Коришћење алата Microsoft Visual Studio C++ 2015 и технике контролисаног извршења програма са циљем отклањања грешака

Увод – Microsoft Visual Studio 2015

Visual Studio интегрисано развојно окружење (енг. ***I**ntegrated **D**evelopement **E**nvironment – **IDE***) нуди скуп алата који омогућавају лакше писање и измену кода програма, као и откривање и исправљање грешака у програмима. Ова вежба објашњава формирање новог стандардног C++ програма и тестирање његове функционалности коришћењем алата доступних у Visual Studio окружењу.

Овом вежбом обухваћене су следеће теме:

- Рад са пројектима (стварање и формирање, осврт на битне опције),
- Коришћење претраживача пројекта/решења (енг. *Solution Explorer*),
- Додавање постојеће и прављење нове датотеке са изворним кодом,
- Поправљање грешака приликом превођења и увезивања,
- Тестирање програма и
- Технике отклањања грешака у програму.

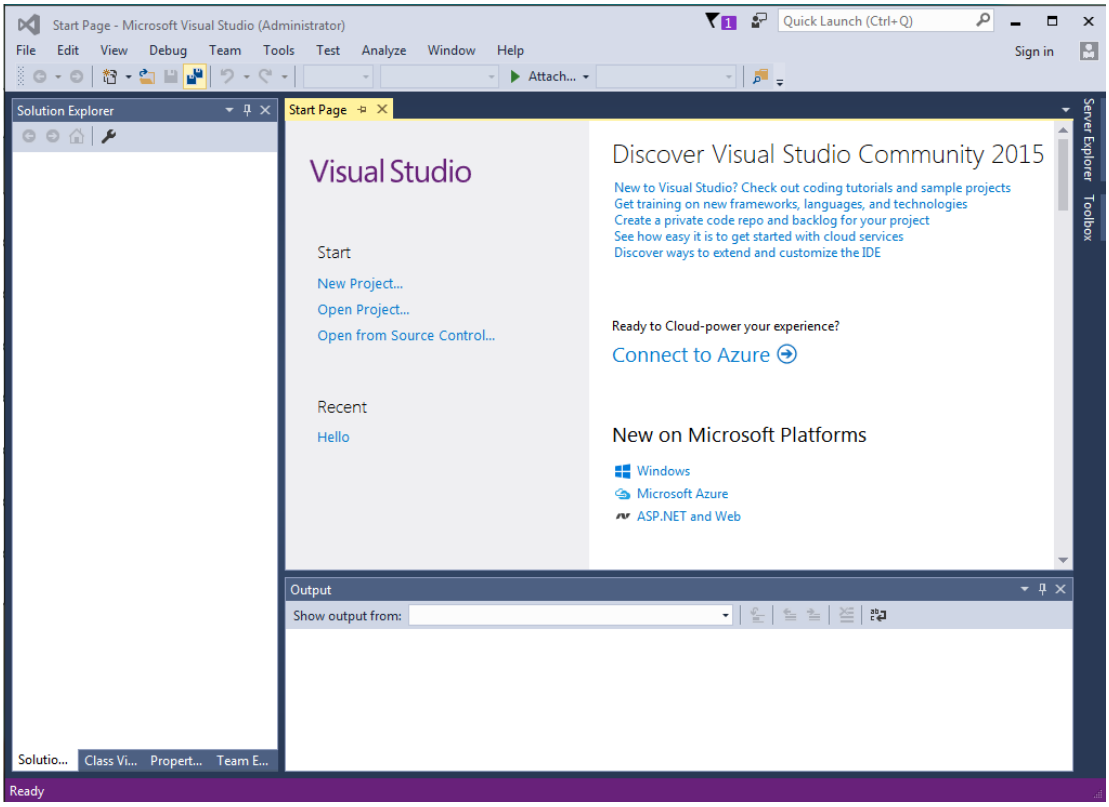
Рад са пројектима

Visual Studio организује рад на изради програма у пројекте и решења (енг. *Solutions*). Решење може садржати више од једног пројекта, као што је, на пример, случај када постоји једна динамичка библиотека (енг. ***D**ynamic-**L**ink **L**ibrary – DLL*) и извршна датотека која се упућује на DLL датотеку. У примерима и задацима на овом курсу обично ће се радити са једним пројектом у једном решењу.

Први корак у писању Visual C++ програма са *Visual Studio* алатом јесте формирање новог пројекта и одабир типа пројекта. За сваки тип пројекта, *Visual Studio* поставља подешавања специфична за сваки преводилац (енг. *compiler*), и по жељи, ствара почетни код.

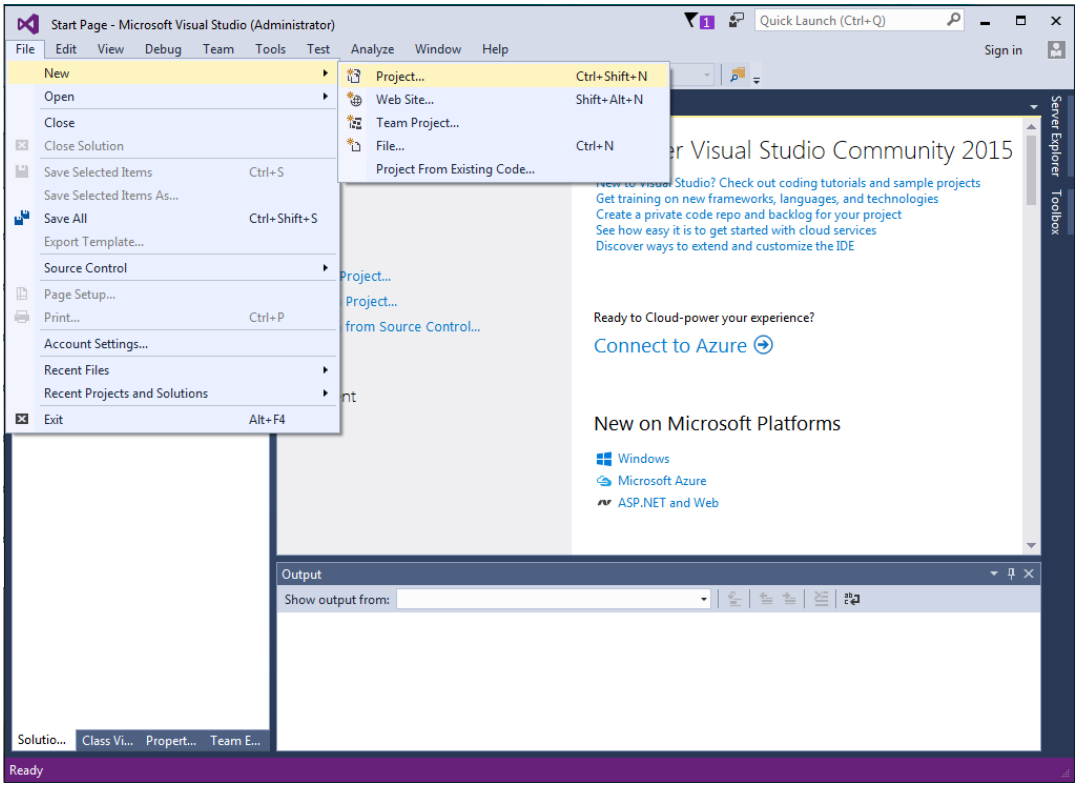
Стварање новог пројекта

1. Најпре је потребно покренути *Microsoft Visual Studio* 2015 са радне површине (енг. *Desktop*) или са путање **Start > All Programs > Microsoft Visual Studio 2015**.
2. По отварању графичког окружења, затворите почетну страницу, Слика 1.



Слика 1 – Графичко окружење – почетна страна

3. Из ***File*** падајућег менија, одаберите ***New*** и кликните на ***Project...***, Слика 2.



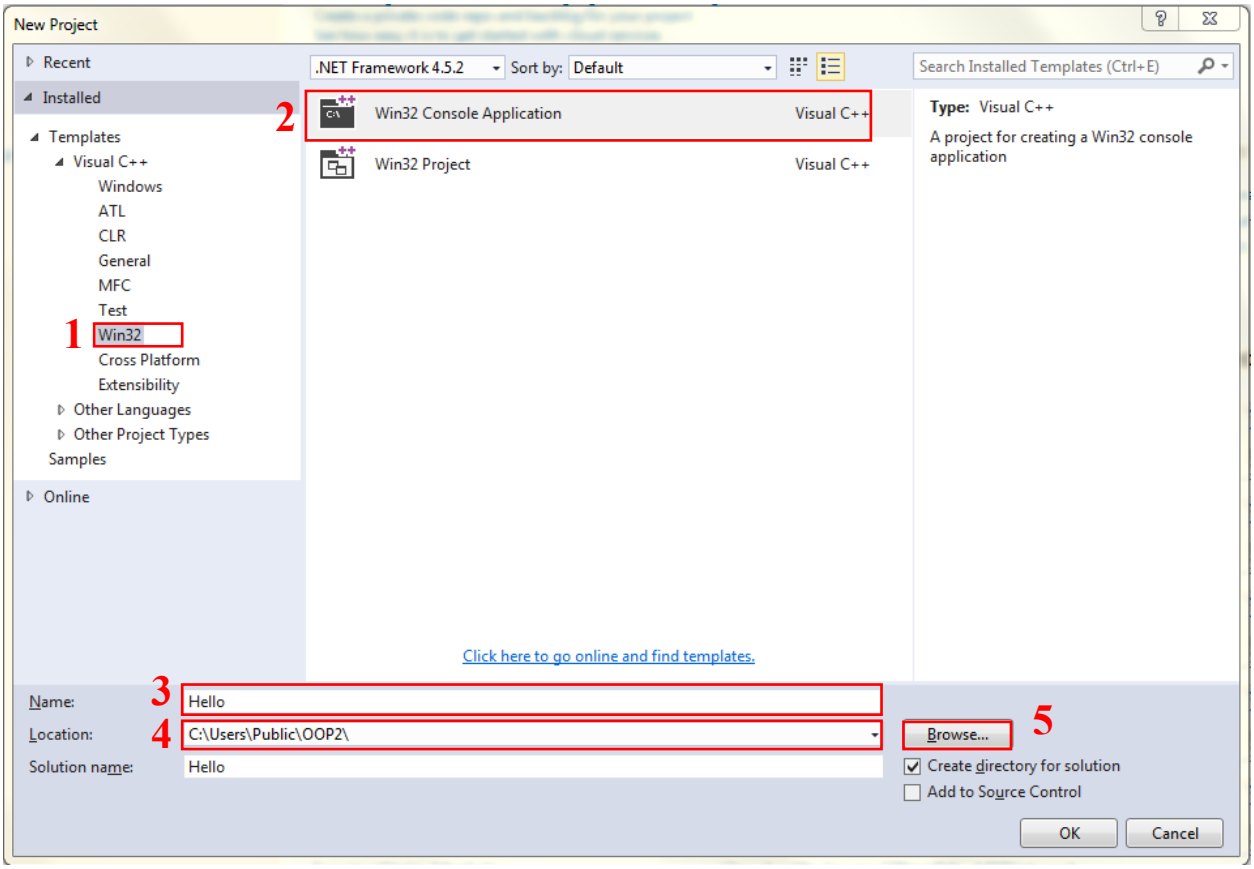
Слика 2 – Отварање прозора *New Project*

4. У области ***Installed, Templates*** под ***Visual C++*** одаберите ***Win32*** Слика 3, тачка 1, а потом, у ***Visual Studio installed templates*** области, одаберите ***Win32 Console Application***, тачка 2.

5. Упишите **Hello** за име пројекта и одаберите **путању** на којој желите да пројекат буде сачуван, Слика 3, тачке 3 и 4.

Када направите нови пројекат, *Visual Studio* смешта пројекат у решење (енг. *Solution*). Подразумевано име решења је исто као име пројекта, али је могуће променити га. Уколико не желите да посебан директоријум буде направљен за решење, можете искључити ту могућност, Слика 3, тачка 5.

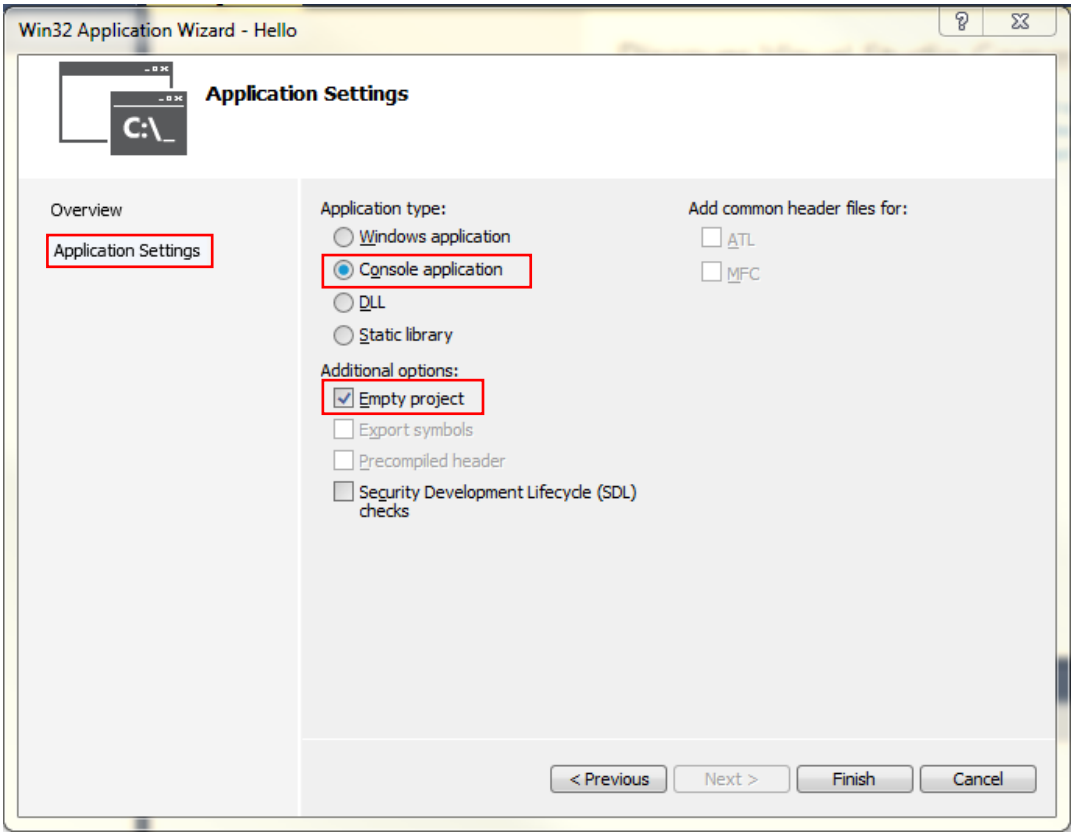
6. Кликните на ***OK*** да бисте покренули ***Win32 Application Wizard***.



Слика 3 – Одабир типа пројекта у прозору *New Project*

7. На ***Overview*** страници ***Win32 Application Wizard*** дијалога, кликните на ***Next***.

8. На страници ***Application Settings***, у групи ***Application type***, оставите одабрано ***Console Application***. У групи ***Additional options*** искључите све опције изузев опције ***Empty Project*** и кликните на ***Finish***, Слика 4.



Слика 4 – Одабир додатних опција новог пројекта

Одабиром опције ***Empty Project*** створили сте пројекат без датотека са изворним кодом.

