Решавање проблема помоћу рачунара

Као што смо већ поменули, готово је сваки проблем могуће решити помоћу рачунара. Ипак, с обзиром на то да рачунар не размишља сам, неопходно је задати му одговарајуће инструкције које ће он извршити и на тај начин решити проблем.

На следећим сликама приказан је ток радњи од тренутка када добијемо проблем који треба да решимо, па све до самог његовог решења.

Слика 1.1. Дијаграм решавања проблема помоћу рачунара

Дакле, почињемо од анализирања проблема, онда формирамо алгоритам, и на крају трансформишемо тај алгоритам у рачунарски програм.

1. ***Који кораци чине анализу проблема?***

Слика 1.2. Дијаграм анализе проблема

2. ***О алгоритмима ће бити више речи у следећој глави.***

3. ***Шта представља трансформисање алгоритма у рачунарски програм?***

Слика 1.3. Дијаграм трансформације проблема

Трансформација проблема на облик погодан за решавање на рачунару

Подсетимо се поступка решавања проблема применом рачунара: прво вршимо анализу проблема, затим формирамо алгоритам за решавање и на крају тај алгоритам трансформишемо у рачунарски програм. Анализа проблема нам помаже да детаљно сагледамо проблем и како да дођемо до очекиваног решења уз помоћ датих података.

Да бисмо радили на рачунару неопходно је да појмове схватамо апстрактно. Математички проблеми су апстрактни, па су погодни за решавање на рачунару. Остале проблеме потребно је прилагодити апстрактним мишљењем, чему служи детаљна анализа. Анализа проблема нам у овом случају омогућава да уочимо важне податке и њихове особине које нас даље доводе до очекиваног решења. Такође, она нам помаже да разложимо поступак долажења до решења на мање кораке, па је самим тим и могућност појављивања грешака мања.

Покажимо на примеру како би требало да изгледа анализа проблема и сам ток нашег закључивања о коме смо говорили.

Пример 1 **Опишимо природним језиком поступак, алгоритам, за израчунавање збира бројева 5 и 6 помоћу дигитрона.**

Прво размислимо о следећем: на дигитрону не постоји дугме на коме пише *збир*, тако да ћемо потражити дугме +

. Следеће што треба урадити јесте укуцати на дигитрону *5 + 6*, међутим још увек постоји шанса да и овај корак буде неуспешан. То ће бити случај уколико нисмо укључили дигитрон, као и уколико смо направили грешку притискајући погрешну дугмад. У случају да је све до сада било исправно, потребно је још само притиснути дугме =

. Сада можемо исписати алгоритам.

1. Проверити да ли је дигитрон укључен, ако није, укључити га
2. Притиснути дугме 5 на дигитрону
3. Ако је грешком притиснуто друго дугме, притиснути дугме C на дигитрону и поновити претходно почевши од радње под редним бројем 2.
4. Притиснути дугме на + дигитрону.
5. Ако је грешком притиснуто друго дугме, притиснути дугме C на дигитрону и поновити претходно почевши од радње под редним бројем 2.
6. Притиснути дугме 6 на дигитрону.
7. Ако је грешком притиснуто друго дугме, притиснути дугме C на дигитронуи поновити претходно почевши од радње под редним бројем 2.
8. Притиснути дугме = на дигитрону.
9. Ако је грешком притиснуто друго дугме, притиснути дугме C на дигитрону и поновити претходно почевши од радње под редним бројем 2.
10. Прочитати резултат.
11. Искључити дигитрон ако је потребно.

Основи алгоритмизације

Алгоритам је поступак долажења до решења неког проблема. Састоји се од коначног низа корака који се одвијају одређеним редоследом.

Алгоритам нам може помоћи не само у случају решавања проблема помоћу рачунара, већ и у решавању многобројних једноставних активности које су уобичајене у нашој свакодневици, као што је случај са припремањем кулинарских специјалитета, чаја и слично.

Алгоритам конструишемо тако што почињемо од једноставне, уопштене идеје, а потом поједине кораке поступно разлажемо. То значи да главни проблем рашчлањујемо на мање проблеме које треба решавати.

Основни кораци алгоритма.

Слика 1.4. Дијаграм основних корака алгоритма

Описивање алгоритма.

Постоји више начина да се опише алгоритам: природним језиком, псеудо језиком, дијаграмом и програмом. У наставку ћемо објаснити сваки од ових типова.

ТИП А. Опис природним језиком

Описати алгоритам природним језиком значи детаљно, јасно, прецизно, недвосмислено и тачним редоследом описати кораке који воде решењу проблема.

Пример 1. **Опишимо природним језиком алгоритам за сабирање произвољна два броја. Нека то буду *a* и *b* за бројеве које сабирамо и *c* за резултат.**

1. Унети вредности за променљиве ***a*** и ***b*.**
2. Израчунати збир променљивих ***a*** и ***b*** и доделити их променљивој ***c*.**
3. Приказати вредност променљиве ***c*.**

Пример 2. **Приметимо да и кување кафе такође можемо представити алгоритмом.**

1. Узети лонче и сипати воду и шећер по потреби.
2. Укључити ринглу.
3. Сачекати да вода прокључа.
4. Кад вода прокључа, сипати кафу и промешати.
5. Сачекати да се ниво напитка подигне и тада склонити лонче са рингле.
6. Искључити ринглу.
7. Сипати кафу у шољицу.

ТИП Б. Опис псеудо језиком

Псеудо језик представља комбинацију природног језика и некогпрограмског језика. Код овог типа описивања алгоритма веома је важно користити исте језичке конструкције на исти начин. Такође, уколико је потребно, псеудокод треба да буде праћен одговарајућим објашњењима.

Пример 3. **Опишимо псеудо језиком алгоритам за сабирање произвољна два броја. Нека то буду *a* и *b* за бројеве које сабирамо и *c* за резултат, као код *примера 1*.**

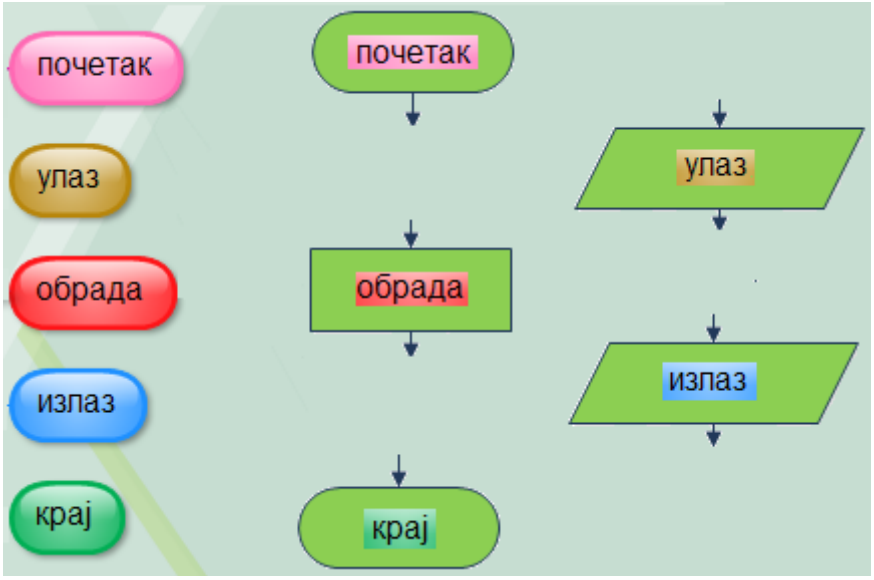
```
Почетак
Упиши a и b
c = a + b
Испиши c
Крај
```

ТИП В. Опис алгоритма блок шемом (дијаграм)

Код овог облика описивања алгоритма користе се међународно договорени симболи прописани ISO

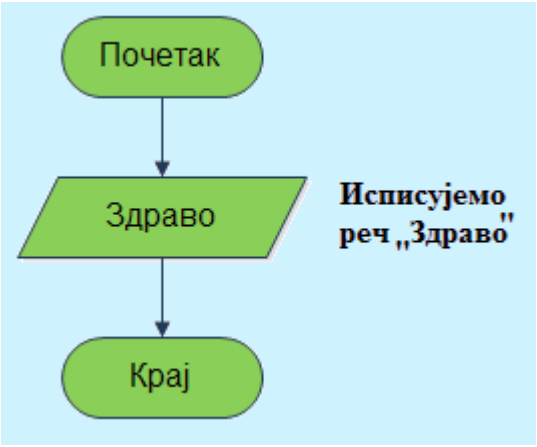
стандардом, а ток рада алгоритма представља се линијама које повезују симболе. Овај тип описивања има предност у односу на описивање псеудо језиком и природним језиком јер не зависи од говорног језика онога ко саставља алгоритам, графички приказ је једноставан, прегледан и лако је пронаћи грешке.

Графички симболи који се најчешће користе за приказ алгоритма дијаграмом су:



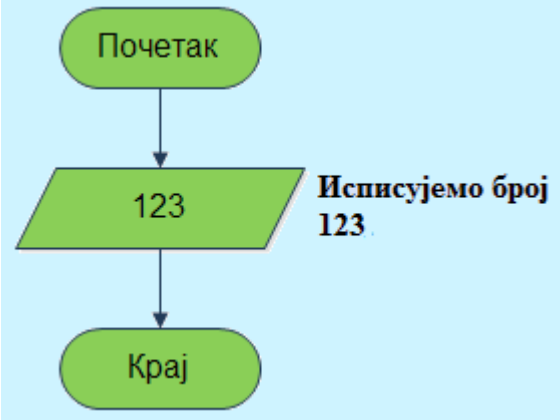
Слика 1.5. Графички приказ неких основних корака алгоритма

Пример 4. **Покушајмо да опишемо дијаграмом алгоритам који исписује поруку „Здраво" на екрану.**

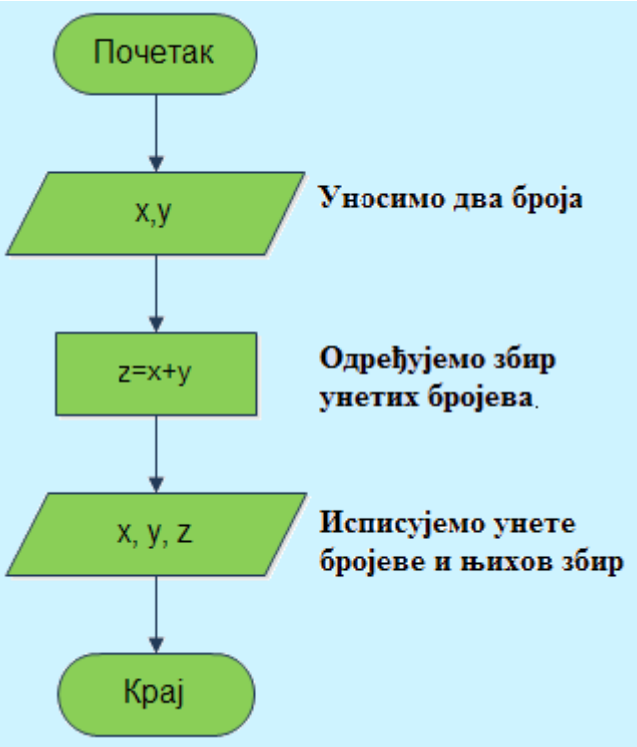


Пример 5. **Покушајте да сами опишете дијаграмом алгоритам за исписивање имена и презимена, на пример Пера Перић.**

