #0 nema nikakvo znacenje.}
  end;

end;
```

7. 5. Компонента GroupBox

Компонента **GroupBox** (оквир за групу) се користи за груписање компоненти типа **CheckBox**, **RadioButton** и других контрола у логичке целине.

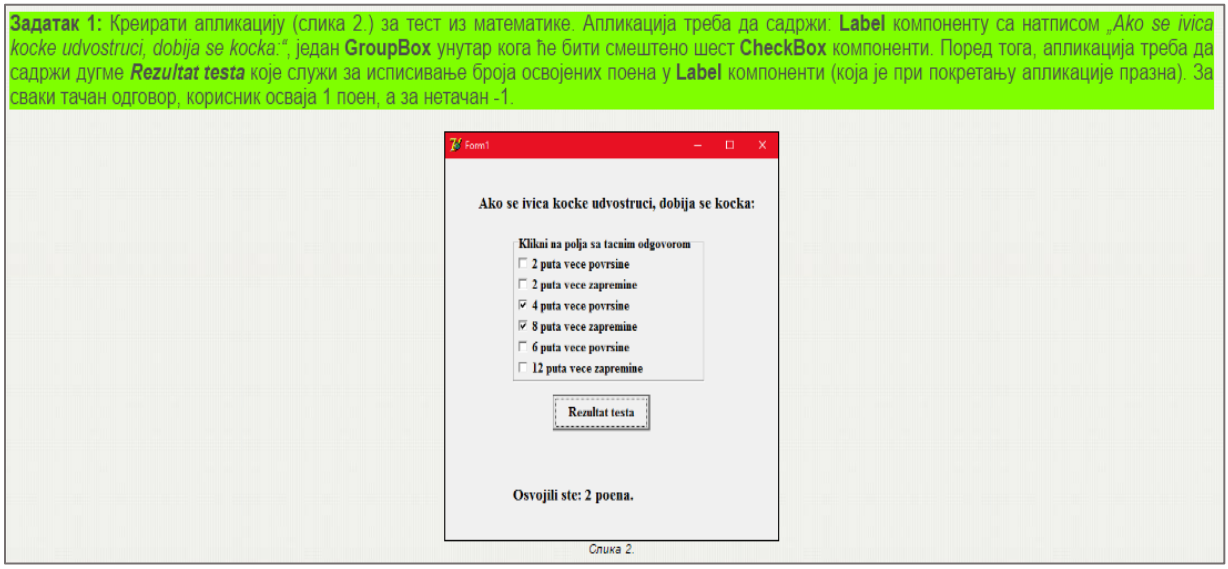
Када се ова компонента постави на форму, на њеној површини се могу додавати друге компоненте простим кликом у оквиру њених граница. Она тиме постаје „родитељ“ нове компоненте, која се унутар ње може померати коришћењем миша, али се не може одвући ван њених граница. Премештање компоненте из једне групе у другу могуће је само операцијама Cut и Paste.

Налази се на картици Standard (слика 54.) у оквиру палете компоненти и поставља се унутар форме као стандардна компонента.

Слика 54. Компонента GroupBox

Компонента постављена унутар оквира за групу (компонента „дете“) може да наследи вредности својстава *Font* и *Color* од „родитеља“ постављањем својстава *ParentFont* и *ParentColor* на вредност True.

У лекцији се налази задатак који чији је захтев креирање апликација која представља тест из математике (слика 55.).



Слика 55. Приказ задатка 1 из лекције компонента GroupBox

Постављање компоненти унутар форме апликације и програмирање дугмета је објашњено у видеу који се налази у лекцији. У оквиру решења је приказан и исечак програмског кода.

```
procedure TForm1.Button1Click(Sender: TObject);
var p : integer; {Promenljiva p će imati vrednost koja je jednaka
ukupnom broju poena.}
begin
  p := ord(CheckBox3.Checked) + ord(CheckBox4.Checked)-
     ord(CheckBox1.Checked) - ord(CheckBox2.Checked) -
     ord(CheckBox5.Checked) - ord(CheckBox6.Checked);

{Funkcija ord je pogodna za brojanje poena, jer je ord(True)=1
     i ord(False)=0.}
{Tačni odgovori odgovaraju trećem i četvrtom CheckBox-u, pa se
zato sabiraju.}