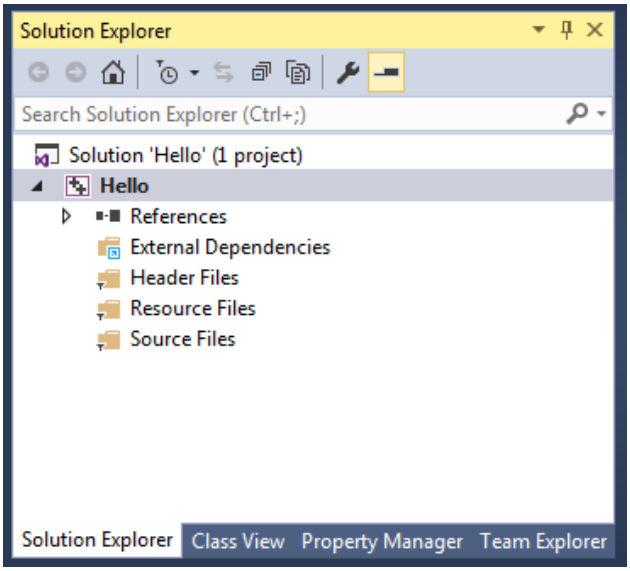
Отварање постојећег пројекта

Постојећи пројекат можете отворити покретањем било пројектне (*Project*) ***.vcproj*** датотеке, било ***.sln*** датотеке са решењем (*Solution*).

Коришћење претраживача пројекта/решења (Solution Explorer)

Уколико прозор *Solution Explorer* није видљив, у ***View*** менију одаберите ***Solution Explorer***.

Помоћу *Solution Explorer*-а могуће је прегледати и организовати постојеће датотеке у пројекту. Датотеке се групишу на основу типа у одговарајућу групу – филтар, Слика 5. Подразумевана су три филтра – *Header Files*, *Resource Files* и *Source Files*. У *Header Files* групи налазе се ***.h*** датотеке – заглавља (енг. *Header Files*), у *Source Files* групи су датотеке са изворним кодом (***.c*** или ***.cpp***), док се у групи *Resource Files* налазе смештају ресурси које обухвата пројекат (током овог предмета ту групу нећемо користити). Филтре је могуће по потреби брисати или додавати нове.

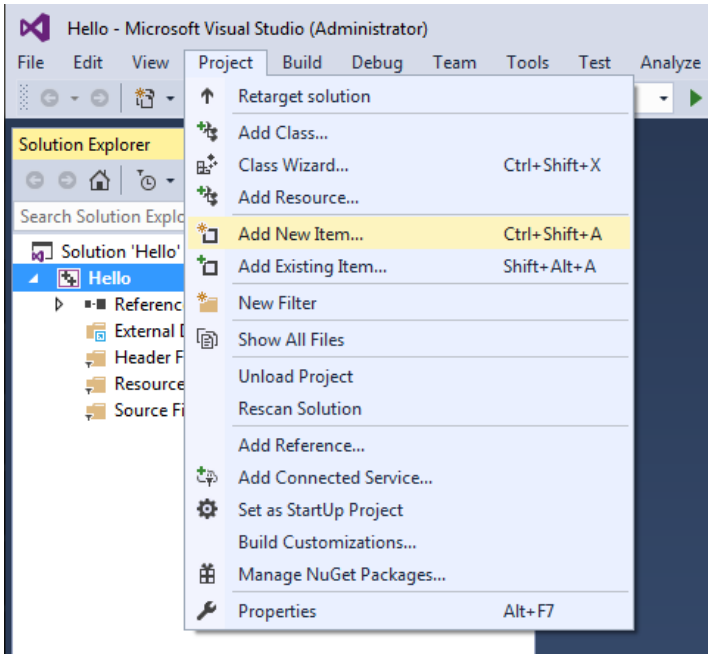


Слика 5 – Solution Explorer

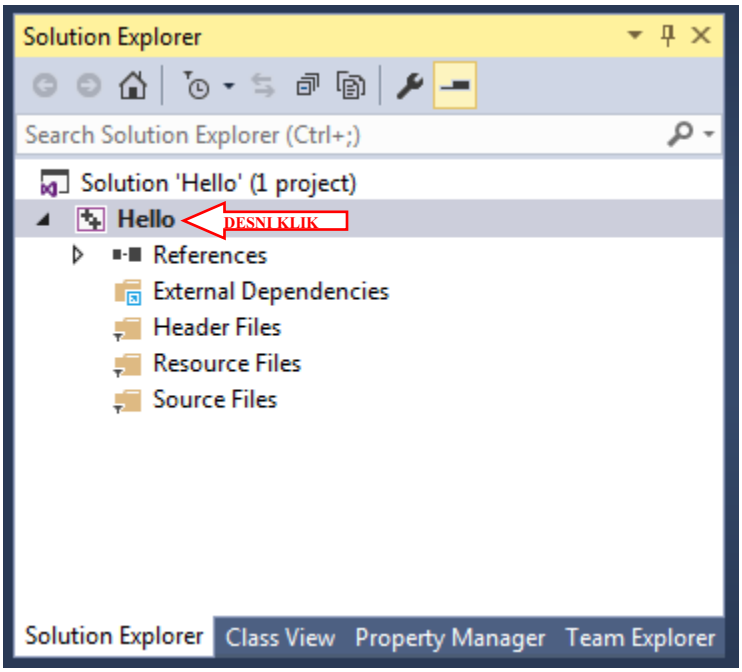
Додавање нове датотеке са изворним кодом

Нове датотеке са изворним кодовима могуће је формирати и додати у пројекат на више начина.

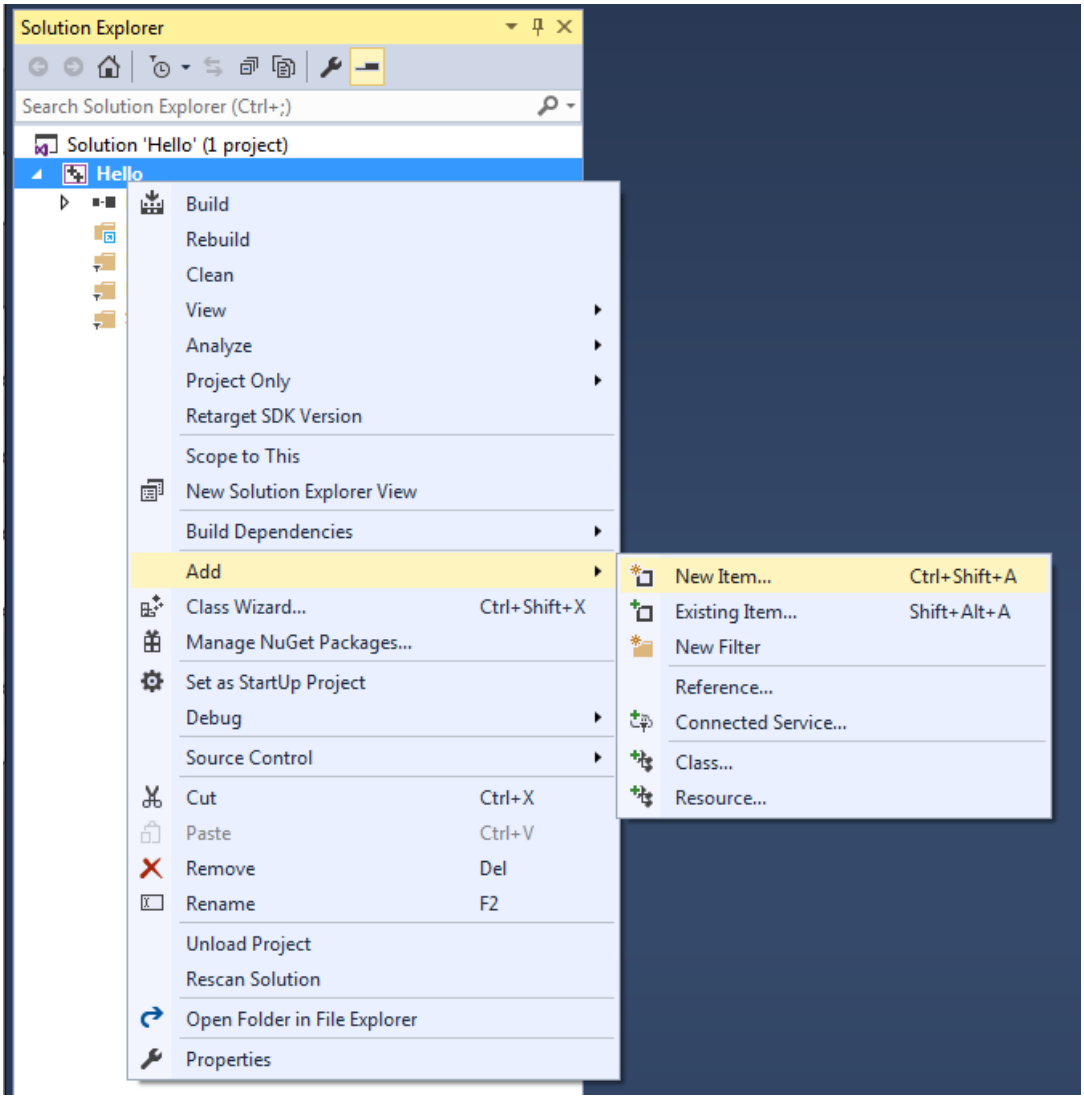
1. *Add New Item* прозор је могуће отворити одабиром опције ***Add New Item...*** из падајућег менија ***Project***, Слика 6. Други начин је у *Solution Explorer*-у, десним кликом на име решења, пројекта или неког од филтара (Слика 7), у зависности од тога где желите да нова датотека буде смештена, у добијеном менију одаберете подмени ***Add*** и опцију ***New Item...***, Слика 8. Уколико је десни клик био на крајњи филтар, датотека ће по формирању бити смештена у њега, уколико је одабран пројекат, датотека ће бити смештена у пројекат и у одговарајућу групу у складу са правилима филтрирања, итд.



Слика 6 – Отварање *Add New Item* прозора из *Project* падајућег менија

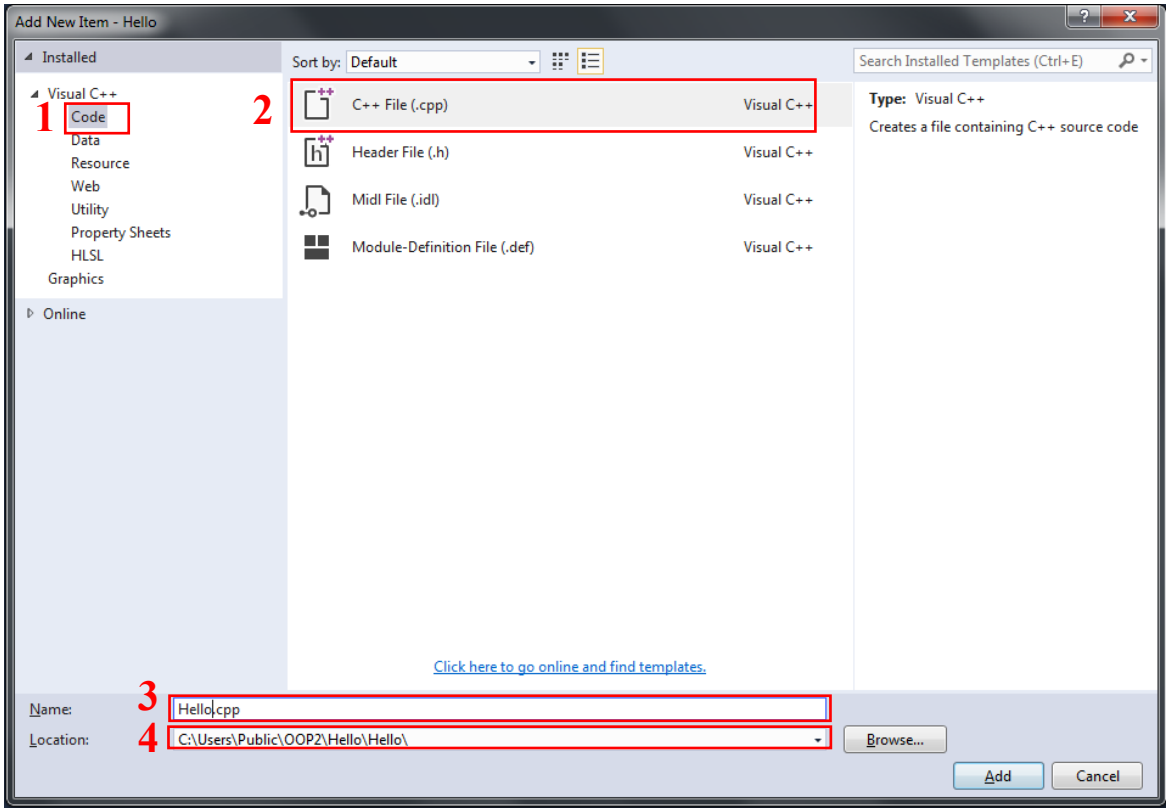


Слика 7 – Одабир одредишта нове датотеке у организацији пројекта



Слика 8 – Отварање *Add New Item* прозора из *Solution Explorer*-а

2. У прозору *Add New Item*, за категорију изаберите ***Code*** (Слика 9, опција 1), а у пољу *Templates* (Слика 9, опција 2) одаберите тип датотеке коју желите да додате у пројекат (најчешће ***C++ File (.cpp)*** или ***Header File (.h)***). У поље ***Name*** унесите назив датотеке заједно са проширењем (нпр. ***Hello.cpp***), Слика 9, опција 3, а у поље ***Location*** унесите путању на коју ће датотека физички бити смештена (Слика 9, опција 4), подразумевано у директоријум пројекта. Кликните на дугме ***Add***, да бисте додали нову датотеку, Слика 9.

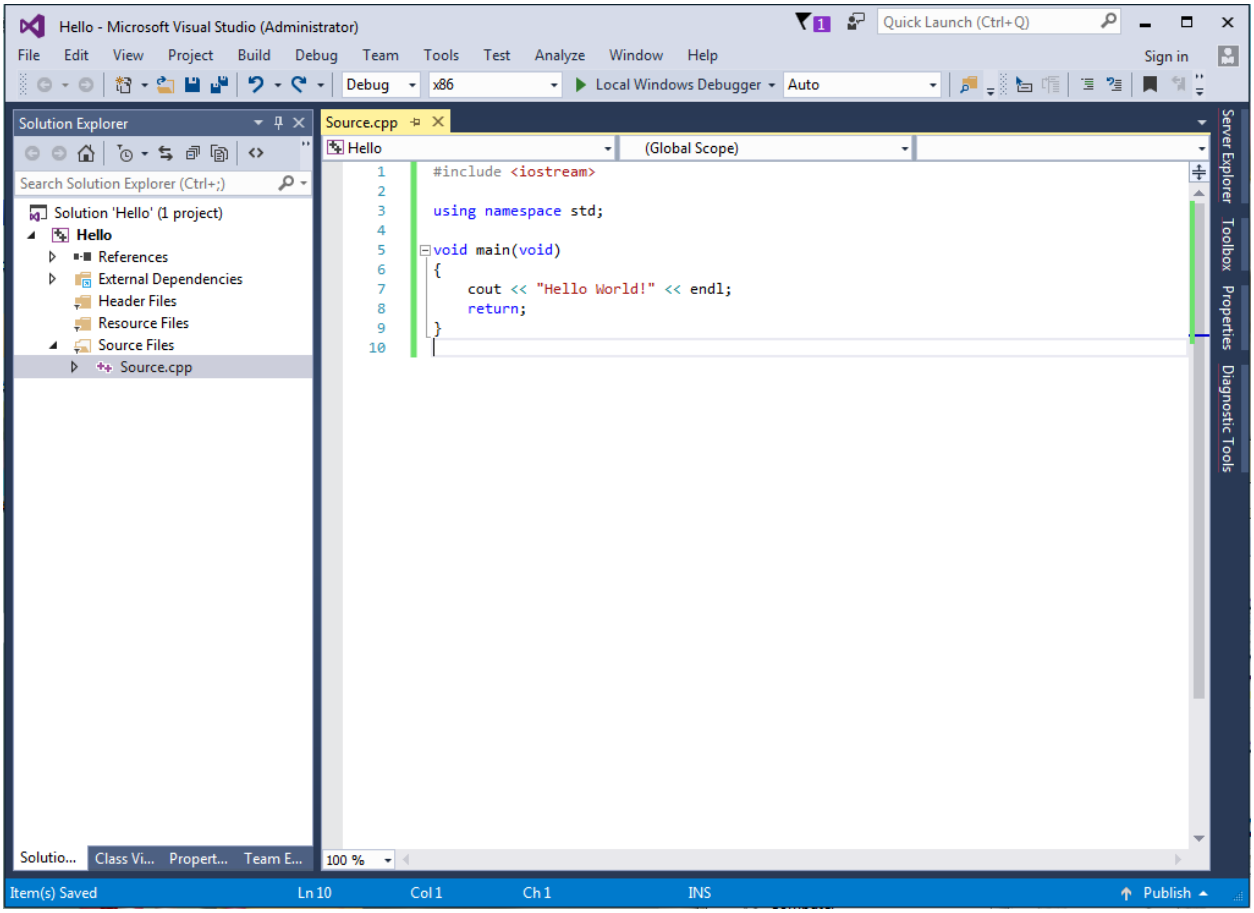


Слика 9 – Опције у *Add New Item* прозору

Напомена: Правилан избор проширења у имену датотеке је веома битан! Нпр. уколико доделите проширење *cpp* датотеци која садржи коректан C програмски код, преводилац ће сматрати да се у њој налази C++ програмски код, што може довести до појаве синтаксних грешака у фази превођења и обрнуто.

