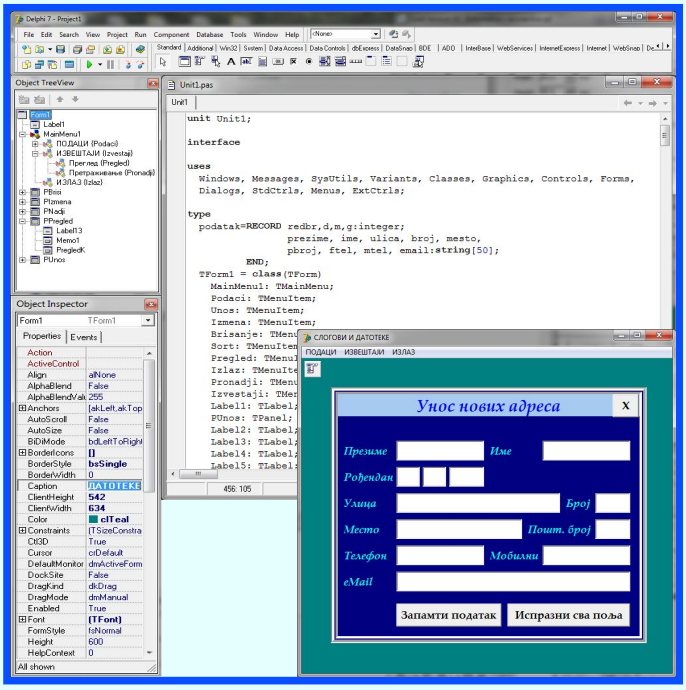
М и р о с л а в И л и ћ

П Р О Г Р А М С К И Ј Е З И К
Д Е Л Ф И
П Р В И К О Р А Ц И

(програмирање од А до Ш)



Београд, 2014.

Програмирање није само написати програм који решава одређени проблем.
Програмирање је више од тога, много више.
Програмирање је уметност.
Сваки, па био то и најједноставнији задатак може се решити тако да буде на ваш начин занимљив, необичан, привлачан, заводљиво оригиналан, ...
Ако желите да се бавите програмирањем радите то из љубави, нека сваки програмски ред буде стих у поеми вашег живота, мисао која плени духом.
Ако и не будете програмери у свом будућем животу, чак и тада посао који обављате нека буде прављење најлепшег букета који ћете поклањати свету.

Садржај

Решавање проблема помоћу рачунара

Рачунари се, у данашње време, могу применити у готово свим областима људског деловања и од њих се очекује да решавају све проблеме на које се може наићи. Међутим, потребно је знати да рачунар сам не може решити ни један (и најједноставнији) проблем. Он само извршава низ инструкција које му корисник задаје. Према томе, само ако корисник разуме проблем и познаје пут за решавање тог проблема до најситнијих детаља, он ће бити у стању да тај пут представи преко низа инструкција које рачунар може да *разуме* и *прихвати*. Тек тада можемо очекивати од рачунара да тај низ инструкција веома брзо и тачно изврши и да дође до резултата који су потребни. То значи да рачунар не олакшава кориснику рад на решавању проблема, већ само извршава задати програм и у веома кратком времену даје одговарајуће резултате.

Када се процес решавања неког проблема спроводи без рачунара, о сваком кораку се мисли непосредно у тренутку када тај корак треба да се спроведе, тј. када покушавамо да решимо неки проблем ми не знамо унапред шта ћемо све радити док не дођемо до решења, већ одлучујемо само о првом следећем кораку. Међутим, када се за решавање проблема користи рачунар, тада цео низ корака које тај процес обухвата мора бити прецизно одређен и унапред задат са свим могућностима које могу да се појаве у току извршавања у зависности од задатих почетних вредности. Тај низ корака мора бити прецизно дефинисан у облику веза и логичких услова. То значи да се, пре него што се приступи коришћењу рачунара за решавање неког проблема, мора написати програм. Са написаним програмом и улазним подацима, који одређују вредности низа параметара који се јављају у програму, рачунар може да добије резултате задатка који се решава.

Основи алгоритмизације

Појам програмирања подразумева дуг процес размишљања о задатку. У том процесу решавања датог проблема користимо познате методе и правила или сами одређујемо поступке решења. Када се заврши мисаони процес решавања проблема, програм је готов. Сада је процес записивања програма само нужна формалност коју и не мора да обави сам програмер. Довољно је да се идеја решења запише у погодном облику да би на основу тог записа нека, за то квалификована особа, кодирала програм - уписала га у меморију рачунара и извршила. Да би се идејно решење проблема могло записати тако да га свако за то квалификован, може разумети морају се поштовати нека прописана и формализована правила понашања. Тај коначан скуп строго формулисаних правила којима је одређен низ радњи за решавање одређеног проблема чини алгоритам.

Алгоритам је у математику увео арапски математичар Мухамед Ал Хорезми (***Abu Abdullah Muhammad bin Musa al-Khwarizmi*** живео око 825. године нове ере). Он је написао књигу ***Ал Хорезми о индијској вештини рачунања*** где у арапску математику уводи индијске цифре и децимални бројни систем. У књизи су прецизно, корак по корак, дефинисане све четири основне математичке операције (сабирање, одузимање, множење и дељење) у облику упутстава која се састоје од елементарних корака. Када је касније књига преведена на латински добила је назив ***Algoritmi de numero indorum***. Од овог лошег латинског превода презимена арапског математичара и потиче сама реч алгоритам, која је дуго затим означавала поступак за рачунање са децималним бројним системом (и индијским, односно, арапским цифрама) и прецизан начин описивања пута до решења проблема у математици. Даљим развојем науке, алгоритам је постао један од основних појмова како у математици тако и у рачунарству.

Алгоритам је коначан низ једноставних корака којим се описује поступак решавања одређеног реалног проблема, али такав да се, после коначно много времена, поступак завршава. Многе реалне ситуације у животу одвијају се по тачно одређеним корацима тако да можемо рећи да су алгоритми саставни део свакодневног живота. Постоји релативно прецизан поступак по коме покрећемо аутомобил, телефонирамо, прелазимо улицу, кувамо кафу... Чак, можемо рећи да се и сам живот одвија по неким, мање или више строгим, правилима. Добар пример алгоритма је сваки кулинарски рецепт; у њему јасно разликујемо шта је улаз (потребни састојци), шта је обрада (прецизно описан поступак за кување) и шта је излаз (готово јело). Алгоритам можемо у најопштијим цртама приказати шемом:

Постоји много начина да се алгоритам прецизније дефинише, али за нас је довољна и једна оваква, не престрога, дефиниција:

Алгоритам је повезани низ елементарних правила, односно, алгоритамских корака у којима се улазне величине поступно трансформишу све док се не добије коначно решење.

Сваки алгоритам красе следеће карактеристике:

- ***одређеност*** - сваки алгоритамски корак мора да буде прецизно и једнозначно дефинисан,
- ***детерминисаност*** - вредност која се добија после извршавања сваког корака једнозначно је одређена вредностима из претходног корака
- ***коначност*** - алгоритам се мора завршити после коначно много алгоритамских корака,
- ***масовност*** - алгоритам се прави тако да се може применити за решавање читаве групе сродних проблема без обзира на различите полазне величине,
- ***ефикасност*** - алгоритам треба да доведе до решења за што краће време и у што мањем броју алгоритамских корака.

Алгоритам је употребљив ако се његовом применом добија резултат у коначном „разумном" времену. Алгоритам који би бирао потез играча шаха тако што ће испитати све могуће последице потеза, захтевао би милијарде година на најбржем рачунару. Поступци израде алгоритама нису једнозначни, иначе би постојали генератори алгоритама који сами решавају све проблеме. Самим тим, алгоритмизација проблема захтева и велику дозу креативности. У пракси алгоритми се на рачунару представљају изворним кодом (***source***) неког вишег програмског језика. Програмски језици имају веома строга правила писања инструкција и дефинисања података која, код различитих програмских језика могу бити мање или више слична, али и битно различита.

Да би се логичко одвијање процеса решавања проблема лакше сагледало, алгоритам се представља дијаграмом тока, тј. графичком алгоритамском шемом.

Према форми веза у шеми алгоритми могу бити:

- ***прост линијски алгоритам*** - сваки алгоритамски корак се остварује тачно једном од горе на доле,
- ***разгранати линијски алгоритам*** - сваки алгоритамски корак се остварује највише једном, тј. има и корака који се неће остваривати за дате полазне величине,
- ***циклични алгоритам*** - постоје алгоритамски кораци који се остварују више пута,
- ***селективни алгоритам*** - у зависности од датог критеријума бирају се следећи алгоритамски кораци.

Међутим, код решавања неког проблема, програмер нема пред собом обавезу да направи један од строго одређених типова алгоритама. Најчешће се дешава да се за решење проблема морају употребити различите, готово све поменуте структуре алгоритама у појединим сегментима програма, зато ову поделу алгоритама схватамо само као теоријску.

Трансформација проблема на облик погодан за решавање на рачунару

Да бисмо решили неки проблем коришћењем рачунара, неопходно је креирати програм у неком програмском језику. Тај процес је комплексан и у њему можемо издвојити следеће фазе:

Поставка проблема. Проблем који се решава треба прецизно и потпуно описати на природном језику. То ради корисник коме је намењен програм. Да не би дошло до грешке при формулисању захтева пожељно је да наручилац буде рачунарски описмењен што неретко није случај, зато се већ у овом кораку мора остварити јака сарадња са програмерима. Јасно се одређује циљ и основна намена програма, анализирају се постојеће информације и врши се избор неопходних почетних информација, описују се сви излазни подаци.

Анализа проблема. У овој фази прецизно се дефинишу улазни и излазни подаци, ограничења њихових вредности и дефинишу све везе између података. У овој фази треба детаљно поставити задатак, уочити и упознати све елементе значајне за решавање задатка. Потребно је критички оценити значај свих елемената и изабрати оне који су битни за решавање постављеног задатка. Дефинисање проблема је у многим случајевима веома једноставно, међутим, често се догађа да је пут до дефиниције проблема веома дуг. Од тога како је проблем дефинисан веома зависи какве ће се тешкоће појавити у конструисању алгоритма и каква ће бити употребљивост програма. Резултат ове фазе је формалан опис проблема, израда математичког модела који се може реализовати на рачунару који ће представљати скелет програма. Ту спада, на пример, избор математичке методе која је примерена датом проблему. Приликом избора методе треба имати на уму да проблем решавамо на рачунару, а не ручно. Тако се, имајући у виду велику брзину рачунара у извршавању неких врста операција, можемо радије одлучити за поступак који захтева велики број врло простијих операција, него за елегантнију и бржу методу која се састоји од *интелигентнијих* операција.

Разрада алгоритма. У овој фази се дефинише начин на који се од почетних, улазних података долази до тражених, излазних резултата. Већина проблема може се решити на више начина, зато треба проанализирати све начине и изабрати онај који је оптималан. Сложеност решења проблема можемо посматрати са аспекта броја потребних операција (времена потребног за извршавање алгоритма) и са аспекта потребних меморијских ресурса да се алгоритам изврши (просторна сложеност алгоритма). Доказ о коректности изабраног поступка за решавање проблема неопходно је прецизно извести. О овим аспектима треба посебно водити рачуна при решавању сложених проблема, док их код једноставних проблема занемарујемо.

Пројектовање опште структуре програма. У овој фази врши се избор програмског језика. Разматра се цео проблем и дели се у логичке целине; свака целина се даље дели у зависности од њене сложености. Прецизно се дефинише начин чувања информација, тј. дефинише се структура података.

Кодирање. У овој фази описују се, у неком програмском језику, подаци и поступак решавања проблема, који су дефинисани у претходним фазама, то је запис алгоритма у облику прихватљивом за рачунар. Овако записан алгоритам назива се програм. Програм записан у неком програмском језику често се зове код, зато се ова фаза и назива кодирање. Програм може бити записан на машинском, симболичком или неком вишем програмском језику. Писање програма представља фазу реализације, јер програм оспособљава рачунар за решавање одређеног задатка. Ако смо у претходним фазама прецизно разрадили алгоритам, структуру програма и структуру података, ова фаза је релативно једноставна за извођење.

Фаза превођења, извршавања и тестирања програма. Подсетимо се да се на рачунару може извршавати само програм који је написан помоћу бинарних цифара (на машинском језику). Зато је потребно програм написан на програмском језику вишег нивоа превести на машински језик; превођење се врши помоћу посебних програма: компајлера, односно, интерпретера. Да би се програм извршавао, потребно је формирати извршну верзију (***exe*** верзију). То се постиже помоћу специфичних програмских алата који се разликују од језика до језика (повезивачи, билдери). Врло често су фаза превођења и фаза прављења извршног фајла обједињене. Тестирањем програма треба проверити да ли програм у потпуности решава постављени задатак. То је врло важна фаза у којој треба открити евентуалне скривене грешке. Ово је крупан проблем у програмирању. Обично се дешава да програм није потпуно исправан, па је потребно пронаћи грешке. На неке грешке нас упозорава рачунар тиме што за програм јавља дијагностичку поруку која садржи листу грешака. Када отклонимо те грешке то још увек

не значи да је програм исправан. Програм се тестира тако што се извршава за одабране скупове улазних величина. Примерима можемо показати да програм није исправан, али не можемо показати да је исправан. Иако је рачунар идеално средство за тестирање, исправност програма не можемо доказивати тако што ћемо програм извршити за сваки могући скуп улазних величина. За велики број програма овакво тестирање би трајало годинама. Зато се проверавање врши са подацима за тестирање које треба одабрати тако да се без већих напора могу предвидети исправни резултати и којима треба да обухватимо све могуће ситуације, свакако и оне кад постављени проблем нема решења. Ако је програм и даље неисправан на то ће нас упозорити неочекивани резултати. Овај поступак може бити веома дуготрајан и да од програмера захтева много спретности и досетљивости. По потреби се враћамо на претходне фазе.

Израда документације. У документацији програма описује се шта програм ради (поставка задатка и алгоритам за његово решавање), дају се упутства како се користи (начин задавања улазних величина и издавања резултата), детаљније се описује алгоритам и објашњавају поједине формуле и поступци када је у питању решавање сложенијих проблема и слично. Документацију никако не треба започињати када је програм написан, већ је треба писати током израде програма, али је свакако треба допунити на крају када је програм комплетно завршен. Детаљна документација о програму омогућава лако коришћење програма, као и евентуалне касније промене у програму.

Одржавање програма. Током коришћења програма могу се уочити неке грешке, које се морају отклонити. Понекад треба извршити неке промене у програму услед нових околности. Ако је програм читко написан и ако има добру документацију, онда је ова фаза једноставна, како за саме ауторе програма, тако и за друге програмере. У супротном, многим програмерима је лакше да напишу нови програм него да преправљају већ постојећи. У овој фази долази и до проширивања програма додавањем нових функционалности. Када се развија велики пројекат, све претходно наведене фазе су једнако важне. Оспособљавање програмера за успешно реализовање великих пројеката је дуготрајан процес. У едукацији полазимо од једноставних примера ка сложенијим. У овом процесу проблеми које решавамо су једноставни, па су све фазе, осим фазе разраде алгоритма, једноставне.

Програмски језици и њихова синтакса и семантика.

Језик је средство комуницирања у процесу размене и обраде информација и података. Постоје природни и вештачки језици. Природни језици су језици људи. Иако су врло богати могућностима изражавања, ови језици су практично неупотребљиви у комуникацији на релацијама човек - машина или машина - машина, бар у данашњим условима. Да би се остварила оваква комуникација приступило се конструкцији вештачких језика. Вештачки језици су једнако приступачни и човеку и машини. У зависности од области и начина примене, конструисани су бројни вештачки језици. Програмски језици су подврста ових језика. Програмски језици су вештачки језици одређени помоћу изабраног скупа симбола. У њима се јављају и неки специфични знаци у односу на језичке конструкције природних језика. *Елементарне конструкције програмског језика* имају одређена значења, али се у програму никада не јављају као самостални елементи који утичу на рад рачунара. То су *константе, подаци, променљиве, низови* и *изрази*. Повезани низ симбола и елементарних конструкција који има одређено значење, а може да представља самосталну целину у програму, односно, може да проузрокује акцију на рачунару је *сложена конструкција програмског језика*. Такве конструкције су *наредбе, програмски редови, потпрограми* и *програми*.

Правила помоћу којих се формулишу елементарне и сложене конструкције програмског језика чине његову *граматику*. Правила помоћу којих се гради и проверава коректност језичких конструкција чине *синтаксу* програмског језика. Програмски језици су тако конструисани да рачунар може да уочава и упозорава на синтаксне, формалне грешке. Значење појединих језичких конструкција проучава *семантика*. Састављање програма, исписивање програмских инструкција, тј. кодирање програма врши се на основу алгоритма. Семантичке грешке настају као последица грешке у алгоритму. Ову врсту грешака рачунар не може уочити.

Синтакса се односи на начине на који појединачни симболи могу да креирају исправне реченице језика (или програме). Синтакса дефинише формалне релације између елемената

језика, тиме пружајући структурне описе различитих израза који чине исправне ниске језика. Синтакса се бави само формом и структуром симбола језика без било каквих разматрања у вези са њиховим значењем. За разлику од природних језика, сваки програмски језик има скуп строго дефинисаних правила којима се описује поступак писања програма. Пошто се ради о језику за комуникацију са рачунаром и најмања двосмисленост доводи до грешака. Синтакса програмског језика се описује помоћу метајезика. Синтаксне дефиниције се често задају помоћу специјалне металингвистичке симболике која је позната под именом Проширени Бекус-Науров запис (***Extended Backus- Naur Form***) или скраћено ***EBNF***. ***EBNF*** омогућава врло прецизан и компактан опис елементарних језика, коришћењем неколико симбола специјалне намене – метасимбола. Најпростији појмови су симболи и резервисане речи језика. Они се називају терминални симболи или терминали и пишу се између наводника. Сви остали појмови, нетерминали, се постепено изводе из терминалних помоћу специјалних металингвистичких формула и стављају се између угластих заграда. За опис синтаксе језика, због веће прегледности, користе се и синтаксни дијаграми. Они се састоје из кружића, овалних симбола и правоугаоника повезаних у одређеном поретку. Кружићи или овали означавају терминале. Правоугаоници означавају конструкције које се накнадно дефинишу (нетерминале), тј. које се ослањају на друге синтаксне дијаграме. Између ***EBNF*** и синтаксних дијаграма постоји веза која омогућава представљање појмова који су записани у ***EBNF*** преко синтаксних дијаграма и обрнуто.

Семантика придружује значење синтаксно исправним нискама језика. За природне језике, ово значи повезивање реченица са неким специфичним објектима, мислима и осећањима. За програмске језике, семантика описује понашање рачунара током извршавања програма написаног на неком језику. Ово понашање може се описати релацијама између улаза и излаза програма или корак-по-корак објашњењем како ће се програм извршавати на стварној или апстрактној машини.

Класификација програмских језика

Према степену зависности од рачунара програмски језици се деле на:

- машински зависне и
- машински независне језике.

Машински зависни језици се деле на:

- машинске и
- машински оријентисане језике.

Машински језици су језици најнижег нивоа. По структури су најближи машини. Толико су зависни од типа рачунара за који су прављени да се не могу применити на било који други тип рачунара. Главна предност ових језика у односу на све друге је брзина извршавања програма. Нема губљења времена на било каква превођења. Оног тренутка када је програм у меморији рачунара он се и извршава. Зато се овај језик користио за писање неопходних системских програма.

Машински оријентисани језици се деле на:

- симболичке и
- макро језике.

У *симболичким језицима* се поједине групе симбола машинског језика замењују мнемоничким симболом, па су зато разумљивији, али још јако зависе од рачунара за који су прављени. Брзина извршења програма се смањила, у односу на машинске језике, јер се програм најпре преводи на машински језик, па тек онда извршава. Међутим, још се ту ради о завидној брзини извршења програма. Због тога се ови језици користе за писање неких системских програма.

Макро језици су највиши ниво машински зависних језика. Код ових језика се група команди симболичког језика замењује једном макро командом. Зато се програми на макро језицима пишу много једноставније него на симболичким, разумљивији су и краћи, али још увек у великој мери зависе од рачунара за који су прављени. Ови језици се, углавном, користе за писање системских програма.

Машински независни језици се деле на:

- процедуралне језике и
- проблемски оријентисане језике.

Процедурални језици подразумевају најпре алгоритамско решавање проблема, а затим формално исписивање програма. Програми писани овим језицима, релативно једноставно, могу да се пренесу са једног типа рачунара на други.

Проблемски оријентисани језици су језици највишег нивоа. Код оваквих језика програмер треба само да назначи проблем и пут за његово решавање, а програм генератор даље преузима посао формирања алгоритамског решења проблема и формалног исписа програма. Због овакве своје структуре, ови језици су за сада незграпни, троше пуно времена и меморије и прилично су спори. Ипак, то су језици будућности и са даљим развојем науке и технике ће се све више усавршавати.

Пошто је написан неки програм, природно је да се он изврши. Да би се неки програм могао извршити, све наредбе се морају превести на машински језик. Само се програми на машинском језику могу извршити непосредно, без превођења. Посао превођења наредби неког програмског језика на машински језик обављају преводиоци. То су посебни програми који повезују програмске конструкције језика са командама машинског језика. Према нивоу језика са кога се програм преводи преводиоце делимо на:

- асемблере,
- макроасемблере,
- компилаторе и
- генераторе.

Симболички језици су асемблери, макро језици су макроасемблери, процедурално орјентисани језици су компилатори, а проблемски орјентисани језици су генератори.

Процес превођења и процес извршавања програма су два битна процеса са којима се сусрећемо приликом рада на неком програму. Они могу бити временски раздвојени или повезани. Према тој временској карактеристици преводиоци се деле на компајлере и интерпретаторе. Код компајлера процес превођења наредби програма је потпуно временски раздвојен од процеса извршења. Код интерпретатора се у процесу превођења свака преведена наредба изврши, а затим се прелази на следећу наредбу.

Компајлерског типа су програмски језици:

- ***fortran*** **- *FORmula TRANslation system***, намењен је пре свега решавању проблема у области математичких наука и технике; први важнији алгоритамски језик у историји програмирања; дизајнирао га је тим програмера америчке компаније ИБМ, 1957. године, на чијем је челу био Бекус (***John Backus***); дизајниран је са идејом да служи потребама научника и научних израчунавања са реалним бројевима (бројевима са покретним зарезом) као и скуповима реалних бројева организованих у један или више низова;
- ***cobol*** **- *COmmon Business Oriented Language***, један од најстаријих виших програмских језика који је и данас у употреби; углавном служи за писање бизнис и финансијских апликација и као подршка административним системима у компанијама и владама; најновија спецификација, кобол 2002, садржи механизме објектно-оријентисаног програмирања и многе друге модерне могућности;
- ***algol*** **- *ALGOritmic Language***, за решавање математичких и научно-техничких проблема; дизајнирао га је комитет америчких и европских научника рачунарства за сврху објављивања алгоритама, али и за рачунарска израчунавања, између 1958. и 1960. године; алгол поседује рекурзивне потпрограме, односно, процедуре које могу саме себе позивати приликом решавања задатог проблема, редукујући га на мањи проблем било које врсте; новост у алголу је блок структура, програм је компонован од блокова и може да садржи и податке и инструкције које имају исту структуру као и сам програм; блок структура је врло брзо постала стандард за конструисање масивних програма од малих компоненти;
- ***lisp*** - ***LISt Processing***, прилагођен симболичкој обради; развио га је и имплементирао Макарти (***John McCarthy***) око 1960. године, базирајући га на математичкој теорији рекурзивних функција; програм развијен у лиспу је функција примењена на податке, а не низ процедуралних корака, као што је случај у фортрану и алголу;
- ***pl/1*** **- *Program Language no. 1***, развијен је са намером да замени алгол, фортран и кобол, али није испунио очекивања, тако да није применљив у свим областима рада;
- ***c*** - универзални програмски језик; развили су га Ричи (***Dennis MacAlistair Ritchie***) и Керниган (***Brian Wilson Kernighan***) 1972. године, у AT&T корпорацији за програмирање оперативних система; у данашње време је један од популарнијих због своје једноставности, природности и широке применљивости; неки системски програми пишу се у њему;

- ***ada*** - универзални, вишенаменски програмски језик високог нивоа који је заснован на Паскалу, развијен по налогу америчког Министарства одбране, крајем седамдесетих година 20. века, са циљем да буде примарни језик овог Министарства и ту се углавном и користи, име добио по Ади (Ејди) Аугусти Бајрон;
- ***gps - General Problem Solver***, прилагођен моделирању; развили су га Невил (***Allen Newell***) и Сајмон (***Herbert Alexander Simon***) 1957. године у покушају да имитирају људски начин размишљања;
- ***simula*** - применљив у моделирању, настао је 60-их године у Норвешкој, развили су га Дал (***Ole-Johan Dahl***) и Најгард (***Kristen Nygaard***); синтаксно је врло сличан алголу; веома је утицао на имплементацију неких објеката у програмским језицима ***C++***, ***Java*** и ***C#***;
- ***pascal*** - развио га је професор Никлаус Вирт (***Niclaus Wirth***), као језик погодан за учење структурног програмирања; именован је по чувеном француском математичару и филозофу Блезу Паскалу (***Blaise Pascal***), творцу прве рачунске машине која је имала могућност извођења операције сабирања; паскал је стандардизован 1983. године од стране Међународног комитета за стандардизацију; данас се користи као почетни програмски језик за обуку будућих програмера, наручито оних који намеравају да раде на програмском језику ***C***

и много других мање или више познатих.