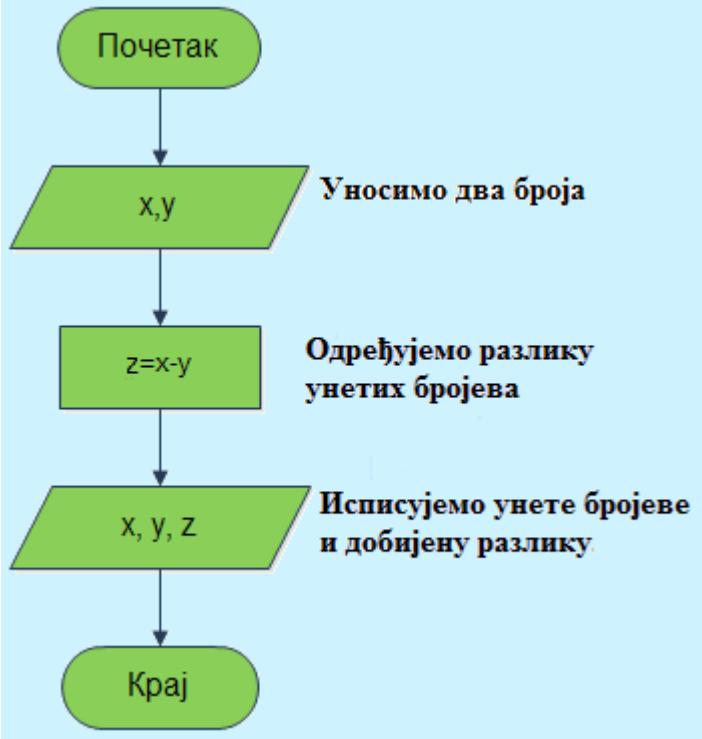
Пример 6. **Покушајте да опишете дијаграмом алгоритам за исписивање броја 123.**



Пример 7. **Опишимо дијаграмом алгоритам за сабирање два произвољна броја.**

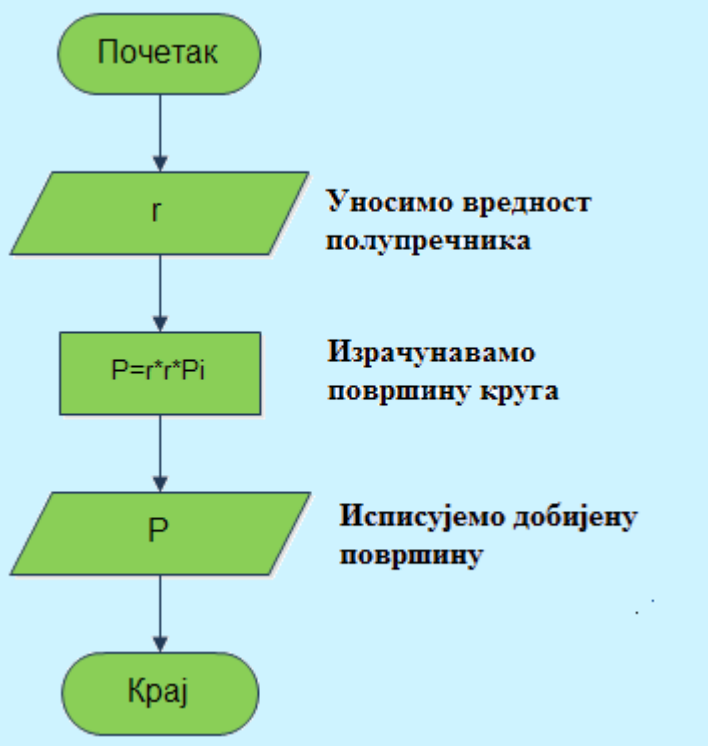


Пример 8. **Опишите дијаграмом алгоритам за разлику, производ и количник два произвољна дата броја, а потом и квадрат задатог произвољног броја.**

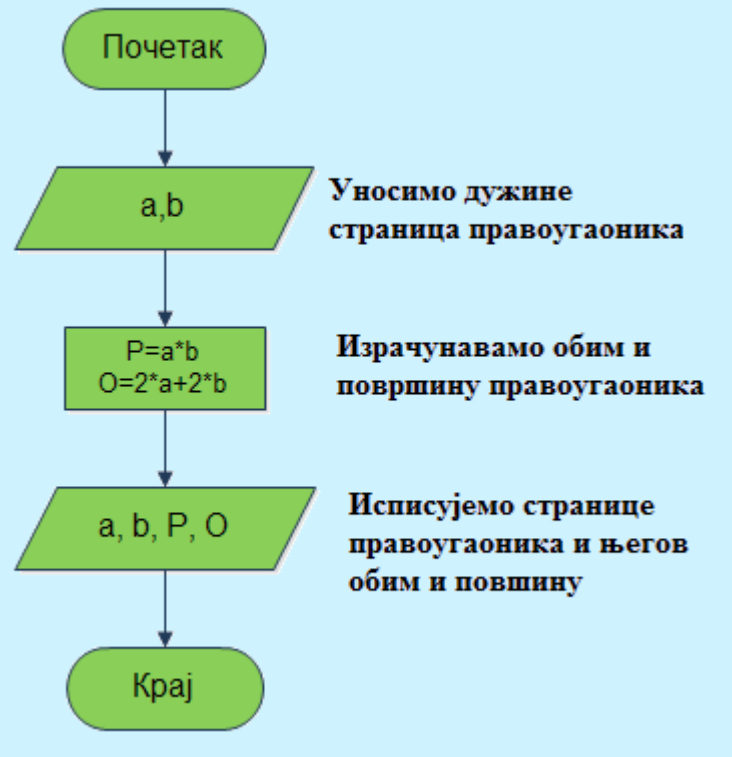


Пример 9. **Опишите алгоритам (дијаграмом) за израчунавање израза *x = a / 3, a* је дат произвољан број.**

Пример 10. **За дати полупречник напишите алгоритам за израчунавање површине круга.**

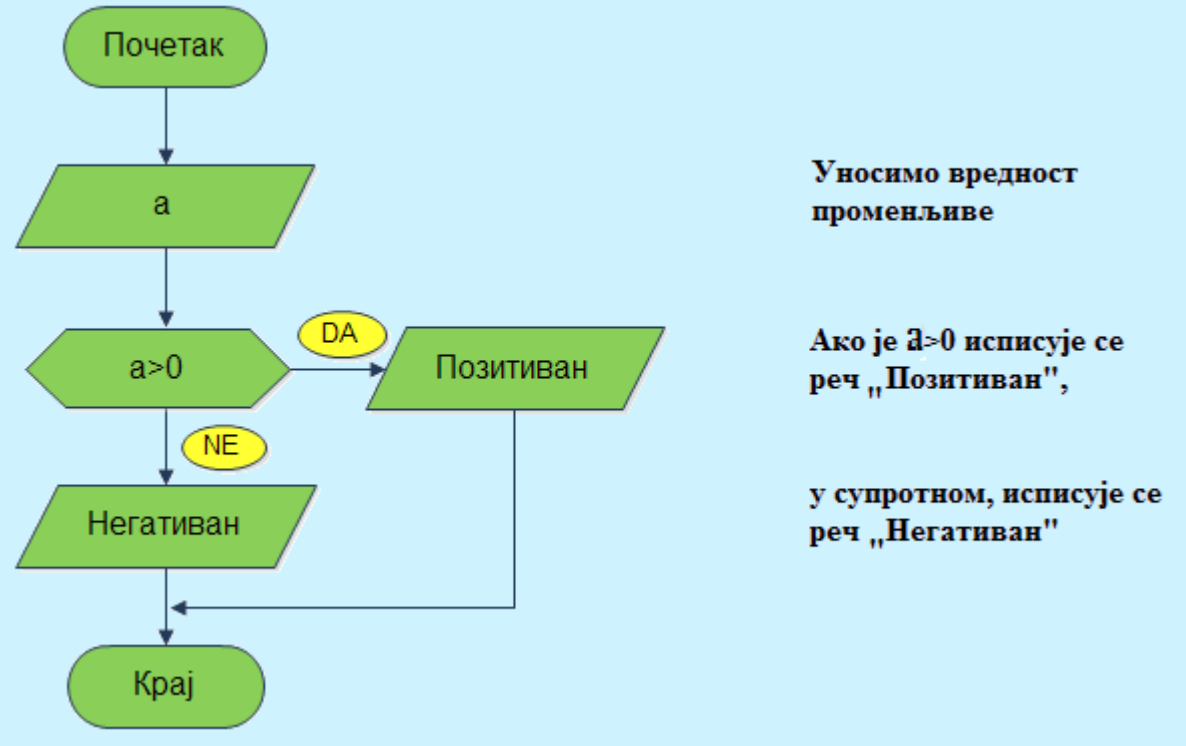


Пример 11. **Дате су дужине страница правоугаоника *a* и *b*. Дијаграмом опишите алгоритам за рачунање површине и обима правоугаоника датих страница, а онда испишите дужину страница, површину и обим правоугаоника.**

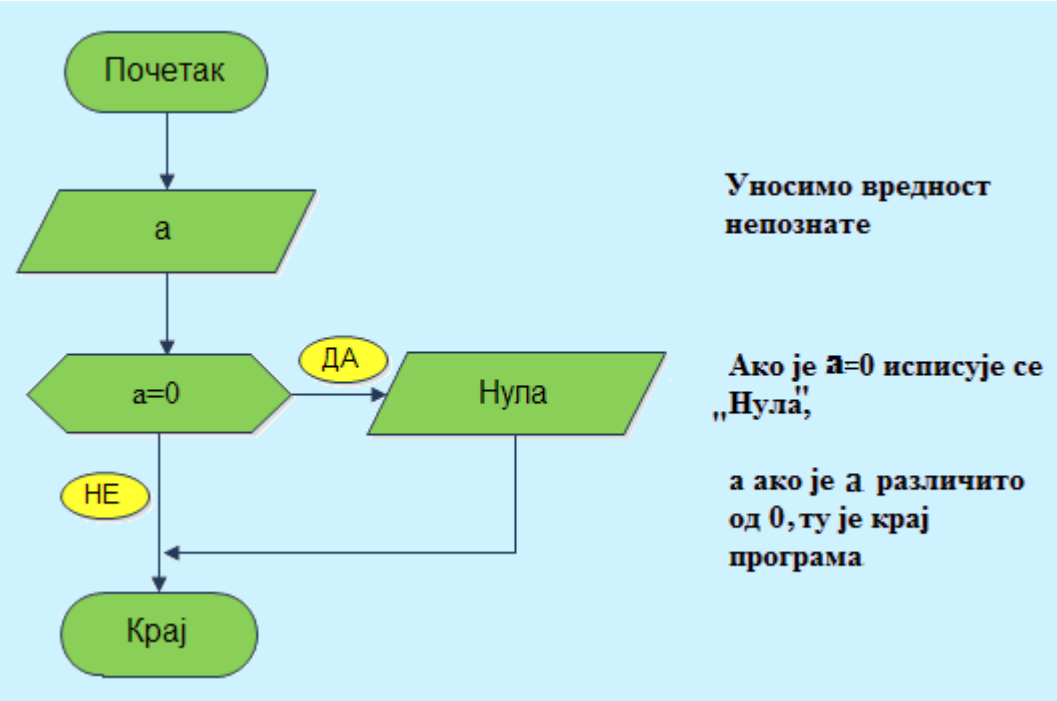


Шта се дешава уколико се од нас тражи да напишемо алгоритам који извршава одређени задатак, али само уколико испуњава одређени услов? Погледајмо следећи пример.

