

Setting development tools - for Programming 1

Grzegorz Ostrek, Rafał Józwiak

October 6, 2017

Abstract

Presenting introduction to development tools settings for Programming 1 students is the aim of this document. We briefly describe chosen compilers and Integrated Development Environments. Essential steps for writing simple programs are shown. Document is not intended to contain comprehensive documentation which is available in respective software sources. The document is work in progress, please report mistakes or suggested improvements g.ostrek@mini.pw.edu.pl.

Contents

1	Introduction	2
2	Microsoft Visual Studio	2
2.1	Solution	3
2.2	Command Line Tools	10
2.3	Passing parameters to program	11
2.4	Debugging	12
3	*nix systems	13
4	Other IDEs	18
5	Documentation	19

1 Introduction

To write command line programs we need a text processor and compiler. These two programs are often bundled, as the edit-compile-run-debug cycle in production is rather tedious when using it separated. We can find many Integrated Development Environments (IDEs for short) that make the cycle easier, but may demand additional steps at the beginning of development process. Advantages of the IDEs are syntax highlighting, keyword completion, revision control, and many more although some of these features can be found in advanced text processors like *Notepad++*, *Komodo*, *Emacs* or *Vi* which have ability to run compilation tools chain. Fast solution is to try online compilers like wandbox.org.

We are assuming a simple *Hello World* example for first program.

```
#include <stdio.h>

int main(){
    printf("Hello _world\n");
    return 0;
}
```

Remember about at least one `\n` in last `printf` instruction to force output on console and new line (enter or return key) as last character in file.

2 Microsoft Visual Studio

Microsoft Visual Studio (VS) can be acquired through MSDN AA University access¹ or from Microsoft product's site². Presented examples are created with a Visual Studio 2010 C++ Express and generally applies to other versions and editions of VS.

In the Programming 1 Laboratory there are VS 2008, 2010, 2012, **2013 and 2015 and 2017** installed.

On 12 November 2014, Microsoft announced Visual Studio Community version. It have similar functionalities to Visual Studio Professional. Unlike, the previous free VS version namely Express edition, Visual Studio

¹<http://e5.onthefhub.com/WebStore/Welcome.aspx?ws=87581f2b-b28b-e011-969d-0030487d8897>

²<http://www.visualstudio.com/downloads/download-visual-studio-vs>

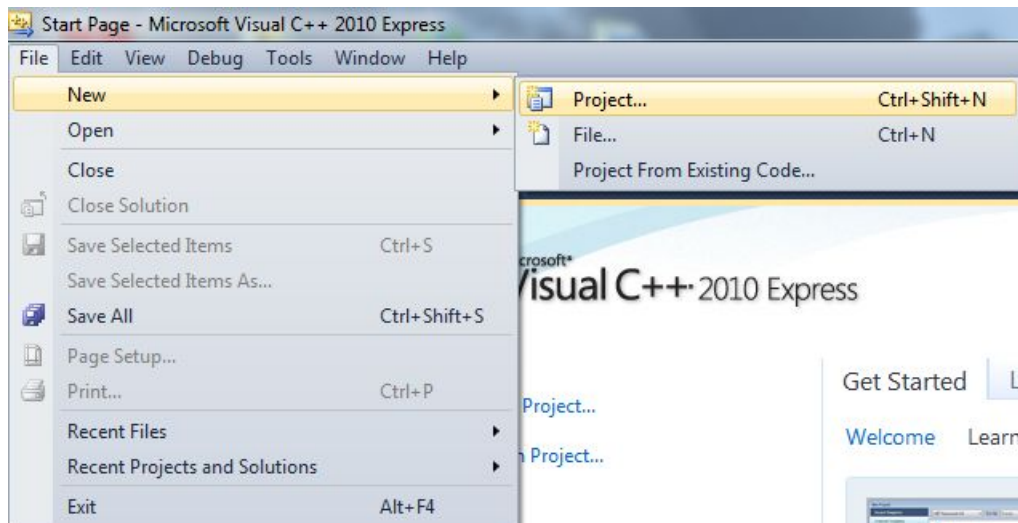


Figure 1: Starting project wizard.

Community supports multiple languages, and provides support for extensions. Visual Studio Community is oriented towards individual developers and small teams³. **What is important Community edition is free for educational purposes .**

To install VS2013 to 2017 community editions please go to <https://www.visualstudio.com/downloads/download-visual-studio-vs> and click proper version under **Visual Studio downloads** (on the left side in the middle of the page) and choose format: web installer or ISO download. This is registerware software so you will have to create Microsoft account (e.g. hotmail email account is sufficient or sign up <http://live.com>). After installation please check and install recent updates (i.e. by Windows update or by <https://www.visualstudio.com/downloads/download-visual-studio-vs>) for VS. ISO Requires burning DVD or an optical device emulator like <http://wincdemu.sysprogs.org>.

2.1 Solution

In VS, project is described as solution, one solution may link to the other projects and workspaces.