Интерпретаторског типа су програмски језици:

- ***basic - Beginners All-purpose Symbolic Instruction Code***, универзални, широко распрострањени програмски језик; то је назив за више програмсих језика чија је заједничка особина да су пројектовани са намером да буду једноставни за коришћење; првобитна верзија овог програмског језика је развијена 1964. године под руководством Кемењиа (***John George Kemeny***) и Керца (***Thomas Eugene Kurtz***). Имплементиран је за рачунаре серије ***G.E.225*** и за велики број некада популарних кућних рачунара био је у склопу оперативног система; план је био да то буде једноставан језик за учење програмирања, да буде почетни корак за студенте који треба да савладају моћније језике као што су фортран или алгол; један од недостатака бејзика је непостојање стандарда, па програми писани за један тип рачунар не раде на другим;
- ***prolog*** - ***PROgramming LOGic***, прву верзију овог програмског језика развио је 1971. године Колмерер (***Alain Colmerauer***); намењен симболичкој обради са елементима вештачке интелигенције; 1977. године Ворен (***David H. D. Warren***) је направио први интерпретатор и компајлер за пролог; јапански научници су почетком осамдесетих година најављивали да ће њихови рачунари пете генерације бити засновани на овом програмском језику; пролог је замишљен као машински језик јапанских рачунара пете генерације; од тада је написано много експертних система на њему; уз лисп, ово је најпознатији језик вештачке интелигенције
- ***planer*** - је дизајнирао Хјуит (***Carl Hewitt***) 1969. године; намењен симболичкој обради са елементима вештачке интелигенције;

и други. Неки програмски језици се јављају у обе варијанте (на пример бејзик).

Најпознатији проблемски оријентисани језици су:

- ***RPG*** - ***Report Program Generator***, развијен је у ***IBM***, 1959. године за пословне апликације са намером да замени и истисне ***Cobol*** и ***PL/1***, постоје верзије за мејнфрејм и микрорачунаре; ово је језик врло високог нивоа;
- ***COGO*** - ***Civil Engineering Coordinate Geometry***
- ***STRESS*** - ***STRuctural Engineering System Solver***
- ***HYDRO*** - ***Hydraulic Engineering Computations***
- ***Mars, Arius, Proza, Graph*** и други.

Програмски језик паскал

Шездесетих година двадесетог века развија се велики број нових програмских језика, а најпопуларнији је *алгол 60*. Као одговор на све веће проблеме у вези са пројектовањем и одржавањем великих програмских система, до тада коришћеним техникама и језицима за програмирање, јавила се жеља да се направи такав универзални језик који би покупио све добре особине свих до тада познатих језика, а отклонио све њихове слабости, језик који би био применљив у свим областима људске делатности. Зато међународна федерација за обраду информација (***IFIP***) 1965. године образује радну групу ***WG2.1*** чији је задатак да на основу

програмског језика алгол 60 начини језик који би био задовољавајући у сваком погледу. На самом почетку заједничког рада радна група се распала на два табора, јер је дошло до неслагања у погледу сложености новог програмског језика. Из редова бројнијег табора изникла је нова верзија програмског језика алгол позната под именом алгол 68 (1968. године је објављена прва верзија овог језика). Остатак групе је, нешто касније, 1969/70. године представио потпуно нов програмски језик који је назван паскал по презимену француског математичара, физичара и филозофа Паскала (***Blaise Pascal***, 1623-1662.) који је један од првих конструктора механичких машина за сабирање. Конструктор програмског језика паскал је професор Високе техничке школе у Цириху (***Swiss Federal Institute of Technology***) Никлаус Вирт (***Niclaus Wirth***), поборник структурираног програмирања.

Структурирано програмирање је скуп техника за израду програма са јасном и лако разумљивом структуром, уз коришћење података са јасно дефинисаним структурама. Овај процес карактерише модуларност и техника писања програма одозго на доле.

- ***Модуларност*** значи да се у оквиру програма разликују одговарајуће целине, које се даље могу разлагати на мање и тако даље све до елементарних целина. Свака целина се може представити у облику потпрограма. На овај начин се један велики, компликовани програм може разделити на неколико релативно независних целина - модула. Сваки од модула је релативно лако анализирати и открити евентуалне грешке. На овај начин се избегавају бескрајна проверавања програма за различите полазне податке.

- ***Програмирање одозго на доле*** подразумева писање програма од општег ка појединачном. Проблем се рашчлани на потпроблеме и сваки од њих се решава као независни проблем. За сваки се пише одговарајући потпрограм који се, затим, повезују у структурни програм.

Програмски језик паскал подржава ове принципе свим својим основним управљачким структурама (селекције и циклуси) и структурираним типовима података (низови, записи, скупови, датотеке). За њега постоји и међународни (***ISO***) стандард. Са појавом паскала остварила се идеја структурираног програмирања, али и формалног доказивања коректности програма. По својој универзалности, једноставности и разумљивости паскал је далеко испред свих до тада познатих програмских језика. Паскал краси и изражајност, преносивост и језгровитост. Популарност коју је стекао не опада ни до данашњих дана. Готово да нема области делатности где се паскал не може применити и где се не примењује.

Стандардна верзија паскала је званично објављена 1979. године. То је језик са једноставном и логичном структуром и због тога се доста лако учи. Врло је погодан за почетнике, за савлађивање основних принципа програмирања, као и за усвајање организованог и разумљивог начина размишљања и изражавања. Стандардни паскал има и неколико озбиљних мана које га чине неподесним за писање великих програмских система. Пре свега, захтева се да целокупан текст изворног програма, укључујући и све потпрограме, буде у једној датотеци. Основна идеја је да се преводиоцу омогући да проверава исправност позивања потпрограма. Једна од најнепријатнијих врста грешака у програмима, које се и врло тешко проналазе у сложеним програмским системима, управо је неслагање типова стварних аргумената потпрограма на местима позивања са типовима формалних аргумената у дефиницији потпрограма. Поред тога, ова мера требала је и да онемогући *паметним* програмерима да свесно наведу стварне аргументе другачијих типова од очекиваних, постижући тиме неке специјалне, нестандардне (и сумњиве) ефекте. Такви програми се косе са једним од основних принципа структурираног програмирања - тешко их је разумети. Друга мана су врло ограничене могућности рада са датотекама. Стандардни паскал предвиђа само секвенцијалне, датотеке са секвенцијалним приступом. Релативне датотеке са директним приступом не постоје, а још мање индексне датотеке које би омогућавале приступ помоћу кључа. Да ситуација буде још гора, имена датотека које програм треба да обрађује, не могу да се бирају у току извршавања програма, нити су уграђена у сам програм. Датотеке се у програмима представљају симболичким именима и препуштено је оперативном систему рачунара на који ће начин та имена повезати са физичким датотекама. На избор датотека корисник, евентуално, може да утиче командама оперативног система пре почетка извршавања програма. Трећа мана је нефлексибилност потпрограма при раду са низовима као аргументима. Бројеви елемената низова приликом сваког позивања морају да буду исти као бројеви елемената предвиђених у дефиницији потпрограма. Чему правити засебан потпрограм за сортирање низова од по 100 елемената као и засебан за оне од по 500 елемената? Стандардни паскал, такође, нема ни могућност графичког приказивања резултата.

Поред стандардне верзије паскала у то време се користе и турбо верзије. Програмски језик **Turbo Pascal** производ софтверске куће **Borland** појавио се на персоналним рачунарима под оперативним системом **DOS** (*Disk Operating System*) са извесним бројем нестандардних

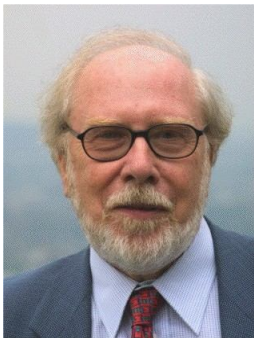
проширења. Нека од тих проширења обезбеђују боље искоришћавање хардверских могућности рачунара коме је језик намењен, док друга отклањају недостатке стандардног паскала. За боље искоришћавање хардверских могућности рачунара уведено је неколико формата целобројних података и неколико формата реалних података, који се међусобно разликују по опсегу вредности и по прецизности израчунавања (стандардни паскал предвиђа само једну врсту целих и једну врсту реалних бројева). Уведени су и специјални оператори за манипулисање са битовима унутар целобројних података. Потреба за писањем текста целокупног програмског система у једној датотеци отклоњена је увођењем програмских модула. Ти модули се преводе одвојено и називају се јединицама превођења (*compilation units*). Свака датотека програмског текста је један модул и дели се на део за везу са осталим делом система и део за дефинисање садржаја потпрограма који се налазе у њој. У првом делу потребно је описати све потпрограме (тип резултата и типове аргумената) и све глобалне променљиве, који се користе и изван те јединице. У току превођења делова програма у којима се користе ти елементи, преводилац ће да консултује делове за везу ради провере исправности коришћења тих потпрограма и глобалних података. Мора се признати да се преводилац може преварити вештим манипулацијама, али знатно теже него што је то био случај у програмским језицима који су се користили пре појаве паскала. Преводиоцу је потребно подметнути датотеку са лажним делом за везу, што у стандардном паскалу није могуће. Рад са датотекама је, такође, побољшан. Уведена је могућност придруживања симболичких имена физичким датотекама у току извршавања програма. Наравно, то уводи елемент зависности програма од рачунара и од оперативног система, детаљ који се желео избећи у стандардном паскалу. Поред тога, уведена је и могућност директног приступања појединим записима у датотекама, без чега не постоји ефикасна обрада датотека. Са појавом верзије 3.0 делимично је решен и недостатак графичког приказа увођењем команди *Turtlegraphic* групе. То јесте била нека врста графике, али није задовољавала захтеве. Зато се у верзијама 4.0 и даљим, уз паскал испоручује и све комплетнији графички пакет ***Graph.tpu***. Актуелна верзија 7.0 уводи могућност рада са мишем, са више програма истовремено, објектно програмирање, рад са прозорима, програмирање у оквирима оперативног система *Windows* и друго. Свака од до сада објављених седам верзија турбо паскала је донела неке новине у односу на претходну, отклањање уочених грешака и увођење нових могућности у језик. У најзначајнија проширења, свакако, спадају увођење подршке за објектно оријентисано програмирање (ООП), почев од верзије 5.5 и подршке за израду атрактивних спрега апликација са корисником, заснованих на прозорима и менијима (*Turbo Vision*) при раду под оперативним системом ДОС.

Објектно оријентисано програмирање је нова методологија израде великих програмских система која се појавила када су програмски системи толико нарасли да технике структурираног програмирања више нису давале задовољавајуће резултате. Док је у центру пажње структурираног програмирања структура програма, у центру пажње објектно оријентисаног програмирања су објекти. Објекти су *интелигентни* подаци који могу да се налазе у одређеним стањима. Та стања могу да се промене применом одређених метода. Скуп објеката са истим особинама чини једну класу. Класе су аналогне типовима података, као што су цели бројеви или реални бројеви, али могу да представљају произвољно сложене апстрактне објекте (на пример геометријске фигуре).

Појава програмског језика паскал и велика популарност коју је врло брзо стекао оставила је дубоке трагове на даљи развој програмских језика. Наиме, из њега су се развили: ***Mesa, Concurrent Pascal, Euclid, Gypsy, PLZ, Modula, Telos, Edison, Ada*** и многи други.

*Аугуста Ада Бајрон, гроfица од Лавлејса (**Augusta Ada (Byron) King, Countess of Lovelace**) је кћи Ана Изабеле Милбанке (**Anna Isabella Milbanke**) и Џорџа Гордона Ноел Бајрона (**George Gordon Noel Byron**), рођена је 10. децембра 1815. године у Лондону, Енглеска. Ада своја прва знања из области математике добија од Августа Де Моргана (**Augustus DeMorgan**, 1806-1871.). Адина мајка је хтела да пробуди интерес за математиком и охрабри рационалне аспекте њеног карактера насупрот романтичним утицајима њеног оца. Када је имала 18 година, Ада је слушала предавање о диференцијалној машини коју је дизајнирао Чарлс Бебиџ (**Charles Babbage**, 1791-1871.). Почевши писмом од 18. јануара 1836., па све до своје смрти, Ада је водила дуготрајну кореспондецију са Бебиџом. У октобру 1842, Луиђи Федерико Менабреа (**Luigi Federico Menabrea**, 1809-1896.), италијански професор на Универзитету машинства у Торину објавио је рад којим описује функцију и теорију Бебиџове*

*аналитичке машине. Ада је превела овај рад са француског на енглески и додала сопствене опсежне коментаре. Превод је објављен 1843. године. И сам Бебиц био је импресиониран неким компликованим програмима које је Ада написала, најсложенији од њих био је одређивање низа Бернулијевих бројева. Тако је Ада стекла репутацију првог компјутерског програмера (иако су и Бебиц и неки други, такође, написали бројне програме за никад завршену аналитичку машину). Оптерећена коцкарским дуговима и раком материце, Ада је умрла у 37. година, 27. новембра 1852. године, а сахрањена је, на лични захтев, поред свога оца у породичној гробници у цркви Свете Марије Магдалене у Хакналу (**Church of St. Mary Magdalene in Hucknall, Nottingham**).*



*Никлаус Емил Вирт **(Niklaus Emil Wirth**), рођен 15. фебруара 1934. године у Винтертуру (**Winterthur**) у Швајцарској је швајцарски информатичар. Звање инжењера електронике је стекао на Швајцарском универзитету технологије у Цириху 1959. а докторирао је на Калифорнијском универзитету Беркли 1963. Сматра се изузетним стручњаком из области рачунарства, а најпознатији је као творац програмског језика Паскал. Осим паскала пројектовао је и програмске језике **Euler, Algol W, Pascal, Modula, Modula-2, Oberon, Oberon-2, Oberon-07**. За свој рад је 1984. године добио Тјурингову награду. Предавао је на универзитету Станфорд на катедри за рачунарске науке (1963 - 1967.) као и на Швајцарском технолошком универзитету. 1975. године је написао књигу „Алгоритми + структуре података = програми“ (**Algorithms + Data Structures = Programs**). Од априла 1999. је у пензији.*

*"Шта је човек у васиони, између две бесконачности: бескрајно великог и бескрајно малог? Ништавило у погледу на бескрајност; све у погледу на ништавило; средина између ничега и свег." - **Блез Паскал**.*

*Блез Паскал (**Blaise Pascal**, 1623 - 1662.) је био француски математичар, физичар и филозоф. Још од малена показивао је интересовање за науку, па је већ са 18 година конструисао прву математичку машину, механички сабирач како би помогао свом оцу у пословању. 1650. године напушта свет науке и окреће се религији, односно како је он написао „разматрању величине и мистерије човека“. Блез Паскал рођен је 19. јуна у Клермон Ферану (**Clermont-Ferrand**), у Француској. као треће дете Етјена Паскала (**Etienne Pascal**) и Антоанете Бегон (**Antoinette Begon**). Мајка му је умрла када је имао само три године, остављајући га са две сестре – Гилберт и Жаклин. Године 1631. породица Паскал напушта Клермон и сели се у Париз. Децембра 1639. године, Паскалова породица је напустила Париз да би живела у Руану где је Етјен био примљен као скупљач пореза за горњу Нормандију. Брзо по смештању у Руану, Блез је написао свој први рад назван Есеј о конусним пресецима, издат фебруара 1640. године. Исте године Паскал је изумео и први дигитални калкулатор са намером да помогне своме оцу у прикупљању пореза и такси. Догађаји 1646. године су били врло значајни за младог Паскала. Те године је његов отац повредио ногу и морао је да се опоравља код куће. О њему су се бринула његова млађа браћа, који су били у религиозном покрету из Руана. Они су имали дубок утицај на Блеза и он је постао јако религиозан. Етјен Паскал је умро у септембру 1651. године. После очеве смрти, Блез је писао једној од својих сестара, дајући, при томе смрти једно дубоко хришћанско значење; и за њега је очева смрт била нешто посебно. Тада је оформио своје идеје које су послужиле као основа за његова писма, обједињена у филозофски рад под називом Мисли (**Пенсеес**). После овога, Паскал је посетио јансенски манастир Порт Ројал де Шамп, који се налазио око 30 км југозападно од Париза. Почео је да издаје анонимна дела на тему религије. 18 провинцијалних писама је било издато у периоду између 1656. године до почетка 1657. године. Све је то било писано као знак одбране од његовог пријатеља Антоана и великог противника Језуита и браниоца јансенизма, који је иначе пре својих студија теологије у Паризу, био осуђиван због свог контраверзног религиозног рада. Паскалов најпознатији рад из филозофије је Мисли, а на издању о својим личним мислима везаним за људску патњу, судбину и Бога, којем је приступио касне 1656. године, наставио је да ради током 1657. и 1658. године. Овај рад садржи и Паскалову опкладу као доказ да је веровање у Бога разумно само са пратећим аргументима. Ако Бог не постоји, онај ко не верује у њега неће изгубити ништа, а ако, пак, Бог постоји, тај исти човек ће изгубити све зато што није веровао у њега. Паскал је у својој опклади користио математичке аргументе и аргументе из вероватноће, али његова главна реченица је: **...ми смо сви присиљени да се коцкамо**.*

Питања на која треба обратити пажњу

- Разлика између решавања проблема у животу и на рачунару
- Појам алгоритма
- Порекло имена појма алгоритам
- Дефиниција алгоритма
- Особине алгоритма
- Врсте алгоритама
- Фазе у решавању проблема на рачунару
- Појам програмског језика

- Граматика програмског језика
- Синтакса програмског језика
- Семантика програмског језика
- Основна подела програмских језика
- Машински језици
- Машински оријентисани језици
- Процедурални језици
- Проблемски оријентисани језици
- Подела преводилаца
- Подела програмских језика према односи процеса превођења и извршавања
- Фазе у извршавању програма
- Интерпретерски програмски језици
- Компајлерски програмски језици
- Структурно програмирање и особине
- Објектно оријентисано програмирање

Програми засновани на прозорима

Апликације су програмски системи пројектовани за решавање одређеног проблема. Поступак израде апликације назива се пројектовање, а предмет тог поступка пројектом апликације или скраћено, само пројектом. Другим речима, пројекат апликације представља апликацију у фази израде или пројектовања. Скица прозора у току пројектовања апликације назива се обрасцем (***form***). Скица главног прозора за време пројектовања је главни образац пројекта.

Основна одлика апликација заснованим на прозорима је комуникација апликације и корисника, заснована на прозорима и другим визуелним компонентама као што су: дугмад, оквири за текст, менији и слично. Помоћу тих компоненти корисник управља радом апликације и прима информације од апликације. Прозори могу да се схвате као оквири који обједињују компоненте у логичке целине.

Свака апликација има барем један прозор који се назива главним прозором, ***main window***. Осим њега може да постоји произвољан број других прозора, ***secondary windows***. Главни прозор је обично стално на екрану, док се секундарни појављују и нестају са екрана према тренутним потребама. Има случајева када се и неки од секундарних прозора дуже задржавају на екрану. По намени, специјалну врсту секундарних прозора чине прозори за дијалог, ***dialog box***. Користе се за саопштавање разних порука кориснику и за прикупљање различитих података од корисника. По завршетку дијалога увек нестају са екрана. Прозори за дијалог су специфични и по томе што корисник, све док их не затвори, не може ништа да уради ни у једном од других прозора дате апликације. Прозори са том особином називају се условљеним (***modal***) прозорима, јер је наставак рада условљен њиховим затварањем. У овом смислу, остали прозори називају се и неусловљеним (***modeless***) прозорима. Условљени прозори су везани за једну апликацију, тј. и у случају да је на екрану отворен условљени прозор једне апликације, увек може да се приђе прозору друге апликације и да се ради у њој.

Када постоји више прозора у апликацији, они се исцртавају један преко другога. Тренутно активан прозор види се у целини, док остали могу да буду делимично или у потпуности заклоњени, у зависности од њихових величина и положаја на екрану. Корисник може помоћу миша да помера независно све прозоре по екрану, мења им мења величине или да их уклони са екрана.

Постоји специјална врста прозора који се налазе унутар радног простора неког другог прозора. Ти прозори називају се децом, ***child window***, оних прозора у чијем се радном простору налазе, а који се, с друге стране, називају родитељима, ***parent window***, те деце. Прозори деца могу да се померају само унутар радног простора прозора родитеља. Померањем прозора родитеља по екрану померају се и његова деца. Променом величине прозора родитеља не мењају се величине прозора деце. Ако после промене величине прозора родитеља у радни простор не стану прозори деце, појавиће се клизачи уз ивице радног простора прозора родитеља, помоћу којих корисник невидљиве прозоре или њихове делове доводи у видно поље.

Основни начин за управљање апликацијама заснованим на прозорима је помоћу миша. Место на коме може да се делује мишем обележава се показивачем миша, ***mouse cursor***. Показивач миша може да има различите графичке представе. Основни облик је стрелица чији врх означава место показивача на екрану. Алтернативни облици могу да се користе за означавање тренутне улоге миша. На пример, изнад разних оквира за текст показивач миша обично има облик танке вертикалне линије са водоравним испустима на горњем и доњем крају. Такав облик је погодан да се означи место између два знака у тексту.

Постоје две основне радње које могу да се обаве помоћу миша:

- Притисак - ***click*** и
- Одвлачење - ***drag and drop***.