3. Унесите програмски код у датотеку *Hello.cpp*. Нпр:

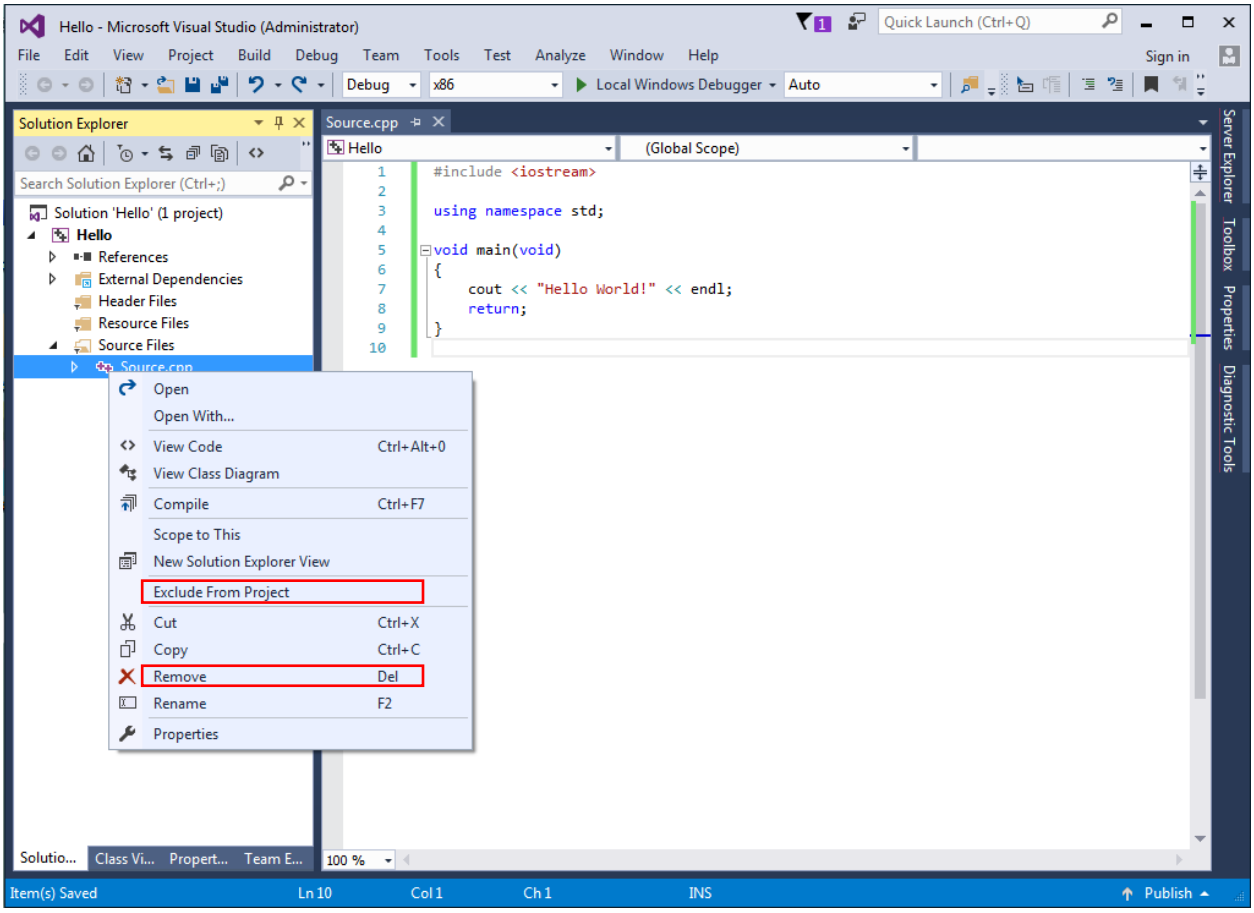
```
#include <iostream>

using namespace std;

void main(void)
{
    cout << "Hello World!" << endl;
    return;
}
```



Слика 10 – Пројекат са додатом датотеком са изворним кодом

Брисање датотеке са изворним кодом

Уколико желите да обришете неку датотеку са изворним кодом, најлакши начини да то урадите је коришћењем *Solution Explorer*-а. Позиционирањем на жељену датотеку и притиском тастера *Delete* на тастатури, или одабиром опције ***Remove*** из падајућег менија добијеног десним кликом на датотеку, Слика 11. Код брисања, *Visual Studio* ће вам понудити две могућности: ***Remove*** – уклањање само из пројекта или ***Delete*** – уклањање и физички са диска. Поред опције ***Remove***, у истом падајућем менију налази се и опција ***Exclude From Project*** (Слика 11), која само искључује датотеку из пројекта, али датотека остаје физички на диску.

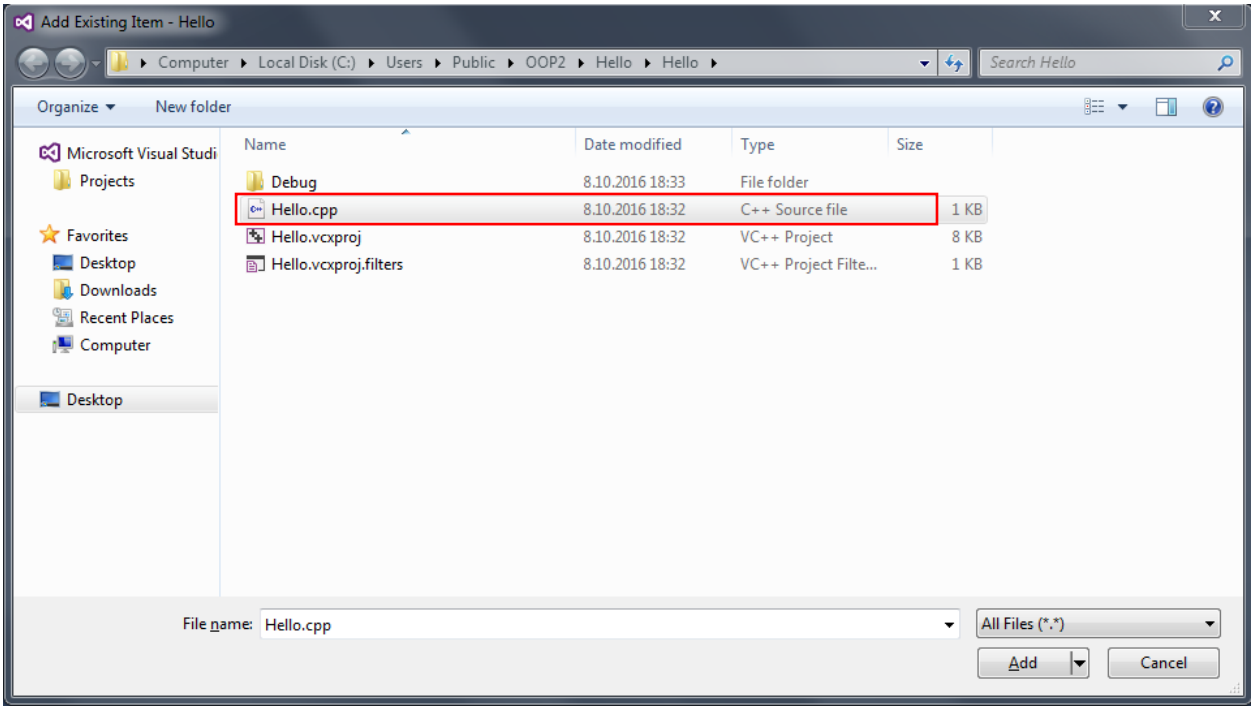


Слика 11 – Брисање постојеће датотеке са програмским кодом

Додавање постојеће датотеке са изворним кодом

Уколико сте у претходном кораку, датотеку само исључили из пројекта, сада је можете додати у пројекат као постојећу датотеку са изворним кодом.

1. Постојећа датотека се додаје на исти начин као и нова, само се бира опција ***(Add) Existing Item...***, уместо опције ***(Add) New Item...***
2. У прозору ***Add Existing Item***, одаберите путању до постојеће датотеке и саму датотеку и потом кликните на ***Add***, Слика 12.



Слика 12 – Додавање постојеће датотеке у пројекат

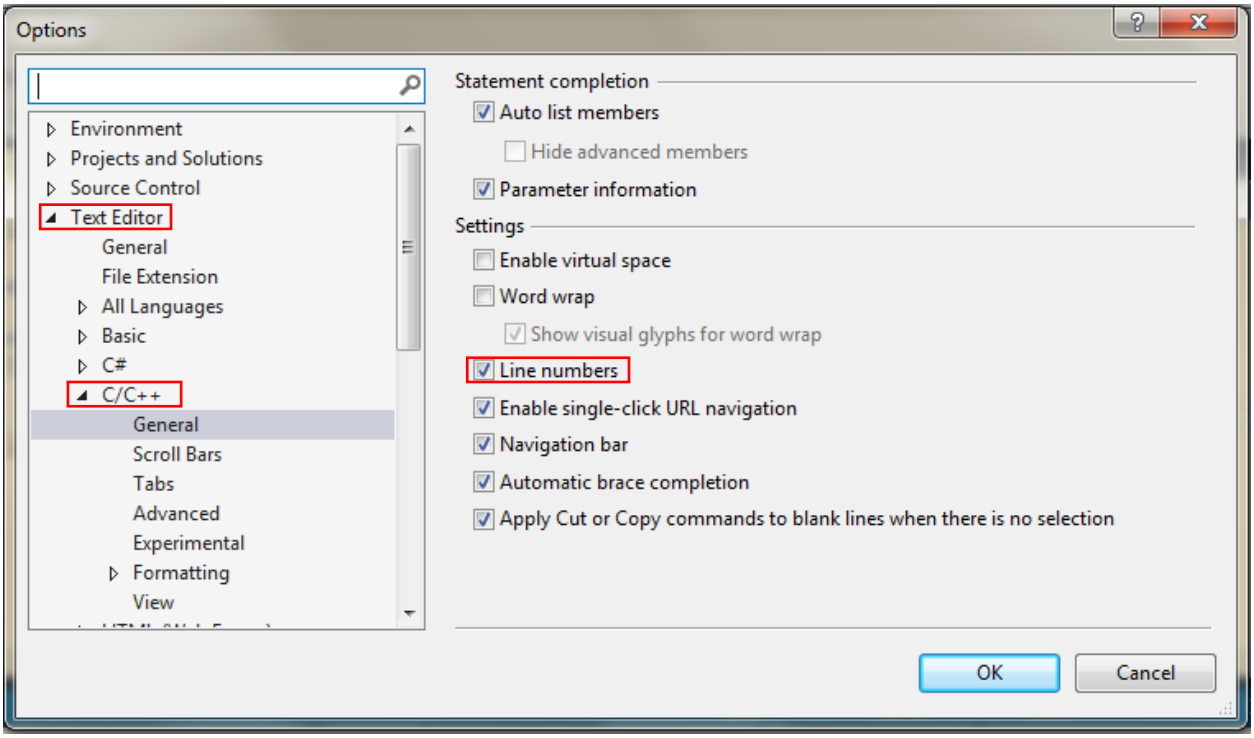
Превођење и увезивање програма и исправљање грешака

У овом кораку, намерно ћемо унети грешку у изворни код у датотеци *Hello.cpp*.

1. У датотеци *Hello.cpp*, у линији 5, обрисаћемо тачку-зарез (;), иза службене речи „return”, тако да добијемо:

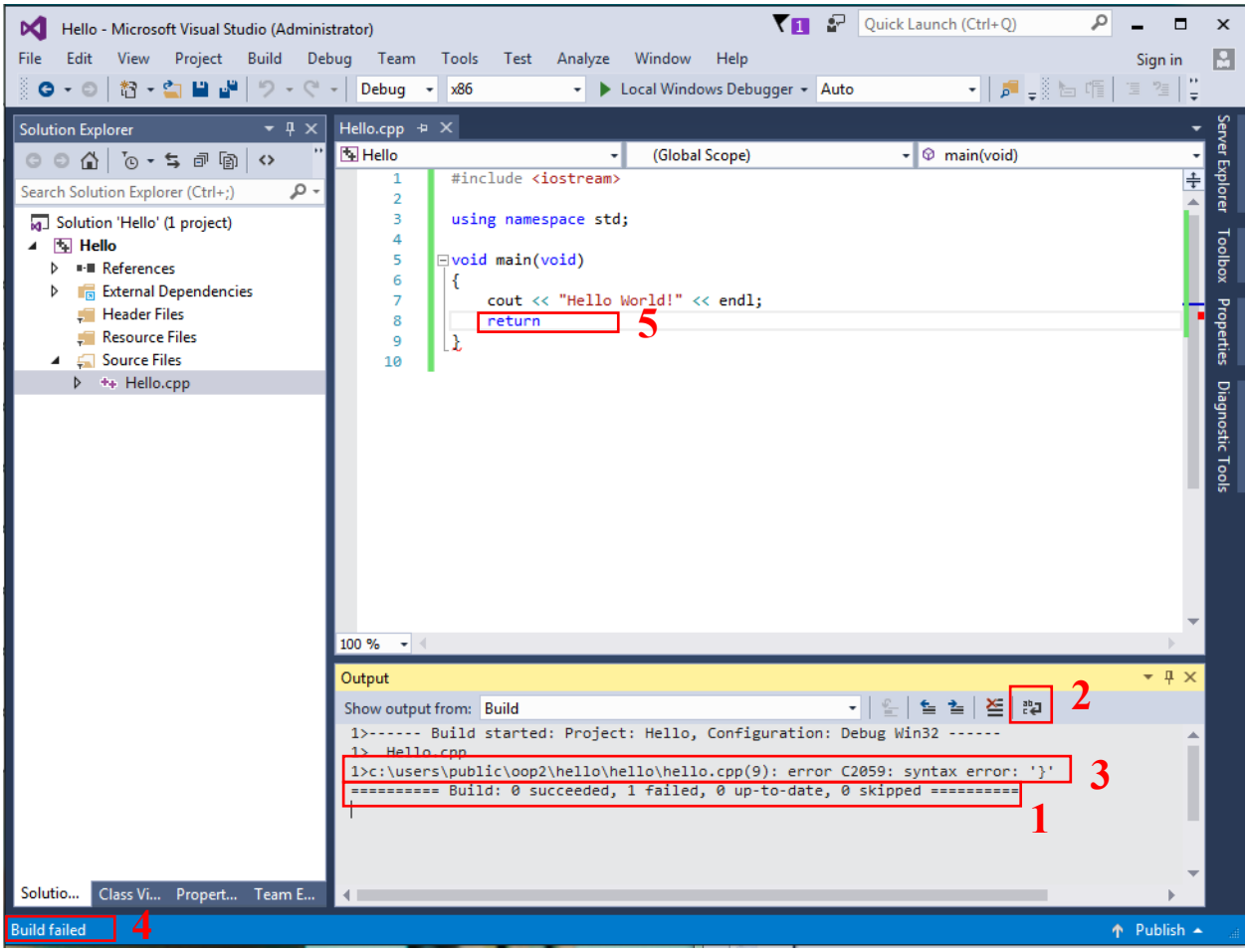
```
    return
```

Уколико се бројеви линија не виде, њихов приказ можете укључити одабиром опције ***Options...*** из ***Tools*** падајућег менија. У прозору *Options* изаберите ***Text Editor***, ***C/C++*** и одаберите опцију ***Line numbers*** у групи *Display*, Слика 13. Потврдите измене кликом на дугме ***OK***.