³https://en.wikipedia.org/wiki/Microsoft_Visual_Studio

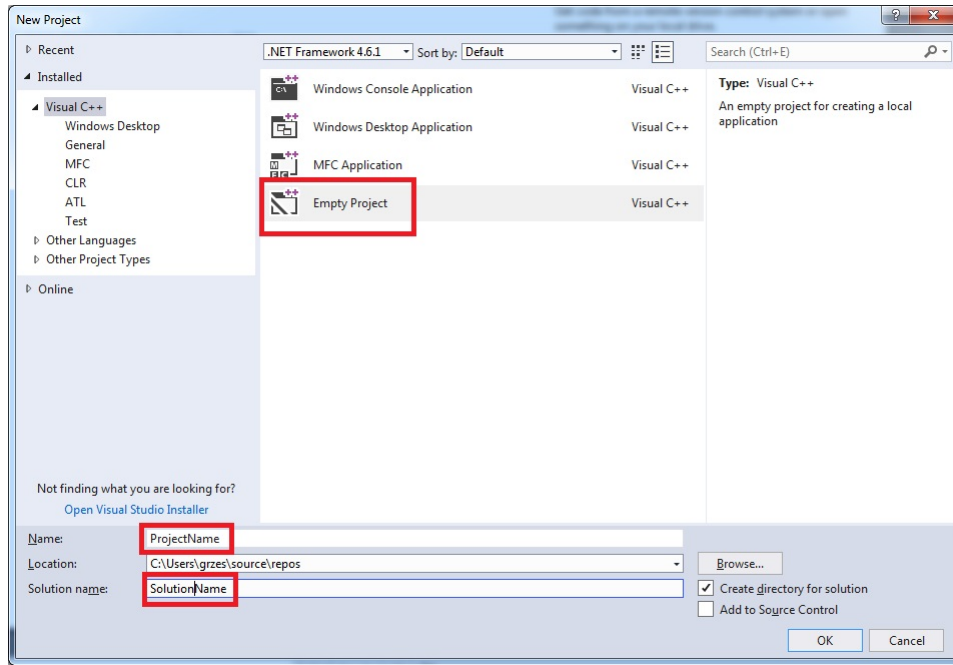


Figure 2: Selecting project type in VS2017.

After installing VS, we can create first project by following steps. First start project wizard with *File* → *New Project* Fig. 1.

Starting from VS2017 there is no Win32 application wizard. Please use Empty Project 2. Here steps suggested steps for VS2017 are described followed by procedure common for previous versions. Add new file to solution by right click Source filter in Solution Explorer (left side of VS window) and add new file with extension `.c` 2. Right click on `ProjectName` in the Solution Explorer window select Properties and change settings as described in Figure 4. SDL option switch security setting for unsafe functions like `scanf` and force using `scanf_s`. Recommended is no. You can add `#define _CRT_SECURE_NO_WARNINGS` before any `#include` in each file. Setting console typ should avoid command line window disappearing. Other method is to use `system("pause")` or `getchar()` as last instruction before return. Then select from installed Visual C++ templates Win32 Console Application Fig. 5, the name of project must be specified and will be used to create subfolder (if checkbox is selected) in Location directory. Solution name is the project name by default and can by changed. In next two windows select

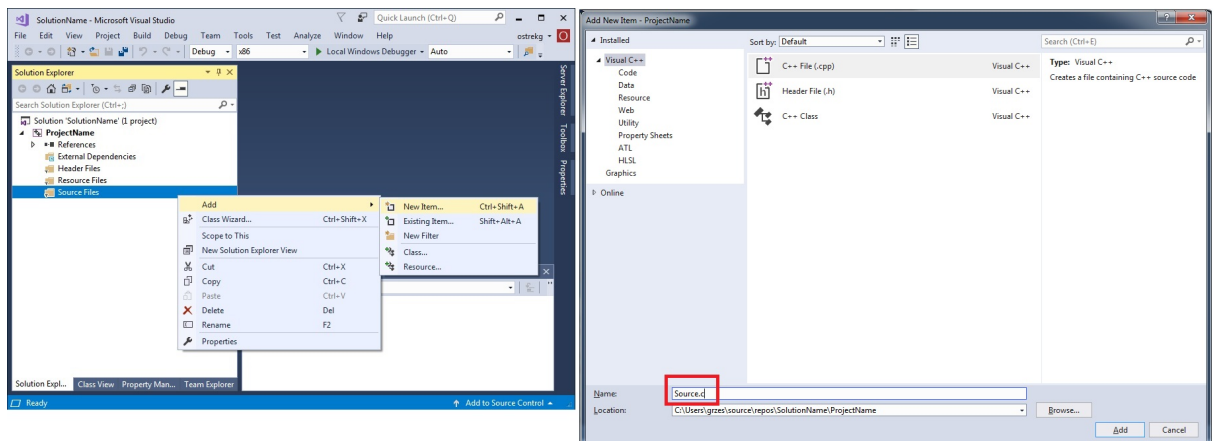


Figure 3: Adding new file Source.c in VS2017.

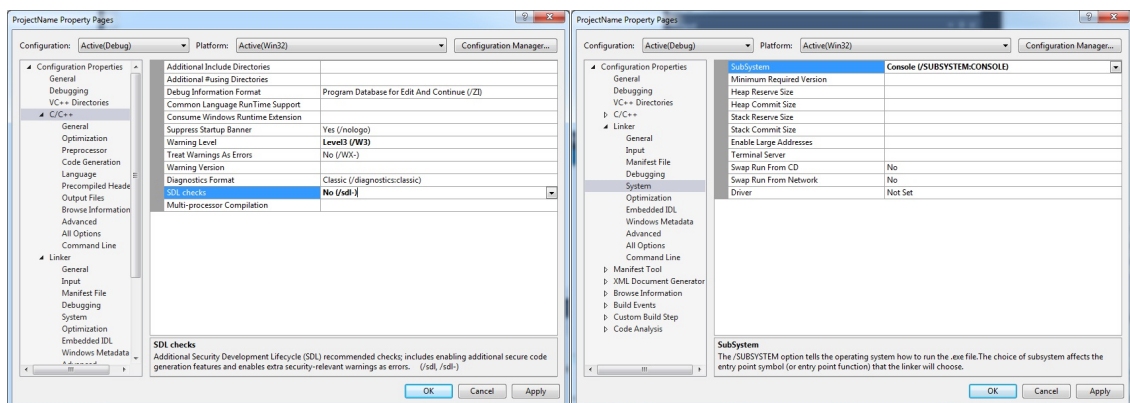


Figure 4: Settings in VS2017.