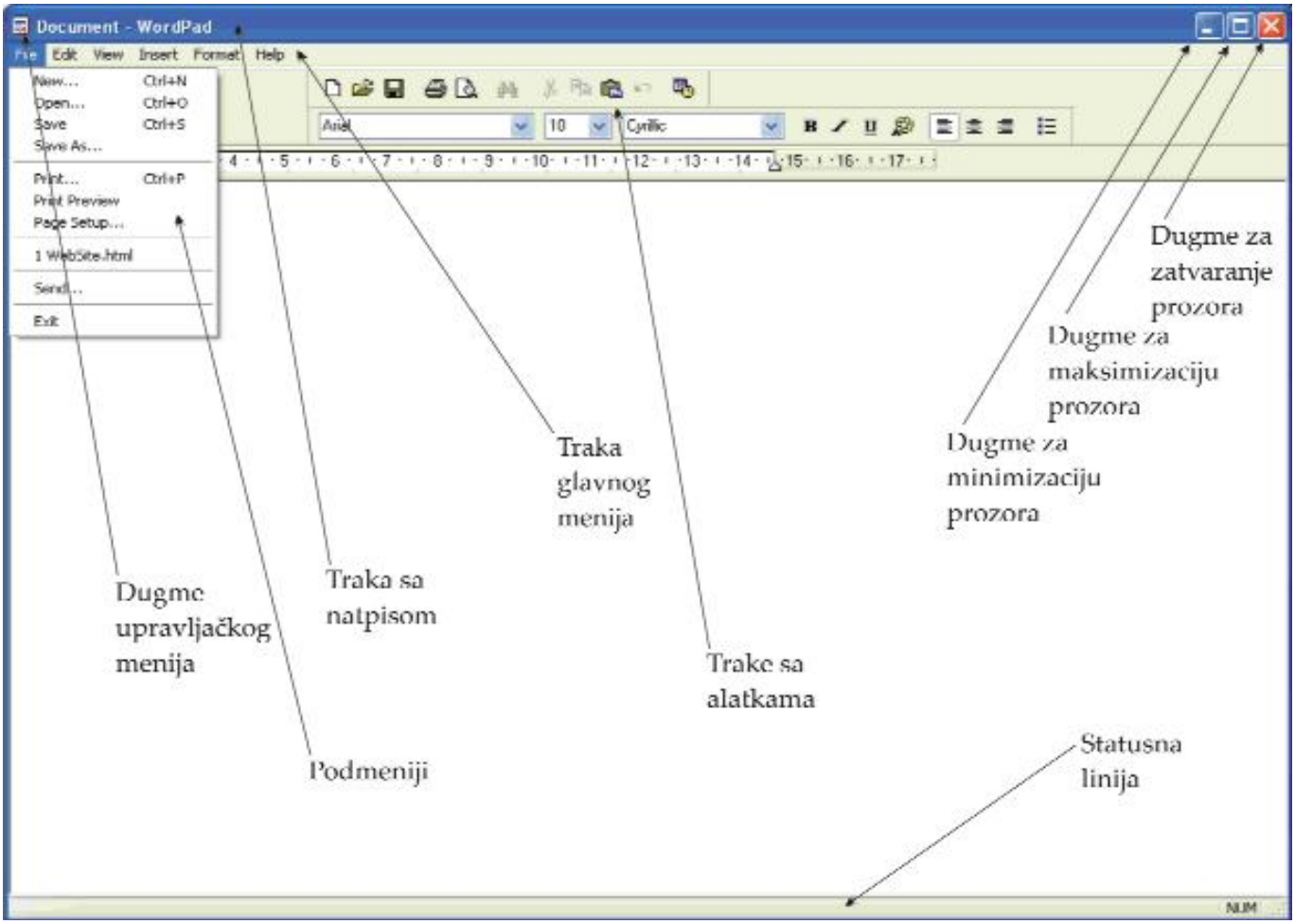
Под *притиском мишем* подразумева се померање показивача миша изнад одређене компоненте и притисак и отпуштање левог тастера на мишу. Тако се на екрану може притиснути неко дугме, одабрати нека ставка у менију, променити стање неког оквира за потврду и слично и тиме дати неки сигнал који ће се касније обрадити у апликацији.

Под *одвлачењем мишем* подразумева се померање прозора по екрану или промена величине прозора. Изводи се тако што се показивач миша доведе изнад натписа прозора, притисне се леви тастер миша, помера се показивач миша уз притиснути леви тастер на неко друго место и отпусти се тастер миша. Ако се на тај начин мишем ухвати ивица или угао прозора, померањем показивача помериће се ивица или угао са суседним ивицама, чиме се мења величина прозора. У зависности од апликације, на овај начин понекад могу да се померају и компоненте унутар прозора, да им се промени величина или да се постигну неки други ефекти.

Осим мишем, радом апликације може да се управља и преко тастатуре. Основна употреба тастатуре је уношење текста у оквире за текст. Место у тексту на коме може нешто да се промени обележава се показивачем текста, ***text cursor***. То је, обично, вертикална трептећа линија која се увек налази између два знака у тексту. Треба јасно разликовати показивач миша од показивача текста. Показивач миша може да се помера помоћу миша по целом екрану, док се показивач текста помера тастерима са стрелицама на тастатури само унутар текста. Притиском мишем на оквир за текст у некој тачки, показивач текста може да се премести на тренутни положај показивача миша. Под неким условима, и преко тастатуре могу да се производе ефекти притиска мишем. За то могу да се користе одређени тастери са словима у комбинацији са тастером ***Alt*** или ***Ctrl***, као и тастери ***Enter*** и ***Esc***. За компоненту која на овај начин може да се активира каже се да јој је придружен знак пречица (***ShortCut***) - ***accelerator character***. Знакови пречице приказују се подвучено у натписима компоненти којима су придружени у менијима, а често се дешава и да је на крају описа у менију исписана и комбинација тастера које треба *откуцати*, нарочито ако је пречица другачија од уобичајених или ако не постоји други начин да се учини видљивом кориснику апликације.

Основне карактеристике програма заснованих на прозорима

Слика приказује главни прозор апликације ***WordPad***, једне од стандардних апликација заснованих на прозорима. На њеном примеру ће бити објашњени елементи о којима треба водити рачуна приликом пројектовања неке апликације за рад у графичком оперативном окружењу.



Прозори апликације могу да имају следеће делове:

1. Трака са натписом - ***Caption bar***, налази се уз горњу ивицу прозора. Састоји се од натписа и управљачких дугмади. Натпис садржи име апликације и, евентуално, име датотеке која се тренутно обрађује. Четири могућа управљачка дугмета су:

- Дугме управљачког менија - ***control menu button***, налази се на левом крају траке са натписом и садржи сличицу - ***icon***, придружену прозору. Притиском на дугме отвара се управљачки мени.
- Дугме за минимизирање прозора - ***minimize***. Притиском на њега прозор се уклони са екрана при чему остане дугме које га представља у палети послова (***Taskbar***) у радном простору - (***Desktop***) виндоуса или се претвара у минимални облик (типично за секундарне прозоре за које обично не постоји придружено дугме у палети послова). Дугме у палети послова садржи сличицу која је придружена прозору и део натписа прозора. Притиском на то дугме левим тастером миша, прозор се поново појави на екрану у првобитној величини и на првобитном месту. Притиском десним тастером на дугме у палети послова отвара се управљачки мени. Минимални облик прозора се састоји од скраћеног облика траке са натписом која садржи сличицу придружену прозору, део натписа и представља дугме за враћање прозора у првобитно стање.
- Дугме за максимизирање прозора - ***maximize***. Притиском на њега прозор ће се раширити тако да попуни цео екран. Само дугме, након ове акције, мења изглед: до притиска имало је изглед квадратића, а сада ће садржати два делимично преклопљена правоугаоника - ***restore***. Поновним притиском на то дугме, прозор ће се вратити на првобитно место у првобитној величини.
- Дугме за затварање прозора - ***close***. Притиском на ово дугме прозор се уклони са екрана. Уклања се и придружено дугме, ако постоји, из палете послова. Ако се ради о главном прозору, апликација се завршава.

2. Трака главног менија - ***menu bar***, налази се испод траке са натписом. Садржи ставке главног менија за избор појединих група радњи које апликација може да обави. Трака главног менија обично постоји само у главном прозору апликације. Апликације без главног менија се врло ретко срећу у пракси.

3. Трака са алаткама - ***toolbars***, налази се испод траке главног менија. Садржи дугмад са сличицама помоћу којих се најчешће коришћене радње могу лакше позвати него помоћу менија.

4. Статусна трака - ***status bar***, налази се уз доњу ивицу прозора. Садржи податке о стању апликације укључујући и кратка упутства или поруке кориснику.

5. Радни простор - ***client area***, заузима сав преостали простор у прозору. У том простору се налазе визуелне компоненте које су смештене у прозору. Уз ивице тог оквира могу да се налазе клизачи помоћу којих се делови прозора који нису стали у радни простор доводе у видно поље, померањем клизача по потреби.

6. Ивице прозора - ***window borders***. Ограђују целокупан прозор. Ивице прозора имају различит изглед у зависности од тога да ли величина прозора може да се мења помоћу миша или не, или се ради о прозору за дијалог.

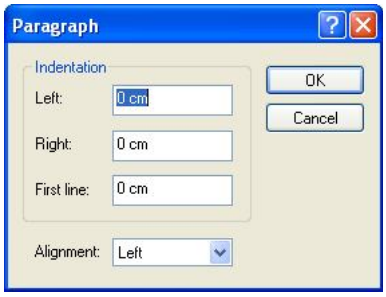
Радом апликација управља се командама помоћу којих се могу подешавати параметри обраде или тражити спровођење појединих радњи које су предвиђене у апликацији. Основни начин издавања команди је помоћу менија - ***menu***, мада рад апликација може да се организује и другачије.

Мени садржи низ ставки од којих једна може да се одабере да би се назначила нека акција апликације. Да листа ставки не би била предугачка, ставке се групишу и образују се менији чија хијерархијска структура садржи више нивоа. Први ниво ставки је главни мени - ***main menu***, чије ставке представљају групе могућих операција. Избором ставки главног менија појављује се подмени - ***submenu*** чије ставке могу да представљају конкретне операције или подменије на следећем нивоу. У пракси најчешће има два нивоа менија, тј. ставке главног менија готово никад не представљају конкретне акције.

Ставке главног менија стално су видљиве у траци главног менија, испод траке са натписом прозора. Ставке подменија приказују се само за тренутно одабрану ставку главног менија, поређане усправно испод одговарајуће ставке главног менија. Евентуални подменији који припадају некој од ставки подменија приказују се десно (или, у ретким случајевима, лево) од подменија, са врхом у висини ставке подменија. Постојање подменија означава се стрелицом у облику попуњеног троугла уз десну ивицу одговарајуће ставке.

Тренутно одабране ставке у менију приказују се истицањем другом бојом подлоге и слова у односу на остале ставке. Пошто ставке у различитим подменијима могу имати исте натписе, нека ставка у менију је једнозначно одређена низом ставки које треба одабрати, почев од главног менија, да би се стигло до ње. Избор ставки главног менија и подменија може да се врши на више начина:

- Притиском на леви тастер миша, када је показивач миша изнад жељене ставке главног или подменија.
- Активирањем главног менија притиском на тастер *Алт*, а затим, тастерима ← и → крећемо се по ставкама главог менија, а тастерима ↑ и ↓ по ставкама тренутно активног подменија. Ако ставка подменија има свој подмени, притиском на тастер → отвара се тај подмени. Обрнуто, враћање на претходни ниво подменија постиже се тастером ←. Избор се остварује притиском на тастер *Ентер* када је изабрана одговарајућа ставка подменија.
- Истовременим притиском тастера *Алт* и тастера подвученог слова у ставки главног менија отвара се одговарајући подмени. После тога треба притиснути тастер подвученог слова у ставки подменија (сада више тастер *Алт* не мора, али може да буде притиснут). Подвучени знакови у натписима ставки менија су знакови пречице.



Избором команди иза чијих се имена у подменију налазе три тачке (...) отвара се посебан прозор за дијалог. Тај прозор садржи образац чијим попуњавањем треба да се задају подаци неопходни за извршавање тих команди. Прозори за дијалог, поред компоненти за уношење или избор вредности потребних података, обично, садрже и два дугмета:

- дугме за потврду - ***OK*** којим се захтева спровођење започете активности са тренутно видљивим вредностима у прозору, и
- дугме за одустајање - ***Cancel*** којим се одустаје од спровођења акције.

Осим мишем, дугме за потврду може да се активира и помоћу тастера на тастатури (у случајевима када је оивичен правоугаоником дебљим од оног око остале дугмади или су слова уоквирена испрекиданом линијом). Дугме за одустајање увек може да се активира помоћу тастера ***Esc*** на тастатури. Одустајање се постиже и затварањем прозора за дијалог помоћу управљачког менија или притиском мишем на дугме за затварање прозора. Понекад прозор за дијалог служи само да би корисник потврдио да стварно жели спровођење одабране активности. Тиме му се пружа још једна прилика за евентуално одустајање. Такви прозори обично садрже текст одговарајућег питања и дугмад ***Yes*** и ***No***. Поред наведених, прозори за дијалог могу да садрже другу дугмад различите намене зависно од функције дијалога, а најчешће и дугме за помоћ - ***Help***. Притиском на то дугме добија се пригодна приручна помоћ, ако је предвиђена у оквиру апликације.

За активирање неких радњи апликције, представљених одговарајућим ставкама подменија, могу да постоје тастери пречице - ***shortcut keys***. За разлику од знакова пречица, помоћу којих се по менију креће корак по корак, као и помоћу миша, притиском на тастер пречицу, ставка менија којој је он придружен активира се у једном потезу. При том се мени не отвара и одабрана ставка се уопште не види на екрану. Не смета ако је у тренутку притиска на тастер пречицу мени отворен са било којом истакнутом ставком, укључујући и ставку која се њиме бира. Тастери пречице могу да буду функцијски тастери од ***F1*** до ***F12*** сами или у комбинацији са управљачким тастерима ***Shift, Alt*** и/или ***Ctrl***, као и тастери са словима у комбинацији са тастером ***Alt*** и/или ***Ctrl***. Постојање тастера пречице означава се уз десну ивицу менија, после натписа ставке менија. За разлику од знакова пречица, чије тумачење може да се мења у току извршавања апликације, тумачење тастера пречица је јединствено у току целог извршавања апликације.

Поред евентуалног главног менија са пратећим подменијима сваки прозор поседује управљачки мени - ***control menu*** који се назива и системски мени - ***system menu***. Управљачки мени активира се притиском на дугме управљачког менија на левом крају траке са натписом прозора. Помоћу управљачког менија прозор може да се минимизира командом ***Minimize***, да се максимира - ***Maximize***, да му се поврати претходна величина - ***Restore***, да се помера по екрану - ***Move***, да му се промени величина - ***Size*** и да се затвори - ***Close***. Све ове радње могу и на други начин да се постигну, па је отварање управљачког менија ретко потребно.

Прозори поред менија садрже и разне компоненте помоћу којих корисник може да утиче на ток извршавања апликације. Међу њима се најчешће користе:

- Ознака - ***label***. Приказује текст, натпис ознаке - ***Caption*** који може да буде нека порука кориснику или опис намене друге компоненте. Корисник апликације не може да делује на ознаке помоћу миша.
- Дугме - ***button***. Корисник може мишем да га притисне и тиме да сигнал за неку акцију. Натпис на дугмету означава његову намену. Уношење података у прозоре за дијалог обично се завршава притиском на једно од дугмади.
- Оквир за текст - ***edit box***. Корисник у овај оквир уноси произвољан текст преко тастатуре. Постоје оквири који могу да садрже само један ред текста и оквири који могу да садрже више редова текста.
- Оквир за потврду - ***check box***. Мали квадрат поред натписа овог оквира може да садржи знак за потврду (✔) или да буде празан. Присуство знака за потврду означава да особина у натпису важи, а одсуство да не важи. Левим кликом миша мења се стање из непотврђеног у потврђено, и обрнуто.
- Радио-дугме - ***radio button***. Мали круг поред натписа ове компоненте може да садржи црну тачку (⊙) или да буде празан (○). Користи се у групама с тим да само једно дугме у групи може да има попуњен кружић. Те групе се користе за избор једне од неколико, медусобно искључивих, могућности.

Неке од компоненти у прозору или неке ставке менија могу да буду привремено недоступне. Такве компоненте обично се приказују бледо - ***dimmed***.

Сваког тренутка само је један од тренутно видљивих прозора на екрану активан. За такав прозор се каже да је у жижи (***focus***). Од осталих прозора се разликује видљиво по другачијој боји траке са натписом прозора и текста у њој. Корисник може да активира прозоре по произвољном редоследу, осим ако је тренутно активан прозор условљени прозор, простим притиском мишем на било коју тачку која припада жељеном прозору. У тренутно активном прозору сваког тренутка је само једна компонента активна. За ту компоненту се каже да је у жижи. Компонента која је у жижи, обично се видљиво разликује од других компоненти исте врсте. Три карактеристична начина разликовања су:

- натпис компоненте уоквирен правоугаоником од танке испрекидане линије, за компоненте које имају натписе;
- трептећи показивач текста унутар оквира за текст или
- уоквирена једна од ставки унутар оквира са листама.

Притиском на неку компоненту мишем или одабирањем помоћу знака пречице, жижа се помери на ту компоненту. Поред тога, жижа може да се помери са једне компоненте на другу према одређеном редоследу, притискањем тастера ***Tab*** (↹) на тастатури, односно, комбинације тастера ***Shift + Tab*** по обрнутом редоследу. Редослед померања жиже са компоненте на компоненту на овај начин зависи од апликације и корисник не може да утиче на његову промену. У добро пројектованој апликацији редослед померања жиже са компоненте на компоненту следи логичан ток уношења података, примерен датој ситуацији. Такође, у добрим апликацијама компоненте су тако распоређене да логичан редослед њиховог обиласка на екрану буде слева удесно и одозго надоле (ово није обавезно, али кориснику у великој мери олакшава коришћење апликације).

Све апликације засноване на прозорима могу да се поделе у две групе:

- Апликације за рад са једним документом (***Single Document Interface Applications - СДИ апликације***). Могу истовремено да обрађују само један документ (текст, слику итд.). У главном прозору се обрађује један документ (текст, слика или други скуп података), а евентуални остали прозори у апликацији служе само као подршка за остваривање те обраде. Сви су прозори обични, у смислу да међу њима нема прозора родитеља нити прозора деце.
- Апликације за рад са више докумената (***Multiple Document Interface Applications - МДИ апликације***). Могу одједном да обрађују више докумената. Главни прозор је прозор родитељ који садржи барем један прозор дете. Сваки прозор дете садржи по један документ који се обрађује у апликацији. Апликација може да има и известан број обичних прозора који служе као подршка за обраду докумената. Појединим радњама у апликацији делује се на документ у прозору детету који је тог тренутка активан.

Делфи је апликација заснована на прозорима, а са друге стране, делфи служи за израду апликација заснованим на прозорима. Препорука пројектантима је да се труде да своје апликације што више уклопе у опште прихваћену шему јер тиме повећавају изгледе да њихов производ лакше прихвате будући корисници. Делфи у потпуности подржава и потпомаже пројектовање стандардних и атрактивних апликација за оперативни систем ***Windows***.

Елементи графичког корисничког интерфејса (*Graphical User Interface*)

Графичко корисничко окружење (***Graphical User Interface***, ***GUI***) је тип корисничког окружења који омогућава људима да рукују рачунаром или уређајима контролисаним рачунаром. То је начин интеракције човека с рачунаром кроз манипулацију графичким елементима и додацима уз помоћ текстуалних порука и обавештења. Графичко корисничко окружење користи иконице, визуелне индикаторе или специјалне графичке елементе (***widgets***) уместо обичних текстуалних менија или ручног уношења команди. Иконице се често употребљавају заједно са текстом или ознакама да потпуно прикажу информацију и радње доступне кориснику. Графички кориснички интерфејс се ослања на ***WIMP*** узор рада корисника са рачунаром. Основни елементи овог модела су:

- ***Window*** (*прозор*) - ограничене правоугаоне области које садрже радну површину, мениje, дугмиће, траке за вертикално померање *скроловање* садржаја прозора и друго. Апликације и датотеке се приказују унутар прозора (мада не мора свака апликација имати свој прозор).
- ***Icon*** (*икона или сличица*) - икона је визуелни репрезент објеката у саставу рачунарског система. Објекти могу бити физички или логички (програми или датотеке са подацима). Асоцијативно делује на корисника: јасно означава функцију, лако се памти и разликује се од осталих икона. Селектовањем иконе покреће се програм или отвара датотека, преко путање до садржаја на меморијском медијуму где су смештени.
- ***Menu*** (*мени*) - основни механизам интеракције корисника са програмом, при чему се кориснику прикаже низ могућности у датој ситуацији, а корисник се опредељује за једну од њих.
- ***Pointer*** (*показивач*) - графички симбол којим управља миш или неки други улазни уређај, а користи се за лоцирање, позиционирање и избор визуелног репрезента објекта на екрану монитора.

Корисник у ***GUI*** окружењу може помоћу миша селектовати и кликнути на жељену иконицу (***icon***), чиме стартује одговарајући програм. Без помоћи ***GUI*** интерфејса корисник би све команде морао да уноси у текстуалном облику преко тастатуре. ***GUI*** програми приказују визуелне елементе попут:

- икона (сличице на десктопу, тј. позадини)
- прозора
- дугмад са текстом и/или сликама (***button***)
- оквири за унос текста (***edit***)
- квадратићи за одабир и типа (могуће је одабрати више квадратића - ***check box***)
- кружићи за одабир или типа (могуће је одабрати само један кружић - ***radio button***)

Претходници ***GUI*** програма су развијани су на Станфордском факултету, на челу са Енглбартом (***Douglas Englebart***) и користили су текстуалне повезнице за управљање. Касније су истраживачи на ***Xerox PARC***-у унапредили и отишли даље од само текстуалних повезница и почели су користити графичко окружење за њихов ***Alto*** рачунар. То је био претходник свих данашњих графичких окружења, звао се ***PUI***. ***PUI*** је и тада користио прозоре, мениje, дугмад, квадратиће за одабир, иконе уз употребу неких показних уређаја као што је миш.

Први пут ***GUI*** је искористио ***Apple*** на својим ***Macintosh*** рачунарима и оперативним системима (1984.), док је касније(1985.) ***Microsoft*** копирао ***Apple***-ове идеје у својим првим верзијама ***Windows*** оперативног система. Примери неких графичких окружења су ***Mac OS***, ***Microsoft Windows***, ***NEXTSTEP*** и ***X Window System*** од којег су настали ***Qt*** (***KDE***), ***GTK+*** (***GNOME***) и ***Motif*** (***CDE***).

Помоћу графичког окружења употреба данашњих рачунара је много једноставнија него у доба ***DOS*** оперативног система који је био прилично негостољубљив према новим корисницима рачунара. Већина данашњих оперативних система има могућност употребе графичког окружења, дакле курсора, икона, прозора и других елемената.

У програмима које ћемо писати, најчешће ћемо користити следеће компоненте:

- ***Form*** (*форма, образац*) - ова компонента ће се користити за креирање спољашњег изгледа нашег програма и на њу ће се постављати остале компоненте.
- ***Label*** (*лабела, натпис*) - може се поставити било где на форми, користи се за приказ текстуалних података.
- ***Edit*** (*едит, оквир за текст*) - може се поставити било где на форми, користи се за унос текстуалних података.

- ***Button*** (*тастер, дугме*) - може се поставити било где на форми, користи се за потврду почетка неке акције.
- ***Memo*** (*мемо поље, оквир за вишередни текст*) - користи се за приказ и унос вишередног текста.

Програми руковођени догађајима

У објектно оријентисаним програмским језицима у центру пажње су догађаји на које могу реаговати распоређене компоненте. Постоји велики број догађаја на које поједине компоненте реагују, а програмом се дефинишу они на које ће поједине компоненте реаговати. На пример, ако је тастер компонента чије реакције на догађаје се обрађују, онда догађаји могу бити: клик мишем на објекат, повлачење објекта, прелазак мишем преко њега, померање миша на горе или на доле, активирање објекта, напуштање објекта, притисак неког тастера на тастатури, отпуштање тастера на тастатури, унос неког карактера помоћу тастатуре и други.

Извор догађаја може бити миш, тастатура или неки од постављених објеката, а програмер дефинише на које од њих ће постављена компонента реаговати и на који начин. Клик на неки тастер може проузроковати затварање апликације, покретање неке процедуре или неку активност те или неке друге компоненте. Промена садржаја оквира за текст може покретати неку процедуру или дефинисати неку акцију те или неке друге компоненте. И само покретање апликације представља један догађај, исцртавање неке компоненте такође је догађај који може утицати на почетак извршавања неке дефинисане процедуре. Овакав начин покретања појединих делова апликације повећава динамичност и непосредност апликације. Померањем миша можемо проузроковати померање објеката на форми чиме се добија могућност креирања атрактивних апликација које одржавају пажњу корисника и усмеравају га на одговарајућу активност.