{Netačni odgovori su svi ostali, pa se zato oduzimaju. }

  Label2.Caption := 'Osvojili ste: ' + IntToStr(p) + ' poena.';

end;
```

8. Наредба за организацију циклуса FOR

Наредба за ораганизацију циклуса (петља) је елемент програмског језика који се користи за вишеструко извршавање акције, све док је задовољен одређен услов.

У већини случајева све петље раде на веома сличан начин и садрже заједничке елементе:

- Почетак;
- Тело петље;
- Завршетак;
- Проверу услова који одређује да ли треба да се заврши петља;
- Опционо саджи наредбе *Break* и *Continue*.

Почетна тачка петље је представљена кључном речју за петљу **`for`**.

Тело петље садржи наредбе које се извршавају приликом сваког пролаза кроз петљу и он може да садржи било који исправан код у виду једне линије или блока наредби. Неопходно је да се блок наредби напише између кључних речи **`begin`** и **`end`**.

Приликом писања наредбе за организацију циклуса, неопходно је омогућити завршетак (излаз из петље). У противном доћи ће до појаве бесконачног циклуса.

Наредба FOR служи за опис циклуса код којих је број понављања наредби унапред познат.

Битан параметар FOR петље представља тзв. **управљачка променљива (бројач)**, чијом се променом вредности пребројава колико пута су извршене наредбе у циклусу.

FOR циклус може имати два различита облика у зависности да ли се управљачка променљива увећава или смањује за 1.

Синтакса FOR циклуса када се управљачка променљива увећава за 1:

```
for ime_upravljacke_promenljive := pocetna_vrednost to krajnja_vrednost do
  naredba;
```

При чему је:

- **ime_upravljacke_promenljive** идентификатор управљачке променљиве. Најчешће се обележава са **i**. Није грешка обележити је неким другим именом, битно је да буде синтаксно исправно;
- **pocetna_vrednost** је целобројна, иницијална вредност коју управљачка променљива садржи у првом проласку кроз петљу. У сваком наредном проласку, вредност је већа за један. Када се достигне **krajnja_vrednost** (која исто мора бити целобројна), врши се још један пролазак кроз петљу, а затим се из ње излази и извршавање програма се преноси на линију кода који се налази иза наредбе петље.

Управљачка променљива се мора декларисати (као и свака друга променљива).

Синтакса FOR циклуса када се управљачка променљива смањује за 1:

```
for ime_upravljacke_promenljive := krajnja_vrednost downto pocetna_vrednost do
   naredba;
```

У оба случаја FOR циклуса је дозвољено циклично извршавање блока наредби. Наредбе које чине блок треба ставити између кључних речи **begin** и **end**. Уколико се изостави коришћење ових кључних речи како би се означио блок, биће извршена само прва наредба.

Тренутна верзија лекција не садржи задатке који се решавају применом FOR циклуса. Могуће је да ће се у будућности појавити допуњена верзија лекција са задацима из ове области.

9. Закључак

Слободно се може рећи да информационо-комуникационе технологије обележавају 21. век. С обзиром на широк спектар могућности које оне нуде, њихова примена пружа могућност образовању да изађе из оквира традиционалне наставе и тиме постане независно од времена и простора. Савремена настава, између осталог, подразумева усмеравање ученика ка различитим изворима информација. Прикупљањем, анализирањем и обрађивањем тих информација могу се стећи знање и вештине из многих области.

На претходним странама овог рада је представљен начин креирања електронских лекција за предмет Рачунарство и информатика. Због јавне доступности, електронске лекције могу бити од користи и професорима и ученицима. Електронске лекције се могу користити као материјал у току наставе и као помоћни приручник ученицима за самостални рад и учење након наставе.

Тренутно у школама, концепт наставе заснован на електронским лекцијама је тек почео да се развија. Масовно креирање електронских лекција у будућности би омогућило размену нових идеја, наставних садржаја и метода међу професорима, а ученицима би била доступна средства за усвајање знања и стицање вештина без обзира на време, физичко и географско место у ком се налазе.

10. Литература

[1] Глушац Д., *Електронско учење*, Технички факултет „Михајло Пупин“ Зрењанин:

http://www.tfzr.uns.ac.rs/Content/files/0/Knjiga%20Elektronsko%20ucenje.pdf

Приступљено септембра 2016.

[2] *Електронски часопис за наставнике*:

http://www.microsoftsrb.rs/download/obrazovanje/pil/Elektronsko_ucenje.pdf

Приступљено септембра 2016.

[3] *Увод у Delphi*, Машински факултет у Нишу:
ftp://ftp.masfak.ni.ac.rs/PROGRAMMING/DELPHI/sa%20predavanja/

Приступљено августа 2016.

[4] *Top 10 Reasons to be a Delphi Developer*, Embarcadero:

http://community.embarcadero.com/blogs?view=entry&id=8654

Приступљено августа 2016.

[5] Марић М., *Програмски језици*, Девета гимназија „Михаило Петровић Алас“, Београд:

https://milenamaric.files.wordpress.com/2015/08/programskijezici.pdf

Приступљено августа 2016.