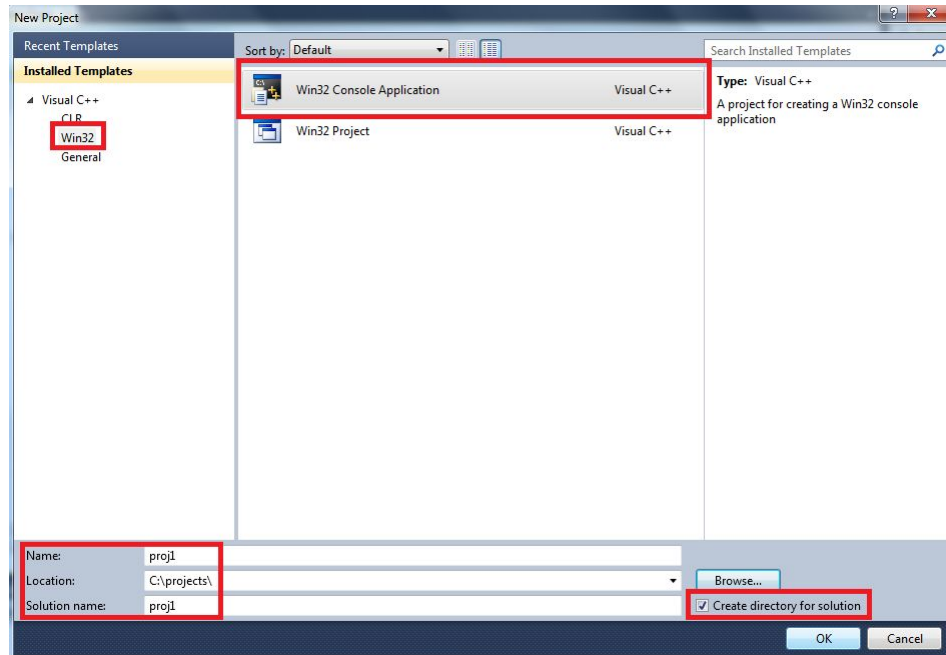


Figure 5: Selecting project type.

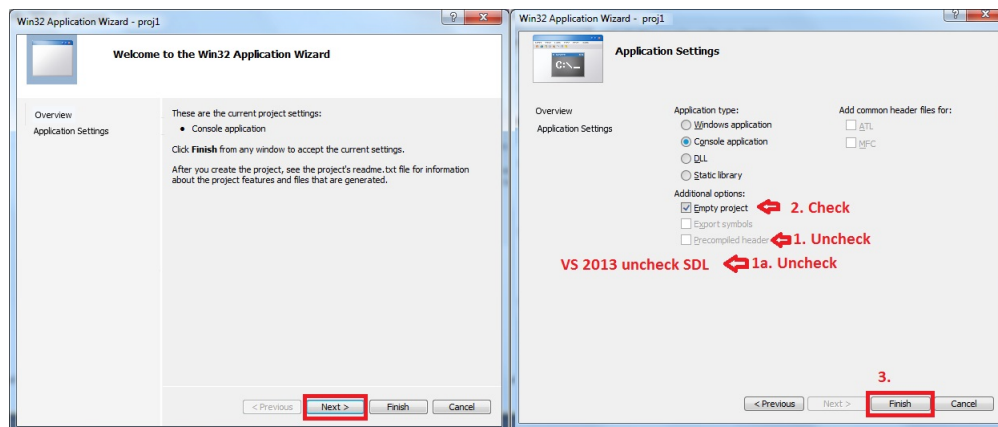


Figure 6: Selecting application settings.

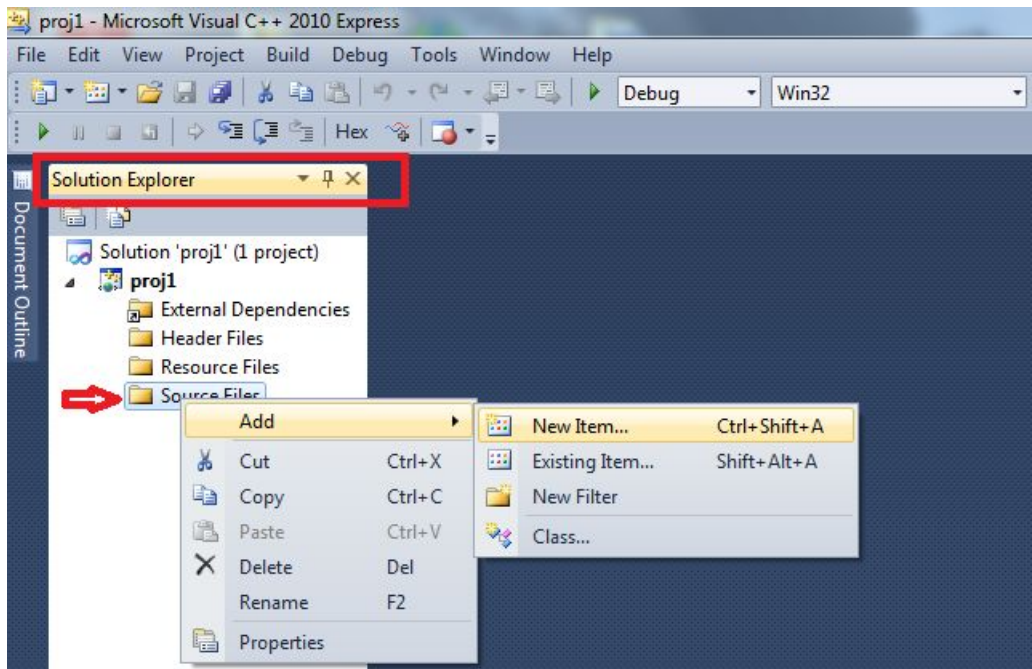


Figure 7: Adding files to project.