Питања на која треба обратити пажњу

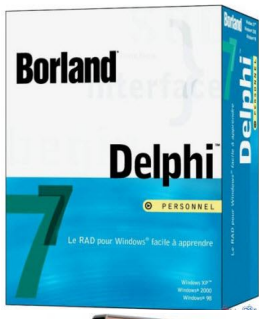
- Одлике апликација заснованих на прозорима
- Врсте прозора у апликацијама и њихове карактеристике
- Карактеристике рада са мишем
- Управљање радом апликације помоћу тастатуре
- Елементи прозора апликације и њихова функција
- Главни мени апликације и систем подменија
- Прозори за дијалог
- Најчешће коришћене компоненте које могу бити елементи апликације
- Подела апликација заснованих на прозорима
- Елементи графичког корисничког интерфејса
- Извори догађаја и утицај њихове обраде на квалитет апликације

Увод у развојно окружење програмског језика

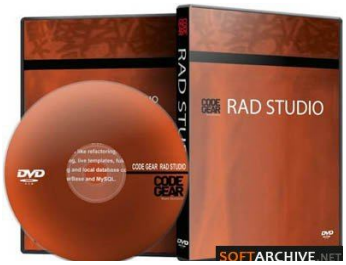
На персоналним рачунарима данас врло је популаран рад под оперативним системом ***Windows*** и развој апликација за ту платформу. Као одговор на ово велико интересовање, софтверска кућа ***Borland*** је *давне* 1994. године објавила своје моћно интегрисано развојно програмско окружење ***Delphi*** за израду апликација за рад под оперативним системом виндоус. Програмско окружење делфи садржи врло моћне алате за брзо и лако организовање и уобличавање комуникације на релацији апликација - корисник у стилу професионалних апликација за виндоус платформу. Ти алати омогућавају визуелно уређивање појединих прозора (образаца) и подешавање параметара компонената у тим обрасцима, померањем компонената по екрану помоћу миша и попуњавањем одговарајућих формулара. У том делу посла нема никаквог писања програма. Поступци које апликација треба да обави као одговор на разне догађаје, као што је притисак мишем на неко дугме у прозору, програмирају се на језику ***Object Pascal***. Објектни паскал представља даље усавршавање турбо паскала у домену објектно оријентисаног програмирања. Уграђена је подршка за непосредно коришћење АПИ (***API - Application Programming Interface*** - рутине за управљање датотекама, приказ података и слично) функција, стандардног алата који фирма *Microsoft* нуди за израду апликација за виндоус окружење. Програмер у делфи окружењу релативно ретко непосредно позива те АПИ функције. Алати које делфи пружа чине то уместо њега.

Програмски језик делфи настао је као први програмски језик са визуелном компонентом. Иако је у то време већ постојао ***Visual Basic*** (ВБ), не може се рећи да је он први програмски језик за визуелно програмирање јер су се програми у ВБ извршавали под контролом преводиоца. У делфију се изворни програмски код преводи директно у машински чиме се постиже већа брзина код извршавања програма и независност програма од програмског преводиоца, тј. програм написан у делфију и преведен у извршни код може се извршавати без обзира да ли је на рачунару инсталиран делфи преводилац.

Прва верзија делфија прављена је за оперативни систем ***Windows 3.1x*** и правила је 16-битне апликације у машинском коду. Могла је да се користи и под оперативним системима ***Windows 95*** и ***Windows NT***, али није могла да искористи нове погодности које су пружала та нова, 32-битна окружења. Резултујуће апликације су биле искључиво 16-битне, без обзира под којим оператиним системом се радило. Креиране апликације су могле да се извршавају у свим поменутим окружењима. Подршка за 32-битно програмирање је уграђена у верзију 2. Осим тога у ова верзија укида ограничење дужине знаковних низова на 255 знакова. Боље управљање меморијом, истовремено извршавање више нити програма (***multithreading***), већи број новоуведених компоненти, бољи рад са базама података и повезивање са ***SQL*** језиком упита су, такође, новине. Ова знатно прерађена и допуњена верзија, која је објављена у фебруару 1996. године, могла је да се користи само под оперативним системима ***Windows 95*** и ***Windows NT***, а и развијене апликације су могле да се извршавају само под тим оперативним системима. Трећа верзија из 1997. године донела је само неколико новина и побољшања већ постојећих карактеристика (много већи скок је направљен између верзија 1 и 2). Пре свега, побољшана је помоћ за писање кода апликација, подршка пакетима кода којима могу приступити различити програми, подршка ***Active X*** контролама, могућност креирања објектног кода (***.obj***) чиме се отвара могућност увоза програмског кода у друге програмске језике. Додате су и интернет компоненте за креирање апликација које могу да користе предности глобалне мреже. Делфи 4 из 1998. године доноси знатна побољшања већ постојећих карактеристика претходне верзије и прилагођавање делфија новим верзијама оперативног система са модернијим изгледом и једноставнијом употребом алатки. Делфи 5 доноси знатна олакшања у раду са базама података и даља унапређења ***Active X*** контрола. Делфи 6 је до крајњих граница поједноставио изградњу апликација увођењем чаробњака који помажу код почетног формирања различитих апликација. Палете са алаткама су препуне новим компонентама. Делфи 7 је прерађена и мало побољшана верзија делфи 6 пакета прилагођена оперативном систему ***Windows XP*** и допуњена пакетом за креирање извештаја ***Rave Designer***.



Убрзо, затим, појавила се верзија Делфи 8 и то је последњи пројекат компаније ***Borland***. Испоставило се да је то још један од пословних промашаја фирме, односно, да готово ништа ново и значајно није донео. Компанија је запала у дугове због овог, али и још неких других пословних промашаја и постојала је оправдана сумња да ће се пројекат Делфи угасити. Спас је дошао са појавом верзије ***Delphi 2009*** која је прилагођена раду у мрежном окружењу и новим 64-воро битним оперативним системима, а која је настала у тренутку када је компанију ***Borland*** преузела компанија ***CodeGear***. Ово је прва верзија делфија која је испоручивана у пакету са ***C++*** билдером, тако да су њиме могли бити задовољни и љубитељи делфија и це-програмери. Даљи развој оперативних система довео је до појаве делфија у верзији ***XE***. Актуелна је верзија ***XE3.***



Карактеристика свих верзија делфија је да генеришу прави машински програм (***native code***), што резултује бржим извршавањем апликација у односу на друга визуелна окружења која генеришу неки међукод који се после извршава интерпретацијом. Коначан резултат пројектовања је једна извршна датотека (са наставком ***.exe***) програма који чини апликацију. Приликом инсталације апликације, осим евентуалних датотека које користи апликација, додатне датотеке нису потребне (на пример, библиотеке са динамичким повезивањем, датотеке са наставком ***.dll***), под условом да апликација не користи подршку за рад са базама података и за израду извештаја. Ако се користи нека од тих подршки, ***BDE*** (***Borland Database Engine***), односно, ***Report Smith*** морају да се инсталирају. Смештање целокупног кода у извршни облик програма има и један недостатак: најмања апликација заузима барем 275 килобајта. Ако се на једном рачунару користи више апликација које су развијене помоћу делфија, долази до знатног утрошка простора на диску (данас је овај проблем знатно ублажен снижењем цене хард дискова и повећањем стандарда на 500 гигабајтова, али средином 90-их година хард дискови су били величине 80 - 500 мегабајтова). Због тога је у верзији 3.0 уведена могућност прављења пакета са динамичким повезивањем (***.dpl - Delphi Package Library***) који по намени одговарају виндоусовим библиотекама са динамичким повезивањем (***.dll***). Библиотека визуелних компонената се испоручује у облику таквих пакета. Пројектант апликације може да назначи који ће од тих пакета да се угради у извршни облик апликације, а који не. Наравно, пакети који нису уграђени морају да се испоруче одвојено кориснику апликације.

Привредна комора Београда, Удружење информатичке делатности у сарадњи са компанијом Ембаракадеро Интернационал и локалним партнером Code Factory, премијерно су представили EMBARACADERO - RAD STUDIO XE3 који садржи програмске алате Делфи/Ц++/ХТМЛ5 за развој софтвера за најновије оперативне системе Windows 8, Mac Mountain Lion, Mobile на стручном скупу: "EMBARACADERO - Нови RAD STUDIO XE3" који је одржан у уторак 11. септембра 2012., са почетком у 11.00 часова у сали VI, зграде Привредне коморе Београда, Кнеза Милоша 12. Нови алат представљен је као најбржи пут до Windows 8 апликација. По њима, Delphi XE3 и C++builder XE3 са компонентама за Метрополис су једини алати који омогућавају надоградњу класичних Windows UI апликација на Windows 8 Style. Апликације раде на WinXP, Vista, Windows 7 и 8, и то како на десктоп тако и на таблет окружењима као што су Slate и Surface Pro. Апликације у делфију XE3 сада подржавају HD/3D графику задивљујућег изгледа. Пакет укључује и најновију верзију FireMonkey компоненати – FireMonkey FM² са којима можемо програмирати апликације изузетаног изгледа за Enterprise и ISV и то за Windows 8, Mac OS X Mountain Lion и Mobile и све то из истог изворног кода. Ту је и најновији механизам за повезивање са базама података. Нови Visual LiveBindings пружа најлакши начин за повезивање корисничког интерфејса са подацима у базама. Приступачније него икада до сада, Rad Studio XE3 Visual LiveBindings омогућава повезивање без писања једног реда кода. Повезивање података и контрола је изузетно.

Већина савремених програмских језика подржава ООП. Сви ООП језици су засновани на три концепта:

енкапсулацији - Енкапсулација представља имплементацију која ће довести до жељеног понашања објеката. Битан њен аспект је и сакривање небитних особина класе, тј. сакривање структуре класе. Свака класа има два дела: интерфејс и имплементацију. Интерфејс класе обухвата само спољашњи изглед класе и преко њега се установљава да ли апстракција има жељено понашање. Са друге стране, имплементација обухвата реализацију свих механизама који доводе до тог жељеног понашања.

наслеђивању - Механизам наслеђивања (***inheritance***) састоји се у томе да једна класа може бити врста друге класе са додатним особинама, наслеђујући њену структуру, понашање и комплетну функционалност те класе и додајући евентуално неке своје особености које је чине аутономним ентитетом.

полиморфизму - Полиморфизам је својство да објекат изведене класе извршава операцију на начин својствен изведеној класи, иако му се приступа као објекту основне класе. Полиморфизам обезбеђује генерички софтверски интерфејс тако да колекцијом различитих типова објеката може да се манипулише униформно.

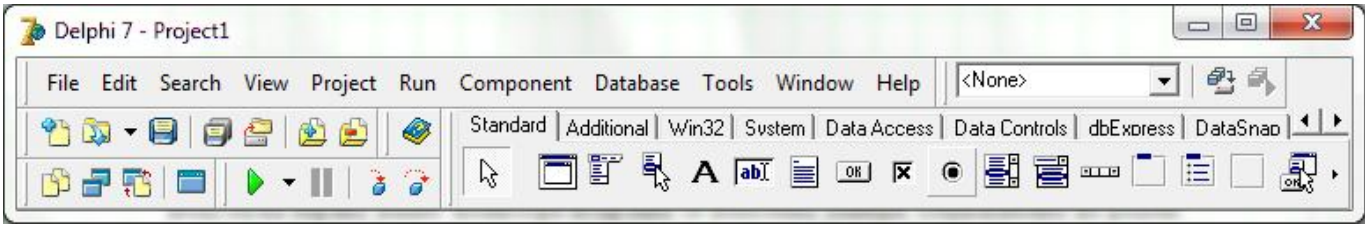
Програмски језик објектни паскал који делфи користи је ООП проширење стандардног језика паскал. Синтаксе програмског језика паскал је обимнија и читљивија од синтаксе, на пример, језика Ц, а његово ООП проширење прати исти приступ и даје му моћ једнаку најновијим ООП језицима Јави или Ц++.

Ако питамо било којег програмера који је програмски језик најбољи добићемо увек исти одговор, онај који он користи јер је ефикасан, једноставан, лак за учење итд. Права истина крије се баш у томе. Најбољи је онај језик који најбоље познајемо. А како онда одлучити шта је најбоље ако не знамо ниједан програмски језик. Нема правог одговора. Ако ћемо сами учити, најбоље је почети од језика за који имамо најбољу литературу (теоријске основе и примере развоја различитих апликација, нарочито оних каквим ћемо се бавити), а ако ће нам неко помагати онда препустимо њему да нас води. Када савладамо неке елементарне технике онда даље можемо и сами, а у зависности од прилика можемо и прећи на неки други програмски језик који ће захтевати конкретни посао.

Почетак рада и управљање развојним окружењем.

Покретањем апликације отвара се неколико прозора које ћемо укратко описати.

Широка трака на врху екрана је главни изборни прозор или главни прозор апликације. Њен изглед можемо прилагодити свом укусу, али и карактеристикама рачунара на коме радимо (монитор са вишом резолуцијом даје нам више могућности распоређивања делова).



Стандардна трака прозора садржи назив програмског модула (***Delphi 7 - Project1***) са иконицом која представља дугме управљачког менија (***Control Menu Button***) и три екранска тастера (стандардни управљачки тастери): ***Minimize, Restore Down / Maximize*** и ***Close***. Подразумева се да знамо да минимизацијом овог прозора са екрана нестају и сви други, зависни прозори делфија и да се у том моду не може радити ништа. Затварањем овог прозора излази се из делфија. Ако оно што смо пре затварања није сачувано програм ће нас упозорити на то и тражити потврду акције или претходно чување нашег рада. Испод ње се налази линија главног менија са стандардним опцијама: ***File, Edit, Search, View, Project, Run, Component, Database, Tools, Window*** и ***Help***. Велики део команди у менијима знамо из рада са неким другим виндоус апликацијама и зато ћемо се кратко задржати на њиховим специфичностима у делфију, а остале нећемо објашњавати јер се не уклапају у програм овог курса.

Из ***File*** мени ћемо користити следеће команде:

New - отвара подмени из кога треба одабрати опцију ***Application***

Open - отвара дијалошки прозор за учитавање постојећих пројеката и других текстуалних датотека које могу бити саставни део пројекта или не. Команду не треба га користити док мало боље не савладамо рад са делфијем

Open Project - отвара дијалошки прозор за учитавање постојећих пројеката

Reopen - отвара подмени са листом од пет последњих пројеката које смо сачували (из које се кликом мишем може изабрати жељени) и дужом листом паскалских програма (јунита) који се могу учитати, али не и покренути из делфија (листу користимо за налажење путање до неког пројекта, а не за његово учитавање)

Save - омогућава чување тренутно активне датотеке са датим именом (команду треба пажљиво користити јер не чува пројекат, па се може десити да касније не можемо отворити и покренути програм за који смо мислили да је сачуван)
Save As - омогућава чување тренутно активне датотеке са промењеним именом; користи се када већ урађен сличан задатак желимо да искористимо да брже урадимо наредни задатак (команду треба ***обавезно*** користити у пакету са наредном да бисмо сачували и пројекат са промењеним именом, а не да у претходно отвореном пројекту само променимо име активне датотеке и тако изгубимо претходни задатак)
Save Project As - омогућава чување активног пројекта са промењеним именом; користи се када већ урађен сличан задатак желимо да искористимо да брже урадимо наредни задатак
Save All - омогућава чување активног пројекта са свим припадајућим датотекама
Close - затвара активни пројекат са припадајућим датотекама (корисније је, у ову сврху употребљавати наредну команду)
Close All - затвара све отворене датотеке
Print - штампа активну датотеку на штампачу
Exit - затвара се отворени пројекат и напушта делфи апликација; ако оно што смо пре затварања није сачувано програм ће нас упозорити на то и тражити потврду акције или претходно чување нашег рада.

Из ***Edit*** менија ћемо користити команде:

Undo/Undelete - поништава последњу акцију или брисање
Redo - враћа последњу акцију поништену претходном командом
Cut - селектовани део активне датотеке брише и премешта на ***clipboard***
Copy - селектовани део копира на ***clipboard*** не бришући га из активне датотеке
Paste - копира садржај са ***clipboard***-а у активну датотеку
Delete - брише селектовани део активне датотеке
Select All - селектује комплетан садржај активне датотеке

Из ***Search*** менија ћемо користити команде:

Find - проналази наведени текст, ако постоји, и маркира га
Replace - проналази наведени текст, ако постоји, и мења га другим наведеним текстом
Search Again - понавља претходно дефинисано претраживање

Из ***View*** менија ћемо користити команде:

Object Inspector - отвара и приказује прозор ***Object Inspector*** ако је претходно био затворен
Object TreeView - отвара и приказује прозор ***Object TreeView*** ако је претходно био затворен
Alignment Palette - отвара и приказује прозор ***Alignment Palette***
Toggle Form/Unit - на врх екрана поставља један од прозора ***form*** или ***unit***

Из ***Run*** менија ћемо користити команде:

Run - преводи написани програм на машински језик и извршава га (ако нема грешака)
Program Reset - насилно прекида извршење програма, ослобађа радну меморију и враћа нас у делфи да бисмо наставили рад на истом или другом програму (користи се када програм из неког разлога не може да се искључи на регуларан начин).

Већина поменутих команди из ових менија има одговарајућу пречицу - комбинацију једног или више тастера којом се може извршити без директног отварања менија и избора команде из њега.

У оквиру овог прозора могу се наћи и бројне тулбар траке које могу бити различито распоређене. На њима се налазе екрански тастери - ***speed buttons*** којима се, као и пречицама, поједине команде могу извршити без директног отварања менија и избора исте команде из њега. Неки од ових тастера не убрзавају рад са одговарајућом командом (јер команда може бити јако сложена), али је приступ команди једноставнији, па су они зато и постављени на одговарајућу траку. Сада ћемо навести тулбар траке које се стандардно налазе у главном прозору делфија и тастере који се налазе на њима без објашњавања одговарајуће команде јер смо команде које ћемо користити већ објаснили кроз систем менија.

Испод траке са називом апликације су тулбар траке (***toolbar***):

- ***Desktop*** са оквиром ***Desktop speedsetting*** и екранским тастерима: ***Save current desktop*** и ***Set debug desktop*** (у првом реду, у продужетку главног менија)

У другом реду, одмах испод главног менија су:

- ***Standard*** са тастерима: ***New, Open, Save, Save All, Open project, Add file to project, Remove file from project***
- ***Custom*** са тастером: ***Help contents***

У трећем реду испод главног менија је:

- ***View*** са тастерима: ***View unit, View form, Toggle Unit/Form, New form***
- ***Debug*** са тастерима: ***Run, Pause, Trace into, Step over***

Са десне стране у продужетку од последња два реда са тулбар тракама налази се палета са објектима које можемо уградити у свој програм. Навешћемо их све, а затим објаснити само оне које ћемо користити у овом курсу (компоненте које су неопходне да би наш програм радио).

- ***Component palette*** са листићима: ***Standard, Additional, Win32, System, Data Access, Data Controls, dbExpress, DataSnap, BDE, ADO, InterBase, WebServices, InternetExpress, Internet, WebSnap, Decision Cube, Dialogs, Win 3.1, Samples, ActiveX, Rave, Indy Clients, Indy Servers, Indy Intercepts, Indy I/O Handlers, Indy Misc, COM+, InterBase Admin, IW Standard, IW Data, IW Client Side, IW Control, Servers*** у оквиру којих се налазе објекти које можемо уградити у свој програм једноставним превлачењем објекта на форму. Део листића се не види, па да би се до њих стигло треба притиснути неколико пута одговарајући тастер са стрелицом

Најчешће ћемо користити листић ***Standard*** јер се ту налазе најједноставније компоненте за комуникацију програма са корисником, у ретким случајевима ћемо користити неке од компоненти са листића ***Additional, Win32, System***. Остале листиће и компоненте нећемо користити.

Селектовање компоненте коју желимо да уградимо у свој програм остварује се кликом миша на њену графичку представу на листићу, а затим кликом на одговарајућу позицију на форми постављамо изабрану компоненту. Ако смо се предомислили, па не желимо ту компоненту можемо:
- кликнути на другу компоненту, ако је потребно уградити је у програм, или
- кликнути на стрелицу на почетку траке са компонентама, ако не желимо да додамо било шта.

Са листића ***Standard*** најчешће ћемо користити компоненте:
- ***Label*** (сличица са великим словом ***A***)
- ***Edit*** (сличица са малим словима ***ab***)
- ***Button*** (сличица са словима ОК)
- ***Memo*** (сличица између едита и тастера),
- ***ComboBox*** (једанаеста сличица не рачунајући стрелицу за деселекцију) и
- ***Panel*** (претпоследња сличица на листићу).

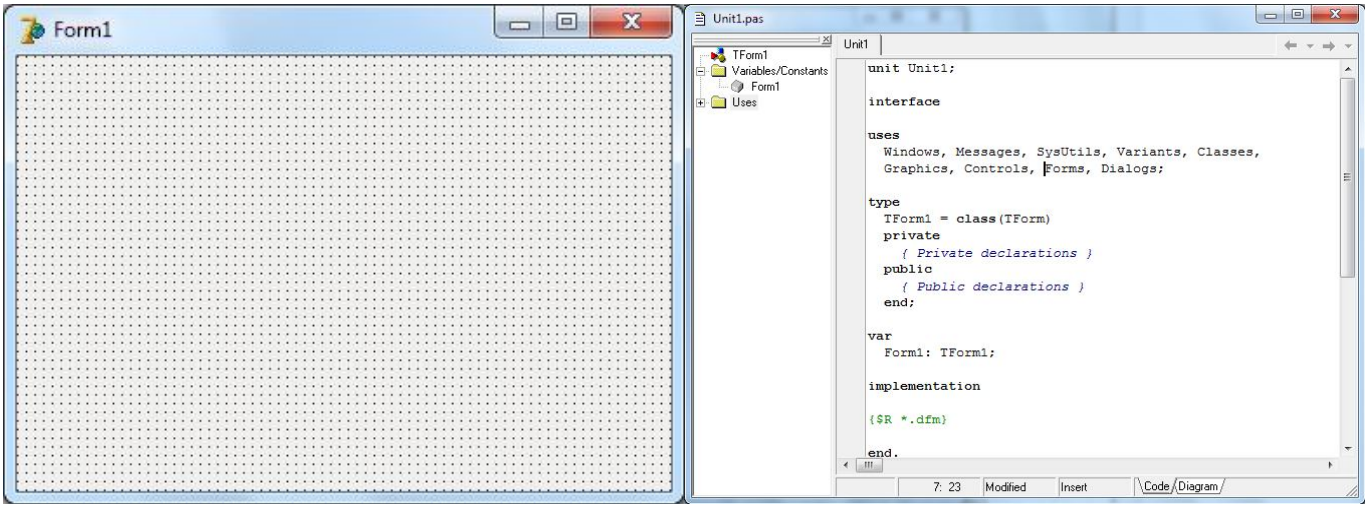
Са листића ***Additional*** најчешће ћемо користити компоненте:
- ***BitBtn*** (компонента личи на тастер са могућношћу комбиновања слике и текста на њој),
- ***Image*** (када желимо да свој програм украсимо сликом) и
- ***LabeledEdit*** (компонента која представља комбинацију лабеле и едита).

Са листића ***Win32*** ћемо користити компоненту ***DateTimePicker*** за читање времена и датума.

Са листића ***System*** ћемо користити компоненту ***Timer***, а занимљиве су и ***PaintBox*** и ***Media Player*** код креирања мултимедијалних апликација, али их нећемо користити на почетку учења.

Централни део екрана заузима прозор са именом ***Form1.*** Форма (образац) представља радну површину програма. То је оно што ће се видети на екрану када се програм извршава. На форму се постављају објекти. Дефинисањем параметара - карактеристика (***Properties***) форме дефинишемо изглед нашег програма. Једна од карактеристика форме је име - ***Form1***, уопштено, аутоматски додељено на почетку рада што значи да се и оно може, по жељи, променити. Затим ширина и висина форме, боја позадине, основни фонт итд. Као и сваки прозор, форма у десном горњем углу има тастере ***Minimize, Restore Down/Maximize*** и ***Close***, дугме управљачког менија, ***Control Menu Button***, у облику иконице која је симбол програмског језика делфи (ови елементи се могу искључити у фази извршења апликације) и назив форме (може бити празан).