Слика 13 – Укључивање приказа броја линија

2. У ***Build*** менију одаберите ***Buid Solution*****,** или притисните тастер F7.
3. Порука у ***Output*** прозору показује да је превођење пројекта било неуспешно, Слика 14, тачка 1. Кликните на ***Go To Next Message*** дугме (зелена стрелица удесно) у ***Output*** прозору, Слика 14, тачка 2. Порука о грешци у ***Output*** прозору (Слика 14, тачка 3) и статус бар области (Слика 14, тачка 4) означава да недостаје тачка-зарез пре затворене велике заграде. Притиском на тастер **F1** можете добити више информација о конкретној грешци. Истовремено, у датотеци *Hello.cpp*, курсор је позициониран на линију у којој се налази грешка, Слика 14, тачка 5.



Слика 14 – Резултати превођења и увезивања

НАПОМЕНА: Ова информација о проблему на који је Visual Studio преводилац наишао не мора увек бити прави узрок проблема. Пример који је дат то већ делимично илуструје, јер линија у којој је грешка пријављена (линија 9) није линије у којој је грешка начињена (линија 8), то јест у којој ће бити исправљена, додавањем; после *return* (**Питање:** Зашто грешка није пријављена у линији 8?). Због тога је важно, у случајевима када пријављени проблем није очигледан, да се код око пријављеног места грешке добро погледа и разуме.

4. Додајте тачку-зарез на крај линије са синтаксном грешком:

```
return;
```

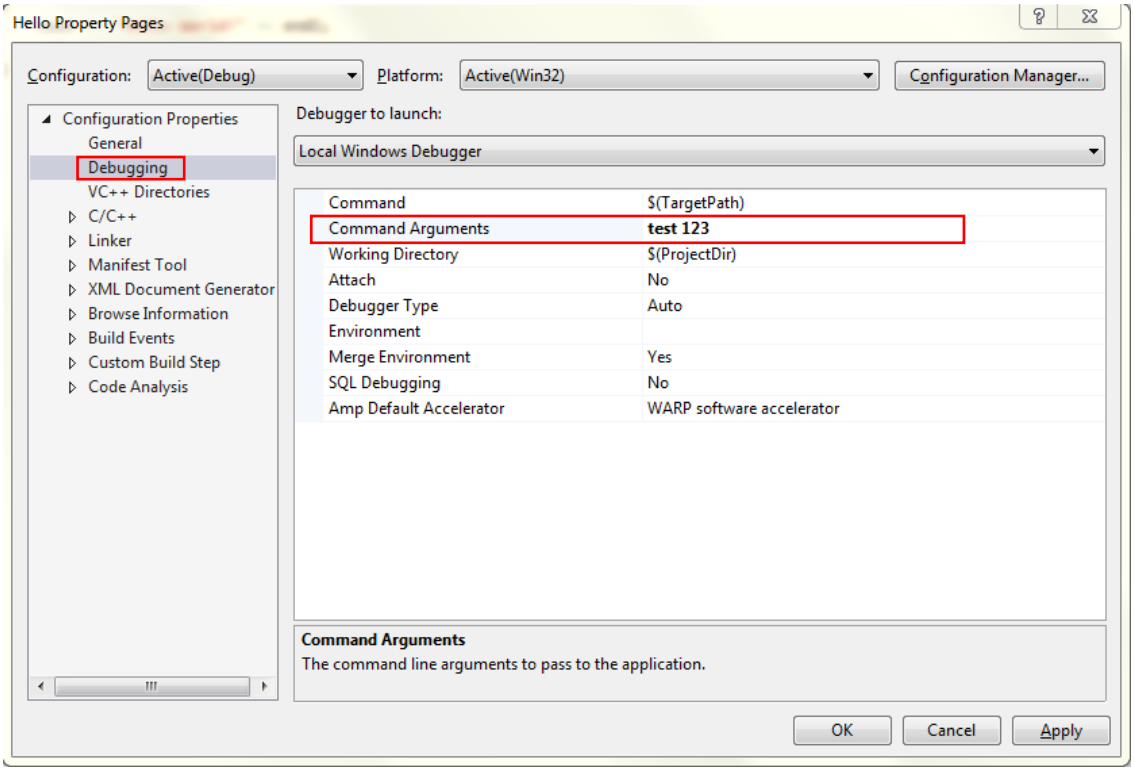
5. У ***Build*** менију одаберите ***Buid Solution***.
6. Порука у ***Output*** прозору показује да је превођење и увезивање пројекта било успешно.

<u>Додатни параметри пројекта</u>

Параметре пројекта можете подешавати у прозору ***<Project Name> Property Pages*** који отварате избором ставке ***<Project Name> Properties...*** из менија ***Project*** или одабиром опције ***Properties*** у падајућем менију добијеном **десним кликом** на жељени пројекат у *Solution Explorer*-у.

Најважније додатне опције/параметре пројекта можете подесити на следећи начин, у групи ***Configuration Properties***:

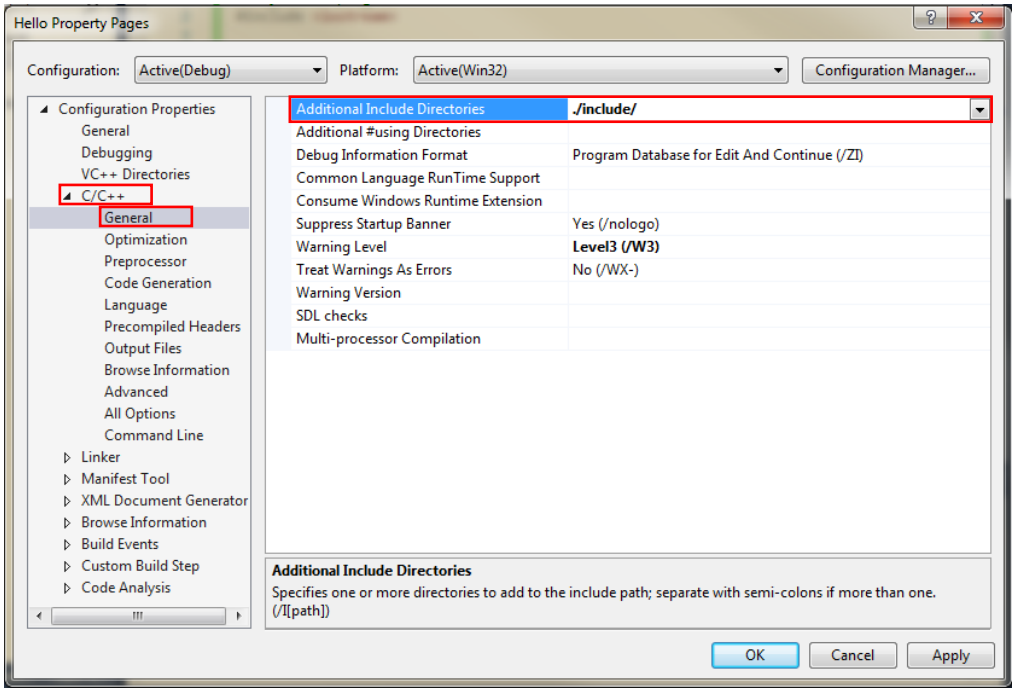
- Постављање аргумената који се приликом покретања програма преносе преко командне линије, ***Command Arguments*** у ***Debugging*** групи, Слика 15.



Слика 15 – Пренос аргумената преко командне линије

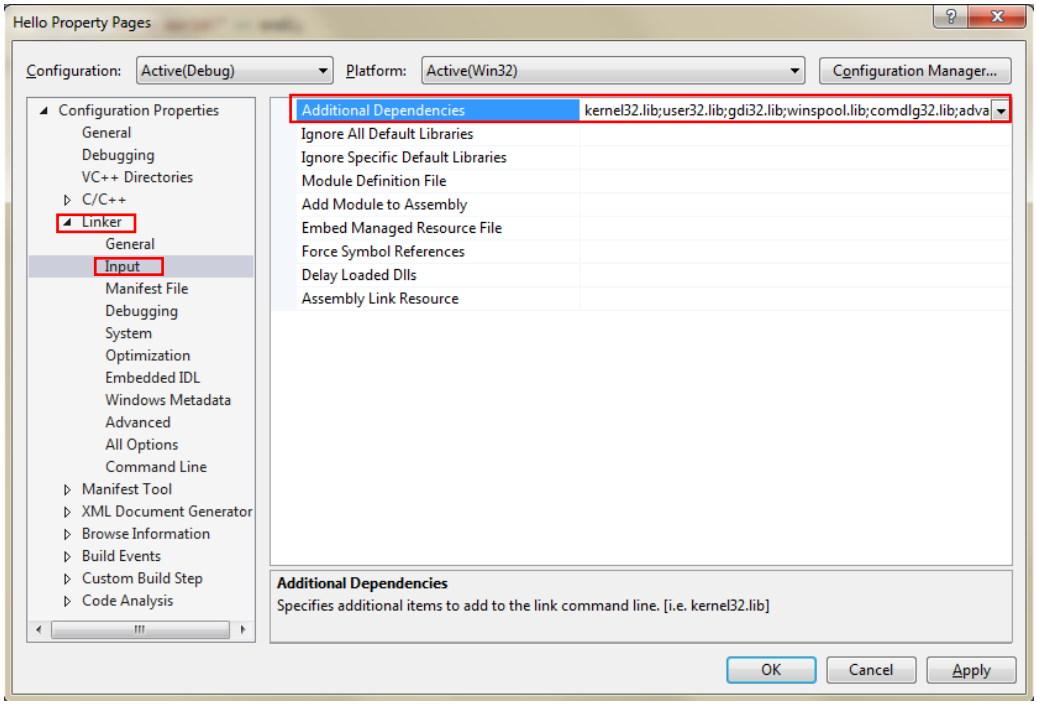
У конкретном примеру са слике, главној (*main*) функцији *Hello* програма биће прослеђена два аргумента – „test” и „123”.

- Додавање путање до директоријума са *include* датотекама, које нису у оквиру пројекта или пројектног директоријума налази се у групи ***C/C++***, ***General*** као опција ***Additional Include Directories***, Слика 16.



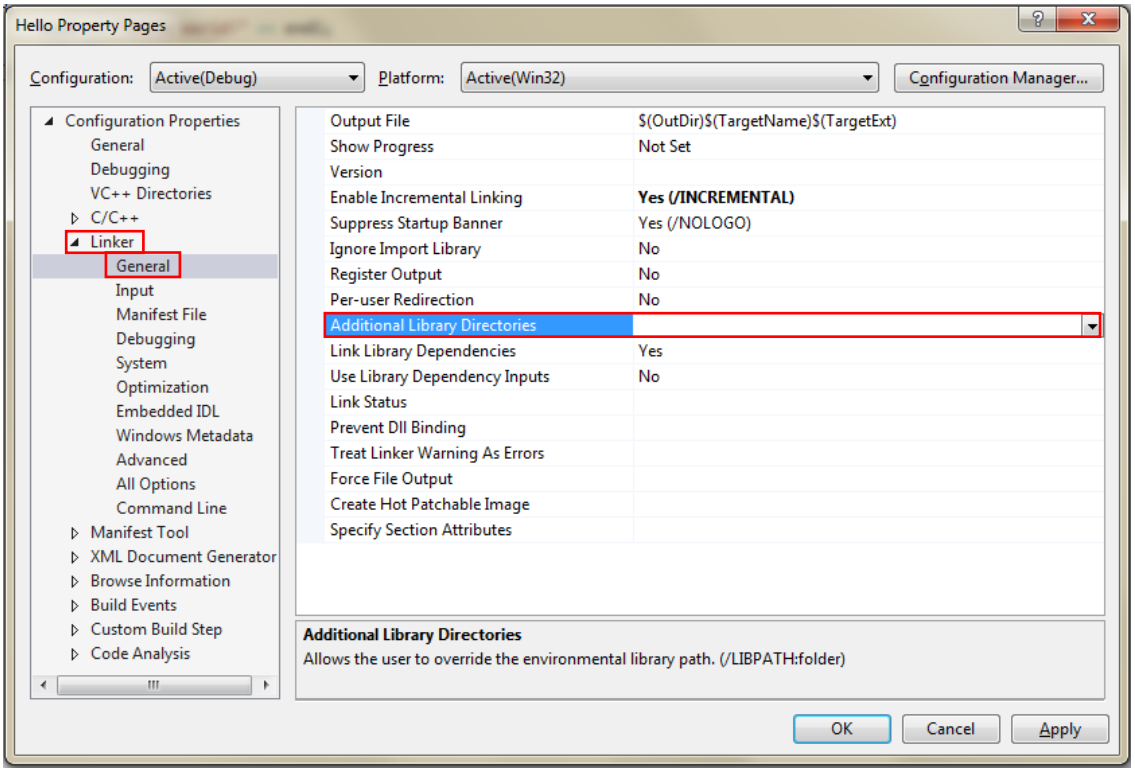
Слика 16 – Додавање путање до директоријума са *include* датотекама

- Уколико пројекат зависи од других библиотека које су потребне при увезивању, оне се могу додати у пројекат. Опција ***Aditional Dependencies*** налази се у групи ***Linker***, ***Input*** и у поље поред, могу се дописати жељене библиотеке (нпр. winmm.lib), Слика 17.



Слика 17 – Укључивање додатних библиотека за увезивање

- Додавање путање до директоријума са додатним библиотекама (енг. *libraries*), које нису у оквиру пројекта или пројектног директоријума налази се у групи ***Linker***, ***General*** као опција ***Additional Library Directories***, Слика 18.



Слика 18 – Додавање путање до директоријума са додатним библиотекама

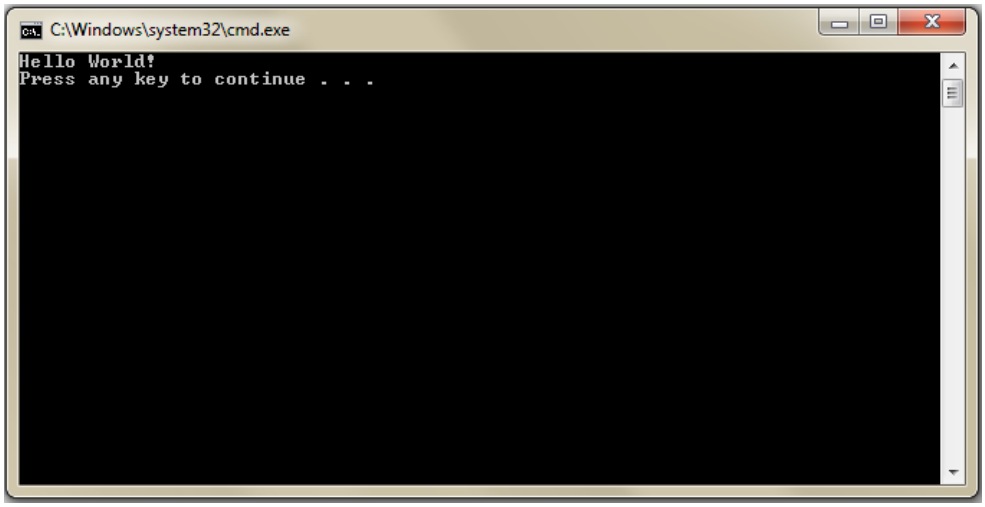
НАПОМЕНА: Сви додатни параметри пројекта доступни су у обе конфигурације – *Debug* и *Release* и морају се засебно подешавати. Тренутно активна конфигурација пројекта користи одговарајуће параметре.

Покретање програма

Након превођења и увезивања, програм се може покренути избором ставке ***Start Without Debugging*** из менија ***Debug*** или комбинацијом тастера **Ctrl+F5**.