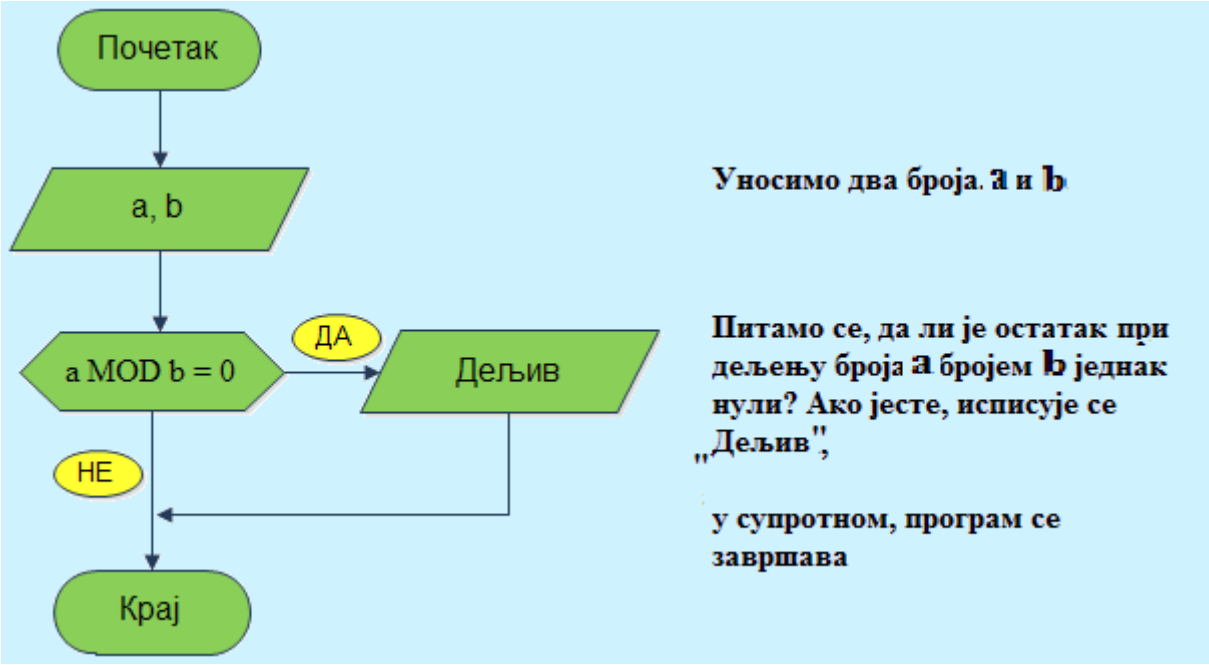
Пример 12. **Дат је произвољан број различит од нуле. Напишимо алгоритам који исписује да ли је број позитиван или негативан.**



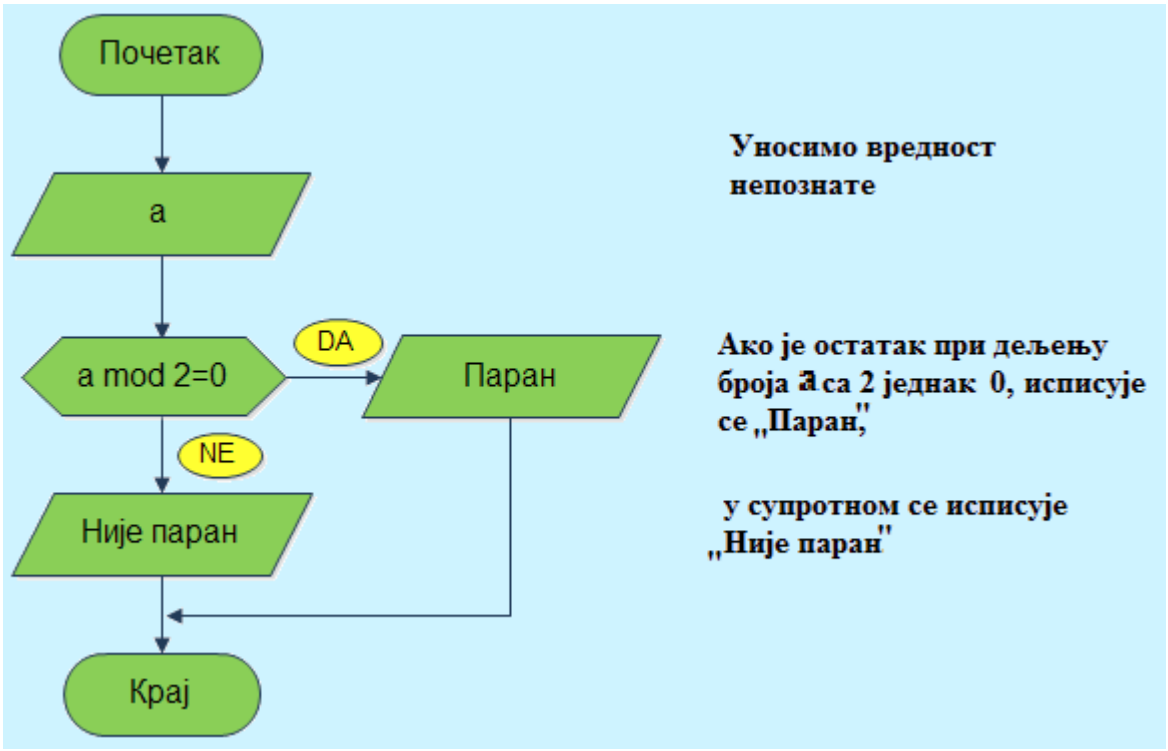
Пример 13. **Дат је произвољан број. Напишите алгоритам који проверава да ли је број једнак нули и исписује „Нула" ако јесте.**



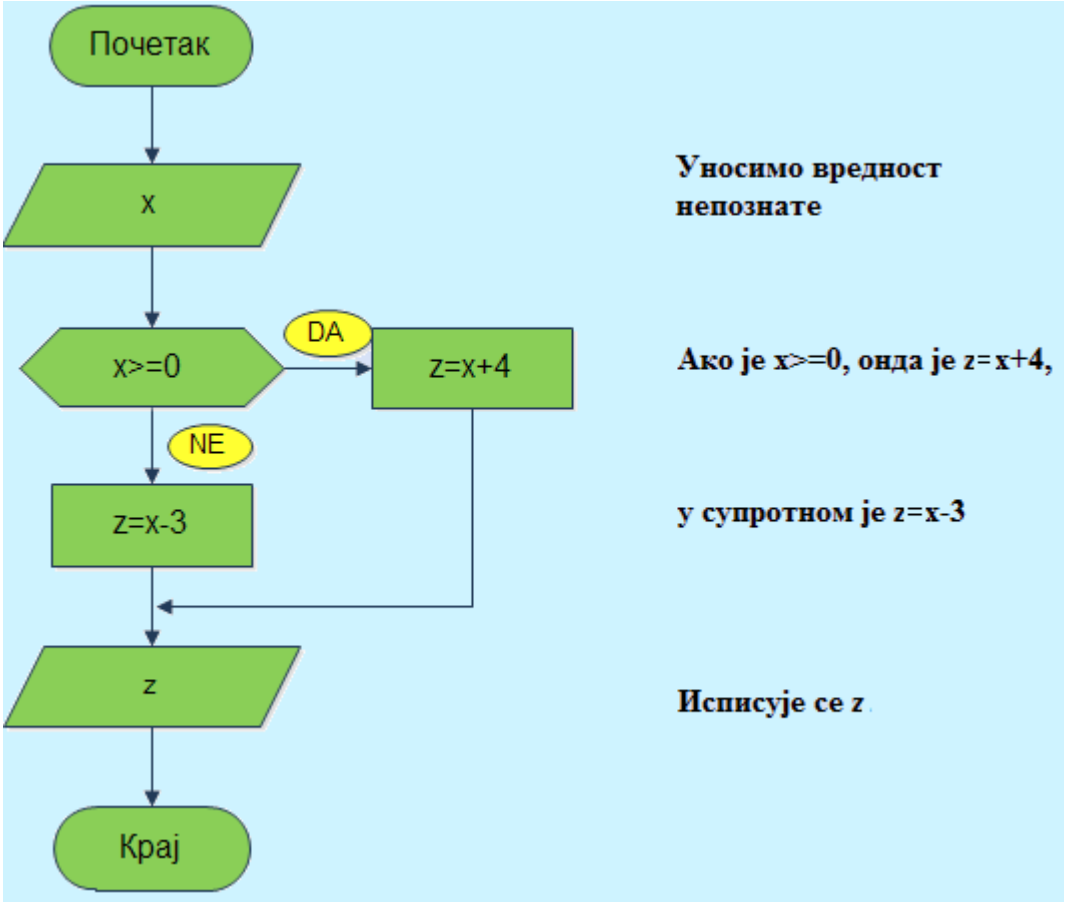
Пример 14. **Напишите алгоритам који проверава да ли је од два дата броја први дељив другим, а потом исписује „Дељив" ако јесте.**



Пример 15. **Напишите алгоритам који проверава да ли је дати произвољни број паран или није, а потом то и исписује.**

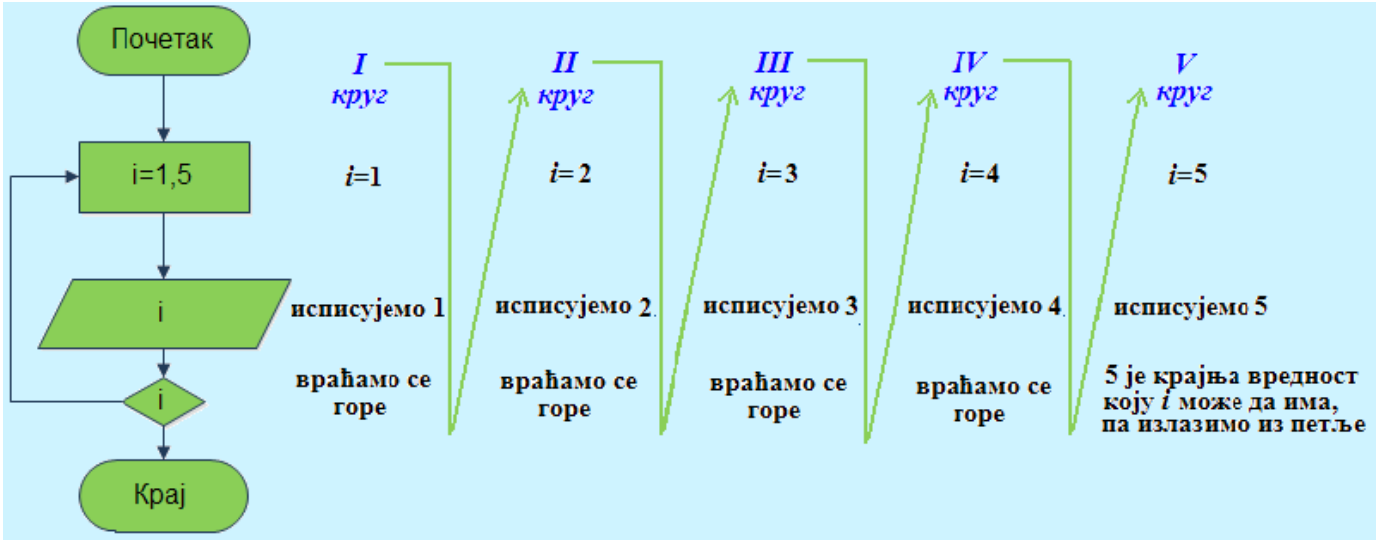


Пример 16. **Дат је произвољан број *x*. Напишите алгоритам који исписује број за 3 мањи од *x*, ако је *x* негативан број, а у противном исписује број који је за 4 већи од *x*.**