Console Application and Empty project Fig. 6. Then add new file by right clicking on Source in Solution Explorer view and Adding New Item shown of Fig. 7. When specifying name for new item ensure that extension is `.c` that sets compilation for C code – in case of `.cpp` files C++ compilation is done. There is no option to select `.c` extension but manual input works, see Fig. 8. Note that new item is saved in subdirectory of project location. Adding more files and headers can be made similarly - all files found in Solution Explorer's Source Files are compiled during solution building. Existing source files can be added as well. Then code can be edited and saved to the created file. Last two steps are building and executing code. Build can be done by selecting *Build* → *Build solution* from menu: 1 in Fig. 10. After successful code generation information in output window appear and running executable by *Debug* → *Start without debugging* show new console window with program results - 2 and 3 in Fig. 10. Building can create output files with debugging symbols or optimized for production purposes - selection is made by setting Debug/Release build mode in toolbar dropdown. Executable files are created in *Debug* or *Release* subdirectories in project location - information about

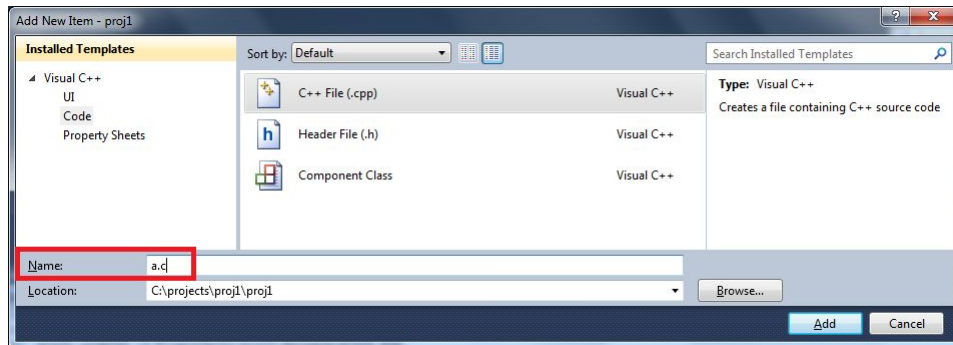


Figure 8: Specifying name.

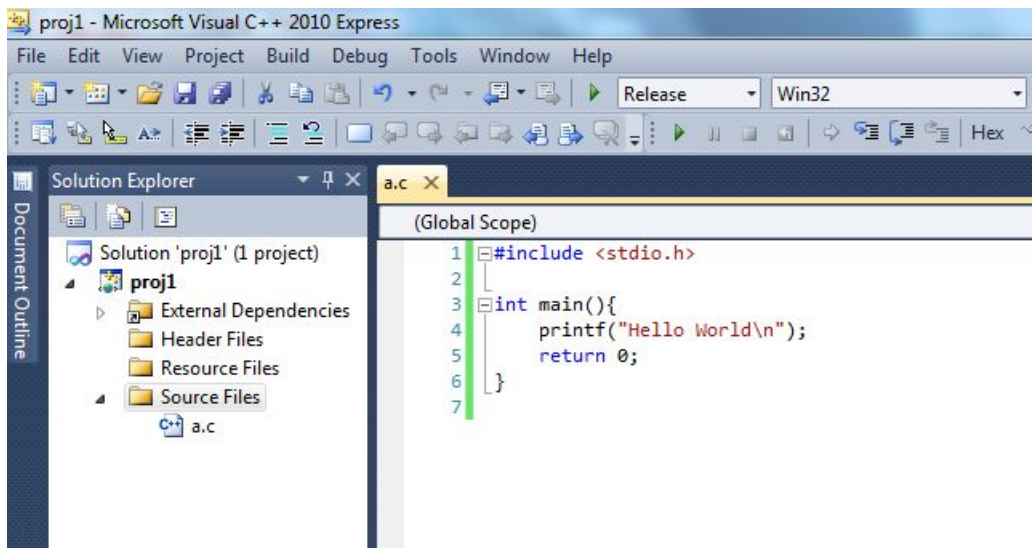


Figure 9: Code input.