Програм из примера ће отворити нови прозор за извршавање конзолне апликације (у складу са одабраним типом пројекта) и у њему исписати:

```
Hello World!
Press any key to continue . . .
```



Слика 19 – Прозор за извршавање конзолне апликације

Контролисано извршавање програма (debugging), *Visual Studio 2015*

Visual Studio 2015 пројекат има посебне конфигурације за коначну (енг. *Release*) верзију и верзију за проверу и отклањање грешака (енг. *Debug*) својих програма. Као што име говори, *Debug* верзија се користи за исправљање грешака, а верзија *Release* за коначно издавање и дистрибуцију.

Ако правите свој програм у *Visual Studio* алату, *Visual Studio* аутоматски прави ове конфигурације и поставља одговарајуће подразумеване опције и друге поставке. Подразумевана подешавања:

- Са *Debug* конфигурацијом ваш програм је преведен са пуним симболичким информацијама за контролисано извршавање програма, праћење променљивих са циљем откривања и отклањања грешака (*debug* информације) и без оптимизације. Оптимизација компликује отклањање грешака, јер је разлика између изворног кода и генерисаних инструкција већа.

- *Release* конфигурација програма не садржи симболичке *debug* информације и доноси потпуну оптимизацију.

- Можете се пребацивати између *Release* и *Debug* верзије, односно конфигурације решења (***Solution Configuration***) одабиром из падајућег менија у стандардној траци са алатима (*Standard toolbar*) или покретањем *Configuration Manager*-а из ***Build*** менија.

За потребе демонстрације контролисаног извршавања програма помоћу *Visual Studio 2015* алата користићемо већ направљени пројекат *Hello*, из претходног поглавља. Да бисмо прошли кроз све могућности које алат пружа, а које су од значаја за овај курс, изврони код у датотеци *Hello.cpp* ћемо допунити према следећем:

```
#include <iostream>

using namespace std;

void main(void)
{
    int test = 0;
    int i;

    cout << "Hello World!" << endl;

    test = 5;

    for (i = 0; i < 50; i++)
    {
        cout << i + 1 << ". ulazak u petlju.\r";
    }

    cout << endl;
    cin.get();

    return;
}
```

Програм из примера исписује поруку „Hello World!“ и након тога 50 пута улази у петљу у којој исписује који је по реду тренутно пролазак кроз петљу. Након тога чека се на притисак било ког тастера на тастатури.

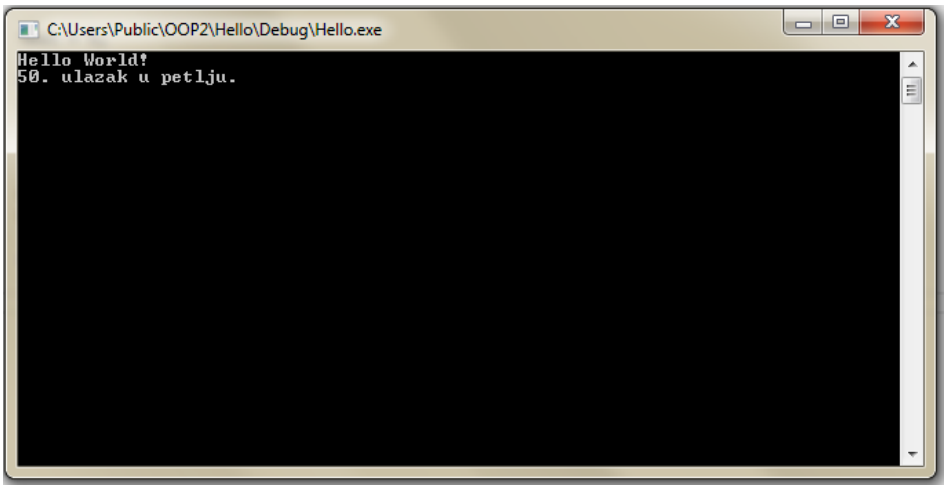
Покретање

Уколико је код измењен према датом примеру, одабрана *Debug* конфигурација и направљена извршна датотека, програм се може покренути са могућношћу контролисаног извршења и праћења промена.

Начини покретања:

- Нормално покретање програма са циљем контролисаног извршења и откривања и отклањања грешака врши се одабиром опције ***Start Debugging*** из менија ***Debug***, притиском на тастер ***F5*** или десним кликом на жељени пројекат у *Solution Explorer*-у, опција ***Start New Instance*** из подменија ***Debug***. Самим покретањем програма са опцијом контролисаног извршења, без других подешавања, програм из примера се секвенцијално извршава од

почетка до краја, као и иначе, приликом покретања изван *Debug* окружења. Програм се на функцији *cin.get()* зауставља и оставља активну конзолу чекајући на унос са тастатуре све док се не притисне Enter дугме, Слика 20. Након уноса каратера, програм се завршава и конзола се затвара.

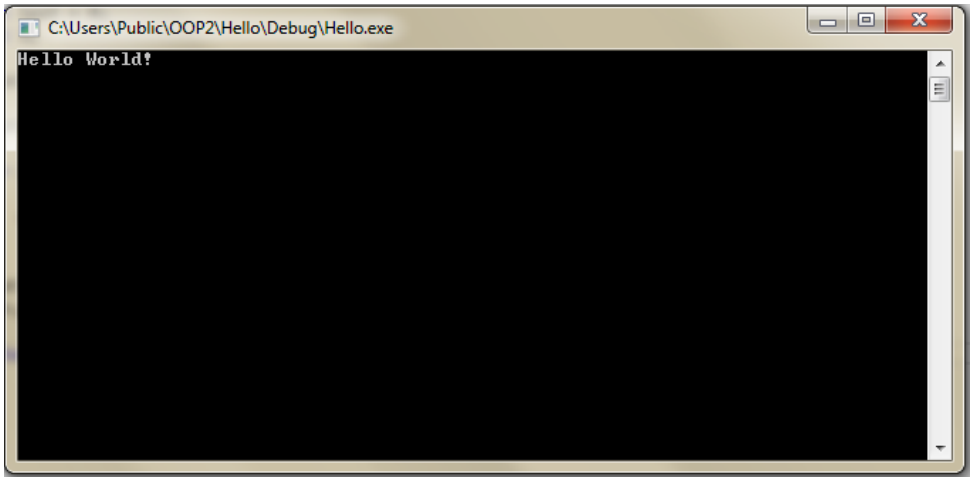


Слика 20 – Покретање контролисано извршаваног програма

- Програм је могуће покренути у моду за контолисано извршавање тако да се изврши до тренутно, курсором обележене линије и ту заустави, коришћењем команде ***Ctrl+F10***. Уколико то учините на линији 12,

```
    test = 5;
```

програм ће се зауставити на тој линији, након исписа „Hello World!“, а тачно пре додељивања вредности 5 променљивој *test*. На то указује и тренутно стање у конзоли, Слика 21.



Слика 21 – Покретање контролисано извршаваног програма до тренутно обележене линије

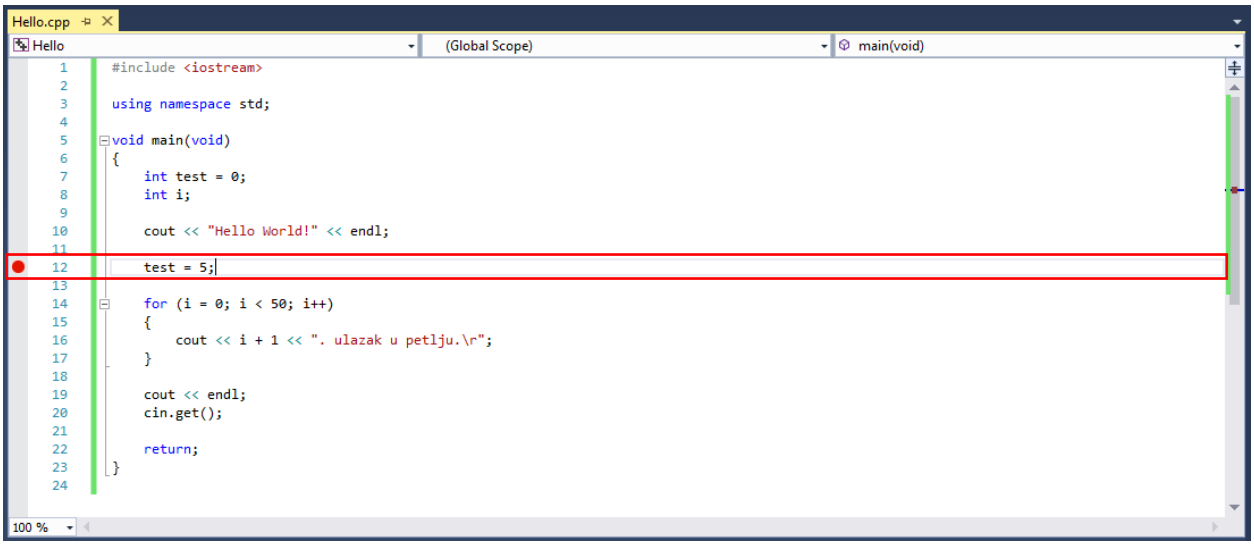
- Притиском тастера **F10**, који означава команду *Step Over* (мало касније ће њен смисао бити детаљније објашњен) програм ће се покренути, али ће зауставити контролисано извршавање на првој наредби *main* фунцкије.

- Контролисано извршење програма се може прекинути у било ком тренутку командом ***Shift+F5*** док је активан прозор *Visual Studio*. Извршавани програм ће бити угашен.

Тачке прекида (breakpoints)

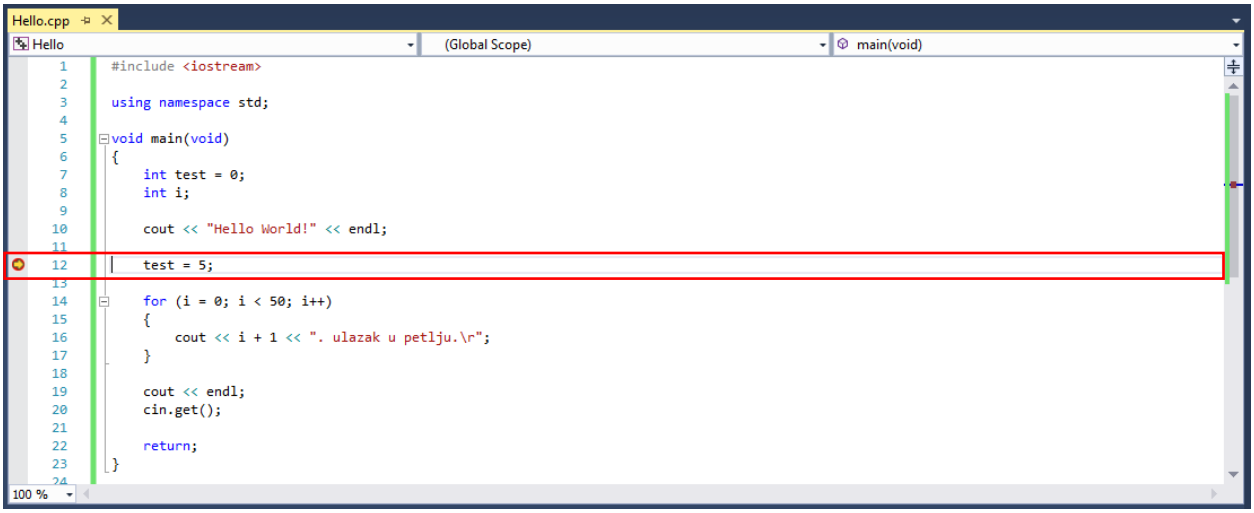
Breakpoints или тачке прекида служе за заустављање извршења програма на линији на којој су постављене, односно пре извршавања прве инструкције дефинисане том линијом. Тачке прекида се могу, како пре покретања, тако и у току извршавња програма, додавати, брисати или привремено деактивирати. Постоји и могућност постављања услова за заустављање извршења програма на линији на којој је постављена конкретна тачка прекида. Услови могу бити логички или услови везани за број заустављања на конкретној тачки.

- **Додавање нове тачке прекида** је могуће извршити на више начина: левим кликом на сиву вертикалну линију у висини линије на коју желимо да додамо тачку прекида, постављањем курсора на жељену линију и притиском на тастер ***F9*** или постављањем курсора на жељену линију и одабиром опције ***Toggle Breakpoint*** из менија ***Debug***. Додату тачку прекида потврђује црвени кружић у вертикалној сивој линији, који се налази на свакој линији за коју је постављена тачка прекида, Слика 22.



Слика 22 – Додавање нове тачке прекида

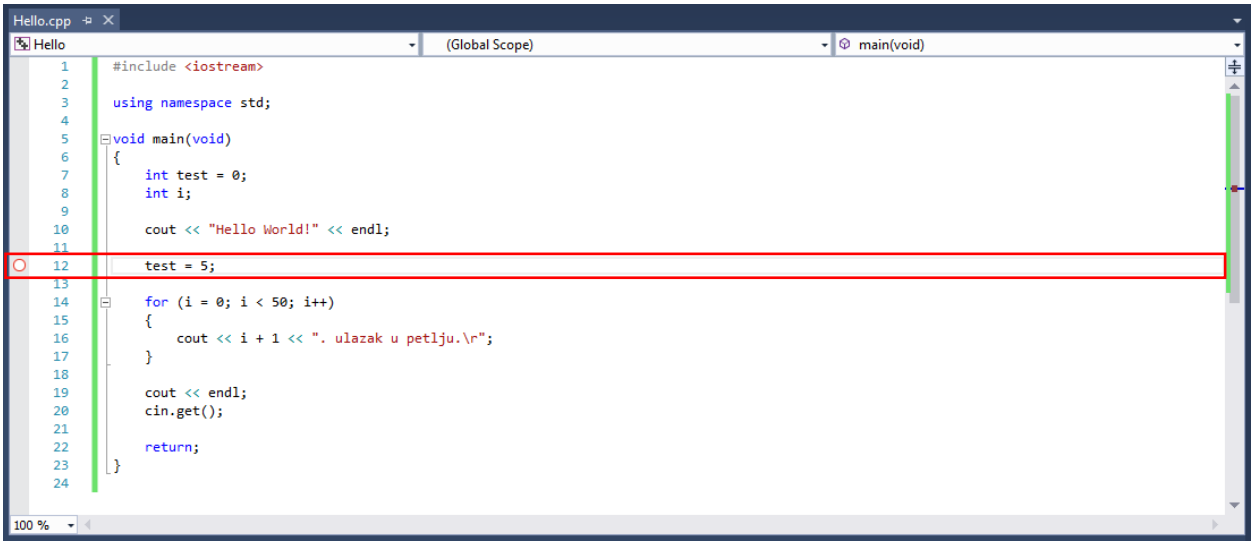
Након додавања тачке прекида, програм покренут са циљем контролисаног извршавања зауставља се на жељеној линији, односно пре извршења инструкција које она дефинише, Слика 23.



Слика 23 – Програм заустављен на првој линији са постављеном тачком прекида

- **Брисање тачке прекида** се врши на сличан начин као и додавање: левим кликом на тачку прекида, десним кликом на тачку прекида и одабиром опције ***Delete Breakpoint*** из падајућег менија, постављањем курсора на жељену линију и притиском на тастер ***F9*** или постављањем курсора на жељену линију и одабиром опције ***Toggle Breakpoint*** из менија ***Debug***. Такође, све тачке прекида из пројекта је могуће одједном обрисати одабиром опције ***Delete All Breakpoints*** из менија ***Debug*** или командом ***Ctrl+Shift+F9***.

- **Привремено стављање ван функције (деактивирање)** се врши десним кликом на тачку прекида и одабиром опције ***Disable Breakpoint***. Такође, све тачке прекида из пројекта је могуће одједном деактивирати одабиром опције ***Disable All Breakpoints*** из менија ***Debug***. Празан кружић у вертикалној сивој линији означава постојање тренутно неактивне тачке прекида, Слика 24.