Испод овог прозора налази се прозор ***Unit1***. У њему се пишу кодови процедура и функција које се користе у програму. Подељен је на два дела. У левом делу, који се зове ***Code Explorer*** и може бити самосталан прозор изван прозора јунита, виде се називи компоненти које смо уградили у свој програм, називи свих променљивих и константи које су декларисане и дефинисане и њихове везе. У десном делу, који се назива ***Edit Window***, су изворни кодови процедура и функција, као и декларације и дефиниције. Наравно, ако програм који пишемо,

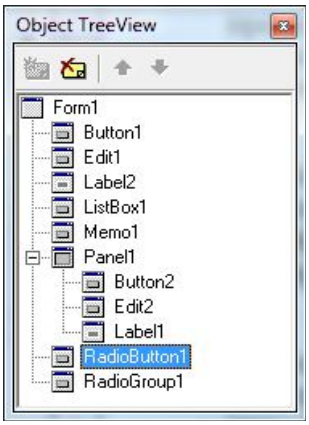


садржи више независних целина, тј. јунита, десни део овог прозора ће имати посебне листиће за сваки од њих, а прелазак из једног у други листић је стандардни клик левим тастером миша над његовим именом. Прелазак из прозора јунита у прозор форме и обрнуто омогућава тастер ***F12***. Уколико имамо више прозора са јунитима прелазак из једног у други омогућава комбинација ***Ctrl+F12***, а ако имамо више прозора са формама прелазак из једног у други омогућава ***Shift+F12***. Наравно, ако су ови прозори видљиви или бар неки њихов део, из једног у други можемо прећи и једноставним кликом миша на неки од њих. Јунит нема карактеристике јер је невидљив када се програм покрене, а назив јунита се даје приликом чувања пројекта.

Лево од ова два прозора (мада се распоред прозора може по жељи мењати, али говоримо о стандардном, изворном распореду прозора - пре било какве акције корисника) су још два стандардно отворена прозора:

- ***Object Tree View*** и
- ***Object Inspector***.

За разлику од претходних прозора, ови се не могу ни минимизирати нити максимизирати помоћу стандардних тастера. Можемо их увећавати или смањивати повлачењем ивица уз помоћ миша или затворити левим кликом на ***X***. Но, то не представља њихов недостатак с обзиром да су то помоћни прозори са дефинисаним садржајем који програмер не може мењати. У случају да су намерно или грешком затворени притиском тастера ***X*** или другачије отварају се притиском тастера ***F11*** (*Object Inspector*) и ***Shift+Alt+F11*** (*Object TreeView*) или одговарајућим командама из ***Edit*** менија.



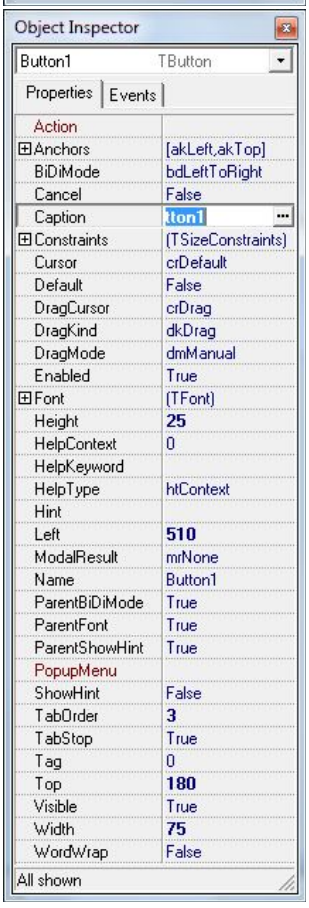
Object TreeView је помоћни прозор у коме се виде називи свих употребљених компоненти програма, њихова повезаност са главном формом и, евентуална, припадност неким другим компонентама које се налазе на главној форми програма. Нарочито је значајан када се пишу компликованији програми са више панела или форми, програми у којима се поједине компоненте, односно, објекти који су постављени на форму преклапају тако да неки чак и нису видљиви (јер су прекривени другим компонентама), па их је зато немогуће селектовати ради постављања или измене њихових карактеристика. Да бисмо селектовали један такав *скривени* објекат довољно је да једноставно кликнемо на његово име у ***Object TreeView*** прозору и онда можемо, иако га не видимо, да мењамо његове карактеристике (наравно, морамо тачно знати који се ефекат постиже изменом неке карактеристике, нарочито ако се она односи на спољашњи изглед објекта који не видимо).

Object Inspector је врло користан помоћни прозор у коме за активну компоненту, тј. објекат чије име се исписује у првом оквиру, дефинишемо карактеристике и његово реаговање на разне догађаје. Овај прозор има два листића: ***Properties*** и ***Events***.

Листић ***Events*** садржи листу свих догађаја на које одређени објекат може "реаговати". Двокликом у оквир десно од назива догађаја аутоматски се отвара процедура коју треба да напише програмер, а која ће представљати реакцију објекта на тај догађај. У случају да је програмер већ дефинисао неке процедуре које могу бити реакција на изабрани догађај, левим кликом на стрелицу на десном крају оквира тог догађаја ће се отворити листа са њиховим именима из које се може изабрати само једно, тј. на један догађај се може реаговати само на

један начин. Листа догађаја објекта може бити променљиве дужине у зависности од врсте објекта чија се реакција дефинише, тј. немају сви објекти исту *осетљивост* на разне догађаје.

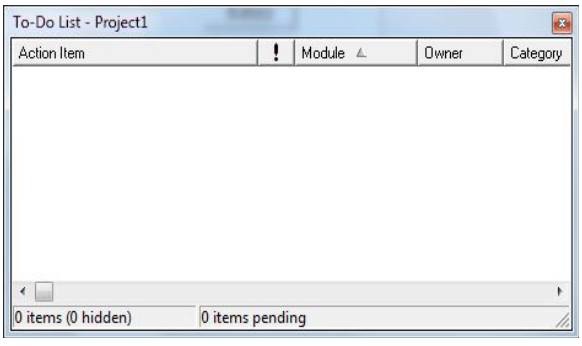
Листић ***Properties*** садржи све могуће карактеристике изабраног објекта (величина, боја, положај, хоризонтално и вертикално уређење, наслов, облик рубова, фонт, име итд) и њихове предефинисане вредности, ако постоје, а код неких карактеристика и све вредности од којих се може изабрати само једна. Програмер има могућност да промени карактеристике објекта не пишући програмски код, већ директним уносом вредности карактеристике у предвиђени оквир. Различити објекти имају различит број карактеристика, па је зато дужина овог листића променљива, осим тога, неке карактеристике које се не мењају често могу бити сакривене, невидљиве или се могу приказивати, по жељи програмера. Неке карактеристике означене су симболом + или - јер представљају сложене, тј. вишедимензионалне карактеристике, на пример: карактеристика Font се састоји из детаља: ***Charset*** - кодна страна, ***Color*** - боја исписа, ***Height*** - висина у тачкама, ***Name*** - назив фонта, ***Pitch*** - растојање између узастопних слова, ***Size*** - величина слова и ***Style*** - стил исписа који је, такође, сложен и састоји се из подкарактеристика ***fsBold, fsItalic, fsUnderline, fsStrikeOut***. Притиском на симбол + отварају се сви детаљи који припадају тој карактеристици, а симбол + прелази у симбол -. И неки детаљи, такође, могу бити означени овим симболима, па ако су означени симболом + на исти начин се отварају. Притиском на симбол - затварају се сви детаљи који припадају тој карактеристици, а симбол - прелази у симбол +. Карактеристике објеката у оквиру овог листића су обично уређене по абецедном редоследу, али могу бити уређене и по категоријама, ако нам је тако једноставније да пронађемо карактеристику коју желимо да дефинишемо. Промену начина уређења карактеристика објеката извршићемо притиском десног тастера на било којем делу ***Object Inspector*** прозора, избором опције ***Arrange*** и, затим, опције ***by Category*** или ***by Name***. Избором опције ***Properties*** можемо дефинисати боје позадине и исписа (што, на почетку учења, није претерано важно, али некоме може нешто и значити) као и шта ће се приказивати код карактеристика. Промена неке карактеристике објекта одмах се приказује на селектованом објекту и задржава се све док се програмски или новом променом на овом листићу не дефинише њена другачија вредност. Ако се промена карактеристике изврши у програму (када се програм стартује) онда она није трајна, тј. поновним стартовањем истог програма карактеристике објеката се постављају на оне које су претходно дефинисане у листићу ***Properties***.

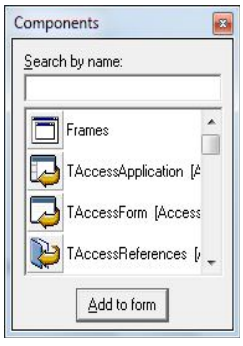


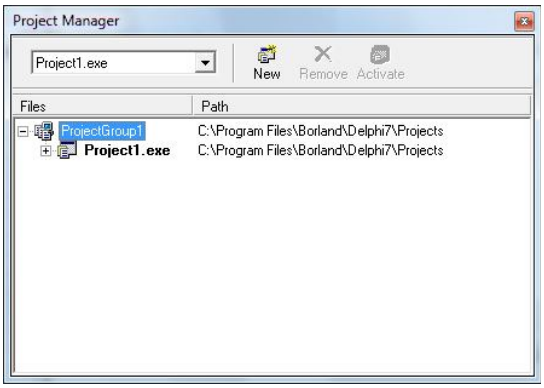
Осим ових, стандардно отворених прозора на радној површини делфија могу се појавити и многи други помоћни прозори. Погледајмо (и опишимо) сада само неке од њих:

To-Do List прозор приказује редослед одвијања операција код извршења програма.

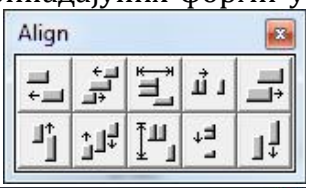
Components (претраживач компоненти је корисна алатка ако знамо назив објекта, али не знамо на којој палети се он налази);





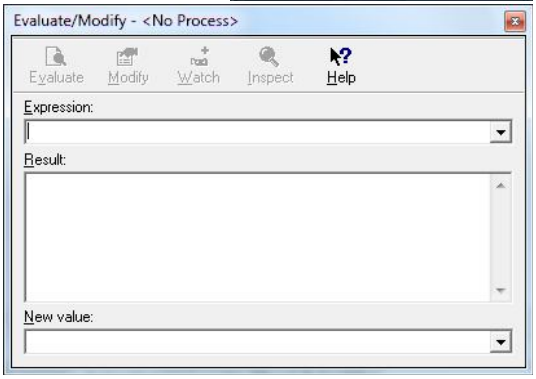


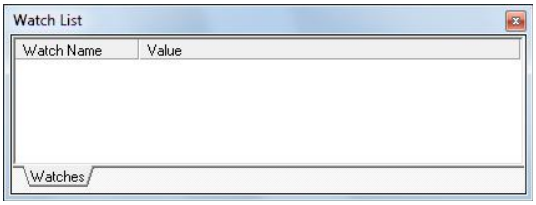
Project Manager је прозор у коме се виде групе пројеката и пројекти у оквиру групе пројеката и омогућава навигацију, кретање кроз пројектне датотеке. Овај прозор можемо користити и код додавања или брисања пројеката из групе као и за активацију пројекта у оквиру групе. Везано за пројекат, помоћу њега се додају, бришу, чувају и копирају датотеке које припадају том пројекту. Овде се могу видети називи свих јунита и припадајућих форми у оквиру пројекта или групе пројеката.



Align - помаже код уређења положаја објеката, леви клик миша на одговарајуће дугме у овом прозору док су селектовани неки објекти на форми помериће неке у смеру стрелице на дугмету до постизања жељеног положаја).

Осим поменутих прозора могу се отварати и други. На пример, разни ***Debug*** прозори отварају се, на захтев, код превођења програма, у њима се исписују евентуалне грешке у програмском коду, у прозору ***Evaluate/Modify*** могу се исписивати и мењати вредности променљивих у појединим фазама програма, у прозору ***Watch*** могу се исписивати вредности променљивих или израза у току извршавања програма и слично што програмеру може помоћи да тестира понашање и отклони недостатке програма. Ове прозоре нећемо сада објашњавати јер нису неопходни за рад на почетку дружења са делфијем.





Уз програмски пакет ***Delphi*** иде детаљно упутство за радознале и оне који би да науче о овом пакету више него што је обимом ове књиге предвиђено. Упутство је прилагођено раду под оперативним системом ***XP***, а да би се користило под новијим оперативним системом треба инсталирати програмски додатак са сајта његовог произвођача (***www.microsoft.com***).

Празан пројекат.

Отварањем делфи апликације отворили смо празан пројекат. И овакав пројекат је виндоуз апликација. Може се покренути (након компајлирања). Има све карактеристике апликације: радни прозор, наслов апликације, системски мени са одговарајућим тастерима. Радни прозор се може премештати, можемо му мењати димензије и облик, али ништа не ради (јер нисмо написали програмски код који би вршио неку активност).

Основне датотеке које садржи сваки делфи пројекат су:

Project1.cfg - конфигурацијски документ који чува параметре пројекта неопходне за рад самог делфија у току израде апликације;

Project1.dpr - ***(Delphi Project)*** датотека - основни програмски код који генерише сам делфи и не би је требало дирати без превелике потребе;

Project1.res - ресурсна датотека, садржи различите ресурсе које користи апликација коју креирамо, дефинитивно је не треба уклањати;

Project1.dof - садржи тренутно важеће параметре за опције у пројекту као што су компајлерски и линкерски параметри, директоријуми, директиве и командни параметри;

Unit1.pas - ***(Delphi Source File)***, изворни код делфи/паскалског програма;

Unit1.dfm - ***(Delphi Form)***, датотека која садржи основне параметре извршног прозора апликације коју креирамо. То је текстуална датотека коју делфи интерпретира графички.

Осим ових могу се јавити и још неке датотеке које се аутоматски креирају (***Unit1.dcu, Unit1.ddp*** и друге), а компајлирањем програма формира се и ***Project1.exe*** - извршна датотека.

Име пројекта се аутоматски додељује - ***Project1***. Име пројекта може се, по жељи, променити, али то није обавезно, ако усвојимо договор да се сваки пројекат чува у засебном фолдеру. Ово ће бити и име извршне верзије програма, па ако се озбиљно желимо бавити писањем програма, онда би требало променити га. Сада ћемо видети структуру сваког програма.

Отвориће мо датотеку ***Project1.dpr*** која се може отворити било којим едитором текста, на пример, помоћу ***Notepad***-а, само пажљиво, да се не би десило да датотеку придружимо овом едитору, па да је касније не можемо, аутоматски, отворити помоћу делфијевог едитора:

```
program Project1;
uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};
{$R *.res}
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Пројект се састоји из јунита и форме. Јунит (***Unit1.pas***) на почетку рада изгледа овако:

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs;
type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
end.
```

Празна форма са предефинисаним димензијама и изгледом и са натписом ***Form1*** дефинише се као текст датотека, неки атрибути могу имати и другачије вредности у зависности од карактеристика рачунара на коме се ради, (датотека ***Unit1.dfm***):

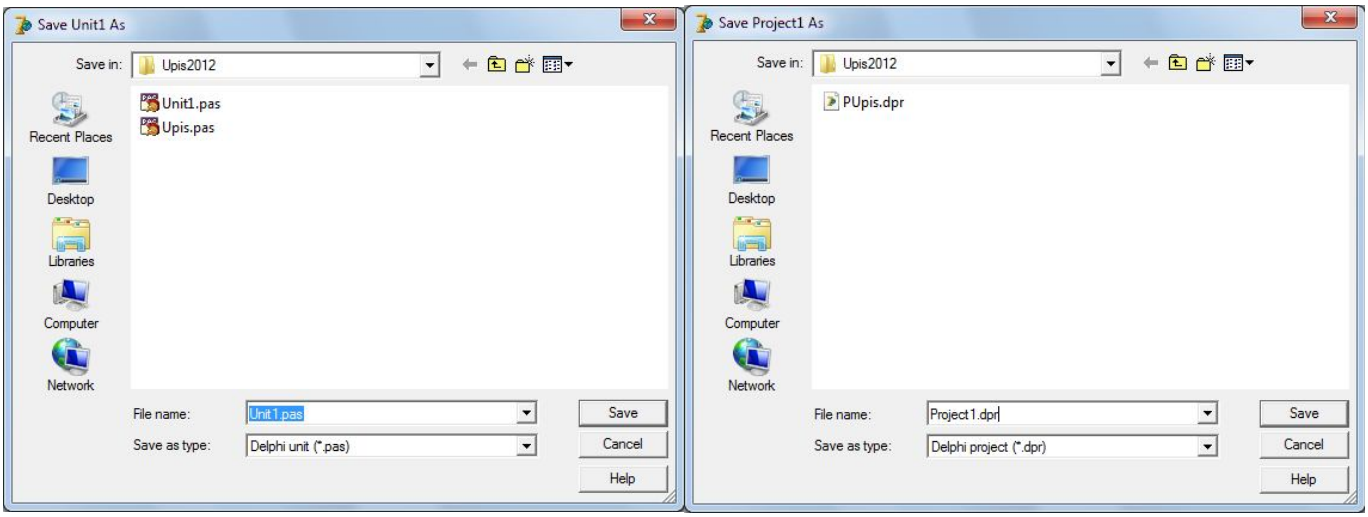
```
object Form1: TForm1
  Left = 192
  Top = 114
  Width = 870
  Height = 640
  Caption = 'Form1'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  PixelsPerInch = 96
  TextHeight = 13
end
```

Покретање ове празне, као и било које друге апликације, остварује се избором команде ***Run*** из одговарајућег менија, кликом на тастер у облику зеленог троуглића (*плеј* дугме) или коришћењем пречице - функцијски тастер ***F9***. Пре покретања апликације, добро је сачувати је како нам се не би десило да изгубимо свој *труд* у случају непланираног и неконтролисаног напуштања делфи апликације било услед грешке у програмском коду, било из неког другог, спољашњег разлога (*код новијих оперативних система, нарочито ако су 64-воробитни, програм је обавезно сачувати да би се могао покренути*). Опште правило у раду са рачунаром је да би требало свој рад чувати често и у фази израде и касније код сваке измене. Учесталост чувања се, обично, одређује обимом посла, тј. кад год урадимо нешто што не бисмо желели да радимо још једном, колико год то било, треба то и сачувати. Празан пројекат је полазна основа у изградњи било које апликације. Наравно, када радимо неки пројекат који мање или више личи на неки претходни, не морамо кренути од празног пројекта и губити време на подешавање истих карактеристика, писање истих или сличних програмских кодова, већ можемо искористити постојећи. У таквом случају, одмах по отварању пројекта који желимо да искористимо за креирање новог потребно је сачувати га са другим именом, како не бисмо изгубили претходни.

Чување и отварање пројекта.

Једна од важнијих ствари које морамо научити пре писања било ког програмског кода је како сачувати свој пројекат. Пошто се сваки пројекат састоји из већег броја датотека морамо, пре чувања пројекта, креирати нови фолдер, па тек онда у њега сачувати свој пројекат. Фолдер се може креирати пре отварања новог пројекта или у току рада на новом пројекту. Објаснићемо други начин (јер се претпоставља да је први познат од раније).

Из менија ***File*** изабраћемо команду ***Save All***. Отвориће се прозор за дијалог ***Save Unit1 As***.

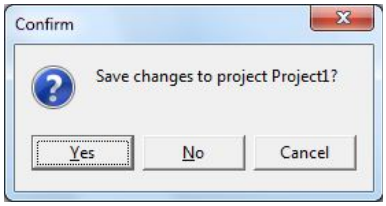


Када први пут чувамо пројекат, најпре ћемо креирати нови фолдер кликом на сличицу фолдера са звездицом у горњем десном углу или десним кликом у празни део прозора и избором опције ***New -> Folder***. Фолдеру треба дати име које ће нас подсећати на проблем који се тим програмом решава овим пројектом (на пример: ***ResavanjePravouglogTrougla***). Име фолдера не треба да садржи размаке, интерпункцијске знаке ни наша слова (сва ћирилична ни латинична слова ш, ђ, ч, ћ, ж). У новим верзијама оперативног система употреба ових карактера неће представљати никакав проблем, али због евентуалног преношења нашег програма на рачунаре са различитим верзијама оперативног система, треба поштовати препоруку). Затим треба отворити тај фолдер, двокликом на његову иконицу или кликом на дугме ***Open*** (које ће се исцртати уместо дугмета ***Save***). У оквиру означеном са ***File name:*** треба уписати име програма (*<u>ово није обавезно, тј. име јунита не мора се мењати, може се оставити понуђено **Unit1**, јер ће у овом фолдеру бити само један јунит</u>*). Код давања имена јуниту не би требало користити размаке, знаке интерпункције, латинична слова ш, ђ, ч, ћ, ж нити наша ћирилична слова. Када смо дали име јуниту кликнућемо на дугме ***Save***.

Сада се отвара нови дијалошки прозор ***Save Project1 As*** у којем треба уписати име пројекта (ово, такође, није обавезно, тј. име пројекта не мора се мењати, може се оставити понуђено ***Project1***, јер ће у овом фолдеру бити само један задатак, односно, само један пројекат). За име пројекта важе исте напомене као и за име јунита, а име пројекта не би требало да буде исто као име јунита.

Код сваке значајне измене програма, без обзира да ли смо на форму додали нове компоненте или смо мењали програмски код, треба сачувати пројекат помоћу команде ***Save All*** при чему се претходни прозори више неће отварати (само ћемо себи уштедети мало времена и много нерава у случају да се нешто непредвиђено деси са рачунаром, па да писање програм не морамо да почнемо од почетка).

У случају да нисмо отворили нови пројекат, већ користимо постојећи за креирање новог (јер има случну форму или се део програмског кода преклапа са новим или из било којег другог разлога) применом команди ***Save*** или ***Save All*** преко старог пројекта сачуваћемо нови. На тај начин бисмо изгубили претходни пројекат. Да се то не би десило, у оваквом случају поступак чувања пројекта биће другачији. Из менија ***File*** изабраћемо прво команду ***Save As***. Отвориће се прозор за дијалог као на првој слици и сачуваћемо јунит како је раније описано (креираћемо нови фолдер и у њему са истим или промењеним именом сачувати јунит). Затим ћемо из менија ***File*** изабрати команду ***Save Project As***. Отвориће се прозор за дијалог као на другој слици и сачуваћемо пројекат како је описано.



Пројекат затварамо изабором команде ***Close All*** из менија ***File***. Ако претходни пројекат нисмо сачували програм ће нас упозорити на то и дати нам могућност да измене сачувамо пре затварања пројекта.

Нови празан пројекат отвара се изабором команде ***New -> Application*** из менија ***File***. И у овом случају, ако је претходно био отворен неки пројекат који није сачуван отвориће се исти дијалошки прозор упозорења који ће нам дати нам могућност да измене сачувамо пре затварања претходног и отварања новог пројекта.

Отварање постојећег пројекта остварује се тако што се из менија ***File*** изабере команда ***Open Project***. Ако претходни пројекат нисмо сачували поново ће се отворити претходно описани дијалошки прозор, упозорити нас на то и дати нам могућност да измене сачувамо пре отварања новог пројекта. Након тога ће се отворити прозор за дијалог као на слици. Треба пронаћи фолдер из којег желимо да отворимо пројекат и отворити га двокликом. Двокликом на само име пројекта или обележавањем простим кликом миша и кликом на дугме ***Open*** отвориће се жељена апликација. У случају да се апликација не отвори онда претходни поступак чувања пројекта није испоштован или је накнадно обрисана нека од важних датотека. У неким од таквих случајева то се може решити активностима у оперативном систему, али у већини случајева то значи да ту апликацију не можемо користити за измене или креирање нове. Зато је од изузетне важности научити правилно чување пројекта.

Отварање неког постојећег пројекта могуће је и из оперативног система, пре покретање делфи пакета, тако што се отвори одговарајући фолдер и двокликом изабере пројекат који се отвара. Оперативни систем ће аутоматски покренути делфи апликацију и у њој отворити изабрани пројекат. У овој верзији делфија нема могућности рада са више пројеката истовремено. У једном тренутку могу бити отворена два или више програма, али то подразумева да смо два или више пута покренули програмски пакет делфи. Код неких врста програма, то може значајно успорити или сасвим онемогућити рад на развоју апликације. Ако желимо да гледамо решење једног задатка и на основу њега решавамо други, сличан, треба отворити јунит у неком текст едитору, на пример у ***Notepad***-у, одакле га можемо или копирати или само гледати решење.