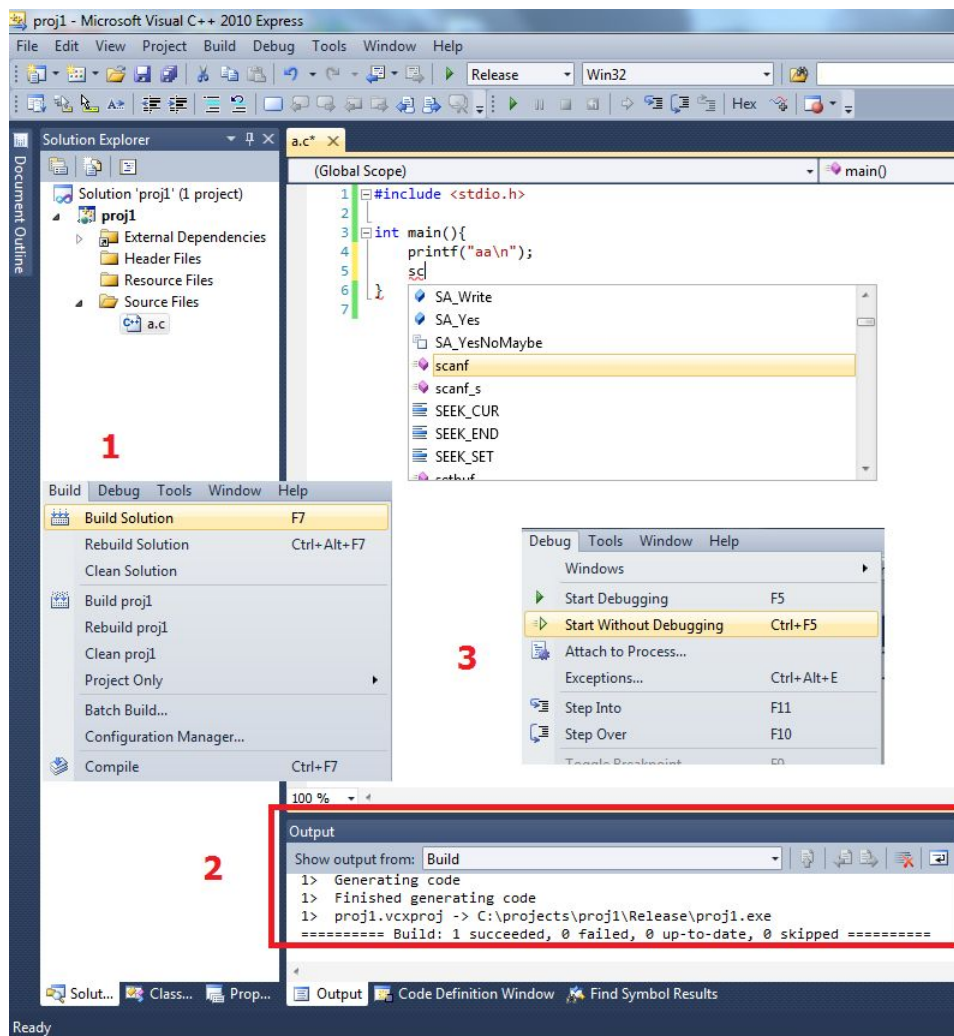


Figure 10: Building file.

executable location is included in output window. Careful reading the output gives information about possible mistakes in code - typo, matching braces, not found identifiers (function or variable name). A code auto completion is triggered with starting letters of keyword : function name, variable name (and Ctrl-Space pressing in VS2008), see Fig. 10. Proper header file must be included to invoke searching matching name - in case of `scanf()` it is declared in `stdio.h` header file. Compiler option can be set by right-clicking on project name under solution in Solution Explorer view and choosing Properties (last option).

Any customization can be made with *Menu → Tools → Options...* window. Switching between programming languages settings (e.g. C++ and C#) need its importing with *Menu → Tools → Import and Export Settings....* To turn on line numbers *Menu → Tools → Options...* from left hand tree *Text Editor → C/C++ → General* and under Display section check Line numbers.

2.2 Command Line Tools

Visual Studio comes with **Visual Studio Command Prompt** which can be found under *Start → Visual Studio XXXX → Visual Studio Tools*. Main task of the **Visual Studio Command Prompt** is setting environmental variables in console which are used by VS compiler named `cl`. Running simple `cmd` doesn't comes with compiler name known, only `%PATH%` value is set in this case. To check if compiler works type following command in the console.

```
cl /?
```

It allow to compile files without need to create VS solution. Using it is similar to *nix with respect to compiler name - please use adjusted instructions from section 3; creating directory command is `md`, printing working directory is not necessary as it is already in prompt line. Output executable file name is same as compiled `.c` file name, so running is just typing the name with `.exe` extension. Example of running HelloWorld.exe example is presented in Fig. 11.

Remember that paths containing a space character should be inside a parentheses "c:\path to source\here". Changing current drive letter is made by calling eg. `d:` in console (for moving to drive D).

```

Visual Studio Command Prompt (2010)
C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC>k:
K:\>cd K:\mini-pr1\dev-intro
K:\mini-pr1\dev-intro>cl helloworld.c
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 16.00.40219.01 for x86
Copyright (C) Microsoft Corporation. All rights reserved.

helloworld.c
Microsoft (R) Incremental Linker Version 10.00.40219.01
Copyright (C) Microsoft Corporation. All rights reserved.

/out:helloworld.exe
helloworld.obj
K:\mini-pr1\dev-intro>helloworld.exe
Hello world
K:\mini-pr1\dev-intro>helloworld
Hello world
K:\mini-pr1\dev-intro>.\helloworld
Hello world
K:\mini-pr1\dev-intro>.\helloworld
'._' is not recognized as an internal or external command,
operable program or batch file.
K:\mini-pr1\dev-intro>

```

Figure 11: Compiling and running an example helloworld.c program.

2.3 Passing parameters to program

Example of utilizing command line arguments is presented in the following listing.