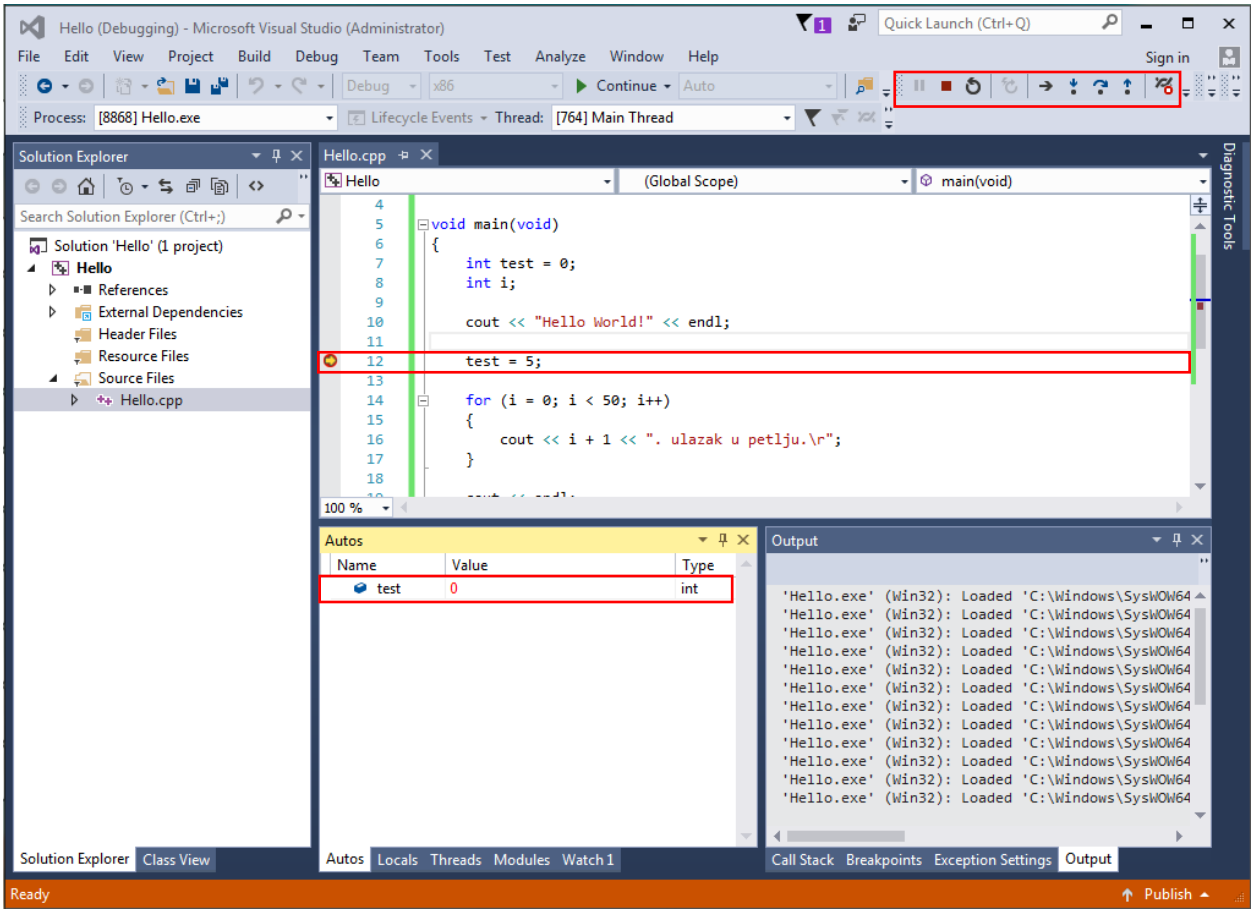
Слика 24 – Пример деактивиране тачке прекида

- Једноставан **преглед постојећих тачака прекида** нуди прозор *Breakpoints* који се отвара преко менија ***Debug***, ***Windows*** и опције ***Breakpoints*** или командом ***Alt+F9***. Овај претраживач, поред могућности прегледања и претраге свих тачака прекида у пројекту, нуди могућност лаког додавања, брисања и деактивирања тачака прекида, брисања и деактивирања свих тачака прекида и скок на линију у коду за коју је одређена тачка прекида везана, кликом на одговарајућу иконицу у горњој линији алата.

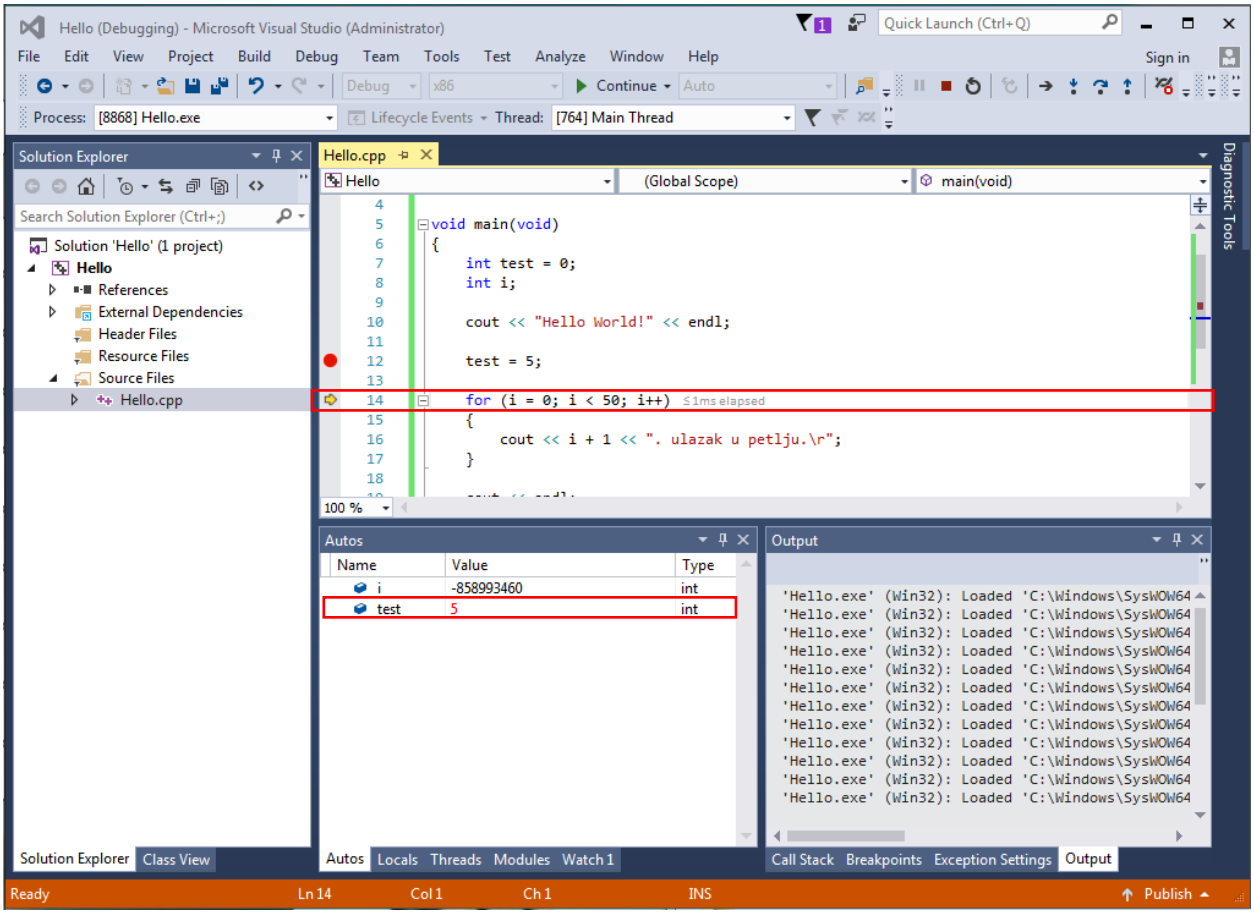
Начини упоредог кретања кроз код и извршавања програма

Након покретања програма са контролисаним извршавањем и заустављања на некој тачки прекида, могуће је на различите начине наставити извршавање програма уз праћење кретања кроз линије изворног кода и промена садржаја промењљивих, меморије и др.

- **Извршавање тренутне линије без уласка у фунцкију** (енг. *Step Over*) може се извршити одабиром опције ***Step Over*** из менија ***Debug***, коришћењем команде ***F10*** или кликом на одговарајућу иконицу у линији са алаткама, Слика 25. Када је програм заустављен на некој линији (линија 12 у примеру, Слика 25), све инструкције које та линија дефинише још увек нису извршене. У доњем делу прозора (Слика 25) обележена је линија у којој је приказан садржај промењљиве *test* који још увек има претходно постављену вредност (*test=0*). Тек након извршења тренутне линије и преласка на следећу, коришћењем фунцкије *Step Over* (Слика 26), инструкције дефинисане из линије 12 су извршене и тада промењљива *test* има нову вредност (*test=5*).



Слика 25 – Заустављање извршавања програма на линији са тачком прекида



Слика 26 – Извршење *Step Over* функције

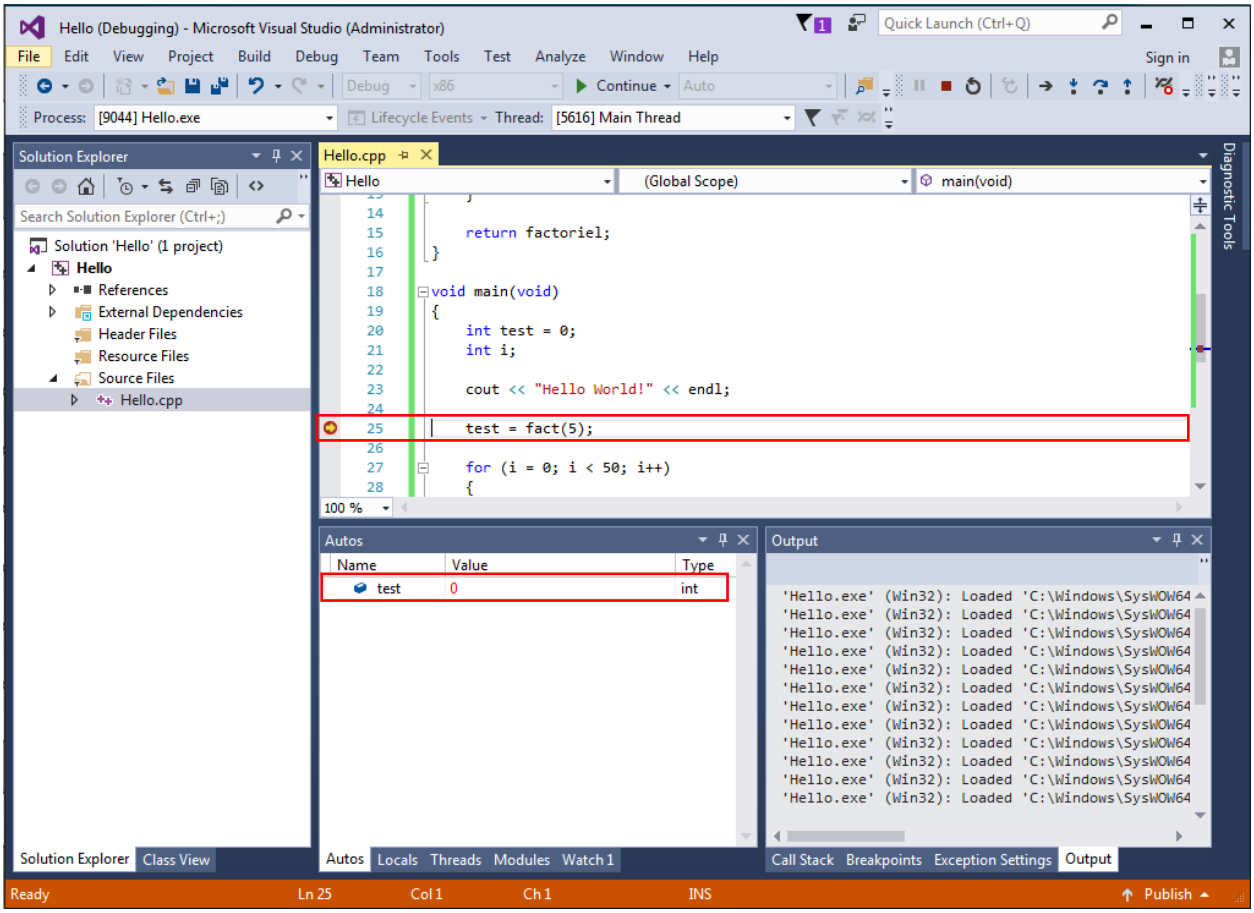
- **Улазак у функцију из тренутно извршаване линије** (енг. *Step Into*) може се извршити одабиром опције ***Step Into*** из менија ***Debug***, коришћењем команде ***F11*** или кликом на одговарајућу иконицу у линији са алаткама.

  Ради лакшег разумевања начина рада фунцкије *Step Into*, користиће се преправљен пример тако да се на линији 12, промењивој *test* додељује повратна вредност функције *int fact(int op)* за прослеђен параметар *5*. Функција *fact* рачуна факторијел броја прослеђеног као једини аргумент, а дефинисана је пре функције *main()*. Код који обухвата наведене измене дат је у продужетку.

```
#include <iostream>

using namespace std;

int fact(int op)
{
    int factoriel = 1;
    int i;
```

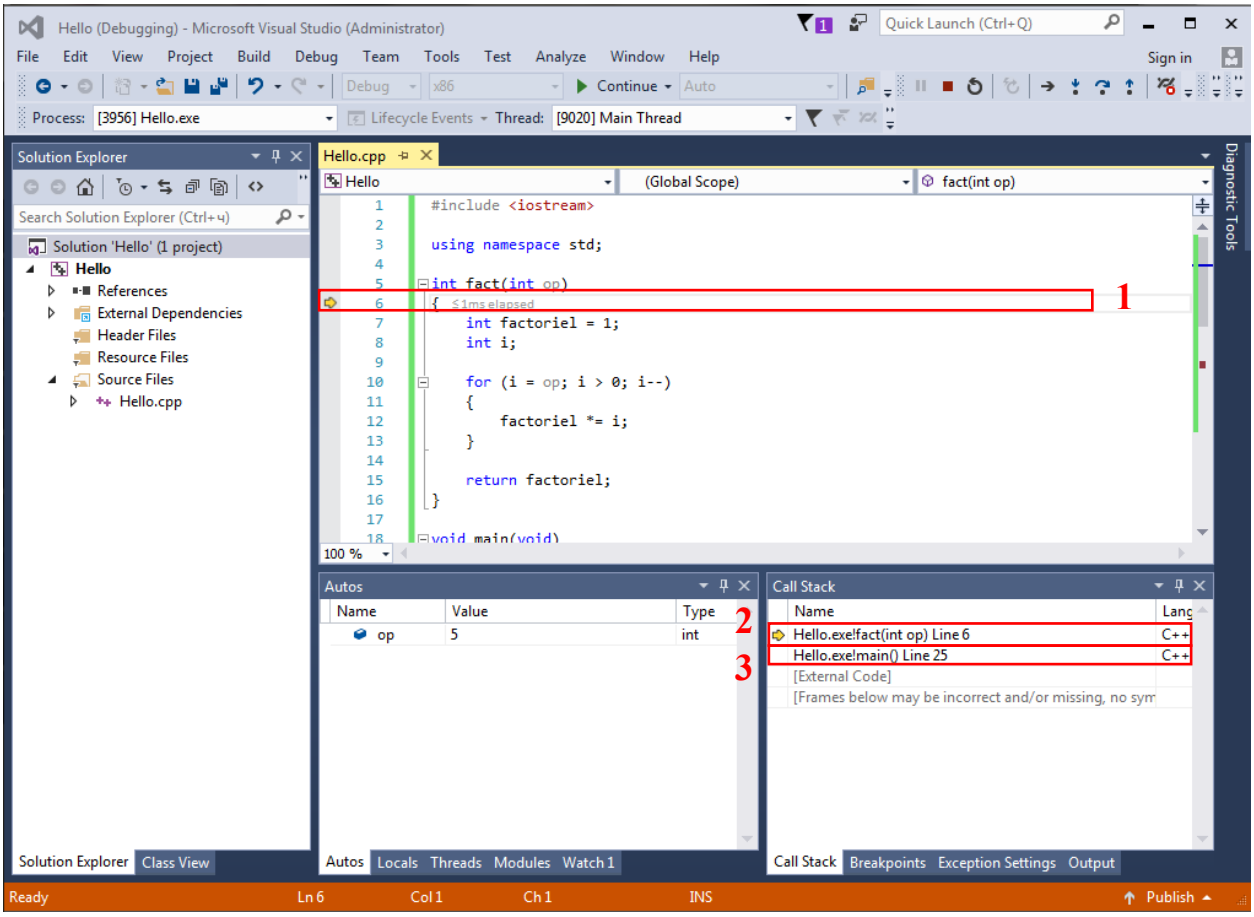
```
    for (i = op; i > 0; i--)
    {
        factoriel *= i;
    }

    return factoriel;
}

void main(void)
{
    int test = 0;
    int i;

    cout << "Hello World!" << endl;

    test = fact(5);

    for (i = 0; i < 50; i++)
    {
        cout << i + 1 << ". ulazak u petlju.\r";
    }

    cout << endl;
    cin.get();

    return;
}
```

Коришћењем тачке прекида на линији 25 и покретањем програма у режиму за контролисано извршавање, извршавање се зауставља на одговарајућој линији, пре позива функције *fact*, Слика 27. У том тренутку, вредност променљиве *test* је 0.