Форма и подешавање њених својстава.

Почетни изглед форме описан је у ***Unit1.dfm*** датотеци и додавањем нових елемената на форму или мењањем карактеристика већ постојећих, ова датотека се допуњава. Ово је текстуална датотека и њен садржај прегледати у неком едитору. Садржај датотеке може се и мењати у едитору без последица по рад програма уколико не направимо неку грешку. Док је програм отворен у делфију не би требало мењати садржај ове датотеке отворене неком у другом едитору. Својства форме се могу, у зависности од самог својства, подешавати директно, помоћу миша (ово се односи само на величину и положај форме) или уписивањем одговарајућих карактеристика на листићу ***Properties*** прозора ***Object Inspector*** (односи се на готово сва својства форме), а могу се поједина својства подешавати и програмски, тј. у фази извршавања пројекта форма може мењати изглед и друге карактеристике. Најчешће се мењају следећа својства:

- Спољашње димензије форме, односно, ширина (***Width***) и висина (***Height***) прозора, али и унутрашње димензије прозора ***ClientWidth*** и ***ClientHeight***, при чему се не могу истовремено користити и спољне и унутрашње димензије јер су зависне једне од других.
- Позиција на екрану, односно, растојање од левог (***Left***) и од горњег (***Top***) руба екрана, а осим са ове две, може се дефинисати и карактеристиком ***Position*** (са вредностима *poDefault,*

poDefaultPosOnly, poDefaultSizeOnly, poDesigned, poDesktopCenter, poMainFormCenter, poOwner FormCenter, poScreenCenter) и карактеристиком ***Align*** (са вредностима *alBottom, alClient, alCustom, alLeft, alNone, alRight, alTop*) при чему ова последња појединим вредностима онемогућава дефинисање ширине или висини прозора апликације.

- Назив прозора (***Caption***), односно, текст који ће се исписивати у ттраци на врху прозора.
- Боја радне површине (***Color***), избор се врши између 20 стандардно понуђених боја или двокликом на ову карактеристику отварамо палету са које можемо да креирамо једну од преко 16 милиона боја.
- Стил рубова прозора (***BorderStyle***), односно, дозвољавамо или забрањујемо кориснику да манипулише позицијом и величином прозора (*bsDialog, bsNone, bsSingle, bsSizeable, bsSizeToolWin, bsToolWindow* су дозвољене вредности, друга и четврта дозвољавају пуну слободу кориснику, а остале делимично или потпуно ограничавају измену прозора апликације).
- Дебљина оквира (***BorderWidth***), карактеристика практично дефинише део радне површине уз рубове прозора који се не могу користити за постављање објеката (нешто слично маргинама у текст процесорима).
- Контролна дугмад прозора ***biSystemMenu, biMinimize, biMaximize, biHelp*** се могу видети на горњем рубу прозора или не, што се подешава у оквиру карактеристике ***BorderIcons***.
- У оквиру карактеристике ***Font*** дефинишемо основни фонт који ће се аутоматски придруживати свим објектима које касније поставимо на форму.
- Стил форме (***FormStyle***) дефинише да ли је прозор апликације зависан или не од неког другог прозора и да ли ће бити увек на врху свих отворених прозора, могуће вредности су *fsMDIChild, fsMDIForm, fsNormal, fsStayOnTop*.
- Облик прозора (***WindowState***) са вредностима *wsMaximized, wsMinimized, wsNormal*.

Додавање компоненти форми.

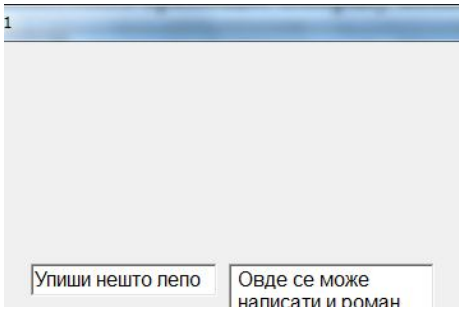
Додавање компоненти форми је врло једноставно. Пронађемо жељену компоненту на некој од палета, кликнемо левим тастером миша на њу, а затим кликнемо на одговарајуће место на форми. Не постоји могућност да додамо истовремено више компоненти, већ се овај поступак понавља жељени број пута. Када смо кликнули на неку компоненту она је селектована и клик на форму је тамо и преноси. Ако смо се предомислили и не желимо ту компоненту, већ неку другу, довољно је клинути левим тастером миша на ту другу компоненту и на тај начин аутоматски деселектовати прву и селектовати другу. Ако не желимо ни једну компоненту у овом тренутку, треба кликнути на прву сличицу на палети (стрелица на горе-лево) и на тај начин деселектовати претходно изабрану компоненту.

Компонента која је постављена на форму уклања се са форме селектовањем и притиском тастера ***Delete*** на тастатури или десним кликом на компонету, па избором команде ***Delete*** из подменија ***Edit*** падајућег менија који смо отворили. Ако смо додали једну компоненту на форму, а желимо још неколико таквих, можемо понављати поступак додавања потребан број пута или додату компоненту селектовати и ископирати потребан број пута. Копирање се може остварити десним кликом миша на компоненту и подменија ***Edit*** падајућег менија који се отвара или селектовањем компоненте помоћу миша и коришћењем пречица помоћу тастатуре (***Ctrl+C*** за копирање објекта у меморију и ***Ctrl+V***, потребан број пута, за постављање копија објекта на форму). Компоненте се на форму могу додати и копирањем са неке друге форме као и програмски (програмер може дефинисати појављивање објекта на форму, али се за овакав начин додавања компоненти захтева виши ниво познавања програмирања у објектно оријентисаним програмским језицима).

Компонента у жижи.

Компонента може бити у жижи и тада јој се могу мењати карактеристике. Да бисмо знали која је компонента активна довољно је да погледамо на врх прозора ***Object Inspector***-а. Њено име је исписано у првом оквиру. Активна компонента је означена и са осам црно обојених квадратића, на угловима и срединама страница.

У извршној верзији апликације, такође, битно је знати која је компонента тренутно активна, односно, у жижи. Када се програм покрене, нема ***Object Inspector***-а, а нема ни црних тачкица на рубовима компоненте и тада активну компоненту можемо препознати по неким другим карактеристикама које зависе од врсте компоненте. Ако се ради о текст компоненти, она је у жижи ако у њој трепери курсор. Ако се ради о графичкој компоненти, она је у жижи ако је око натписа на њој видљив испрекиданим линијама исцртан правоугаоник. На слици је чекбокс компонента у жижи (што препознајемо по тачкицама исцртаном оквиру око текста).



У једном тренутку, само једна компонента може бити у жижи. Жижу са компоненте на компоненту премештамо тастером ***Tab*** на тастатури или кликом миша на изабрану компоненту. Неке компоненте не могу бити у жижи. Да би компонента могла бити у жижи мора поседовати карактеристику ***TabStop***. Карактеристиком ***TabStop*** дефинишемо редослед постављања компоненти у жижу. Овај, *природни*, редослед може се пореметити кликом мишем на одговарајућу компоненту. Добро организовани програм има редослед доласка компоненти у жижу који прати алгоритамско решење.

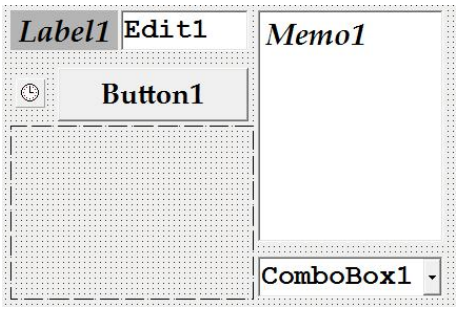
Једноставне компоненте

Селектовањем неке компоненте, у ***Object Inspector***-у на листићу ***Properties*** можемо изабрати и подесити одговарајућу карактеристику те компоненте. Карактеристике компоненти могу се подешавати и из саме апликације у току извршавања, ако се у програмски код угради одговарајућа рутина. Већина карактеристика компоненти може се подесити из апликације (у фази извршавања програма) простом наредбом:

компонента.карактеристика:=вредност;

Најчешће коришћене компоненте су:

- ***Label***
- ***Edit***
- ***Button***
- ***Memo***
- ***Timer***
- ***Image***



Натпис (Label)

Label, натпис је компонента која омогућава једносмерну комуникацију на релацији програм-корисник, односно, помоћу ње се кориснику програма саопштава нешто; корисник не може да мења садржај лабеле нити на било који начин да утиче на њен изглед и величину осим када је то програмом дефинисано, односно, када је испрограмирано да се на неки начин промене карактеристике задате у фази дизајнирања програма. Када се програм извршава, ова компонента не може бити у жижи. Компонента се налази на листићу ***Standard*** на четвртом месту (велико слово А). Као и сваки други објекат, лабела има одређене особине које се могу подешавати како би се постигли одговарајући ефекти. Најчешће ћемо подешавати следећа својства:

Align - положај лабеле у односу на рубове форме
Alignment - уређење текста у оквиру лабеле
AutoSize - аутоматско подешавање величине лабеле

Color - боја лабеле
Caption - текст који је исписан на лабели
Font - врста слова, начин исписа, величина, боја и ефекти
Height - висина објекта
Layout - вертикално уређење текста у лабели
Left - растојање објекта од левог руба форме
Name - име објекта
Top - растојање објекта од врха форме
Transparent - провидност позадине лабеле
Visible - видљивост објекта
Width - ширина објекта
WordWrap - пренос текста на лабели у нови ред

Оквир за уношење и приказивање текста (Edit)

Edit, оквир за текст је компонента која омогућава двосмерну комуникацију на релацији програм-корисник, односно, помоћу ове компоненте програм може да саопштава поруке кориснику и корисник може да се обраћа програму уносом одговарајућих података; кориснику се програмом може дозволити или забранити измена садржаја ове компоненте. Када је ова компонента у жижи у њеном оквиру трепери курсор. Компонента се налази на листићу ***Standard*** на петом месту (мала слова аб). Најчешће ћемо подешавати следећа својства:

CharCase - исписивање малих и великих слова у едиту
Color - боја едита
Enabled - да ли је дозвољено приступити едиту
Font - врста слова, начин исписа, величина, боја и ефекти
Left - растојање објекта од левог руба форме
MaxLength - максимални број карактера који се може уписати
Name - име објекта
ReadOnly - да ли је дозвољено мењање садржаја едита
TabOrder - редни број објекта
Text - садржај едита
Top - растојање објекта од врха форме
Visible - видљивост објекта
Width - ширина објекта

Дугме (Button)

Button, тастер или дугме је компонента којом се корисник обавештава о могућности извршења неке акције у програму за коју је потребна конкретна активност корисника, односно, кликом мишем на ову компоненту корисник програму саопштава да жели да се нека акција изврши; корисник нема права да мења изглед и функцију ове компоненте јер је то програмом дефинисано. Компонента се налази на листићу ***Standard*** на седмом месту (сличица са словима ОК). Најчешће ћемо подешавати следећа својства:

Caption - текст који је исписан на тастеру
Enabled - да ли је дозвољено приступити тастеру
Font - врста слова, начин исписа, величина, боја и ефекти
Height - висина тастера
Left - растојање објекта од левог руба форме
Name - име објекта
TabOrder - редни број објекта
Top - растојање објекта од врха форме
Visible - видљивост објекта
Width - ширина објекта
WordWrap - пренос текста на тастеру у нови ред

Мемо поље (Memo)

Мемо поље (***Memo***) је компонента којом се омогућава унос и приказивање вишередног текста. Ова компонента представља неки облик елементарног едитора текста без могућности напредног уређивања текста. Компонента се налази на листићу ***Standard*** на шестом месту (сличица која подсећа на лист папира из свеске). Најчешће ћемо подешавати следећа својства:

Align - положај објекта у односу на рубове форме
Alignment - уређење текста у оквиру поља
Color - боја позадине поља за текст
Enabled - да ли је дозвољено приступити пољу
Font - врста слова, начин исписа, величина, боја и ефекти
Height - висина објекта
Left - растојање објекта од левог руба форме
Lines - садржај мемо поља
MaxLength - максимални број карактера који се може уписати
Name - име објекта
ReadOnly - да ли је дозвољено мењање садржаја мемо поља
TabOrder - редни број објекта
Top - растојање објекта од врха форме
Visible - видљивост објекта
Width - ширина објекта
WordWrap - пренос текста у нови ред

Часовник (Timer)

Компонента која омогућава извршавање програмских елемената периодично, тј. у тачно одређеном тренутку у односу на њено активирање. Може да се користи код симулација кретања као и код других аудио и визуелних ефеката који треба да се појављују у једнаким временским интервалима. Једна компонента може да контролише само једну програмску целину. Дозвољена је употреба произвољног броја ових компоненти. Пошто су на извршном екрану невидљиве, ове компоненте неће нарушити уређење излазне форме апликације колико год да их је активирано. Компонента се налази на листићу ***System*** на првом месту (часовник). Подешаваћемо следећа својства:

Enabled - да ли је тајмер активан, укључен
Interval - период у милисекундама који активира неки догађај
Name - име тајмера

Оквир за графички објекат (Image).

Излазни екран (лице) апликације може бити ефикасније ако се додају различити графички објекти, цртежи или слике. Да би се омогућило такво опремање форме потребно је искористити компоненту оквир за слику - ***Image***. Компонента се налази на листићу ***Additional*** на шестом месту (плави пејзаж). Најчешће ћемо подешавати следећа својства:

Align - положај слике у односу на рубове форме
AutoSize - аутоматско подешавање величине оквира слике
Center - центрираност слику у оквиру
Enabled - да ли је дозвољено приступити оквиру слике
Height - висина оквира слике
Left - растојање од левог руба форме
Name - име оквира за слику
Picture - слика која се приказује у оквиру
Stretch - да ли слика испуњава оквир за слику
Transparent - провидност оквира за слику
Top - растојање од врха форме
Visible - видљивост оквира са сликом
Width - ширина оквира за слику

Компоненте избора

Коришћење програма подразумева унос неких почетних података чијом се обрадом добија жељени резултат. Унос података може се остварити слободним уносом, тј. корисник уноси било шта или се кориснику унос ограничи на неколико података од којих треба да изабере један. У првом случају се користе едити и мемо поља, у другом се користе компоненте избора. Најчешће коришћене компоненте избора су:

- ***RadioButton***
- ***CheckBox***
- ***ListBox***
- ***ComboBox***

Радио-дугме (RadioButton)

Радио дугме је компонента која дозвољава унос једне вредности од понуђених. Ако постоји само једно дугме, корисник може изабрати тај податак или га не изабрати. Много чешће се радио дугмад групишу, па се кориснику дозвољава шири избор, тј. од више понуђених података може изабрати један. Контрола избора је на програму, тј. избором једног податка аутоматски се поништава претходни избор. На слици је приказана радио група којом би се могао унети одговор на питање: *На којој планети живимо?*. Могуће је изабрати само један од понуђених одговора. Радио дугмад могу бити распоређена хоризонтално, верикално или у облику табеле. Најчешће се подешавају следеће карактеристике:

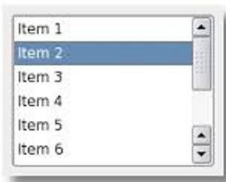
Caption	- наслов (назив опције)
Color	- боја позадине
Top	- удаљеност од горње ивице форме
Left	- удаљеност од леве ивице форме
Width	- ширина, ако је краћа од дужине наслова део наслова се неће видети
Height	- висина компоненте
Font	- врста фонта који се користи за испис опција
Checked	- дефинише да ли је опција селектована или не

Оквир за потврду (CheckBox)

Оквир за потврду је компонента која дозвољава унос једног или више података од понуђених. Као и у претходном случају, може постојати више података при чему се кориснику допушта да изабере један или више података. На слици је приказана група оквира за потврду којом би се могао унети одговор на питање: Коју бисте планету волели да посетите. Могуће је изабрати један или више од понуђених одговора. Оквири могу бити распоређени хоризонтално, верикално или у облику табеле. Најчешће се подешавају следеће карактеристике:

Caption	- наслов (назив опције)
Color	- боја позадине
Top	- удаљеност од горње ивице форме
Left	- удаљеност од леве ивице форме
Width	- ширина, ако је краћа од дужине наслова део наслова се неће видети
Height	- висина компоненте
Font	- врста фонта који се користи за испис опција
Checked	- дефинише да ли је опција селектована или не

Оквир с листом (ListBox)



Оквир са листом је компонента помоћу које можемо изабрати више од понуђених одговора, опција. Број понуђених опција није ограничен. Вишеструки избор омогућен је комбинованом употребом миша и тастера ***Shift*** или ***Ctrl***. Помоћу првог тастера селектујемо све одговоре од првог селектованог до другог селектованог, а помоћу другог тастера можемо селектовати одговоре који нису један поред другога. Ако на листи има више

одговора него што може стати у оквир са десне стране се појављује клизач којим се могу откривати одговори који се не виде. Најчешће се подешавају следеће карактеристике:

Color - боја позадине
Top - удаљеност од горње ивице форме
Left - удаљеност од леве ивице форме
Width - ширина оквира
Height - висина оквира
Font - врста фонта који се користи у оквиру
Border Style - врста ивице оквира
Items - опције, ставке у листи
MultiSelect - дозвољава или забрањује вишеструко селектовање опција листе
ItemIndex - редни број опције у листи (бројање почиње од 0)
Sorted - дефинише да ли ће опције бити абецедно уређене

Комбиновани оквир (ComboBox)

Комбиновани оквир је компонента која се састоји из оквира у који се може уписати податак и стрелице на крају оквира којом се отвара контејнер у који су смештени подаци које је дозвољено унети. Контејнер може бити ограничавајући или информативни. У првом случају кориснику се не дозвољава да унесе ништа осим понуђеног, а у другом случају корисник може изабрати нешто од понуђеног или уписати нешто друго. Падајућа листа може бити сакривена и по потреби се отвара, а може бити и стално видљива. Ако број понуђених одговора не може да стане у предвиђени оквир падајуће листе са десне стране се појављује клизач којим се могу откривати тренутно невидљиви одговори. Користи се нарочито у ситуацијама када је врло битан начин исписа податка који се уноси или ако су подаци који се уносе јако компликовани, а могућности (број различитих уноса) су ограничене. Компонента се налази на листићу ***Standard*** на једанаестом месту (сличица едита са спуштеном листом за избор). Најчешће ћемо подешавати следећа својства:

AutoComplete- аутоматско исписивање податка који одговара унетом првом слову (или групи слова)
AutoDropDown - аутоматско спуштање листе када објекат постане активан
CharCase - исписивање малих и великих слова у оквиру
Color - боја позадине поља за текст
Enabled - да ли је дозвољено приступити оквиру
Font - врста слова, начин исписа, величина, боја и ефекти
Height - висина објекта
ItemHeight - висина податка са листе
ItemIndex - редни број податка са листе који је изабран
Items - листа података
Left - растојање објекта од левог руба форме
MaxLength - максимални број карактера који се може уписати
Name - име објекта
Sorted - да ли су подаци на листи уређени
TabOrder - редни број објекта
Text - податак који је исписан у главном оквиру
Top - растојање објекта од врха форме
Visible - видљивост објекта
Width - ширина објекта

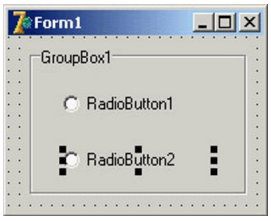
Контејнерске компоненте

Контејнерске компоненте се користе за визуелно издвајање и груписање компоненти које су повезане неким поступком, особином или чине посебну целину у програму. Овим компонентама се постиже боља прегледност маске за унос и приказивање података и ефикаснија контрола компоненти из програма.

Овде ћемо поменути две компоненте:

- ***GroupBox*** и
- ***Panel.***

Оквир за групу (GroupBox)



Када желимо да неколико истих или различитих компоненти повежемо јер су део једног процеса или акције која треба да се изврши онда користимо компоненту оквир за групу. Оквир за групу има своје име, а димензије могу да се мењају како би се компоненте које се повезују распоредиле тако да следе природни ток алгоритма. Могу се повезивати разнородне компоненте, радио дугмад, дугмад, едити, лабеле, чек боксови и други. Доступност компоненти из програма можемо да контролишемо преко оквира за групу. На тај начин можемо једном командом да искључимо све компоненте у групи или да их привремено уклонимо са форме и у погодном тренутку поново вратимо. Специјална врста оквира је ***RadioGroup*** коју чине само радио дугмад. Најчешће се подешавају следеће карактеристике:

Caption - наслов који објашњава функцију компоненти унутар оквира
Color - боја позадине
Top - удаљеност од горње ивице форме
Left - удаљеност од леве ивице форме
Width - ширина оквира
Height - висина оквира
Font - врста фонта који се користи у оквиру (ако је дефинисан, може да се преноси на све компоненте које се стављају на оквир, а може и да се наслеђује од компоненте на којој се налази)

Плоча (Panel)

Када желимо да постигнемо специјалне ефекте, да маске за унос и приказивање података начинимо ефектнијим користимо панеле. Панел може да се распростре по читавој форми или да заузима само део ње, може да има рубове који дају утисак тродимензионалности, могу да се покрећу, да се појављују или нестају, да буду доступни или не. Све компоненте које су смештене на једном панелу деле судбину панела. Панели се могу да користе увек, али се најчешће користе код мало сложенијих програма у којима има јако пуно компоненти и опција које не би требале истовремено да се приказују јер би форма била пренатрпана. А понекад се користе и за добијање специјалних ефеката, на пример, код приказа слика, помоћу панела, можемо додати рам за слику. Ако су два или више панела истовремено на форми могу се неке компоненте визуелно раздвајати и груписати по својим функцијама и употреби. Најчешће се подешавају следеће карактеристике:

Caption - наслов (најчешће на панелу нема наслова)
Color - боја позадине
Top - удаљеност од горње ивице форме
Left - удаљеност од леве ивице форме
Width - ширина панела
Height - висина панела
Font - врста фонта који се користи на панелу (ако је дефинисан, може да се преноси на све компоненте које се стављају на панел, а може и да се наслеђује од компоненте на којој се налази)
Align - позиција, поравњање панела у односу на форму
BevelInner - изглед унутрашњег дела оквира панела
BevelOuter - изглед спољашњег дела оквира панела
BevelWidth - дебљина оквира панела
BorderStyle - врсте ивица панела
BorderWidth- дебљина ивице панела

Визуелно издвајање компоненти на форми може се постићи и коришћењем компоненте ***Bevel*** (оквир), али ово није контејнерска компонента јер само уоквирава компоненте, али не

дефинише њихову припадност и не може директно да преноси своје карактеристике на уоквирене компоненете.

Догађаји компоненти и обрада догађаја.

У објектно оријентисаном програмирању тежиште је на објектима (као што и само име каже) и оно што може да се догоди са или око објеката, односно различити догађаји. Шта може бити догађај? Све што урадимо директном акцијом помоћу миша или тастатуре, али и све што је последица извршавања програма или неког његовог дела, може бити догађај. Клик мишем или само притисак или отпуштање тастера на мишу, прелазак показивача миша преко објекта, превлачење објекта, промена димензија, притисак неког тастера на тастатури, унос неке вредности, појава или нестајање објекта и друго, све су то неки догађаји који се могу *осетити* или не у зависности од врсте објекта. Задатак програмера је да одлучи на које догађаје ће, поједини објекти који су саставни део апликације реаговати, односно, на које догађаје ће бити *осетљиви* и какво ће бити њихово понашање, тј. шта ће се дешавати. Ако ниједан догађај није обрађен, онда написани програм неће радити ништа, тј. он је бескористан.