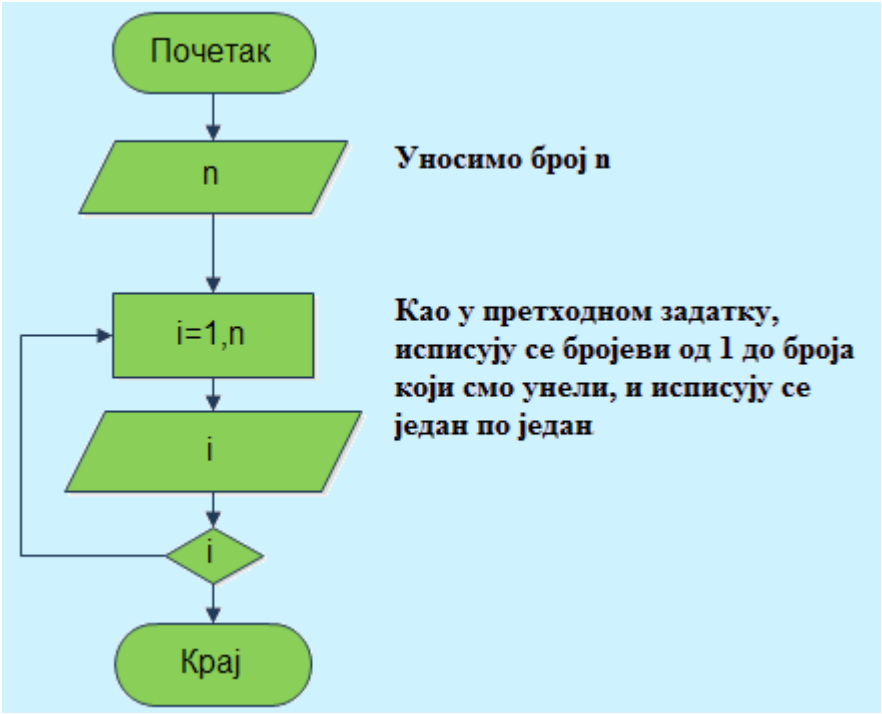


Често се дешава да нам се тражи у неком задатку да више пута поновимо исту радњу. Код програмирања и алгоритмизације се то на једноставан начин може избећи. У следећем примеру се од нас тражи да напишемо алгоритам који ће исписивати све природне бројеве од 1 до 5. Ово би могло да се реши тако што ћемо уносити редом све те бројеве, слично као у примеру 6, али проблем настаје када се од нас тражи да унесемо и испишемо првих 1000 бројева, на пример. Погледајмо како се такви проблеми једноставно решавају.

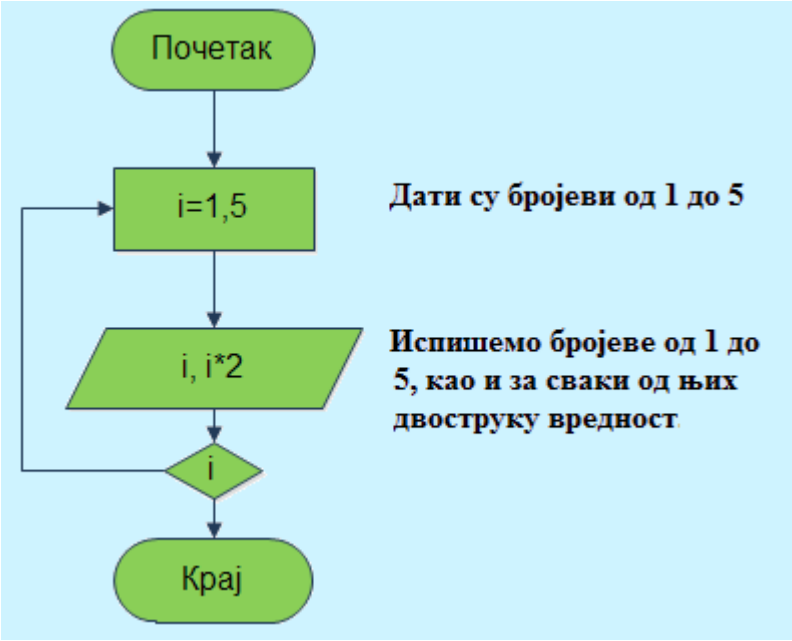
Пример 17. **Написати алгоритам који исписује редом све природне бројеве од 1 до 5.**



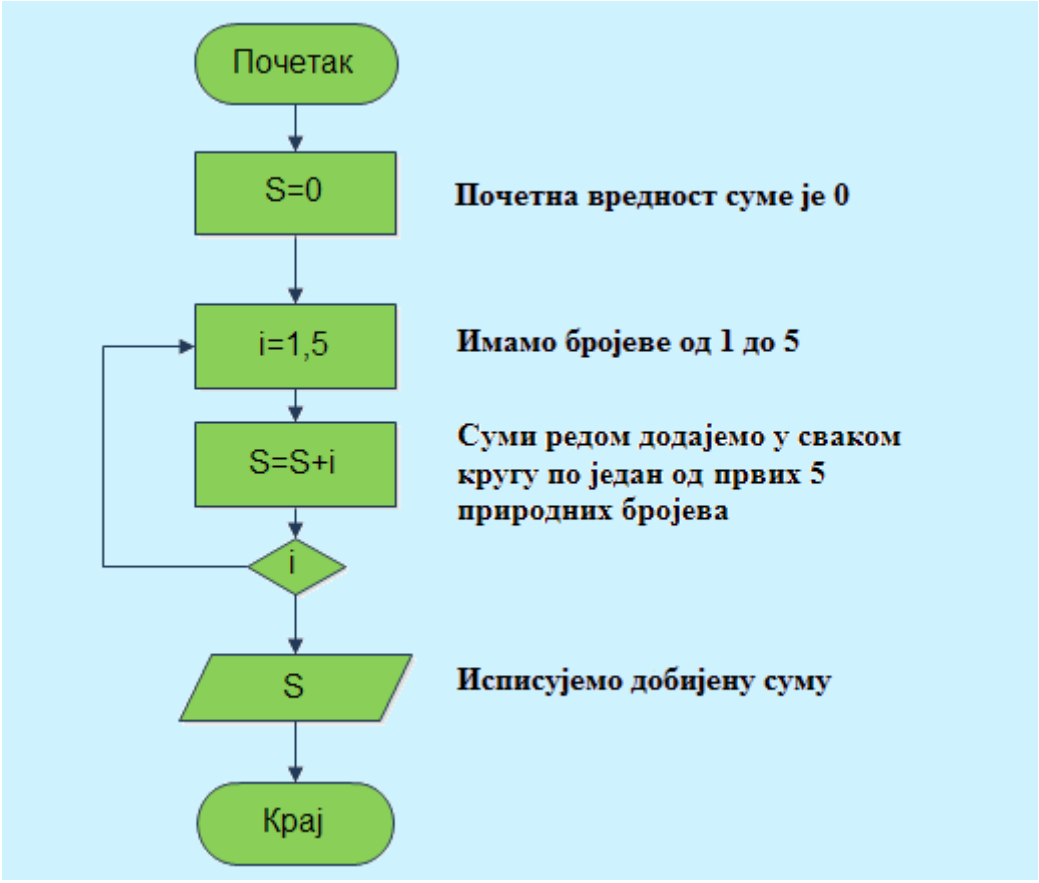
Пример 18. **Напишите алгоритам који за произвољно *n* које се задаје исписује првих *n* природних бројева.**



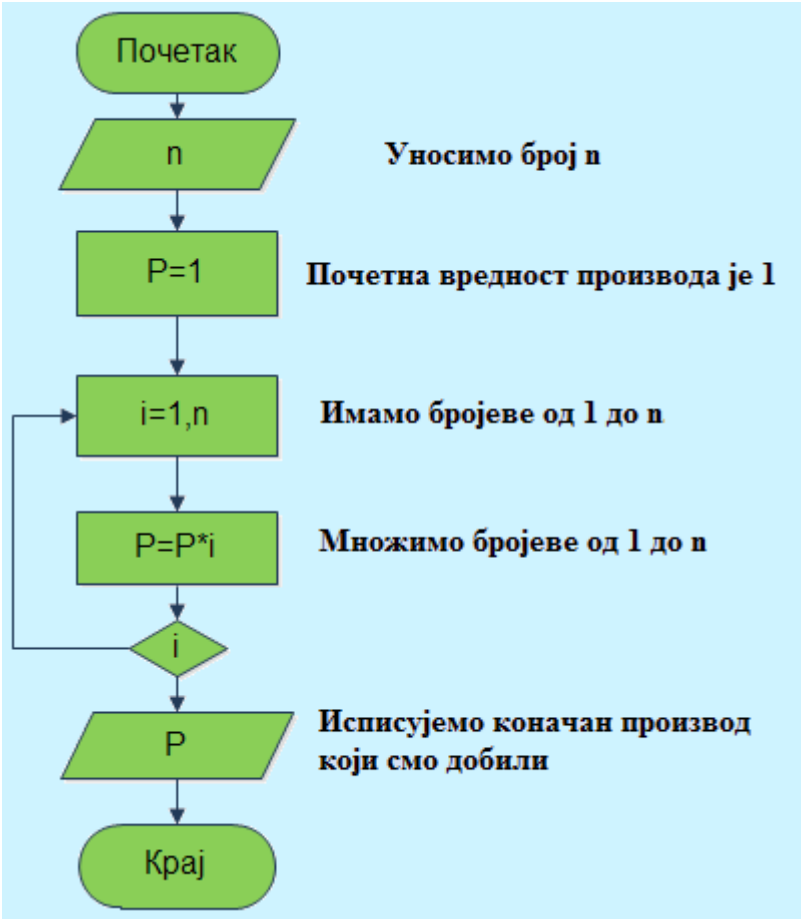
Пример 19. **Напишите алгоритам који исписује редом првих пет природних бројева заједно са њиховом двоструком вредношћу.**



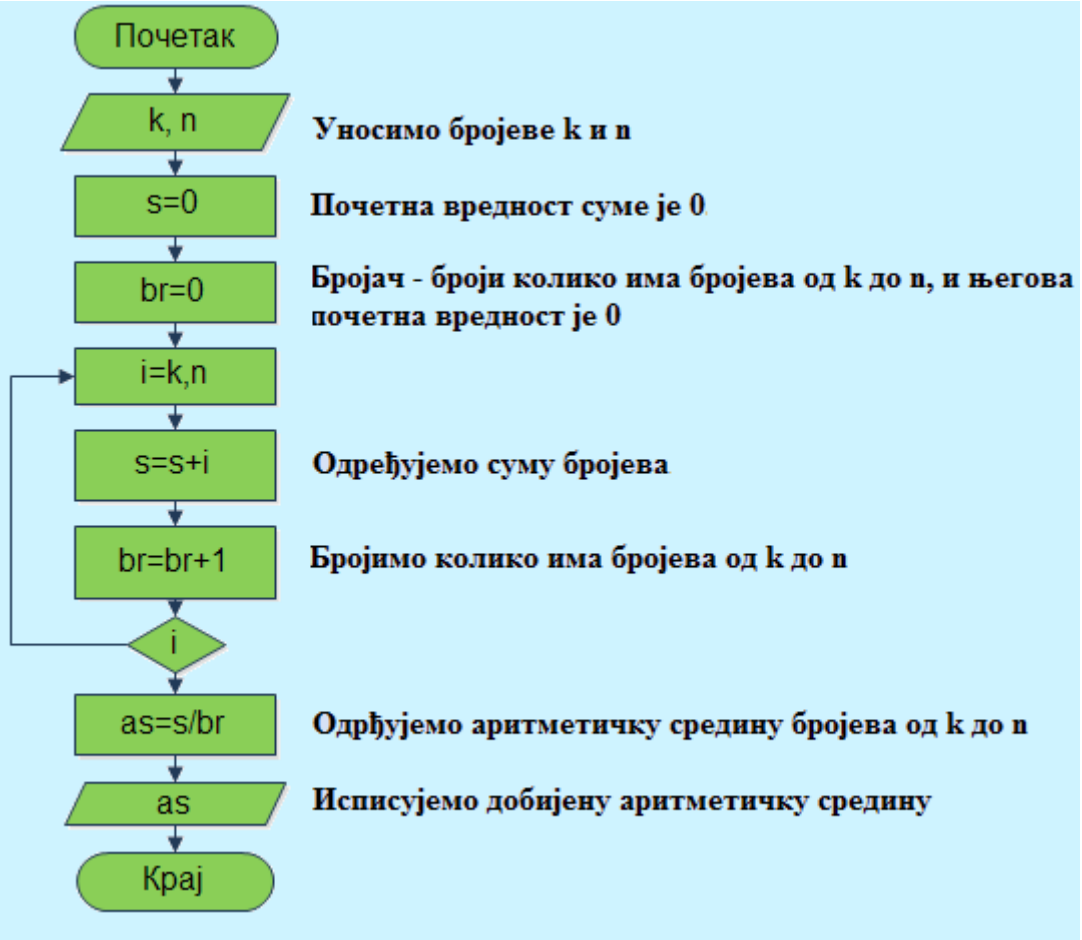
Пример 20. **Опишите дијаграмом алгоритам који исписује суму првих пет природних бројева.**



Пример 21. **Описите дијаграмом алгоритам који исписује производ првих *n* природних бројева. Број *n* је задат и произвољан природни број.**



Пример 22. **Дата су два природна броја, мањи *к и већи n*. Напишите алгоритам за исписивање аритметичке средине бројева од *к* до *n* укључујући и те бројеве.**



Синтакса и семантика програмских језика

Постоје две врсте језика: природни и вештачки. Природни језици се користе у свакодневној комуникацији људи, али су неупотребљиви када се ради о комуникацији међу машинама или човека и машине. Због тога можемо говорити о вештачким језицима који управо због тога постоје. Један тип вештачких језика јесте ***програмски језик***.