Слика 27 – Стање пре избора команде уласка у наредну функцију (*Step Into*)

Након извршења команде *Step Into*, извршавање програма није заустављено на следећој линији функције *main()* (линија 26), већ на првој линији функције *fact* (линија 6 изворног кода у датотеци *Hello.cpp*), Слика 28, тачка 1. У прозору *Stack Frame* у доњем десном углу, види се да је програм тренутно у фази извршења линије 6, у функцији *fact(5)*, Слика 28, тачка 2 и да је та функција позвана са линије 25, функција *main()*, Слика 28, тачка 3.



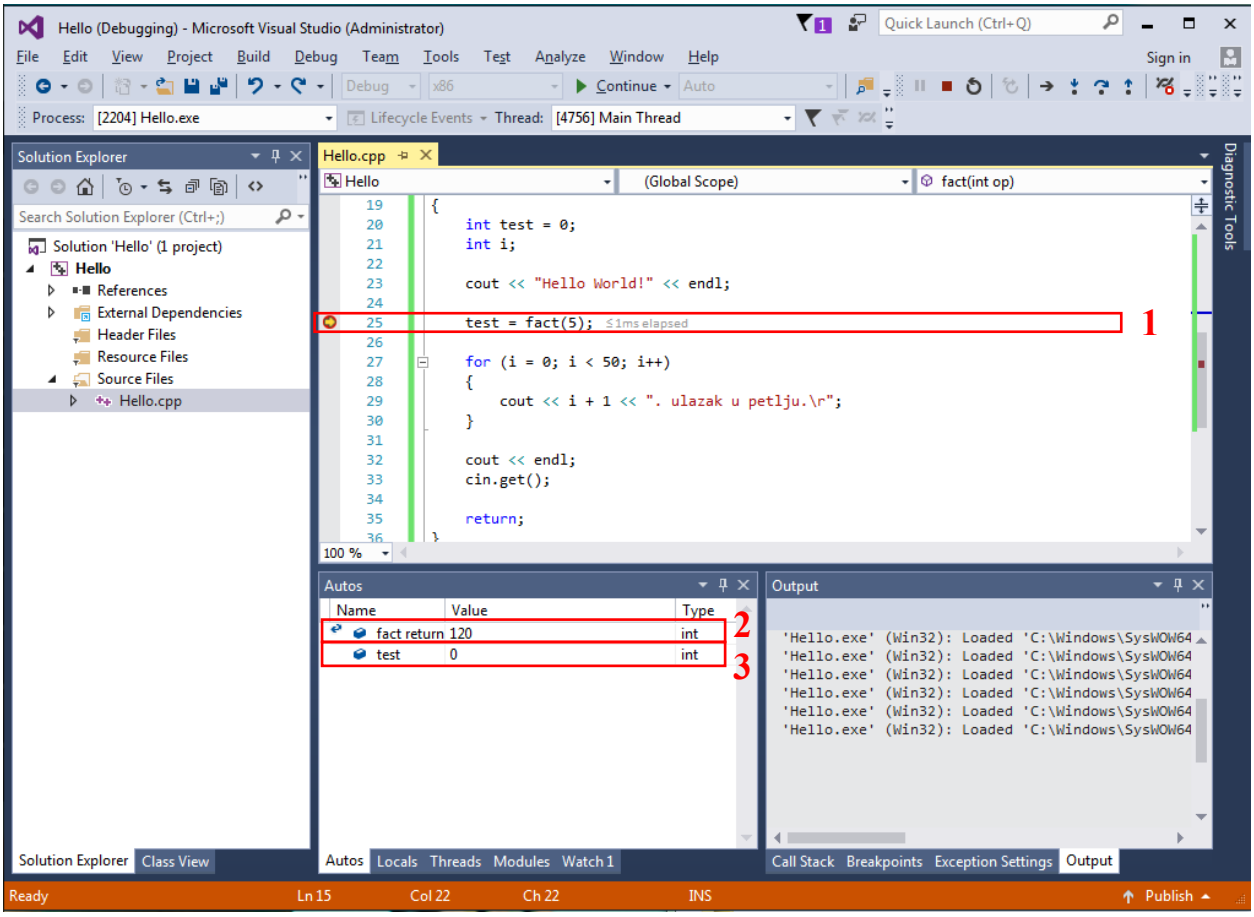
Слика 28 – Стање после избора команде уласка у наредну функцију (*Step Into*)

- **Излазак из функције и повратак на место позива функције** (енг. *Step Out*) може се извршити одабиром опције ***Step Out*** из менија ***Debug***, коришћењем команде ***Shift+F11*** или кликом на одговарајућу иконицу у линији са алаткама.

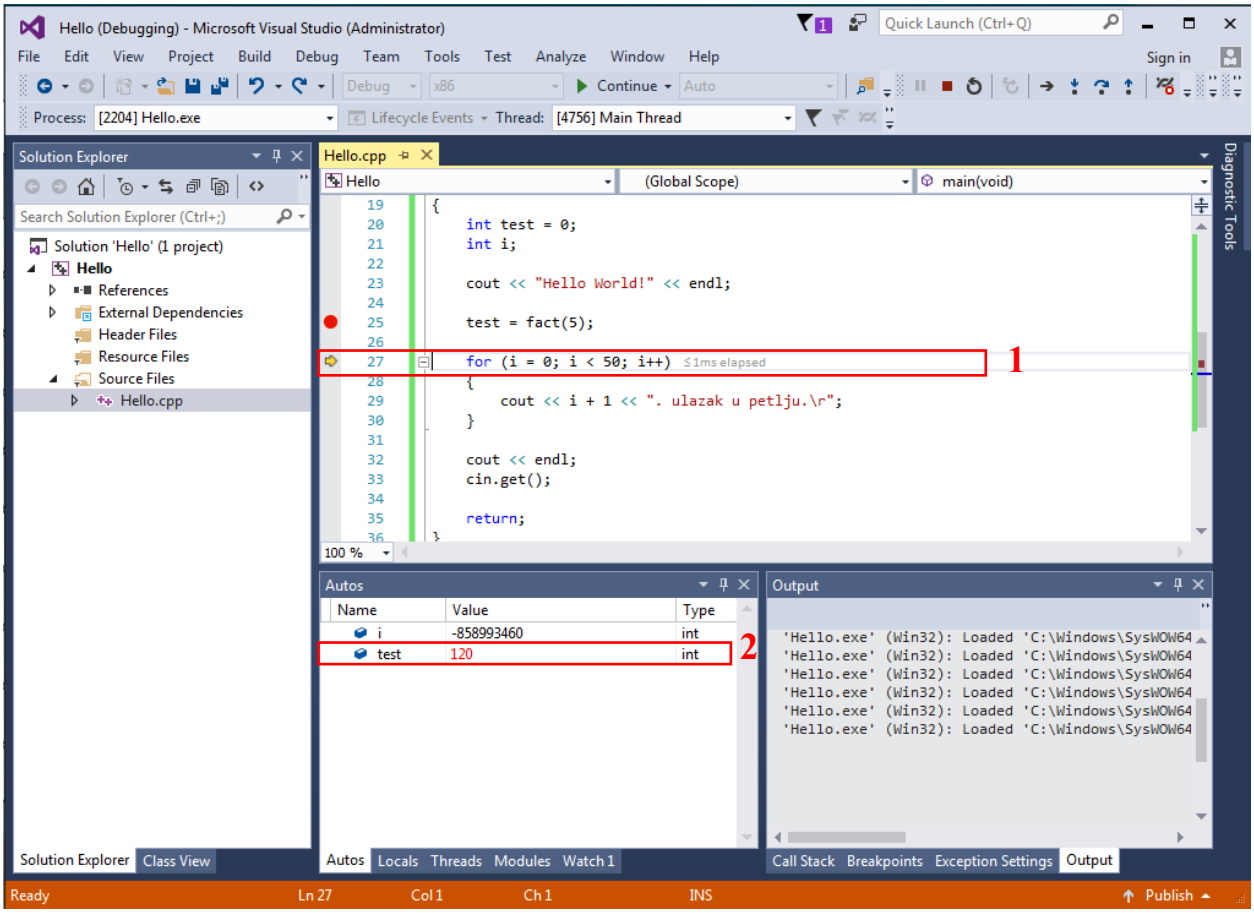
  Настављајући извршење програма из претходне ставке (*Step Into*), позивом команде ***Step Out***, програм се враћа на линију у којој је позвана тренутно извршавана функција *fact*, линија 25, Слика 29, тачка 1.

  Након повратка на линију 25, функција *fact(5)* јесте извршена на шта указује обавештење „*fact returned 120*“ у прозору Autos, Слика 29, тачка 2, али вредност 120 још увек није додељена промењливој *test*, односно операција доделе још није извршена, па је њена вредност и даље 0, Слика 29, тачка 3.

  Тек након издавања команде ***Step Over*** и извршења остатка инструкција из линије 25 и преласка на линију 27 (Слика 30, тачка 1), вредност промењиве *test* је промењена у повратну вредност функције *fact(5)*, односно 120, Слика 30, тачка 2.



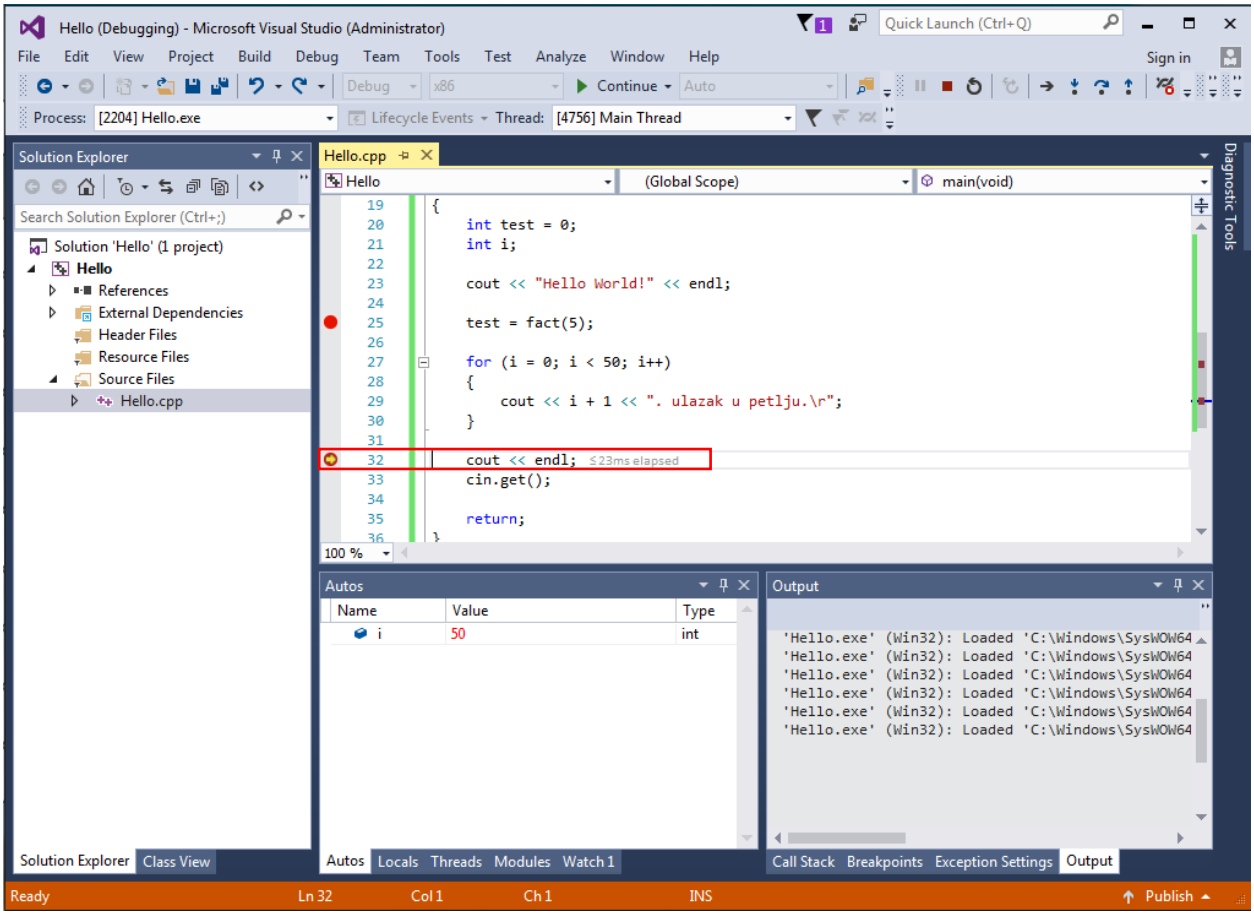
Слика 29 – Стање након избора команде изласка из функције и враћања у позивну линију (*Step Out*)



Слика 30 – Стање након извршења остатка инструкција из линије 25 (*Step Over*)

- **Наставак извршавања** до следеће тачке прекида или до терминације програма (енг. *Countinue*) може се извршити одабиром опције ***Continue*** из менија ***Debug***, коришћењем команде ***F5*** или кликом на одговарајућу иконицу у линији са алаткама.

  Настављајући извршење програма из претходне ставке (*Step Out*) са линије 25 и додавањем тачке прекида на линију 32, позивом команде ***Continue***, програм наставља са извршавањем и зауставља се на првој следећој тачки прекида, у линији 32, Слика 31. Следећим позивом команде ***Continue*** програм ће се извршити до краја јер више нема постављених тачака прекида у јединој грани којом извршавање програма може ићи до краја.



Слика 31 – Стање након извршења програма до следеће тачке прекида (*Continue*)

Праћење промењивих и меморије у току извршења програма

Од велике је користи могућност прегледања садржаја меморије, као и свих промењивих, у сваком тренутку контролисаног извршења програма. Једино ограничење је да се могу гледати вредности само промењивих видљивих из тренутно извршаване функције, а то су у уопштеном случају све глобалне промењиве и све локалне промењиве за ту функцију. Такође, могуће је на више начина приказати садржаје свих врста промењивих укључујући низове, структуре и атрибуте инстанце класе.

