C++ Program Design

-- IDE_VC_Win32ConsoleApplication

Junjie Cao @ DLUT

Summer 2016

http://jjcao.github.io/cPlusPlus

1. Setting up a Console Application in VC 10

① Create a Console Project

② Solution Explorer

③ Add New Source File

④ Add Code

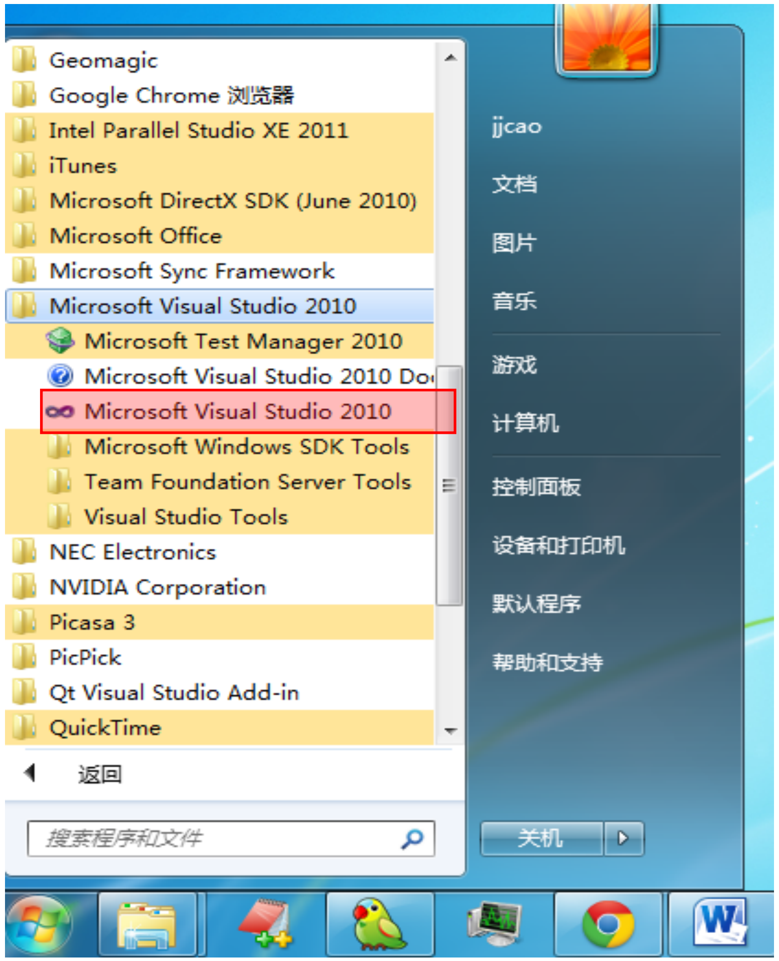
⑤ Class View

⑥ Setup Intermediate Directory (Optional)