Програмски језик и рачунарски програм

Програмски језик **је вештачки језик који нам омогућава задавање *инструкција* рачунару на њему разумљив начин.**

Инструкција **коју задајемо рачунару помоћу програмског језика је *рачунарски програм*.**

Рачунарски програм је опис алгоритма у неком програмском језику.

Синтакса и семантика

Да бисмо формирали програмски језик, односно његове конструкције, треба да постоји **скуп правила на основу којих је то могуће**. Тај скуп правила назива се ***синтакса*** **програмског језика**. Применом синтаксних правила утврђујемо да ли је одређена конструкција правилна.

Као што су код природног језика **слова**, **речи** и **реченице**, тако су код програмског језика **симболи**, **лексеме** и **изрази**.
Синтаксне грешке су: погрешно откуцана реч, спојене две речи и сл.

За разлику од ***синтаксе*** која се односи на ***изглед*** конструкција одређеног програмског језика, ***семантика*** **одређује** ***значење*** **конструкција, односно програма у целини.**
Семантичке грешке су логичког типа и тичу се разумевања тога како програм функционише.

Покажимо на примеру разлику у синтакси два програмска језика исте семантике. У питању су делови програма који омогућавају исписивање текста ,,Здраво свете!" на прозору рачунара.

Pascal

```
writeln('Zdravo svete!')
```

C#

```
System.Console.Write("Zdravo svete!");
```

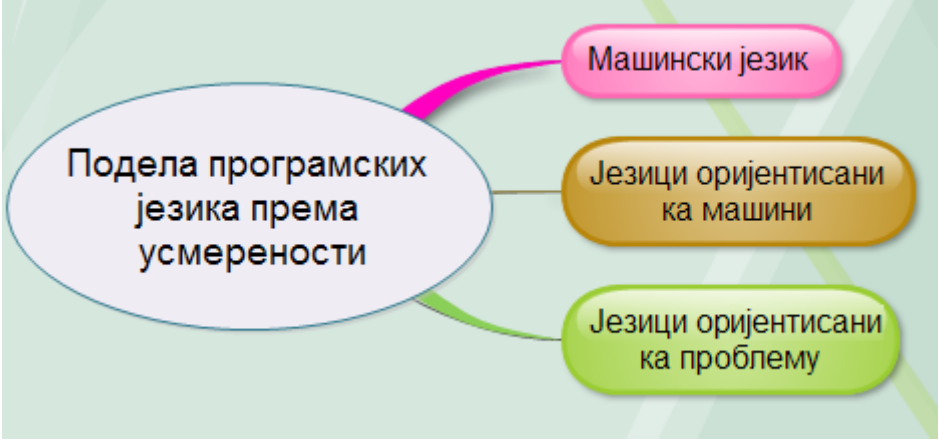
Подела програмских језика

Програмске језике можемо разврстати на основу више различитих критеријума.

I Прва подела је врло природна и везана је за генерацију рачунара.

Слика 1.6. Дијаграм поделе програмских језика према генерацији

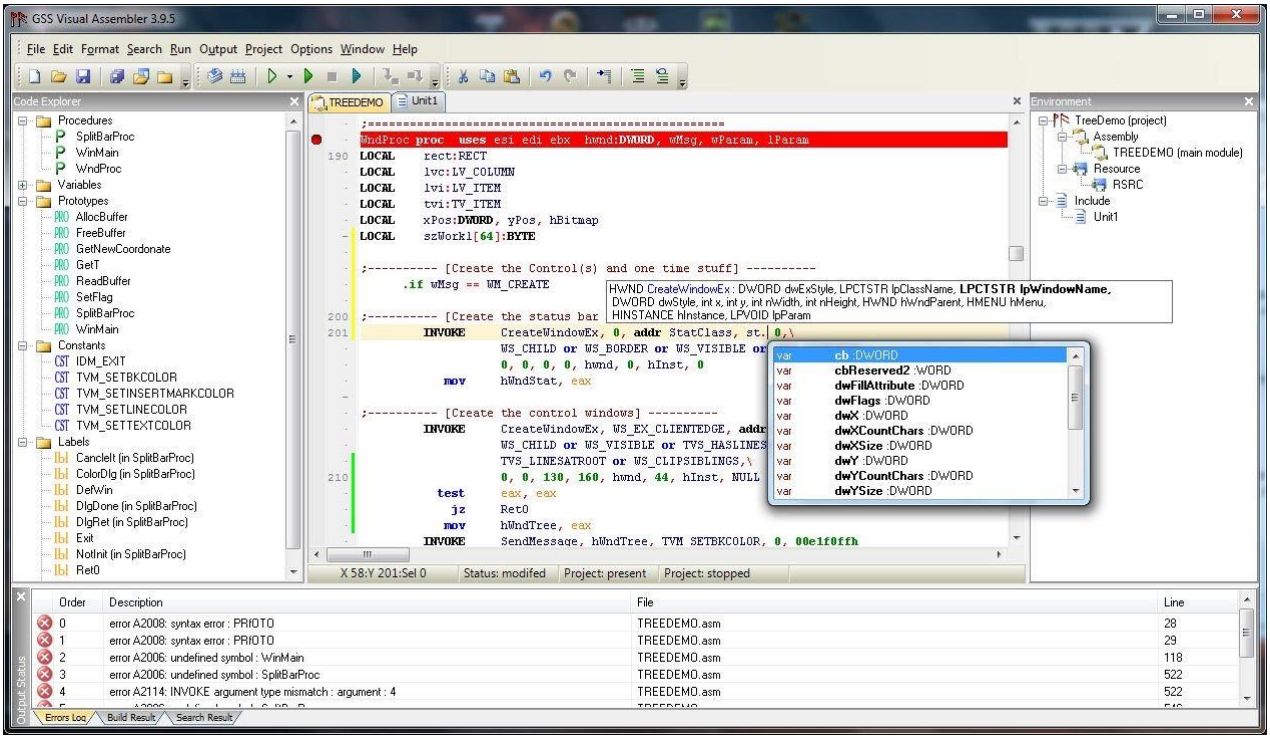
II Следећа подела програмских језика је према њиховој усмерености.



Слика 1.7. Дијаграм поделе програмских језика према усмерености

1) *Машински језик* је скуп инструкција које један процесор може директно, без превођења, да изврши. Најнижи тип програмског језика јесте машински језик. Изграђен је над бинарном азбуком (0,1). Инструкције су шаблони битова, где сваки шаблон одговара одређеној команди која се задаје машини. Није га потребно преводити јер сваки модел централног процесора има сопствени машински језик или сет инструкција.

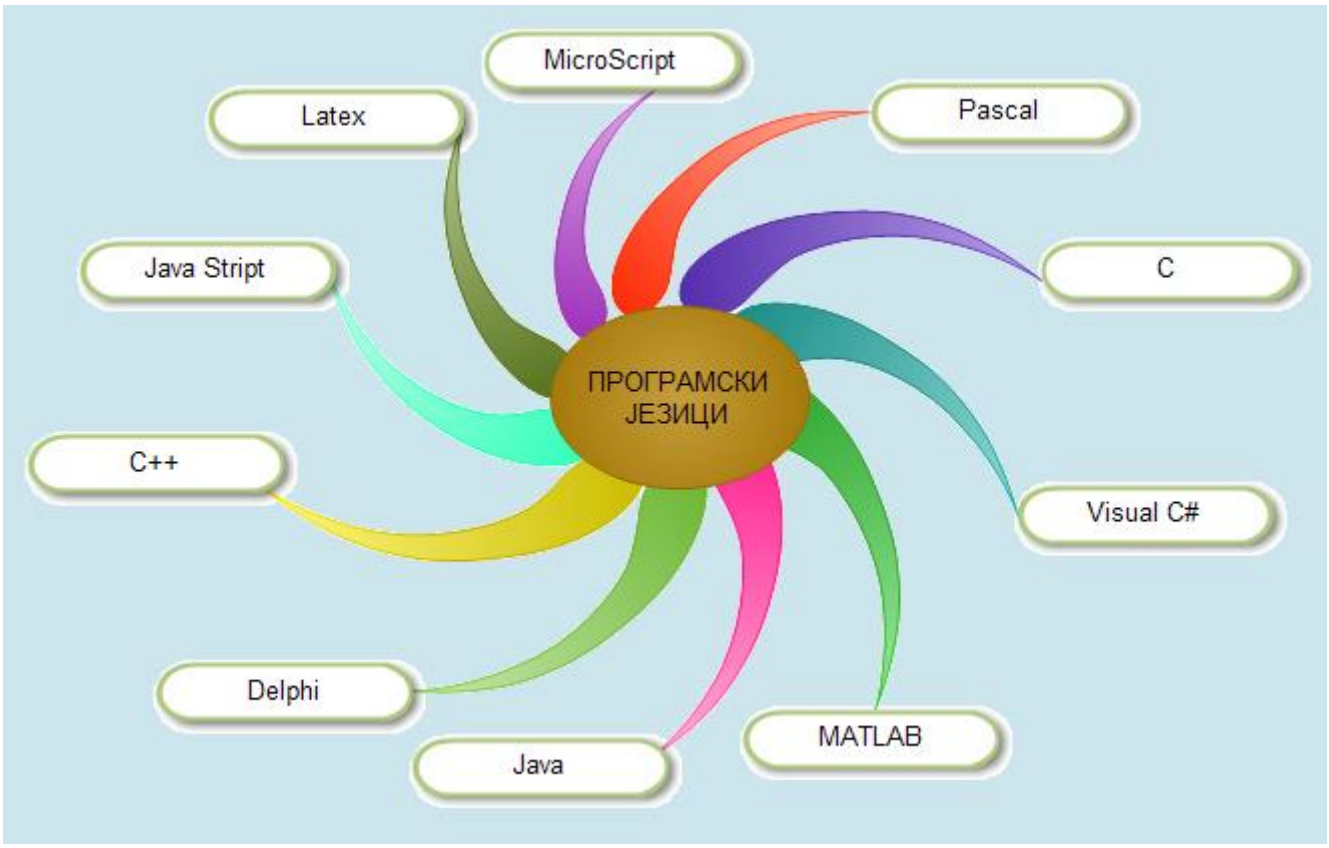
2) Језици оријентисани машини су *асемблери*. Асемблер је нижи симболички програмски језик који машински језик представља у облик читљив људима. Свака инструкција у асеблеру представља једну инструкцију машинског језика. Захваљујући асемблерима програмер не мора директно да кодира машинске инструкције у одговарајући бинарни код и слично.