За потребе демонстрације приказа промењивих и меморије у току контролисаног извршавања програма, користиће се код са неколико дефинисаних промењивих различитих типова.

```
#include <iostream>

using namespace std;

typedef struct STRUCT1
{
    int atr1;
```

```
    char atr2;
} T_STRUCT1;

class class1
{
    char m_c;
    int m_i;
public:
    class1() {m_c ='0'; m_i = 0;}
    void setC(char _c) {m_c = _c;}
    void setI(int _i) {m_i = _i;}
    ~class1() {m_c = m_i = 0;}
};

void main(void)
{
    int* test;
    int iArray[5] = {1, 2, 3, 4, 5};
    T_STRUCT1 struct1;

    cout << "Hello World!" << endl;

    test = new(int);

    *test = 0x20;

    struct1.atr1 = 15;
    struct1.atr2 = 'c';

    cout << endl;

    delete(test);

    {
        class1 c1;
        c1.setC('x');
        c1.setI('x');
        c1.~class1();
    }

    cin.get();

    return;
}
```

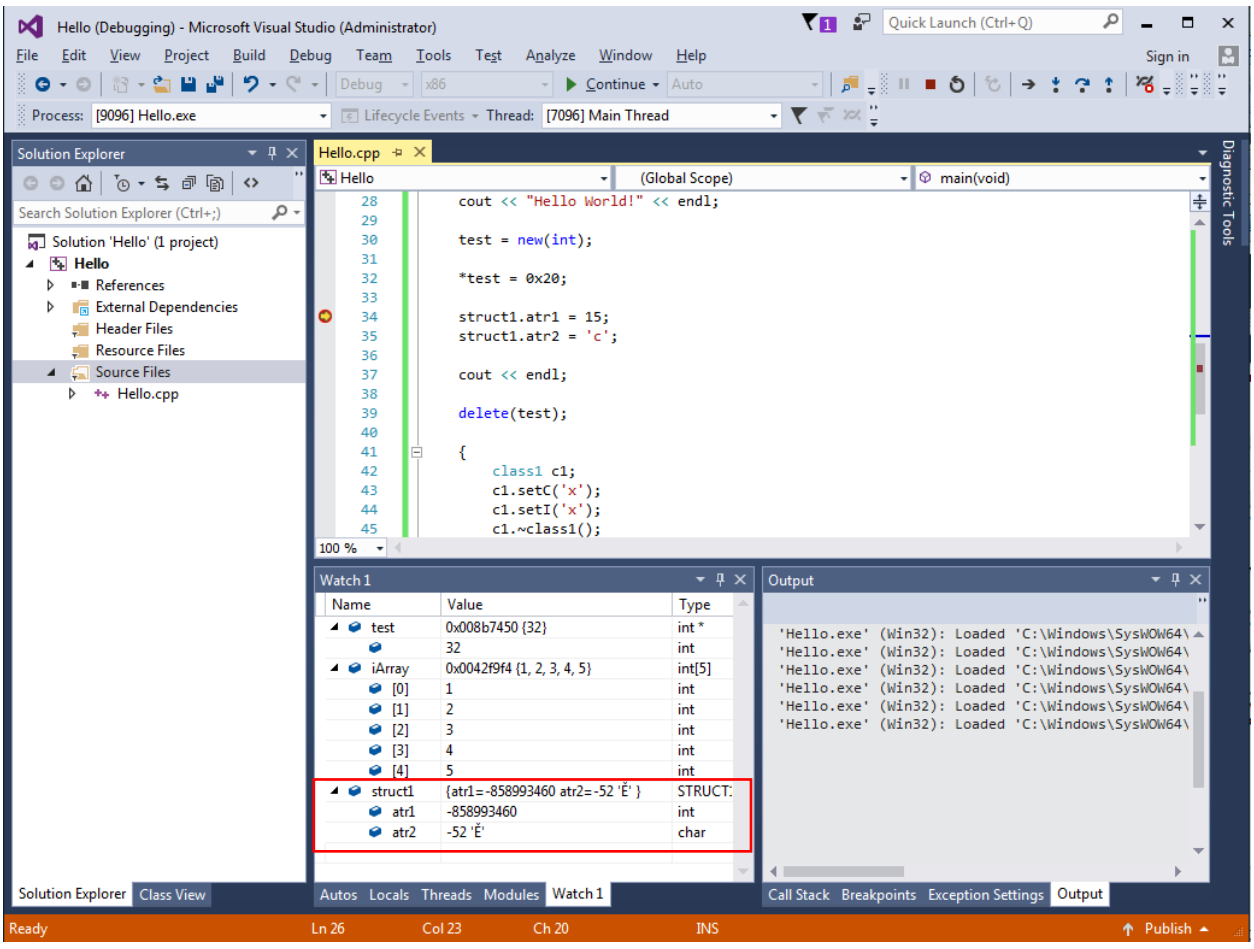
- Праћење промењљивих

  Постоје више начина за приказ садржаја промењљивих.

  Најбржи начин приказа појединачне промењљиве је задржавањем курсора изнад жељене промењљиве. На овај начин може се пратити само једна промењљива у тренутку и доста је непрегледан.

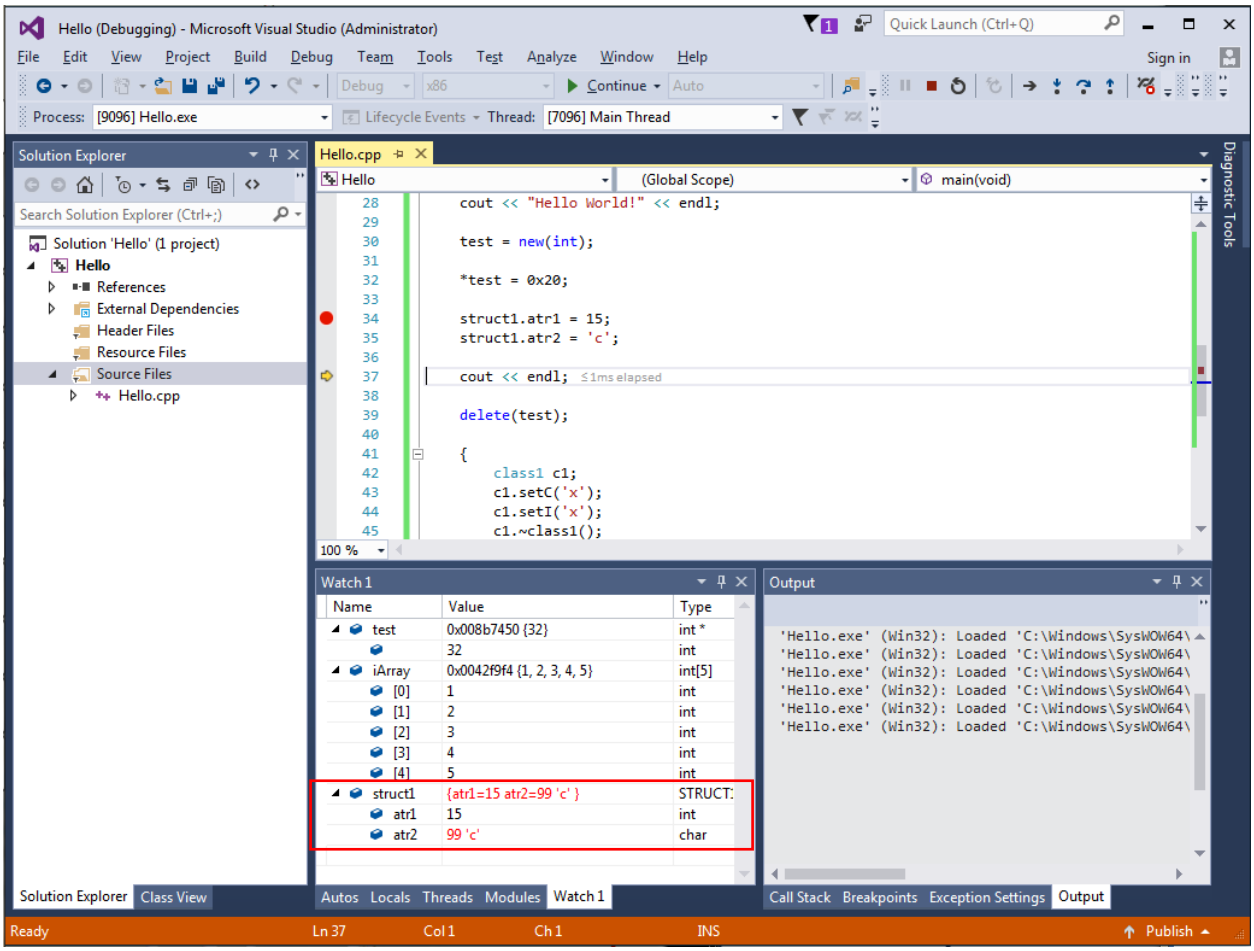
Други начин је праћење у прозору *Autos* у који *Visual Studio* по сопственом нахођењу додаје тренутно активне промењљиве.

Трећи и најчешћи метод је коришћење ***Watch*** опције и прегледање промењљивих у *Watch* прозору. Уколико није отворен, *Watch* прозор се може отворити командом ***Alt+3*** или одабиром било које ставке из менија ***Debug***, ***Windows***, па ***Watch***. Промењљиве које се посматрају се додају у листу тако што се обележе курсором, па се из менија након десног клика одабере ***Add Watch***, превлачењем промењљиве у *Watch* прозор. Заустављањем на линији 34, додавањем промењљивих *test*, *iArray* и *struct1* у *Watch* прозор и посматрањем *Watch* прозора се закључује да низ *iArray* има тачне вредности на које је иницијализован, показивач *test* има за вредност адресу локације која му је додељена алокацијом меморије, а у проширењу се види да је у локацију на коју показује смештена вредност 32 или 0x20. Промењљива односно структура *struct1* није иницијализована па има неке насумичне вредности, Слика 32. Како је приказан објект класе?



Слика 32 – Приказ промењљивих

Након извршења наредне две линије, променљива *struct1* је иницијализована и има вредности које су јој у коду и додељене. Види се да *Watch* прозор нуди и могућност приказивања вредности у виду *ASCII* карактера. Црвена боја приказане променљиве означава да је то променљива чија је вредност промењена у последњем кораку, Слика 33. Шта се дешава са атрибутима објекта класе када се позову одговрајуће методе чланице?



Слика 33 – Приказ променљивих након њихове иницијализације

ТЕХНИКЕ КОНТРОЛИСАНОГ ИЗВРШАВАЊА ПРОГРАМА СА ЦИЉЕМ ОТКЛАЊАЊА ГРЕШАКА

Откривање грешке

Најпре је потребно открити грешку, односно проверити да ли се програм (не)исправно понаша.

Једноставним покретањем и надгледањем

Уколико је једноставнији проблем у питању, исправност се може проверити једноставним покретањем за различите улазе и надгледањем резултата.

Прављење малих тврдњи – теорема (assertions)

За компликованије проблеме је пожељно писати тестове, најбоље у виду тврдњи, односно малих теорема.

У том случају, потребно је само покренути програм довољан број пута, користити притом различите улазе, а резултат тестирања ће одмах бити доступан, на основу тврдњи, у смислу да ли је нека тврдња истинита или није. Додатке у коду за потребе тестирања је најбоље на неки од могућих начина одвојити од остатка корисног кода (нпр. коришћењем претпроцесорских директива или коришћењем функција које саме по себи испуњавају овај услов), да се не би нашли и у коначној верзији програма као сувишне, а да би увек биле доступне. Оне су неопходне само у процесу развоја.

Намерно изазивање грешке

Један од могућих начина је и намерно изазивање грешке и провера да ли се програм очекивано понаша. Овај метод најчешће се користи код покривања свих могућих ситуација у интеракцији програма и корисника, где је потребно проћи кроз све могуће ситуације које је корисник у стању да изазове и проверити одговоре програма у таквим ситуацијама.

Одабир улаза

Најчешће је немогуће (предуго траје и тражи велики напор) испробати програм покретањем са целим скупом свих могућих улаза. Већ за мало сложеније проблеме, овај скуп могућих улаза (комбинација) постаје јако велик. Могуће су следеће стратегије коришћења мањег броја улаза и тестова, уз задржавање поузданости и сигурности за покривање највећег дела могућих случајева (свих репрезентативних група):

- Гранични случајеви – провера са само граничним и репрезентативним случајевима; најчешће су то крајњи елементи подскупова логички (према конкретној примени) издељеног скупа улаза,
- Очекиване вредности – провера исправности коришћењем само оних вредности које се сматрају исправним у нормалном раду програма (нпр. унос карактера на месту где се захтева унос имена),

- Неочекиване вредности – провера исправности рада програма уношењем вредности које нису очекиване да се користе у нормалном раду (нпр. унос бројева на месту где се захтева унос имена),

- Тест оптерећења – тестирање програма великим оптерећењем, задавањем великог броја произвољних улаза или улаза таквих да изазивају велико оптерећење и провера коректности извршења у тим случајевима,

- Насумично – задавање улаза одабраних псеудо-случајним путем.

Већ када је програм мало компликованији, најбоља је комбинација свих ових стратегија.

Одређивање места (узрока) грешке

Након откривања постојања грешке, долази се до најчешће много тежег задатка – проналажење грешке односно узрока грешке.

Секванцијално извршавање

Најједноставнији метод, али веома лош већ и за мало компликованије ситуације. Основни проблем је што овакво проналажење грешке може јако дуго да траје, тражи велику концентрацију, надгледање много различитих променљивих. Посебан проблем је што подразумева извршавање програм секвенцијално, од почетка, а грешка може бити нпр. на самом крају. Тада је узалудно утрошено време за проналажење грешке јако велико.

Логичко дељење кода на целине и одређивање критичних тачака

Бољи метод подразумева дељење кода на неке логичке целине за које се сматра да је вероватноћа постојања конкретне грешке већа. Најчешће је то један вид рекурзивног процеса где се:

1. одређују сви могући узроци,
2. елиминишу могући узроци редом, док се не дође до правог,
3. одређују се места која могу да доведу до неочекиваног понашања и изазивања касније грешке или се региони деле на мање регионе,
4. испитује се постојање грешака у тим регионима и/или се проналазе нови узроци.

Проналажење грешке у фиксираном региону

У одређеном региону, грешке је могуће пронаћи или додавањем нових тврдњи у том делу, додавањем кода или додавањем нових условних тачака прекида. У ређим

случајевима могуће је и само једноставним контролисаним секвенцијалним извршавањем тог дела кода.

Додавање кода и коришћење тачака прекида

Ако су могућности за изазивање грешке многобројне или компликоване, могуће је упростити проблем дељењем, односно додавањем дела кода и изоловањем мање групе или само једног потенцијалног узрока.

Најчешће се додају додатни услови, тврдње и/или исписи.

Пожељно је што мање мењати код пре утврђења тачног места грешке. Најбоље је не мењати га уопште, уколико је могуће.

Ако је неопходно додавање дела кода за откривање узрока тачно утврђене грешке, пожељно је „обухватање“ овако додатог кода претпроцесорским директивама за изузимање тог дела кода од превођења у току прављења коначне верзије програма. Овај део кода је вишак, служи само за откривање грешака и није неопходан за нормалан рад програма, па ни не треба тада да буде присутан, у циљу уштеде ресурса.

Прављење малих тврдњи – теорема (assertions)

Конкретну сумњу могуће је отклонити додавањем нове тврдње на већ описан начин. И у овом случају код не сме да се мења пре утврђивања тачног извора грешке.

Отклањање грешке

Отклањању сваке појединачне грешке се сме приступити тек када су тачно место и конкретан узрок грешке одређени. Пре тога, никако се не сме приступити покушају „отклањања грешке“ случајним погађањем или неком сличном методом. Таквим поступцима ретко се отклони грешка, а врло често се уведе још једна или више нових грешака, или се чак привидно отклони грешка, а заправо грешка само остаје маскирана, најчешће неком новом грешком.

Отклањање грешке је најчешће једноставан процес али је током њега неопходно задржати концентрацију. Потребно је размислити где се све иста или слична грешка можда провукла, али се никако не сме прећи на мењање кода и на тим местима, већ само треба бити свестан о тим потенцијалним грешкама, можда их записати на папир или у текстуалну датотеку (у виду неке TODO листе) и касније их проверити у току тестирања. Такође, потребно је размислити и о међузависности, да ли сада та измена тражи још неку нову измену. Уколико постоји документ о праћењу промена у коду (дневник), потребно је додати информацију о начињеној измени.