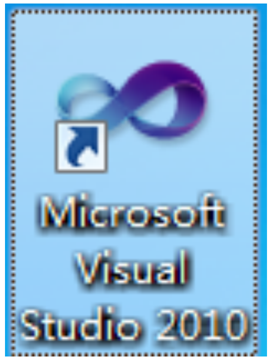
⑦ Build Project

⑧ Run the Program

Start VC IDE



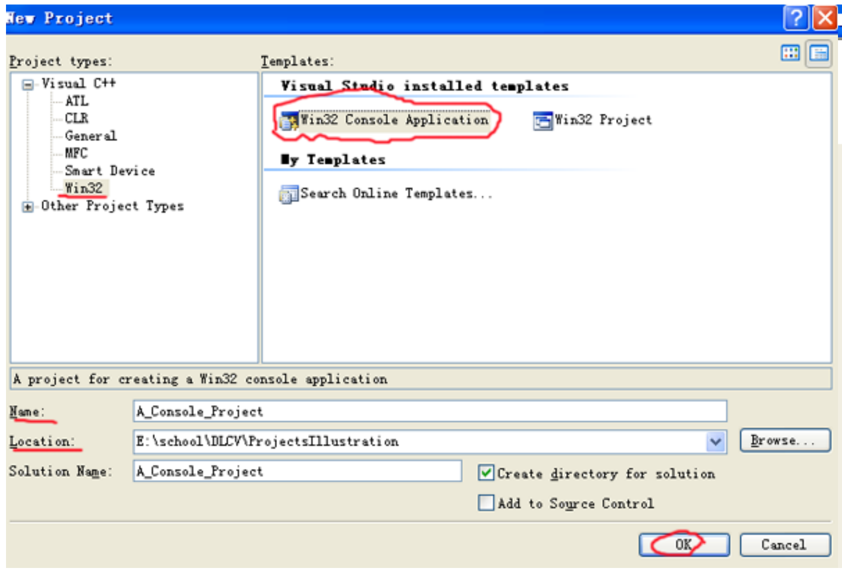
Geomagic
Google Chrome 浏览器
Intel Parallel Studio XE 2011
iTunes
Microsoft DirectX SDK (June 2010)
Microsoft Office
Microsoft Sync Framework
Microsoft Visual Studio 2010
Microsoft Test Manager 2010
Microsoft Visual Studio 2010 Do
Microsoft Visual Studio 2010
Microsoft Windows SDK Tools
Team Foundation Server Tools
Visual Studio Tools
NEC Electronics
NVIDIA Corporation
Picasa 3
PicPick
Qt Visual Studio Add-in
QuickTime
返回
搜索程序和文件
jjcao
文档
图片
音乐
游戏
计算机
控制面板
设备和打印机
默认程序
帮助和支持
关机



Microsoft Visual Studio 2010

Step 1: Create a Console Project

1. Choose **File -> New -> Project** from the VS210 menus => **Win32 Console Application** (chosen from the templates on the right side).
2. Set the location to someplace on your drive and give the project a name, such as **A_Console_Project**.
3. Click **OK**.
4. Go to the **Application Settings** tab. Make sure **Console application** and **Empty project** are selected.
5. Click **Finished**.



A project for creating a Win32 console application

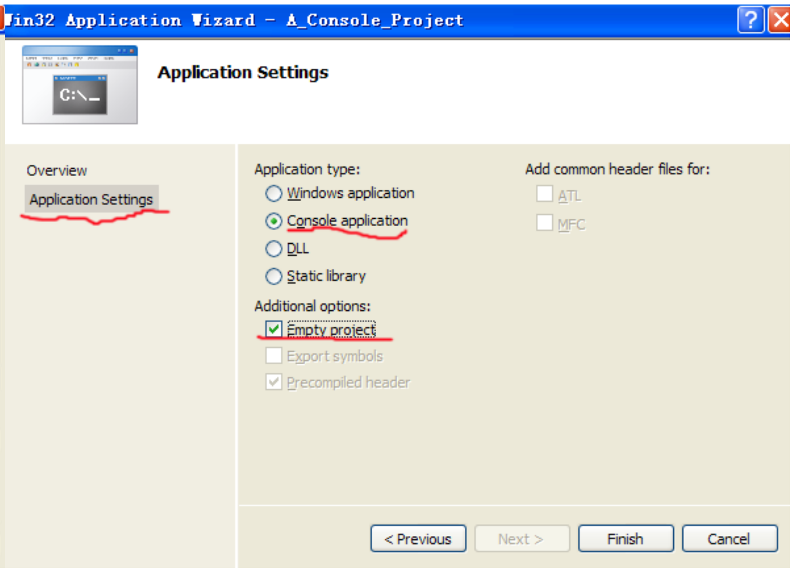
Name: A_Console_Project

Location: E:\school\DLCV\ProjectsIllustration

Solution Name: A_Console_Project

[✓] Create directory for solution

[ ] Add to Source Control



Application type:

- ○ Windows application
- ● Console application
- ○ DLL
- ○ Static library

Additional options:

- [✓] Empty project
- [ ] Export symbols
- [✓] Precompiled header

Add common header files for:

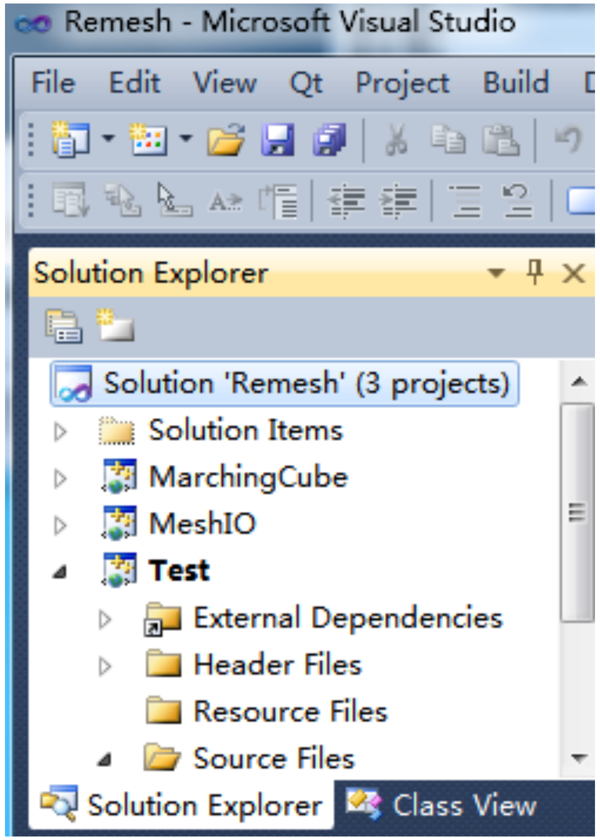
- [ ] ATL
- [ ] MFC

Step 2: Solution Explorer

1. Choose **View- Solution Explorer** from the VS menus

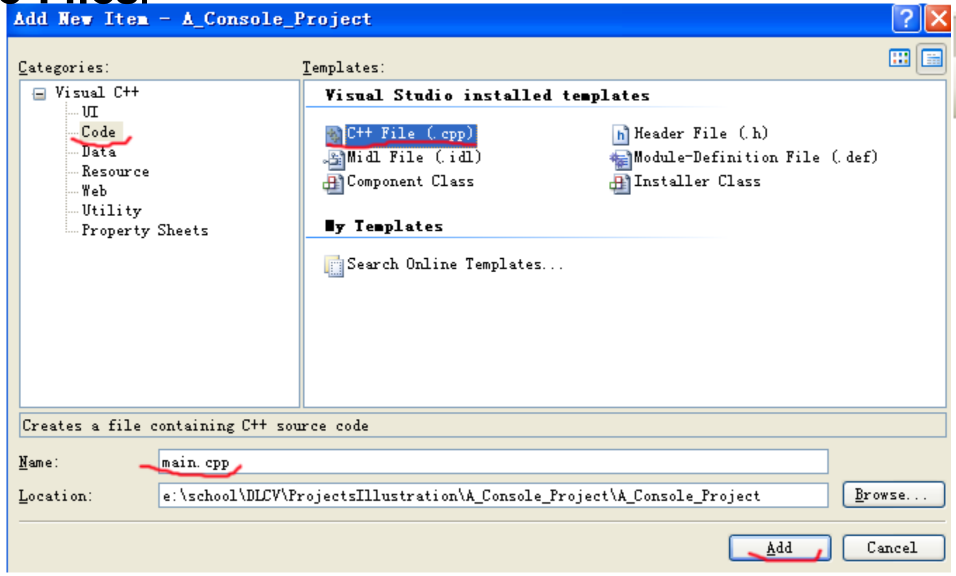
2. It shows you a **tree** representing your current solution.

3. **Solutions** are made of one or more **projects**, which in turn are composed of one or more files. We currently have one project in our solution, namely **A_Console_Project**



Step 3: Add New Source File

1. Right click on our project **A_Console_Project** at **solution explorer** and choose **Add->Add New Item**.
2. We want a C++ source file so choose the **Code** category and **C++ File (.cpp)** from the **Templates**.
3. Enter a name for our source file, **main.cpp** in our case. The location should already be under the project you've created.
4. Click **Add**.
5. You should now see a blank source file in the **editor** titled main.cpp.
6. **Solution Explorer** shows that **A_Console_Project** has currently a single file **main.cpp** under **Source Files.**


Add New Item - A_Console_Project

Categories:
- Visual C++
  - UI
  - Code
  - Data
  - Resource
  - Web
  - Utility
  - Property Sheets