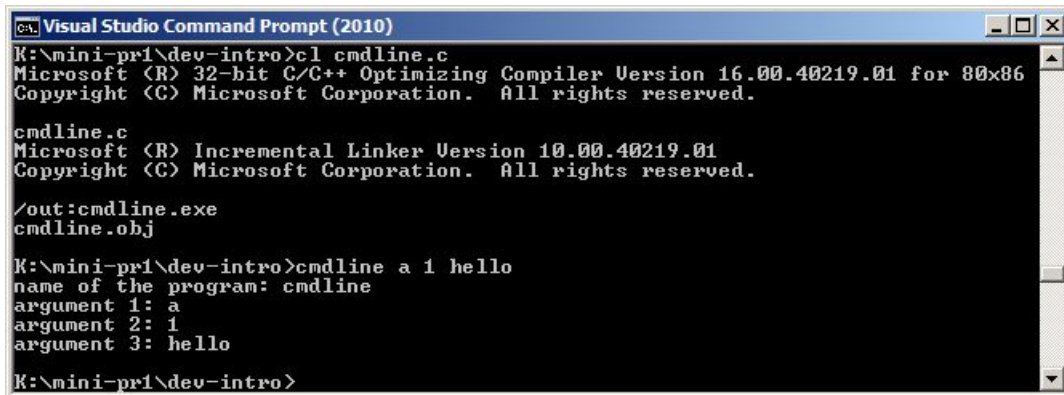
```

#include <stdio.h>

int main(int argc, char *argv[]) {
    int i;
    printf("name_of_the_program: %s\n", argv[0]);
    for(i = 1; i < argc; i++) {
        printf("argument %d: %s\n", i, argv[i]);
    }
    return 0;
}

```

Main function takes two parameters which actual values are passed when program is started. Example of running cmdline program with parameters: a, 1 and 'hello' is presented in Fig. 12. Each parameter is stored in array terminated with a '\0', and argv[] argument stores pointer to such arrays



```
Visual Studio Command Prompt (2010)
K:\mini-pr1\dev-intro>cl cmdline.c
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 16.00.40219.01 for 80x86
Copyright (C) Microsoft Corporation. All rights reserved.

cmdline.c
Microsoft (R) Incremental Linker Version 10.00.40219.01
Copyright (C) Microsoft Corporation. All rights reserved.

/out:cmdline.exe
cmdline.obj

K:\mini-pr1\dev-intro>cmdline a 1 hello
name of the program: cmdline
argument 1: a
argument 2: 1
argument 3: hello
K:\mini-pr1\dev-intro>
```

Figure 12: Compiling and running an example `cmdline.c` program.

on indexes 1 to `argc-1`. First pointer stores name of the program (name of executable file).

Under Visual Studio IDE command line arguments can be passed by setting workspace properties. Chose `Alt-F8` or `Menu → Project → <project name> Properties` (last item), or mouse right click on workspace (under solution) and chose `<project name> Properties` (last item) to open the project property pages. Then select `Debugging` in the tree on the left side and place arguments in *Command arguments* edit field (cell next to blue highlighted one) shown in Fig.13.

2.4 Debugging

Selecting `Debug` build mode allow to track program's execution step-by-step. Program can be stopped on selected instruction or under given condition. Selecting code instructions where program flow should stop is done by clicking on the gray bar left from a source code editor what cause a red dot to appear - its mean that a breakpoint is set in the program. When breakpoint is set then calling menu `Debug → Start Debugging` (or `F5`) result in execution interrupt before instruction selected by breakpoint. VS view after debugging is started is presented on Fig. 14. Current program position is marked by a yellow arrow placed on the red dot in Fig. 14. At the bottom are placed `Local` and `Stack` views. `Local` contain variables present in current scope (block), `Stack` gives information about functions calls. Additional names also can be tracked in `Watch` window, also array view is possible as presented in Fig. 15

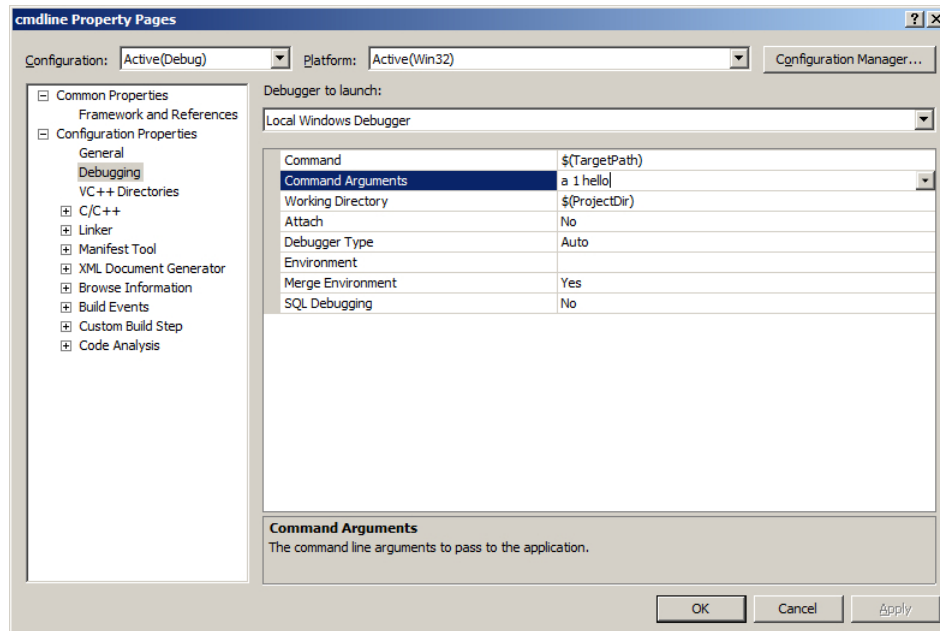


Figure 13: Setting project properties to pass command line arguments in VS.

Continuing debugging goes to next breakpoint hit, other options to follow execution is *Debug* → *Step* [*Into*—*Over*—*Out*]. Variable which value has changed between breakpoint hits is marked with red color, as presented in Local window in Fig. 16.