Неки од важнијих могућих догађаја објеката су:

- ***OnActivate*** - када објекат постане активан
- ***OnClick*** - када се мишем кликне изнад објекта
- ***OnClose*** - када се објекат затвори
- ***OnCreate*** - када се објекат креира
- ***OnDblClick*** - када се двапут кликне мишем изнад објекта
- ***OnDeactivate*** - када објекат престане да буде активан
- ***OnDestroy*** - када се објекат избрише
- ***OnDragDrop*** - када се објекат превуче и отпусти
- ***OnDragOver*** - када се објекат довлачи
- ***OnHide*** - када се објекат сакрије, тј. објекат још увек постоји, али се не види.
- ***OnKeyDown*** - када се на тастатури притиска било који тастер
- ***OnKeyPress*** - када је на тастатури откуцан неки тастер
- ***OnKeyUp*** - када се на тастатури отпушта неки тастер
- ***OnMouseDown*** - када се миш креће на доле преко објекта
- ***OnMouseMove*** - када се миш креће преко објекта
- ***OnMouseUp*** - када се миш креће на горе преко објекта
- ***OnMouseWheel*** - када се покрене точкић на мишу
- ***OnMouseWheelDown*** - када покрећемо точкић на мишу према себи
- ***OnMouseWheelUp*** - када покрећемо точкић на мишу од себе
- ***OnResize*** - када се објекту мења величина

Наравно, не могу сви објекти реаговати на све догађаје. Неки објекти су *осетљивији*, тј. могу да реагују на више догађаја, а неки, опет, нису, тј. реагују на врло мали број догађаја. Пажљивим одабиром објеката и дефинисањем њихове *осетљивости* могу се направити врло атрактивне апликације (наравно, да би то било успешно треба имати много искуства, знања, маште, воље и времена, али се исплати).

Код обраде догађаја, пошто неки могу бити сложени и састојати се из неколико других догађаја, мора се водити рачуна да једна реакција не поништава другу. Може се десити и да једна реакција производи други догађај, па се поништи претходна реакција или се програм заблокира. Зато треба бити јако пажљив и, пре писања неке реакције, добро размислити о свим последицама које реакција може произвести, а то је већ уметност програмирања.

Питања на која треба обратити пажњу

- Објектно оријентисано програмирање, основни концепти
- Изглед развојног окружења делфи пакета
- Елементи главног прозора делфи пакета
- Функције помоћних прозора и рад са њима
- ***ObjectTreeView***, карактеристике
- ***ObjectInspector***, карактеристике и функције
- Подешавање својстава форме пројекта
- Чување новог пројекта
- Отварање новог и постојећег пројекта
- Рад са палетом са компонентама
- Најчешће коришћене компоненте и њихова својства
- Додавање компоненти на форму пројекта
- Селектовање компоненте
- Врсте догађаја на које компоненте могу да реагују

Типови података

Основни елементи програмског језика

Азбуку програмског језика чине сви дозвољени знаци који се користе приликом кодирања програма и то су:

- велика и мала слова енглеске абецеде
- цифре декадног бројног система: 0,1,2,3,4,5,6,7,8,9
- специјални знаци: + - * / = ^ < > () [] {} . ,: ; ‘ # $

За математичка израчунавања се користе ознаке:

*	за множење
/	за дељење
+	за сабирање
-	за одузимање
( )	за промену приоритета извршења операција

У конструкцији програма се користе ознаке:

:=	у наредбама додељивања
.	ознака краја програма
;	ознака краја наредбе
#	**ASCII** кодови
$	хексадецимални бројеви
(* *) или	
{ }	коментар или директива преводиоцу
+	укључење директиве преводиоца
-	искључење директиве преводиоца
‘	почетак или крај карактера или стринга
,	граничник између две променљиве

За упоређивање или тестирање се користе

=	једнако
<	мање од
>	веће од
<=	мање или једнако
>=	веће или једнако
<>	различито

За структуру података се користе:

^	показивач или контролни кодови са тастатуре
(. .) или [ ]	индексирање и димензионисање низова
( )	листа параметара
.	граничник записа поља
. .	распон (интервал)

Резервисане речи су интегрални део језика и оне се не могу редефинисати ни користити као имена променљивих или константи у програму. Резервисане речи у делфију су:
And, Array, As, Asm, Begin, Constructor, Case, Class, Const, Div, Destructor, Dispinterface, Do, Downto, Else, End, Except, Exports, File, Finalization, Finally, For, Function, Goto, If, Implementation, In, Inline, Inherited, Initialization, Interface, Is, Label, Library, Mod, Nil, Not, Object, Of, Or, Out, Packed, Procedure, Program, Property, Raise, Record, Resourcestring, Repeat, Set, Shl, Shr, String, Then, Thread, Var, To, Try, Type, Unit, Until, Uses, Var, While, With, Xor

Стандардни идентификатори су речи дефинисане у програмском језику, али се могу и редефинисати, мада их је боље користити у изворном облику да не би долазило до забуне. На

пример, можемо дефинисати ***integer*** као скуп речи, али је нормалније оно што је дефинисано програмским језиком, да ***integer*** представља целе бројеве.

Граничници су симболи који служе за раздвајање појединих елемената језика. Дозвољени граничници у делфију су:

;	ознака за крај наредбе
,	крај идентификатора у набрајању
.	крај програма или поља у слогу
eol	(**end-of-line**, крај реда) не види се на екрану - означава се притиском тастера Ентер
//	коментар у једном реду
{ ... }	вишередни коментар
space	(празнина) означава се притиском размакнице

Програмска линија је низ идентификатора максималне дужине 255 карактера рачунајући и празнине. У паскалу, све иза 255-ог карактера се игнорише, у делфију програмски ред може бити и дужи од 255 карактера, али се тако записани програми тешко читају.

Структура програма

Најједноставнија дефиниција програма је:

Програм је запис алгоритма за решавање неког проблема на одговарајућем програмском језику.

Да би се написао програм, потребно је познавати проблем који се решава како би се могао сачинити алгоритам и познавати детаље програмског језика на којем ће се описати алгоритам. Основни елементи сваког програма су конструкције програмског језика које служе за описивање података и радњи - то су идентификатори.

- ***Наредба*** је основни елемент програма који описује неко дејство података или изазива акцију рачунара. Наредбе могу бити описне и извршне.
- ***Коментар*** је основна елементарна конструкција програма помоћу које се објашњавају поједини делови програма. Намењен је програмеру, а не кориснику. Добар програм има одговарајућу, добро одмерену подршку у облику коментара. Требало би да су сви кључни детаљи програма објашњени коментаром.

Делфи је наследио структуру програма из паскала. Паскал је програмски језик у коме се програми, осим функционалношћу и ефикасношћу, могу похвалити и лепотом. Наравно, сасвим је свеједно да ли сте написали леп или ружан програм ако он ради оно што је замишљено, али, кад већ можете, дозволите себи *луксуз* да програм визуелно као и програмским решењима буде леп. Визуелна лепота и склад програма нису бескорисни. При евентуалном тражењу грешака (а код великих и компликованих програма и најбољи програмери греше) много је лакше пронаћи грешку ако је програм уредно исписан, са издвојеним структурним целинама и пропраћен добрим и одговарајућим коментарима. Зато ћемо се трудити да научимо да пишемо лепе и корисне програме.

Сваки програм у паскалу састоји се из два дела: *заглавља* и *тела* програма. Они могу бити редуковани, али се не могу изоставити. ***Заглавље програма*** почиње именом програма и декларацијама променљивих и дефиницијама константи и других података који ће бити коришћени у програму:

```
PROGRAM MojPrviProgram;
...
```

Програмска линија са именом програма може се изоставити, али, чини се да програм много комплетније и лепше изгледа, ако се на почетку представи. *Назив програма* може бити произвољне дужине, али је важно напоменути да се у њему не могу налазити знаци интерпункције, наша слова нити размак. Најбоље је да се називом најкраће саопшти шта програм ради, са две до три речи које се раздвајају или тако што се пишу великим почетним и осталим малим словима или цртицом за подвлачење. На пример:

```
PROGRAM ProracunUbrzanjaSatelita; или
PROGRAM Proracun_Ubrzanja_Satelita;
```

Наравно, програмер може и сам да осмисли начин на који ће написати назив програма, али је овако читљивији.

У делфију су велика и мала слова равноправна тако да то да ли наредбе пишемо великим или малим или великим почетним, а осталим малим зависи само од укуса програмера и ефекта који се жели постићи када се само *баци око* на програм. Наравно, треба бити доследан у коришћењу неког начина записивања програма јер на тај начин помажемо и себи код тражења неких детаља или грешака у програму.

Део за декларације и дефиниције садржи:

- Одељак за декларација типова, почиње резервисаном речи ***type***.
- Одељак за декларација променљивих, почиње резервисаном речи ***var***.
- Одељак за дефинисање константи, почиње резервисаном речи ***const***.
- Одељак за дефинисање лабела, почиње резервисаном речи ***label***.
- Одељак за дефинисање кориснички дефинисаних функција, почиње резервисаном речи ***function***.
- Одељак за дефинисање кориснички дефинисаних процедура, почиње резервисаном речи ***procedure***.

Редослед одељака није битан, важно је само да декларације и дефиниције налазе испред тела програма, односно, пре програмског реда који почиње резервисаном речи ***begin***.

Тело програма почиње почетком (***begin***), а завршава крајем (***end***):

```
begin {овде се пишу наредбе, а ово је коментар}
      ...
end.
```

Тачка на крају програма се не сме изоставити. Испред тела програма (функције или процедуре) не смеју се писати никакве наредбе. Програмске наредбе се једна од друге раздвајају знаком **;** који се може изоставити само у специјалним случајевима које ћемо нагласити касније, када се са њима будемо сусретали. Будући да је делфи креиран тако да сам гради основну структуру програма, нећемо морати много да бринемо о њеном поштовању.

Подела типова података

Програми се пишу да би се обрадили неки подаци који могу бити бројеви, речи или нешто друго. У зависности од типа података дозвољене су различите операције са њима. Не могу се две речи помножити или бројеви делити на слогове. Програмски језици су, углавном, строги у односу на типове података и не дозвољавају њихово мешање.

Типови података могу да се деле разне начине, а најчешћа подела је на:

- стандардне и
- изведене.

Стандардни типови су они који су дефинисани и уграђени у структуру програмског језика и могу се одмах и увек користити у свим програмима. Изведени тип података, по потреби, дефинише корисник у свом програму и важи само у том програму, тј. ако га желимо користити у неком другом програму морамо га у том програму поново дефинисати. Тип података се дефинише именом које се не сме поклапати са неком од резервисаних речи програмског језика и навођењем свих елемената који припадају том типу. Након дефиниције неког типа у програму он се може користити као стандардни, али само у том програму.

Друга подела типова података (која не искључује прву) је на:

- просте,
- стрингове,
- структурне и
- показиваче.

Просте типове података чине:

- уређени типови и
- реални тип.

За уређене типове (***ordinal types***) карактеристично је да се за сваки њихов елемент зна који је испред, а који иза њега. Деле се на:

- ***char*** - слова или у општем случају, карактери (јер и симболе 1, 2, 3, *, / и друге можемо третирати као словни податак, а нико за њих неће рећи да су слова);
- ***integer*** - цели бројеви (... , -2, -1, 0, 1, 2, ...);
- ***boolean*** - логички тип, (вредности *true* и *false*);
- набројани тип и
- интервални тип

Прва три уређена типа су стандардни типови података, а последња два немају унапред задати домен већ се дефинишу у зависности од потребе корисника.

Структурни типови су:

- ***Set*** - скуповни тип;
- ***Array*** - низовни тип;
- ***Record*** - слоговни тип;
- ***File*** - датотечни тип;
- ***Class*** - класе

Структурни типови могу да се користе као стандардни ако је њихова примена могућа без посебно дефинисаног имена, али врло често се дефинишу у складу са захтевима програма.

Важно је напоменути да у оквиру појединих од поменутих типова података постоје подтипови који се међусобом разликују по броју елемената и прецизности, а за све подтипове истог типа важе исте операције и у неким случајевима дозвољено је њихово мешање, односно, у неким случајевима променљивама једног подтипа могу се додељивати вредност променљивих другог подтипа истог типа података.

Целобројни тип (integer)

Integer је прости уређени тип података чије вредности могу бити сви цели бројеви који се могу представити у меморији рачунара. Добро је уређен, тј. за било који елемент овог типа зна се који елемент је испред и који елемент је иза њега. У оквиру типа ***integer*** постоје бројни подтипови. Основни су:

Integer -2147483648 .. 2147483647, тј. -2^{31} .. $2^{31}-1$. Овај подтип има и позитивни и негативни део бројне праве

Cardinal 0 .. 4294967295, тј. 0 .. $2^{32}-1$. Овај подтип има само позитивни део бројне праве.

Остали подтипови типа ***integer*** приказани су у таблици. Они могу да се поделе у две групе, прва четири подтипа имају и позитивне и негативне вредности, а последња три имају само позитивне вредности. Ако се у програму захтева неки од ужих подтипова, а унесе се број ван дозвољеног опсега програмски преводилац нас упозорава на грешку типа. Најчешће се у задацима користе променљиве типа ***Integer*** јер је то довољна прецизност за већину проблема са којима се сусрећемо.

подтип	распон вредности подтипа	меморија
ShortInt	-128 .. 127	8 битова
Smallint	-32768 .. 32767	16 битова
Longint	-2147483648 .. 2147483647	32 бита
Int64	-2^{63} .. $2^{63}-1$	64 бита
Byte	0 .. 255	8 битова
Word	0 .. 65535	16 битова
LongWord	0 .. 4294967295	32 бита

Аритметичке операције и стандардне функције са овим типом података су:

Pred претходни
Succ следећи
+ сабирање
- одузимање
***** множење
Div целобројно дељење
Mod остатак при целобројном дељењу

Abs апсолутна вредност
Sqr квадрат броја

Дељење (/) није затворено у односу на скуп целих бројева, јер се може десити да количник два цела броја не буде цео број, зато ова операција није дозвољена унутар целобројног типа. Целе бројеве је дозвољено делити, али резултат дељења не може се доделити целобројној променљивој, чак ни ако смо сигурни да ће резултат бити цео број. Ради затворености операција претходни и следећи по дефиницији испред првог елемента је последњи елемент, а иза последњег је први елемент. Због затворености операција +, - и * у овом скупу као резултати њихове примене могу се добијати *чудни* резултати ако се не води рачуна о величини резултата ових операција, наиме, када резултат пређе преко највеће могуће вредности целог броја враћа се преко негативног дела потребан број пута. На пример, ако је највећи дозвољени број 127 и ми на њега додамо 1, програм се неће прекинути и пријавити грешку, већ ће као резултат исписати -128 што представља најмањи број у овом случају.

Приоритет операција наслеђен је из математике, тј. најпре се извршавају једночлане операције и све су истог приоритета, а затим двочлане, при чему су множење и дељење вишег приоритета у односу на сабирање и одузимање.

Када домен и кодомен функције није исти скуп података, такве функције зовемо претварачким (или конверзионим). Такве су:

Ord - редни број, која за аргумент има променљиву типа карактер, а као вредност даје редни број тог карактера у ***ASCII*** табели, односно, број од 0 до 255 и
Chr - карактер, која за аргумент има број од 0 до 255, а као резултат даје карактер из ***ASCII*** табеле чији је редни број једнак аргументу функције.

Програмски језик са којим радимо, не дозвољава унос и исписивање нумеричких података. Зато, да би се могли уносити нумерички подаци и исписивати резултати примене различитих операција над њима, морају постојати функције које нумерички податак претварају у текст и обрнуто. Овакве функције су, такође, претварачке јер претварају текст у број или обрнуто. Са целобројним типом постоје три такве функције и то су:

StrToInt(tekst) - функција којом се унети текст претвара у цео број. Једноставна функција, али има и својих мана. Ако је текст празан или не може да се преведе у цео број преводилац ће открити грешку и програм ће се прекинути. Да би се ово избегло потребно је направити рутину која ће обрађивати такве грешке. Мада то није посебно тешко, овде се тиме нећемо бавити.

StrToIntDef(tekst,broj) - функција којом се текст претвара у цео број с тим да је уведена аутоматска контрола исправности уноса, тј. ако унети текст не може да се претвори у цео број (јер садржи неки карактер који није цифра) или је празан као вредност функције узима се задати број (као предефинисана вредност, најчешће се узима 0 или 1, али то није обавезно).

IntToStr(broj) - функција којом се нумерички податак претвара у текстуални. У овом случају не може доћи до грешке приликом претварања јер сваки број може да се посматра и као неки текст.

Осим функција, за претварање нумеричких података у текстуалне и обрнуто могу се користити и две стандардне процедуре:

Str(broj, tekst) - процедура којом се нумерички податак из променљиве ***broj*** претвара у текстуални податак у променљивој ***tekst***. Аргумент ***broj*** може бити променљива, неки израз или конкретна вредност, а аргумент ***tekst*** мора бити променљива типа текст. Код ове процедуре не може доћи до грешке (јер сваки број може да се посматра и као неки текст), а главна разлика у односу на одговарајућу функцију, је могућност форматирања текста, али о томе ћемо касније, када нам та предност буде затребала.

Val(tekst, broj, kod_greske) - процедура којом се текстуални податак из променљиве ***tekst*** претвара у нумерички податак у променљивој ***broj***. Ако је текст празан или нема структуру целог броја преводилац ће регистровати грешку и њен редни број записати у променљиву ***kod_greske***, али програм се неће прекидати (наравно добијени резултати, у случају грешке, неће бити коректни). Зато се у овом случају мора тестирати вредност променљиве ***kod_greske*** и у зависности од те вредности бирати наредне кораке у програму. Ни ово није тешко, али захтева увођење питања и условних корака о чему ће бити речи мало касније.

Све промењљиве које се користе у програму морају бити претходно декларисане. У општем случају промењљиве се декларишу на следећи начин:

```
var    ImePromenljive:TipPromenljive;
```

Декларација промењљивих целобројног типа може изгледати овако:

```
var    a,b,c,d:integer;                          // листа промењљивих са заједничким типом
var    a:integer; b:integer; c:integer; d:integer; // свака промењљива са својим типом
var   a:integer;                  // свака промењљива са својим типом, сваака у новом реду
      b:integer;
      c:integer;
      d:integer;
var a:integer;        // испред сваке промењљиве са својим типом пише се службена реч var
var b:integer;
var c:integer;
var d:integer;
```

Али никако не може овако:

```
var a: integer; b: integer; a:integer;
// промењљива a декларисана је два пута (промењљива се не сме два пута декларисати)
```

Промењљиве могу бити:

- глобалне, тј. могу се користити у читавом програму, свуда могу добијати и мењати вредност и
- локалне, тј. могу се користити само у функцијама и процедурама у којима су декларисане.

Глобалним промењљивама се може доделити иницијална, почетна вредност приликом саме декларације, тј. не мора се чекати акција корисника којом се промењљивој додељује вредност.

```
var a: integer = 5; // глобалној промењљивој a додељена је почетна вредност 5
```

Групно иницијализовање промењљивих није дозвољено, односно, групи промењљивих не може се доделити почетна вредност на следећи начин:

```
var x,y,z: integer = 1;
```

Свакој промењљивој мора се посебно додељивати почетна вредност (без обзира на то што је иницијална вредност иста за све промењљиве), овако:

```
var    x: integer = 1;
      y: integer = 1;
      z: integer = 1;
```

Локалним промењљивама није дозвољено додељивати почетне вредности.

Реални тип (real)

Реални тип је прости тип података кога чине сви реални бројеви који се могу представити у меморији рачунара. Реални бројеви се у меморији рачунара памте у експоненцијалном запису:

```
      mantisaE±eksponent
0.0123456         1.23456e-2
1.23456           1.23456e+0
1234.56           1.23456e+3
```

Мантиса и експонент се чувају у посебним меморијским локацијама, зато реални тип података заузима више меморије од целобројног. За чување експонента резервисано је увек два бајта меморије (без обзира на величину експонента), а за мантису се користи различита количина меморије у зависности од величине бројева који се памте.

Основни тип ***real*** обухвата бројеве у распону од 5.0×10^{-324} .. 1.7×10^{308} са максимално 15-16 значајних децимала. У оквиру реалног типа разликујемо неколико подтипова. У следећој таблици они су дати са количинама меморије коју захтевају и максималним бројем значајних децимала:

подтип	распон вредности подтипа	децимала	меморија
Real48	2.9×10^{-39} .. 1.7×10^{38}	11 - 12	6 бајтова
Single	1.5×10^{-45} .. 3.4×10^{38}	7 -8	4 бајта
Double	5.0×10^{-324} .. 1.7×10^{308}	15 - 16	8 бајтова
Extended	3.6×10^{-4951} .. 1.1×10^{4932}	19 - 20	10 бајтова
Comp	-2^{63}.. 2^{63-1} (-9223372036854775808..9223372036854775807)	19 - 20	8 бајтова
Currency	-922337203685477.5808 .. 922337203685477.5807	19 - 20	8 бајтова

- ***Real48*** треба избегавати јер није у складу са архитектуром интелових процесора, па знатно умањује перформансе рачунара у току извршавања програма.
- ***Single*** и ***Double*** се најчешће користе, нарочито у једноставнијим програмима у којима немамо потребу за превеликим или премалим вредностима и прецизношћу.
- ***Extended*** подтип даје већу прецизност од осталих подтипова, али са њим долазе и проблеми са преносивошћу програма на различите рачунарске платформе.
- ***Comp*** је најприроднији за интелове процесоре и мада представља 64 - воробитни цео број припада реалном типу јер се не понаша као уређени (***ordinal***) тип већ као реални без децимала. Пошто се вредности овог типа чувају као целобројне нема дела меморије резервисаног за чување експонента (зато овај тип заузима мање меморије од претходног).
- ***Currency*** је реални тип који је због мањих грешака код заокругљивања посебно прилагођен програмима везаним за финансијско пословање и банкарство. Подаци овог типа се чувају у фиксном зарезу, тј. нема дела меморије за чување експонента. Подаци овог типа увек имају четири децимале (Вредности су исте као код претходног типа само подељене са 10.000).

И код реалног типа важи да се ужи подтипови не могу мешати са ширим (променљивој типа ***Single*** не сме се доделити вредност из ***Extended*** типа).

Реални бројеви који се могу представити у рачунару чине низ интервала на бројној правој, тј. није могуће у меморији представити све реалне бројеве са бројне праве. Вредности у таблици за све подтипове обухватају само позитивни део реалне праве, а подразумева се да је у рангу и негативни део реалне праве симетричан датом позитивном делу. Изузетак од овог правила је подтип ***comp*** који из у таблици датог интервала бројева узима само целобројне вредности. Колона у таблици у којој је дат број цифара подразумева задату математичку прецизност израчунавања са одређеним подтипом при чему се на екрану приказује највише 15 цифара и експонент, тј. једна цифра испред децималног зареза, 14 децимала и експонент са четири цифре. Пошто сви ови типови у основи имају реални број могу се међусобно упоређивати и могу се вредности преносити са ужег на шири (овде је и обрнути пренос вредности дозвољен, али се при томе мора водити рачуна о рангу, у противном се у превођењу пријављује грешка, тј. програм неће радити или ће оног тренутка када вредност променљиве изађе изван ранга програм бити прекинут). Наравно, у формалном исписивању вредности променљивих ових подтипова неће бити никакве разлике, уколико се ради о вредностима које су у рангу свих променљивих, односно, ако се ради о вредностима изван ранга, разлика ће се појављивати само на граничним децималама.

Аритметичких операција и функција са реалним типом има јако пуно, неке су стандардне и могу се увек користити, а неке су смештене у посебну библиотеку математичких функција ***Math*** и могу се користити само када је та библиотека укључена у програм (дописана у реду који почиње са ***uses*** у датотеци ***Unit.pas***). Овде ћемо поменути само стандардне функције јер се најчешће користе. То су:

+	- сабирање
-	- одузимање
*****	- множење
/	- дељење
frac	- издваја децимални део броја, резултат је реални број чији је целобројни део 0
int	- издваја целобројни део, без заокругљивања, резултат је реални број без децимала
abs	- апсолутна вредност броја
sin	- синус угла датог у радијанима
cos	- косинус угла датог у радијанима
arctan	- угао у радијанима чији је тангенс дати број
ln	- природни логаритам броја
exp	- степен броја ***e*** датим бројем
sqr	- квадрат броја
sqrt	- квадратни корен броја
random	- генерише псеудослучајан број између 0 и 1 (не рачунајући граничне).