Слика 1.8. Пример асемблера

3) Језици оријентисани проблемима су *виши програмски језици* и они уважавају начин људског мишљења. Свака програмска наредба се мора превести у низ машинских инструкција. Они су ближи

природном језику, читљивији и лакши за писање програма. На следећој слици налазе се имена неких од програмских језика.



Слика 1.9. Неки програмски језици

Програми засновани на прозорима

Да би се управљало програмом заснованом на прозорима који има развојно окружење, мора се знати језик којим се тај програм служи. Због тога је, тренутно, најважније питање: *који* програмски језик треба да одаберемо као и *шта* добијамо улагањем напора да би исти савладали. У циљу решавања ове мистерије

особине програмских језика: **Delphi, Java i C#.**
Осим напредних особина програмског језика C# наведених у табели, оно што га издваја од других програмских језика су и: **једноставност** коришћења (већ на самом почетку учења можемо нешто да направимо) као и **чешће коришћење** (у односу на друге програмске језике) у даљем раду и будућности.

Delphi **Java** **C#**

Слика 2.1. Delphi, Java и C#

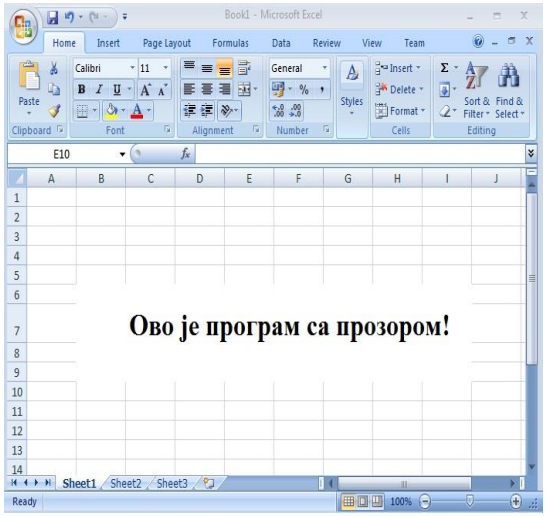
Особине	Delphi	Java	C#
1. 64 - битни компајлер	нема	***има***	***има***
2. Сакупљач отпадака - garbage collector	нема	***има***	***има***
3. Енуератори - групе константи	нема	нема	***има***
4. Делегати	нема	нема	***има***
5. Индексери - специјалне синтаксе за преклапање оператора	нема	нема	***има***
6. Прави правоугаони низови	нема	нема	***има***
7. Петља ***foreach***	нема	нема	***има***
8. ***Linq*** библиотека	нема	нема	***има***
9. Аутоматско утврђивање типова ***var***	нема	нема	***има***
10. Комплексни и високо прецизни децимални бројеви	нема	нема	***има***
У табели су приказане само неке од напредних особина програмског језика C#, оно што га заиста чини језиком будућности јесте способност да иде у корак са временом и новим технологијама.			

Основне карактеристике програма заснованих на прозорима

Пре него што се детаљније позабавимо програмирањем програма заснованих на прозорима, требало би да схватимо разлике између њих и конзолних програма.

• Програми засновани на прозорима

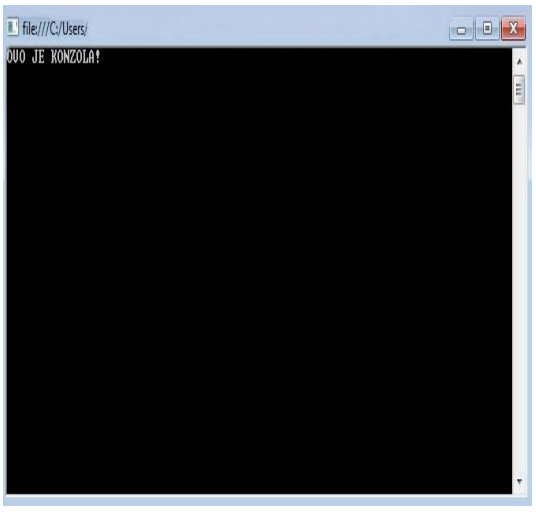
Рад оваквих програма се заснива на ономе што корисник уради са графичким корисничком интерфејсом. Бирањем чланова менија, дугмади или коришћењем миша/ тастатуре изазивамо дешавања унутар програма. У сваком тренутку имамо велики број интеракција које су нам доступне, а од којих свака резултује другачијом програмском акцијом. Док ми нешто не урадимо, није познат програмски код који следећи треба да се изврши.



Слика 2.2. Радна површина Microsoft Office Excel-a

• Конзолни програми

Код конзолних програма, ми покрећемо програм, а програмски код одређује след догађаја. Уносимо податке када се то затражи, а програм избацује резултате када то жели, па је најчешће све унапред одређено. У било ком тренутку је познато који ће се програмски код следећи извршити.

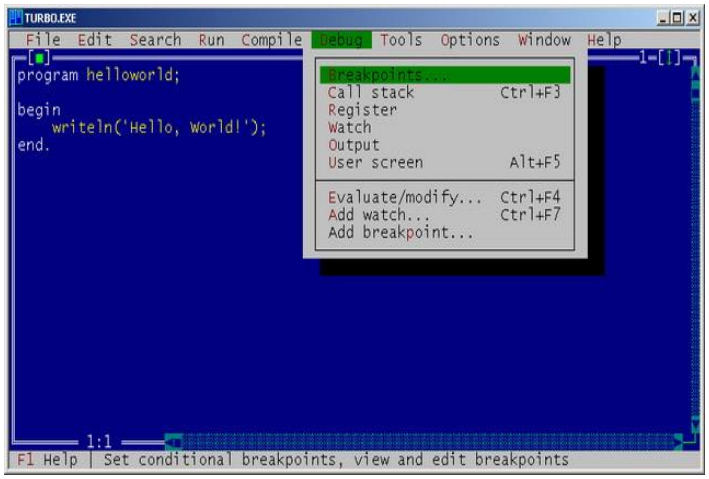


Слика 2.3. Конзола

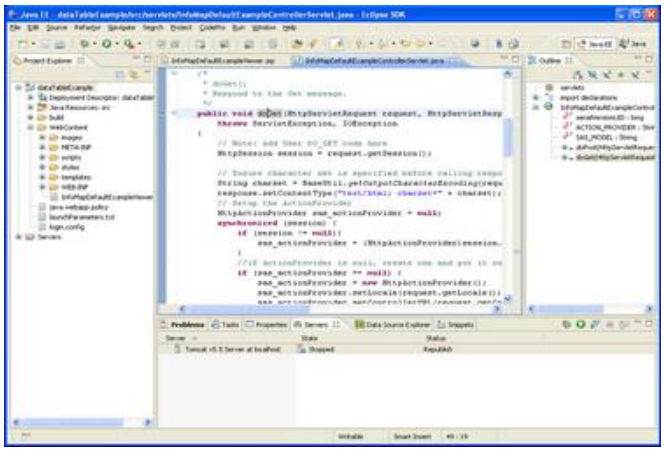
Основне карарктеристике

Основна одлика програма заснованих на прозорима јесте сама комуникација на релацији корисник-апликација. За разлику од програма командног типа где се проблем уноси искључиво преко тастатуре у виду одређених команди и чије решење добијамо у истом облику као и унос, код програма заснованих на прозорима уочавамо и друге визуелне компоненте које су од суштинског значаја. То су најчешће дугмад, оквири за текст, менији, и слично. Помоћу тих компоненти корисник управља радом апликације и прима информације од апликације.

Графичка разлика између ова два начина програмирања је приказана на упоредним сликама 2.4. и 2.5. где нам слика 2.4. приказује како се некада програмирало у графичком окружењу командног типа *Turbo Pascal*, а слика 2.5. нам приказује једно модерно програмирање са елементима графичког корисничког интерфејса у окружењу *Eclipse* где се најчешће програмира *Java* програмски језик.



Слика 2.4.Turbo Pascal



Слика 2.5.Eclipse

Свака апликација мора имати минимум један прозор који се обично назива **main window**. Наравно, поред њега може да постоји и неколико других прозора који се називају **secondary windows**. Главна разлика ових двеју врста прозора јесте у значају па самим тим и у положају на екрану. Главни прозор је увек приказан на екрану док се секундарни прозори по потреби појављују и нестају са екрана сходно потреби у том тренутку. Конкретно, једна специјална врста секундарних прозора јесте прозор за дијалог, **dial box**. Прозор за дијалог се иначе појављује када се кориснику требају саопштити разне поруке које су од значаја у том тренутку. Значај ових прозора је толики да докле год корисник не затвори овај прозор не може даље наставити са радом. То наравно значи да докле год се не обрати пажња која је порука исписана не може се наставити са даљим радом.

Уколико је у апликацији потребно радити истовремено са више прозора, они ће бити отворени један преко другог где се тренутно активни прозор једини види у целости, а остали су испод њега по редоследу како су отварани. Наравно, увек је могуће променити позицију самих прозора као и њихову величину уз помоћ миша, чији ћемо значај и функцију свакако обрадити.

Дакле, основни начин за управљње самом апликацијом и компонентама у њој јесте помоћу миша. То је такође једна од главних разлика између програма командног типа и програма заснованих на прозорима. Место на коме може да се делује мишем обележава се показивачем миша, **mouse cursor.** Тако, у зависности од функције која је тренутно додељена курсору, он може бити различито графички представљен. Најчешће је у облику стрелице.

Постоје две основне радње које могу да се обаве помоћу миша а то су притисак **click** и одвлачење **drag and drop**. Нећемо детаљно објашњавати ове две радње.

Поред миша, апликацијом можемо да управљамо и преко тастатуре. Код програма заснованих на прозорима основна употреба тастатуре јесте само за уношење текста у оквире за текст. Показивач текста тј. место у тексту на коме може нешто да се промени назива се **text cursor**. Она је графички представљена као вертикална трептећа линија. Под неким условима преко тастатуре можемо да производимо ефекте миша уз комбинацију специјалних тастера **Alt** и **Ctrl** са одређеним тастерима са словима.