3 *nix systems

As *nix systems are described : commercial Unix, BSD, Darwin (Apple) and Linux distributions. Creating programs in *nix systems is easy as most distributions comes with compiler and text processors preinstalled. The compiler is GNU C Compiler which is available as *gcc* command and popular editors are *gedit*, *kedit* or other distribution's favorite one. In the Laboratory there is ARCH Linux distribution with gcc 4.8 with Xfce window manager, *Vim*, *Emacs*, and *Leafpad*. To check what compiler is installed and its version run in your command line interface (shell,terminal).

```
gcc -v
```

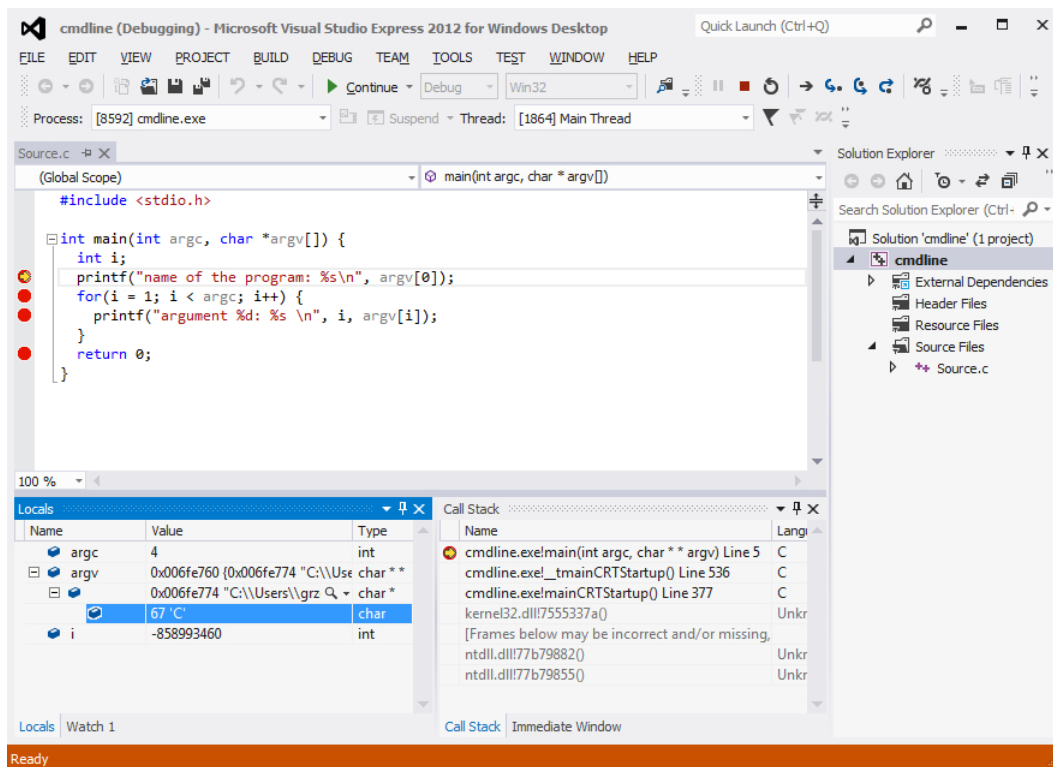


Figure 14: VS debugging mode.

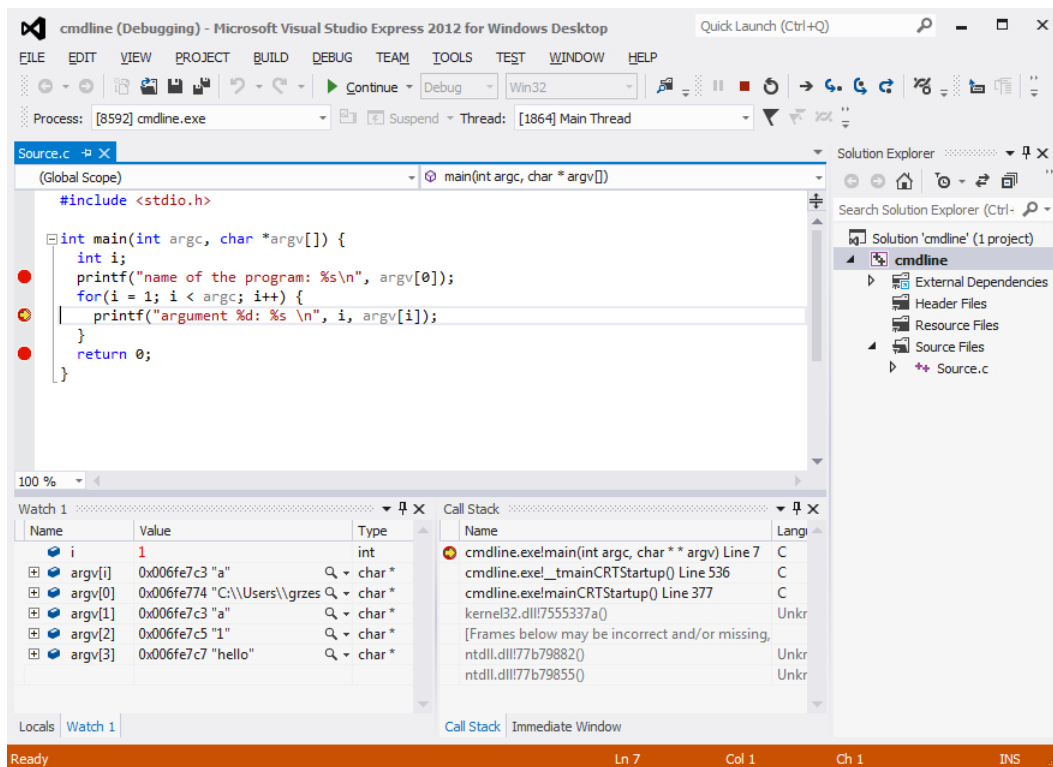


Figure 15: After setting Watch and continue debugging.