Код формирања (генерисања) псеудослучајних бројева у паскалу, приметићемо да се на свим рачунарима у истом програму добијају увек исти бројеви и увек истим редоследом. Да се то не би дешавало употребићемо паскал процедуру ***Randomize*** која мења основу за формирање случајних бројева. Ова процедура се мора употребити једном у програму пре прве употребе функције ***Random***.

Претварачке функције са реалним типом су:

round заокругљени целобројни део реалног броја
trunc издваја целобројни део, без заокругљивања, резултат је цео број

Да би се реални бројеви могли уписивати и исписивати у делфију користићемо следеће претварачке (конверзионе) функције:

FloatToStr - функција која реални број претвара у текст (стринг) да би се он могао исписати у едиту, лабели или неком другом објекту за приказивање текстова.

StrToFloat - функција која текст написан у едиту претвара у реалан број, при чему сва ограничења поменута код функције ***StrToInt*** важе и овде.

StrToFloatDef - функција која текст написан у едиту претвара у реалан број, при чему све напомене и сва ограничења поменута код функције ***StrToIntDef*** важе и овде.

Стандардне процедуре које имају исту сврху (исте као и већ поменуте код целобројног типа):

Str(broj, tekst) - процедура којом се нумерички податак из промењљиве ***broj*** претвара у текстуални податак у промењљивој ***tekst***. Аргумент ***broj*** може бити промењљива, неки израз или конкретна вредност, а аргумент ***tekst*** мора бити промењљива типа текст. Код ове процедуре не може доћи до грешке (јер сваки број може да се посматра и као неки текст), а главна разлика у односу на одговарајућу функцију, је могућност форматирања текста, али о томе ћемо касније, када нам та предност буде затребала.

Val(tekst, broj, kod_greske) - процедура којом се текстуални податак из промењљиве ***tekst*** претвара у нумерички податак у промењљивој ***broj***. Ако је текст празан или нема структуру целог броја преводилац ће регистровати грешку и њен редни број записати у промењљиву ***kod_greske***, али програм се неће прекидати (наравно добијени резултати, у случају грешке, неће бити коректни). Зато се у овом случају мора тестирати вредност промењљиве ***kod_greske*** и у зависности од те вредности бирати наредне кораке у програму. Ни ово није тешко, али захтева увођење питања и условних корака о чему ће бити речи мало касније.

Код слагања разних операција, такође, као и код целобројног типа, треба водити рачуна о приоритету операција да би програм радио оно што се од њега очекује. Као једночлане, операције *Frac, Int, Round, Trunc, Abs, Sin, Cos, ArcTan, Ln, Exp, Sqr* и *Sqrt* су највишег приоритета, затим операције множење и дељење да би, на крају, биле операције сабирање и одузимање које су најнижег приоритета. Уколико нам приоритет операција не одговара, редослед извршавања ћемо променити заградама ( ).

У задацима које ћемо решавати промењљиве ћемо декларисати основним типом ***real***. Декларација може изгледати овако:

```
var    a,b,c,d:real;                      // листа промењљивих са заједничким типом
var    a:real; b:real; c:real; d:real;   // свака промењљива са својим типом
var   a:real;       // свака промењљива са својим типом, сваака у новом реду
      b:real;
      c:real;
      d:real;
var a:real;     // испред сваке промењљиве са својим типом пише се службена реч var
var b:real;
var c:real;
var d:real;
```

Али никако не може овако:

```
var a: integer; b: real; a:real;
// промењљива a декларисана је са два типа (промењљива може бити само једног типа).
```

Корисно је још знати и да је у програмски језик стандардно уграђена константа пи (π) која се врло често користи у задацима са тригонометријским функцијама, задацима везаним за круг и делове круга и сличним и треба је користити јер има много тачнију вредност од приближне 3,14. Ознака ***pi*** користи се као реална константа π чија је вредност **3.1415926535897932385**

Логички тип (boolean)

Код неких програма које ћемо писати биће нам важније да ли је нешто тачно или не, а мање важно ће нам бити шта је или колика је вредност промењљиве. Када се говори о таквим односима између података онда се ради о њиховим логичким вредностима. Неке задатке, које

ћемо касније решавати, без логичког типа не бисмо никако могли испрограмирати (било да се користе логичке променљиве или само логички изрази).

Логички тип је прости уређени тип података који има две вредности, ***true*** и ***false***.

И логички тип има четири подтипа, то су:

- ***Boolean*** (заузима 1 бит, тј. има две могуће вредности, 0 и 1),
- ***ByteBool*** (заузима 8 битова или 1 бајт меморије, тј. има 256 вредности),
- ***WordBool*** (заузима 16 битова, 2 бајта или једну реч меморије, тј. има 65536 вредности) и
- ***LongBool*** (заузима 32 бита, 4 бајта или две речи меморије, тј. има 4294967296 вредности).

Но, сви они имају исте вредности (***false*** је 0, а ***true*** све остале вредности), а направљени су да би се обезбедио интегритет података у раду са виндоус апликацијама. У виндоус апликацијама логичке променљиве могу добијати вредности различите од 0 и 1, па су зато неопходни различити подтипови у складу са целобројним типом, тако су подтипови ***Boolean*** и ***ByteBool*** осмобитни, ***WordBool*** је шеснаестобитни, а ***LongBool*** тридесетдвобитни, но како постоје само две логичке константе, то се константи ***false*** придружује вредност 0, а све остале вредности се придружују константи ***true***. У задацима које ћемо радити ћемо користити само подтип ***Boolean***.

Стандардне логичке операције и функције са подацима овог типа су:

not - негација, има вредност ***true*** ако исказ није тачан
or - дисјункција, има вредност ***true*** ако је бар један од исказа тачан
and - конјункција, има вредност ***true*** ако су оба исказа тачна
xor - ексклузивна дисјункција, има вредност ***true*** ако је само један од исказа тачан
pred - претходни
succ - наредни

Претварачке функције са подацима овог типа су:

odd - непарно, има вредност ***true*** ако је аргумент непаран број у противном је ***false***.
ord - редни број, при чему је ***Ord(false) = 0*** и ***Ord(true) = 1***.
BoolToStr - претвара логичку вредност у текст.
StrToBool - претвара текст у логичку вредност.
StrToBoolDef - претвара текст у логичку вредност уз могућност корекције уноса.

Релације са подацима овог типа су:

< мање
<= мање или једнако
<> различито
= једнако
>= веће или једнако
> веће
in релација повезује скуповни тип са логичким, тачна је ако елемент припада скупу.

Приоритет операција је стандардан: једночлане операције ***Not, Odd, Pred, Succ*** и ***Ord*** су истог, највишег приоритета, затим је операција ***And***, па операције ***Or*** и ***Xor*** које су истог најнижег приоритета. Логичке релације **<, <=, <>, =, >=, >** и ***in*** приоритет добијају одговарајућим ограничавањем заградама. Важно је напоменути да операције ***Not, And, Or*** и ***Xor*** раде само са простим уређеним типовима података (*Char, Integer* и *Boolean*), набројаним и интервалним типом.

Можда би било корисно запамтити следеће релације:

- ***false < true***
- ***Succ(false) = true***, ***Pred(false) = true***
- ***Pred(true) = false***, ***Succ(true) = false***

У задацима које ћемо решавати променљиве ћемо декларисати основним типом ***boolean***. Декларација може изгледати овако:

```
var   a,b,c,d:boolean;                              // листа променљивих са заједничким типом
var   a:boolean; b:boolean; c:boolean; d:boolean;          // свака променљива са својим типом
var   a:boolean;                        // свака променљива са својим типом, сваака у новом реду
      b:boolean;
      c:boolean;
      d:boolean;
var a:boolean;             // испред сваке променљиве са својим типом пише се службена реч var
var b:boolean;
var c:boolean;
var d:boolean;
```

Знаковни тип (char)

Карактер тип (***char***) је прости уређени тип података чије вредности могу бити сви знаци из ***ASCII*** табеле (односно сви елементи азбуке програмског језика). За било који елемент овог типа зна се који елемент је испред, а који иза њега. Редослед елемената одређује њихов код или редни број из ***ASCII*** табеле. Ради затворености операција претходни и следећи, по дефиницији, испред првог елемента је последњи елемент, а иза последњег је први елемент. Овај тип има 256 елемената.

Стандардне операвије и функције са подацима типа карактер су:

Pred - претходни, одређује елемент који претходи елементу који је аргумент функције,
Succ - наредни, одређује елемент који следи иза елемента који је аргумент функције
UpCase - велико слово, као резултат даје велико слово ако је аргумент функције мало слово.

Претварачке функције су оне код којих су област дефинисаности и скуп вредности функције различитог типа. Претварачке функције са подацима овог типа су:

Ord - редни број, која за аргумент има променљиву типа карактер, а као вредност даје редни број знака у ***ASCII*** табели, односно, број од 0 до 255 и
Chr - карактер, која за аргумент има број од 0 до 255, а као резултат даје знак из ***ASCII*** табеле чији је редни број једнак аргументу функције.

У задацима које ћемо решавати декларација променљивих може изгледати овако:

```
var    a,b,c,d:char;                                    // листа променљивих са заједничким типом
var    a:char; b:char; c:char; d:char;                  // свака променљива са својим типом
var    a:char;                    // свака променљива са својим типом, сваака у новом реду
       b:char;
       c:char;
       d:char;
var a:char;                // испред сваке променљиве са својим типом пише се службена реч var
var b:char;
var c:char;
var d:char;
```

Стринг тип

У раду са знаковним, карактер подацима чешће се срећемо са подацима који се састоје из више од једног знака. Такви подаци могу се обрађивати знак по знак, али би то додатно оптерећивало програм. Уместо једне променљиве радили бисмо са низом променљивих што оптерећује и компликује писање програма. Зато је уведен тип ***String*** (*енгл. string - ниска, низ карактера, реч*). Максималан број карактера који може бити вредност променљиве типа ***String*** у паскалу је 255.

У делфију постоји још неколико подтипова овог типа:

ShortString који се поклапа са паскалским типом ***String*** (максимална дужина 255),
AnsiString има максималну дужину 2^{31} карактера,
WideString има максималну дужину 2^{30} карактера и има подршку за националне симболе (***Unicode***).

Подтипови типа стринг могу да се мешају и размењују вредности, али се мора водити рачуна о њиховим специфичностима приликом позива функција и процедура. Подаци овог типа се памте као низови карактера с тим што је, будући да је ово стандардни тип података, програмском преводиоцу остављено да води рачуна о дужини.

Код декларације променљивих овог типа могуће је унапред дефинисати максималну дужину низа или задржати уграђену дужину (правила за декларацију променљивих су иста као и код претходних типова података само се овде појављују у две варијанте):

```
VAR    a:STRING;
{декларисали смо променљиву типа String која може имати највише 255 карактера
 дужине, тј. може имати мање, али не може имати више од 255 карактера, рачунајући
 и размаке и специјалне знакове}
VAR    b:STRING[11];
{декларисали смо променљиву типа String која може имати највише 11 карактера
 дужине, тј. може имати мање, али не може имати више од 11 карактера, рачунајући
 и размаке и специјалне знакове}
```

Променљивама типа ***String*** може се додељивати вредност која не одговара њиховој декларацији, тј. променљивој ***a*** може се доделити вредност која има више од 255 карактера

дужине, а променљивој ***b*** вредност која има више од 11 карактера и програмски преводилац неће пријављивати грешку, али ће се у меморији сачувати само вредност декларисане дужине, остали карактери неће се памтити и са њима програм неће радити. Зато у раду са овим типом треба пажљиво осмислити декларацију променљивих како не бисмо добијали *чудне*, односно, неочекиване резултате.

Стандардне функције/операције са подацима овог типа су:

+ - слепљивање, надовезивање стрингова, вредност функције се мора доделити некој стринг променљивој, на пример: ***a:=b+c+d; a:=a+b+c+d;***

copy(a,n,m) - функција за копирање дела стринга, из стринга ***a*** се од позиције ***n*** копира ***m*** карактера, вредност функције се мора доделити некој стринг променљивој; применом ове функције стринг ***a*** остаје непромењен

length(a) - функција за одређивање дужине, броја карактера у стрингу ***a*** може имати вредности од 0 до 255

pos(b,a) - функција за одређивање позиције стринга ***b*** у стрингу ***a***, вредност функције је цео број 0 ако се ***b*** не налази у стрингу ***a*** или цео број 1-255 ако ***b*** почиње од карактера са тим редним бројем у стрингу ***a*** (функција је осетљива на мала и велика слова, па треба о томе водити рачуна код писања програма)

LowerCase - функција којом се сва велика слова енглеске абецеде у речи која је аргумент функције замењују малим словима (функција не реагује на слова других националних абецеда)

UpperCase - функција којом се сва мала слова енглеске абецеде у речи која је аргумент функције замењују великим словима (функција не реагује на слова других националних абецеда)

Стандардне процедуре за рад са подацима типа стринг су:

concat(a,b,...) - процедура којом се надовезују стрингови (слично као **+**), први аргумент процедуре мора бити променљива на коју се надовезују вредности осталих аргумената који могу бити функције и променљиве (на пример: ***Concat(a,b,c)*** исто је што и ***a:=a+b+c***)

delete(a,n,m) - процедура за брисање дела стринга, из стринга ***a*** се од позиције ***n*** брише ***m*** карактера, тј. стринг ***a*** се мења.

insert(b,a,n) - процедура за уметање стринга у стринг, стринг ***b*** се умеће у ***a*** од позиције ***n***.

val(tekst, broj, kod_greske) - процедура за претварање стринга у број (процедура је иста као код типа ***Integer*** и ***Real*** где је и детаљно објашњена)

str(broj, tekst) - процедура за претварање броја у стринг (процедура је иста као код типа ***Integer*** и ***Real*** где је и детаљно објашњена)

Код рада са стринговима задате дужине морамо водити рачуна да њиховим слепљивањем не изгубимо део њиховог значења јер ако се добије стринг дужине веће од декларисане узима се само почетни део стринга до декларисане дужине.

Набројиви тип и интервални тип

Када решавамо неке типове задатака, може се десити да нам стандардни типови нису довољни. У таквом случају можемо сами дефинисати нови тип података тако што ћемо му дати име и навести све елементе који му припадају. Такав тип зове се ***набројиви*** (јер су сви елементи који му припадају набројани у дефиницији). Спада у уређене типове, а редослед података дефинисан је редоследом навођења елемената у дефиницији типа.

Примери:

```
Type DaniUNedelji=(pon, uto, sre, cet, pet, sub, ned);
Type Godina=(jan, feb, mart,apr, maj, jun, jul, avg, sept, okt, nov, dec);
Type Boje=(pik, karo, herc, tref);
```

Када смо дефинисали неки тип, у даљем раду, понашамо се као да је тај тип одувек постојао, тј. можемо помоћу њега декларисати променљиве. Променљиве се могу декларисати и директним навођењем елемената набројивог типа, без претходне дефиниције типа:

```
Var dan:DaniUNedelji;
Var mesec:Godina;
Var znak:Boje;
Var boja:(crvena,zuta,zelena,plava);
```

Са набројивим типом радимо као и са другим, стандардним типовима података. Да бисмо исписали вредност промењиве овог типа у делфи апликацији морамо дефинисати функцију која ће овај тип конвертовати у тип стринг.

Са подацима овог типа дозвољене су само следеће три функције:

Ord - редни број елемента у дефинисаном типу,
Pred - претходни, одређује елемент који претходи елементу који је аргумент функције,
Succ - наредни, одређује елемент који следи иза елемента који је аргумент функције.

Друге операције, које би нам могле затребати, морамо претходно сами да дефинишемо.

Када желимо да постигнемо неке специјалне ефекте приликом решавања појединих задатака можемо елементе стандардно дефинисаних уређених типова издвојити у посебан тип. Ако су елементи узастопни онда ће такав тип бити ***интервални***. Пошто настаје од уређених типова овај тип је, такође, уређен. Редослед података наслеђује се од основног типа, а такође и дозвољене операције и све друге особине. Предност увођења овог типа је незнатна уштеда меморије за чување података и лепша структура програма.

Погледајмо примере:

```
Type Jednocifreni=0..9;
Type Minut=0..59; Cas=0..23;
Type Godina=1..365;
```

Када смо дефинисали неки тип, у даљем раду, понашамо се као да је тај тип одувек постојао, тј. можемо помоћу њега декларисати промењиве. Промењиве се могу декларисати и директним навођењем елемената интервалног типа, без претходне дефиниције типа:

```
Var cifra:Jednocifreni;
Var m:Minut;c:Cas;
Var dan:Godina;
Var LotoBroj:1..39;
```

Скуповни тип

У задацима са издвајањем појединих слова користићемо појам скуп или колекција - сет (***set***). И други задаци могу се елегантније решавати коришћењем овог типа. Скуповни тип је посебна врста података. То је скуп свих подскупова скупа вредности основног типа. Основни тип скупа може бити било који прости уређени тип података ***boolean, char, integer*** или *набројани тип*. Максималан број елемената у скупу је 255 (то значи не могу ни сви цели бројеви бити елементи скупа). У раду са скуповним подацима важе правила алгебре скупова. Елементи скупа се не могу понављати.

Скуповни тип се дефинише у делу за декларације и дефиниције на следећи начин:

```
TYPE  slova = SET OF 'a'..'z';
      cifre = SET OF 0..9;
      mesec = (jan,feb,mart,apr,maj,jun,jul,aug,sept,okt,nov,dec);
      godina = SET OF mesec;
```

Промењива скуповног типа се декларише помоћу претходно дефинисаног скуповног типа или директно:

```
VAR   a:slova;
      cifra:cifre;
      znak:SET OF (pik,karo,herc,tref);
      ceobroj:SET OF 0..255;
```

Основне операције са подацима скуповног типа су:

+	унија,
*	пресек и
-	разлика.

Основне релације над подацима скуповног типа су:

=	једнако ,
<>	неједнако ,
< или <=	подскуп,
> или >=	надскуп и
in	елемент скупа.

Празан скуп дефинише се командом:

```
c:=[];
```

Додавање елемента **e** неком скупу **c** остварује се командом:

```
c:=c+[e];
```

Ако су два скупа декларисана истим типом то никако не значи да су им елементи аутоматски придружени и да су скупови једнаки. Елементи се морају придружити скуповима у програму и тек онда можемо говорити о једнакости та два скупа. Додељивање елемената скупу може се извршити вишеструком употребом команде за додавање елемената скупу или њиховим директним навођењем:

```
c:=['А','Е','И','О','У'];
```

Наравно, два скупа су једнака ако се састоје из истих елемената, односно, сви елементи једног скупа су сви елементи другог скупа (*ознака a=b*), у противном, тј. ако постоји бар један елемент било ког од та два скупа који није елемент другог скупа, та два скупа су неједнака (*ознака a<>b*). Неки скуп је подскуп другог (*ознака a<b или a<=b*) ако је сваки његов елемент уједно и елемент другог скупа. У том случају кажемо да је други скуп надскуп првог скупа (*ознака a>b или a>=b*). Неки скуп је прави подскуп (*ознака a<b*) другог ако су сви елементи првог скупа елементи другог и осим њих, други скуп има бар још један елемент. Неки скуп је прави надскуп другог (*ознака a>b*) ако постоји бар један елемент првог скупа који није елемент другог и сви елементи другог скупа су истовремено и елементи првог скупа.

Класа и методе класе – основни појмови

Објектни језици полазе од податка као доминантног концепта и уз податак везују операције – то представља *објекат*. Објекат може бити, на пример, број 5 са свим операцијама које се могу применити на њега, на пример *Додај(параметар), Одузми(параметар), Помножи(параметар), Подели(параметар)*, али објекат може бити и нека сложена структура – на пример ОСОБА, па и цео програмски систем са скупом свих операција које се на њега могу применити. Класа представља апстрактну слику скупа неких објеката који имају исте особине. То је структурирани кориснички дефинисани тип података који обухвата податке (атрибуте) који описују стање објеката класе и функције којима се дефинишу операције над подацима класе. Објекат класе је један примерак класе, а функције које су део класе називају се методе. Класа се дефинише навођењем података и функција које јој припадају. ***Класа*** је скуп објеката са истим својствима (тзв. *инстанцним променљивим* које се могу се замислити као поља неке сложене структуре), и понашањем, односно, операцијама. На пример, ***Integer*** може бити једна класа. Сваки објекат је ***примерак*** неке класе. Објекти између себе комуницирају ***порукама*** којима се од објекта који их прима захтева да изврши неку од операција дефинисаних у класи којој припада. Поруке су догађаји на које објекти могу да реагују. Објекат на примљену поруку одговара извршавањем методе, односно, операције за коју само објекти из те класе знају како се извршава (само је њима позната ***имплементација методе***).

Подаци могу бити дефинисани (иницијализација) или само декларисани и не могу бити типа класе која се помоћу њих дефинише (јер би то била циркуларна дефиниција), али могу бити било којег другог типа, па и нека друга, претходно, дефинисана класа. Исто важи и за функције, тј. могу бити само декларисане (наводи се само заглавље) или дефинисане (имају и заглавље и тело са наредбама). Ако су само декларисане, онда мора постојати и њихова дефиниција која се налази негде изван дефиниције класе. Аргументи и резултат функције могу бити било којег типа, па и тип класе која се дефинише. Класа мора бити дефинисана тамо где се и користи, тј. мора бити саставни део кода програма у коме се користи. Пример класе коју користимо сваки пут када пишемо било коју апликацију:

```
type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

Као што видимо, иза резервисане речи ***class*** следе објекти, елементи који припадају класи, а ту могу бити наведене и методе (под методама овде подразумевамо и функције и процедуре) које припадају класи. У деловима означеним са ***private*** (приватне, односно, невидљиве) и ***public*** (јавне, односно, видљиве) наводимо декларације функција које припадају класи.

У овом тренутку неће нам бити значајна карактеристика видљивост методе, па када будемо писали апликације нећемо размишљати о томе где ћемо писати декларацију методе коју додељујемо класи. Општи метод ће бити да функције и процедуре које представљају реакцију на неки догађај декларишемо у делу изнад резервисане речи ***private*** (зато што их делфи аутоматски исписује на том месту), а оне које дефинишемо независно од неког догађаја писаћемо изнад резервисане речи ***public*** (односно, биће приватне, не зато што тако мора него само да бисмо направили визуелну разлику између ове две врсте функција и процедура).

Резервисана реч ***end;*** обавезна је на крају дефиниције класе и иза ње можемо писати друге декларације и дефиниције које су нам потребне у програму.

У објектном моделу примењује се ***скривање информација***, тј. објекту се може приступити само преко порука тј. метода које су дефинисане за класу којој објекат припада. Сама структура објекта, односно, његова својства (инстанцне променљиве) нису видљиве другим објектима. Објектни модел карактерише ***учаурење података и операција*** у објекат што омогућује једноставну измену структуре објекта и класе и имплементације операција без ефекта на апликације које тај објекат користе (све док су дефиниције операција непромењене, тј. са истим називима и параметрима). Нова класа може се дефинисати као ***подкласа*** већ дефинисане класе (своје надкласе). Подкласа наслеђује операције и својства од своје надкласе, али може имати и своје сопствене.

На пример, за дефинисану класу ОСОБА може се дефинисати подкласа СТУДЕНТ, која има сва својства класе ОСОБА, али и нека своја специфична. Програми које ћемо писати у оквиру овог курса неће захтевати увођење нових класа.

Питања на која треба обратити пажњу

- Подела типова података
- Целобројни тип и операције са целим бројевима
- Подтипови целобројног типа
- Реални тип и операције са реалним бројевима
- Подтипови реалног типа
- Карактер тип и операције са подацима тог типа
- Подтипови типа карактер
- Логички тип, операције и релације
- Подтипови логичког типа
- Набројани тип
- Интервални тип
- Скуповни тип
- Декларација променљивих
- Конверзија типова података
- Дефинисање новог типа података
- Шта је класа
- Комуникација између објеката
- *Public* декларације
- *Private* декларације
- Однос између класе и подкласе