Елементи графичког корисничког интерфејса

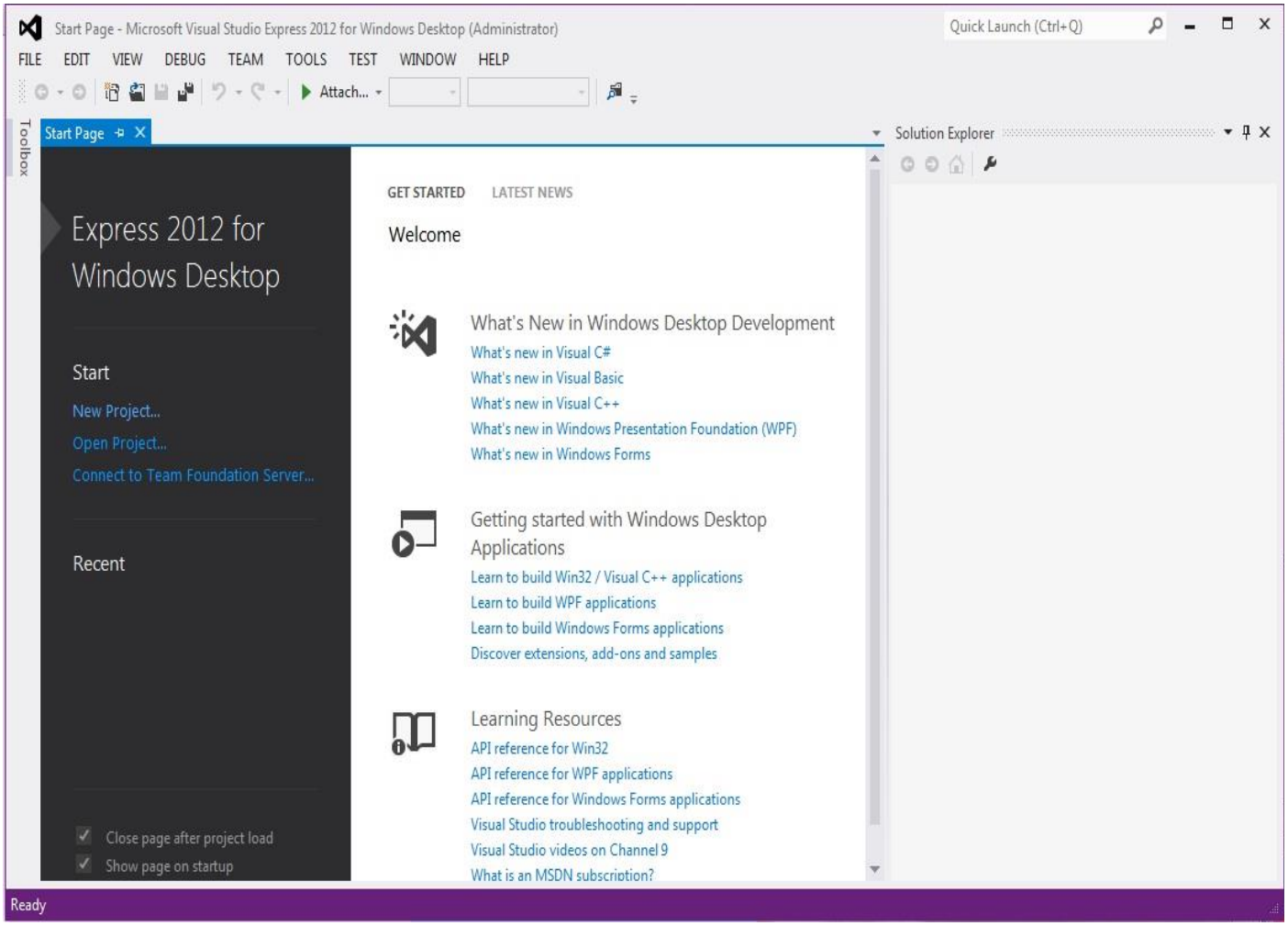
Графички кориснички интерфејс GUI (Graphical User Interface) чини прозор и окружење са којим корисник ради.
• Да бисмо видели како изгледа прозор и који су његови делови на примеру Visual Studio 2012 најпре ћемо проћи следеће кораке:

1. Покренимо ***Microsoft Visual Studio .NET***

•

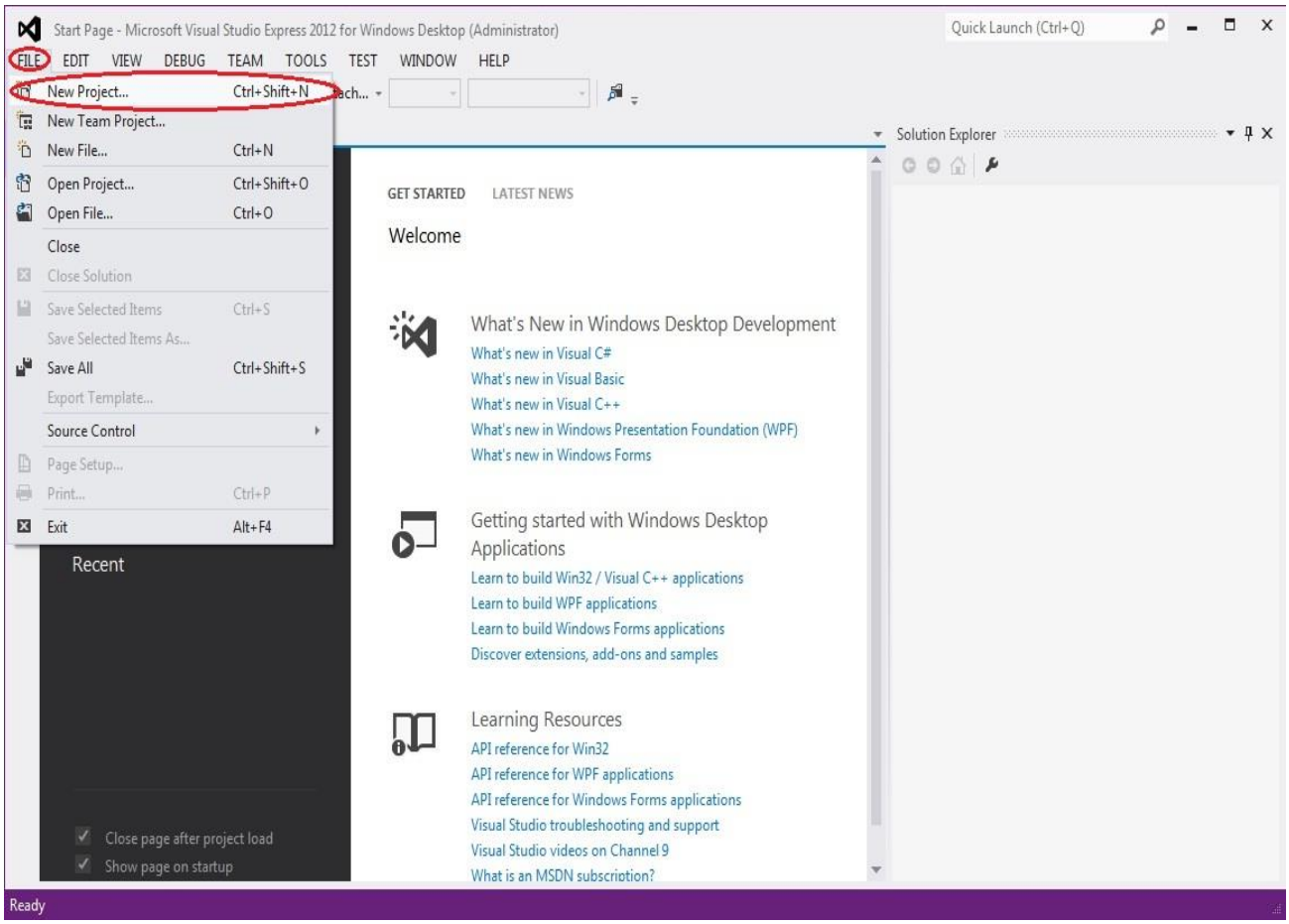
Слика 2.6. VS 2012

• Почетна страна



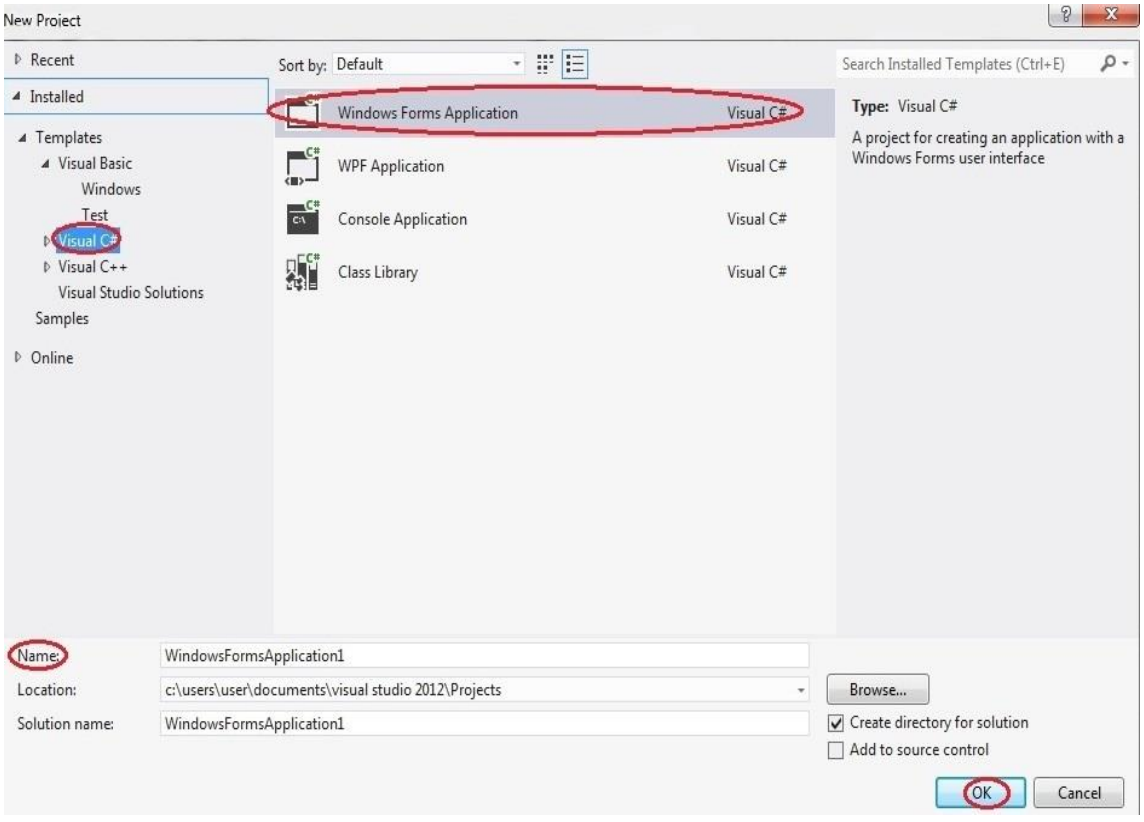
Слика 2.7. Почетна страна

- Изаберимо ***FILE->New Project*** или кликнимо на 



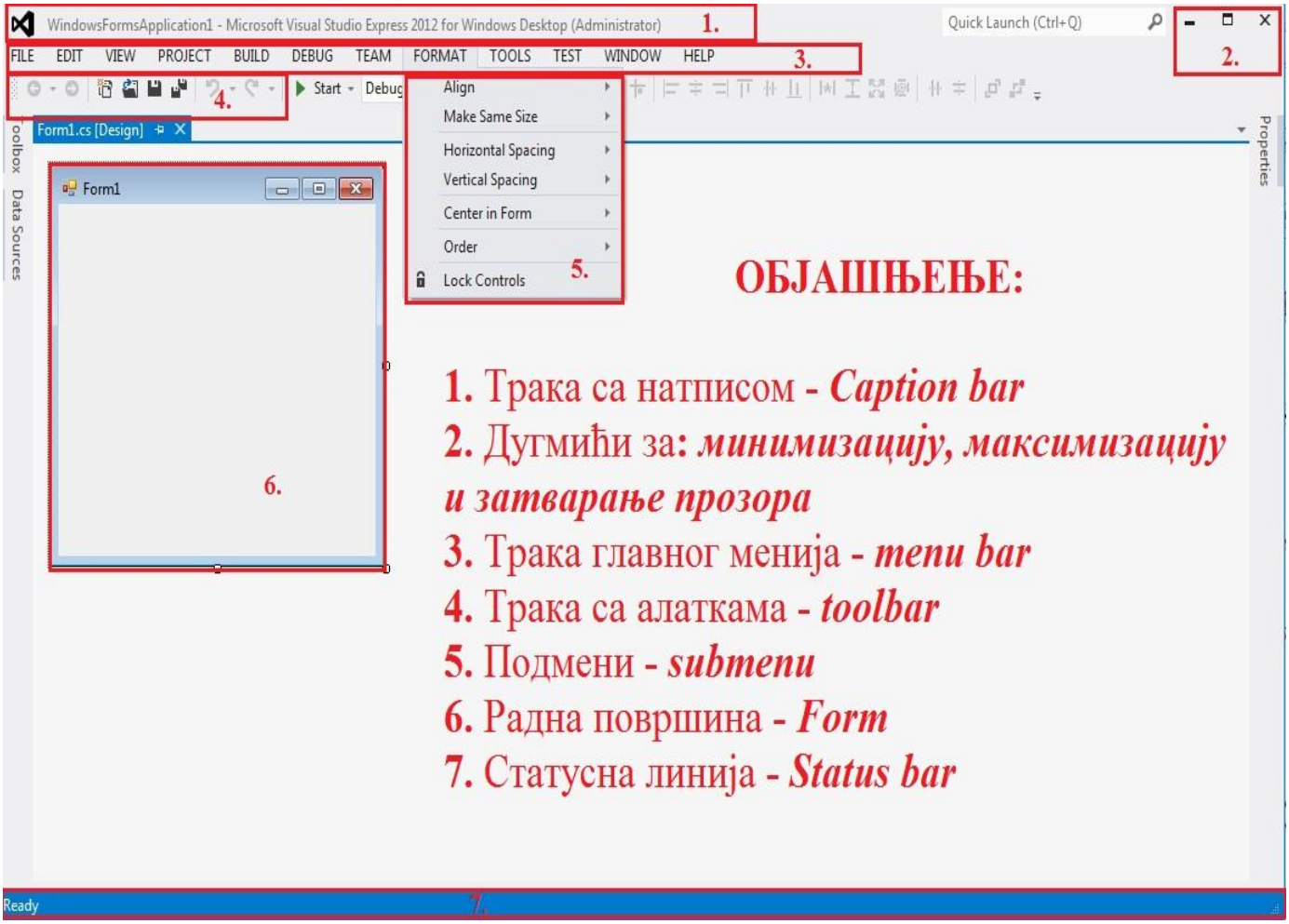
Слика 2.8. Прозор за отварање новог пројекта