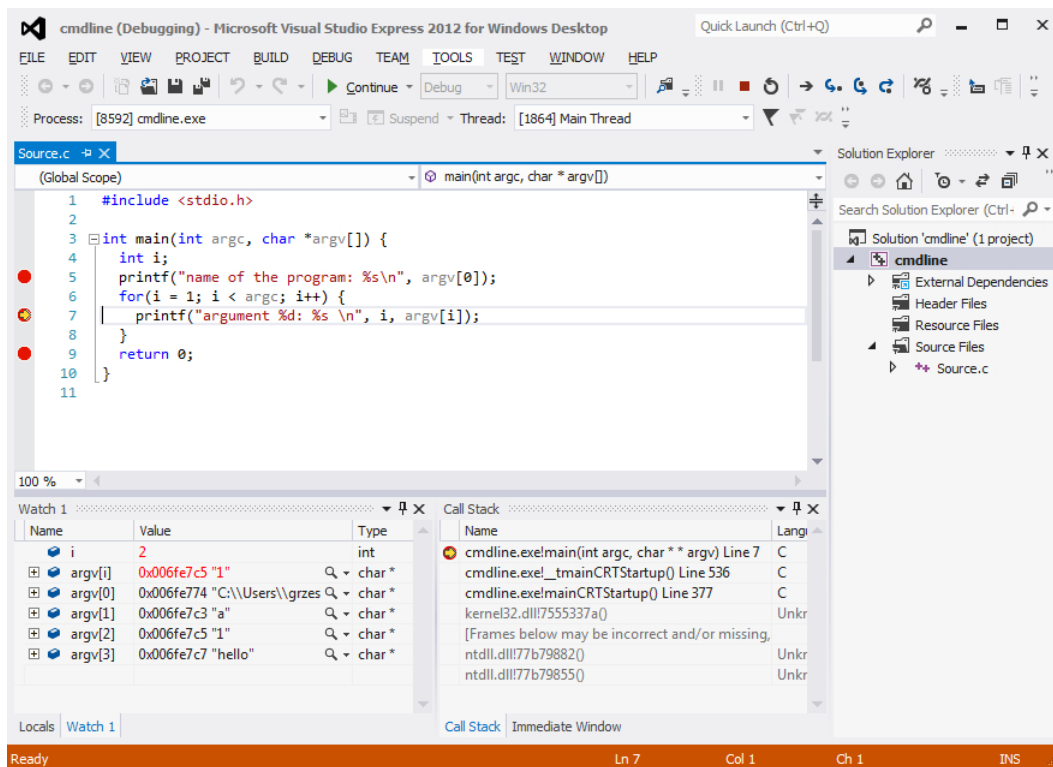


Figure 16: After 2 time continue step.

If previous command not working consult your distribution manual for installing software (in devel version with included headers, in some cases) Recommended is working under designated directory where source file and output executable will be created, so create new with

```
mkdir ~/Programming1
```

Change your working directory

```
cd ~/Programming1
```

Check current working directory with print working directory command *pwd*

```
pwd
```

Create empty file and/or open it in available editor and write a Hello World code.

```
touch proj1.c
```

To compile file just:

```
gcc proj1.c
```

If everything passed OK executable named *a.out* is created in the working directory. To run it type

```
./a.out
```

If you wish different output name use compiler's option *-out filename* (extension *.out* is not needed)

```
gcc proj1.c -out proj1
```

then to run it:

```
./proj
```

When compiling few files, its names should be listed as parameters. Problems may arise when trying compile *.cpp* file with gcc as extensions says compiler how to treat code inside.

4 Other IDEs

Other IDEs that can be used for programming. Presented is only general information as these tools are not fully supported in the Laboratory.

Eclipse, NetBeans

If you familiar with Java IDEs, there are plugin for C/C++ compilers usage. Only NetBeans is available in Laboratory under Windows but is not initially configured. Both configured IDEs are available under Linux. Further descriptions and configuration TODO if needed. (usually have to play with `vcx86_64vars` compiler variables).

CBlocks

Project creation idea is similar to VS, please remember to choose C project type. <http://www.codeblocks.org/> Use version bundled with mingw <http://sourceforge.net/projects/codeblocks/files/Binaries/12.11/Windows/codeblocks-12.11mingw-setup.exe> this includes GNU C Compiler windows port. Available under Linux in the Laboratory. http://www.cprogramming.com/code_blocks/

Changing console runner settings is demanded:

- choose *Menu* → *Settings* → *Environment*
- in General settings find **Terminal to launch console programs:**
- change *xterm - T \$TITLE -e* to **xfce4-terminal -T \$TITLE -x**

DevCpp

Project creation idea is similar to VS, please remember to choose C project type. Contain quite outdated version of GCC. <http://www.bloodshed.net/devcpp.html> Only Windows, not available in the Laboratory.

Other

The Laboratory Linux have Geany and Anjuta IDEs installed.

5 Documentation

<http://c-faq.com/>

<http://en.cppreference.com>

<http://www.cplusplus.com>

In *nix functions description is available by *man function_name* .

Windows resources

<https://msdn.microsoft.com/magazine/msdn-magazine-issues>

<https://code.visualstudio.com/shortcuts/keyboard-shortcuts-windows>.

pdf

<https://code.visualstudio.com/docs/getstarted/keybindings>

<https://channel9.msdn.com/Shows/C9-GoingNative/GoingNative-33-C-Refactoring-in-Vi>
time=04m37s