- Отворен је **Project dialog** прозор: изаберимо **Visual C#** у листи **Installed**, затим бирамо **Windows Forms Application**, у текст поље **Name** унесимо име нашег пројекта или можемо прихватити већ понуђено име. Потврдимо са ***ОК***.



Слика 2.9. Прозор за отварање новог пројекта

- ***Ово је наш прозор!***



Слика 2.10. Радна површина

Слика 2.10. приказује главни прозор апликације Visual Studio 2012 и на њој су објашњени основни елементи за рад у овом окружењу.

1.Трака са натписом-Caption bar се налази уз горњу ивицу прозора. Састоји се од натписа (који садржи име апликације, у нашем случају: *WindowsFormsApplication1 - Microsoft Visual Studio*) и управљачких дугмета.

2.Управљачка дугмета: minimize, maximize, close. Притиском на ове дугмиће прозор ће се: смањити, раширити или затворити.

3.Трака главног менија- menu bar се налази испод траке са натписом и садржи ставке за избор појединих група радњи: File, Edit, View, Project, Build, Debug, Data...

4.Трака са алаткама - toolbar се налази испод траке главног менија и садржи дугмад са сличицама за брзо извршавање најкоришћенијих радњи.

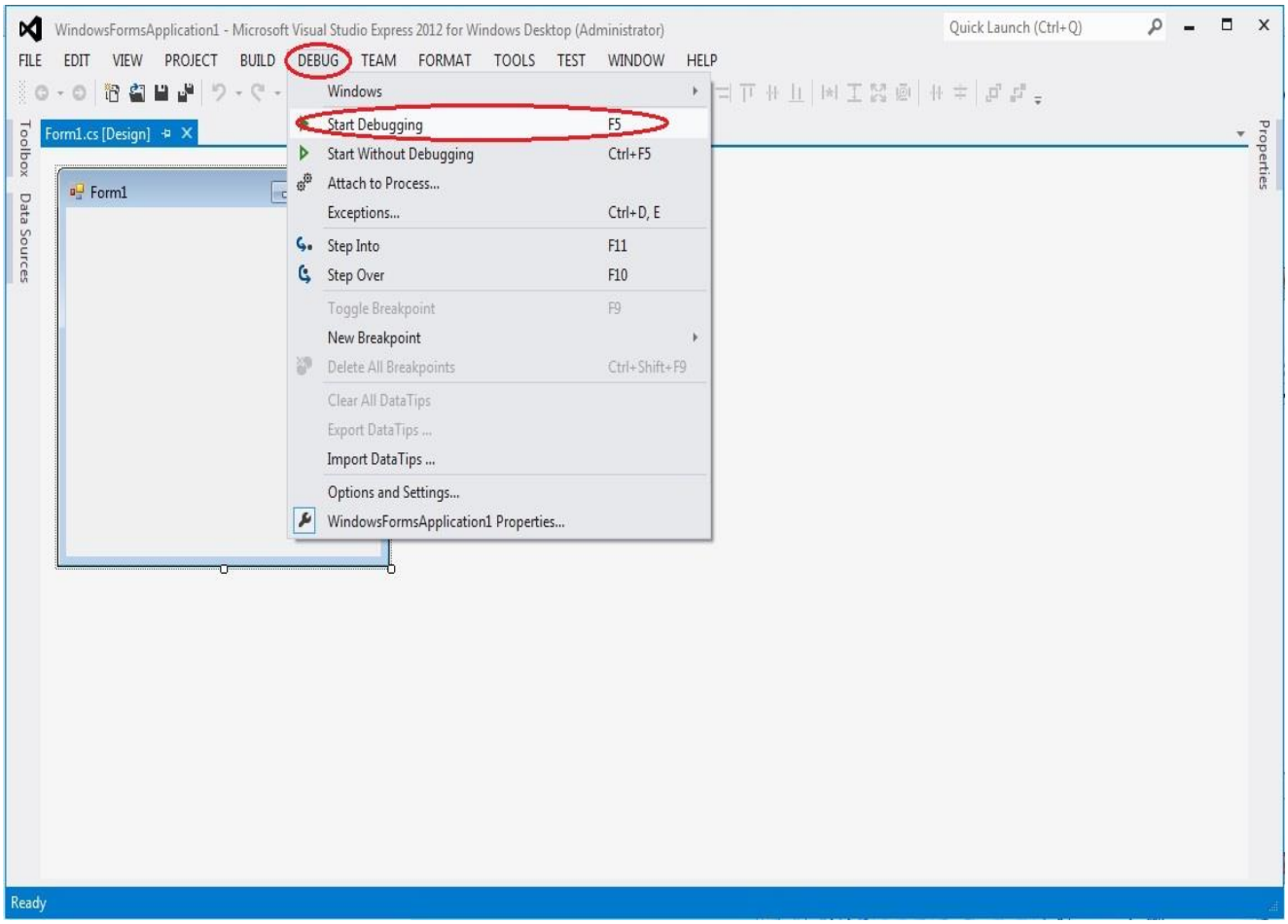
5.Подмени - submenu је видљив избором ставке из главног менија и садржи конкретне операције везане за исту.

6.Радна површина - Form заузима централни део екрана и види се када се програм извршава. Могуће је мењати њено име и друге параметре (боју, ширину, висину...)

7.Статусна линија - Status bar се налази уз доњу ивицу прозора и садржи податке о стању апликације (у нашем случају: Ready, што значи да је Visual Studio спреман за рад).

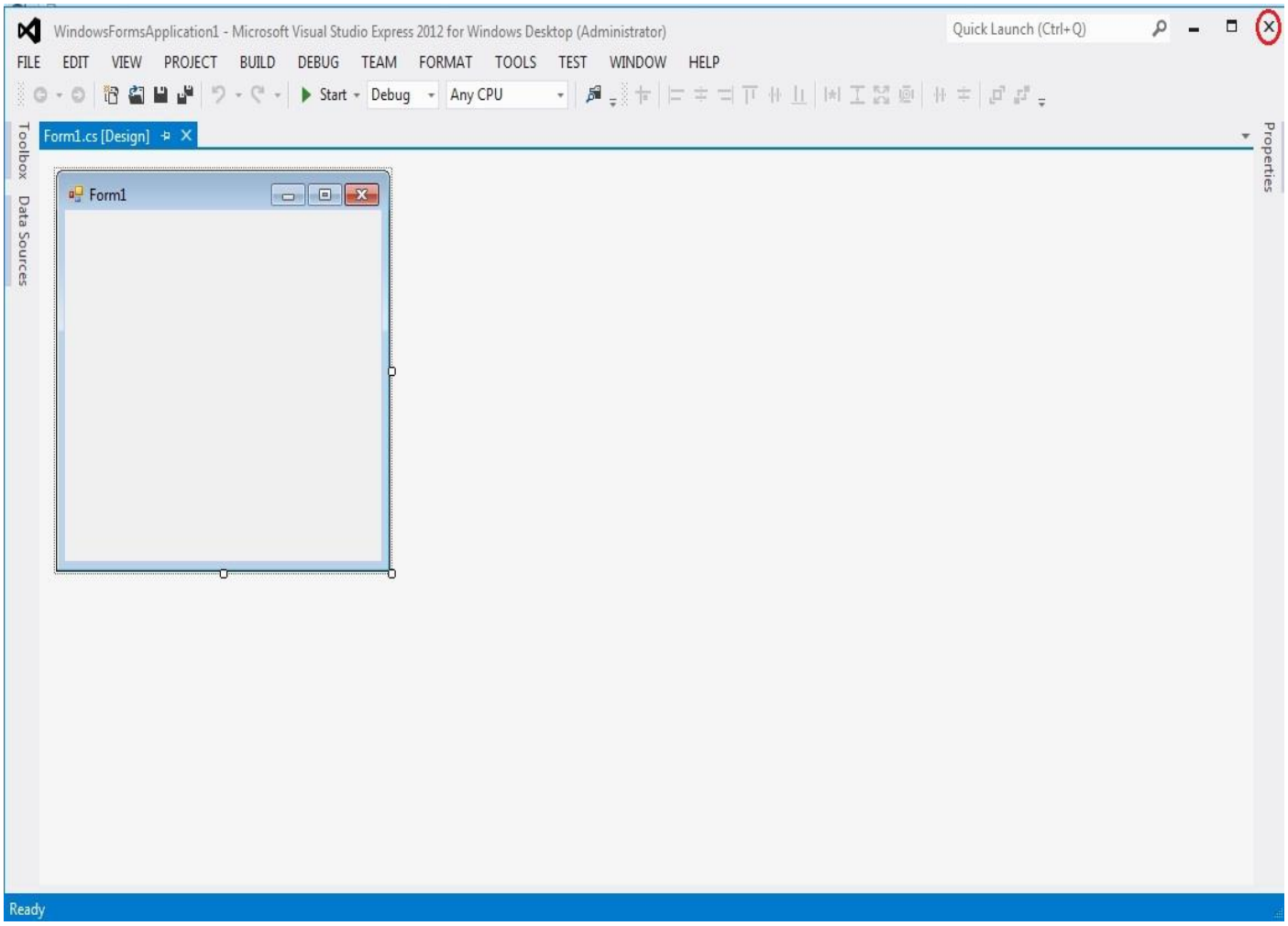
Ако се подсетимо раније научених апликација као што су нпр. Word или Excell и упоредимо са Сликом 2.10, јасно је да међу њима постоје велике сличности, које кориснику омогућавају лако сналажење и коришћење.

- Да бисмо превели и покренули апликацију притиснемо F5 или одаберимо из главног менија ***DEBUG->Start Debugging***



Слика 2.11. Радна површина

- ***Затварамо*** га на дугме 



Слика 2.12. Радна површина

Програми руковођени догађајима (догађаји, извори догађаја и обрада догађаја)

Пошто је у објектно оријентисаном програмирању тежиште на: ***објектима*** и ономе што може да се догоди са њима или око њих (односно различитим ***догађајима***) морамо разумети шта све може бити догађај и како да њиме рукујемо. Догађај је све оно што урадимо помоћу миша, тастатуре, али и све оно што је последица извршавања програма или неког његовог дела. Тако да су: клик миша, прелазак показивача миша преко објекта, промена димензија, превлачење објекта, унос вредности и др. догађаји.


Слика 2.18. Тастатура и миш