Templates:

Visual Studio installed templates

C++ File (.cpp)	Header File (.h)
Midl File (.idl)	Module-Definition File (.def)
Component Class	Installer Class

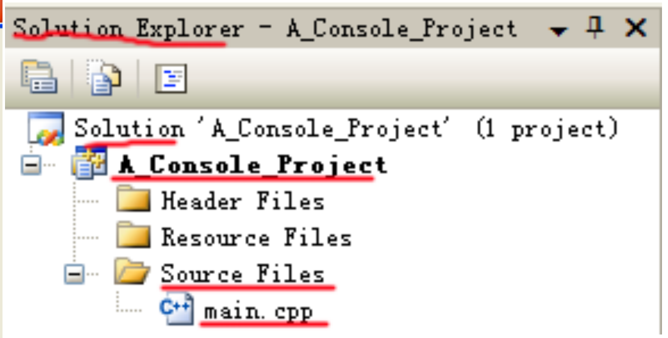
My Templates

Search Online Templates...

Creates a file containing C++ source code

Name: main.cpp

Location: e:\school\DLCV\ProjectsIllustration\A_Console_Project\A_Console_Project


Solution Explorer - A_Console_Project

- Solution 'A_Console_Project' (1 project)
  - A_Console_Project
    - Header Files
    - Resource Files
    - Source Files
      - main.cpp

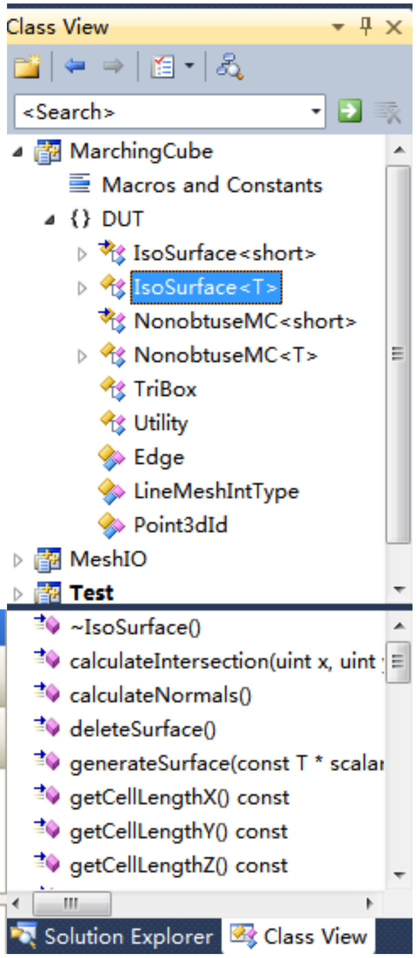
Step 4: Add Code

Enter the following code into **main.cpp** which should already be open in the main source window.

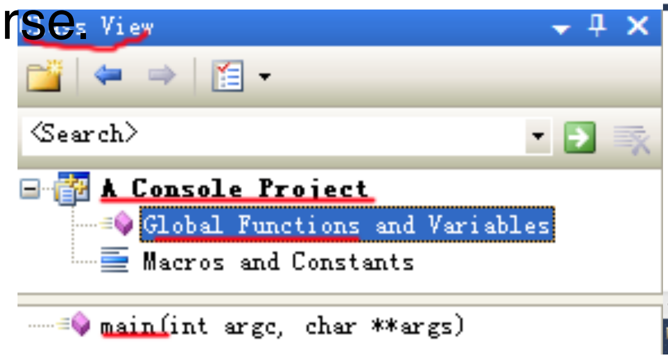
```
#include <iostream>
using namespace std;
int main(int argc, char** args){
        cout << "Hello world" << endl;
        return 0;
}
```

Step 5: Class View

1. We have a single function: **main()** currently, so it is easy to locate the code. However, the project can become rather large and finding the function you want to modify may not be so easy.
2. Choose **View -> Class View** => **Class View** window, which is similar to **Solution Explorer**. Hence you'll probably want to **place them together**.
3. It breaks the project up by **class and function rather than by file**. Right now we have only a single function **main** but later your projects will grow to multiply classes each containing many functions. The Class View window will prove invaluable when navigating such projects later in the course.



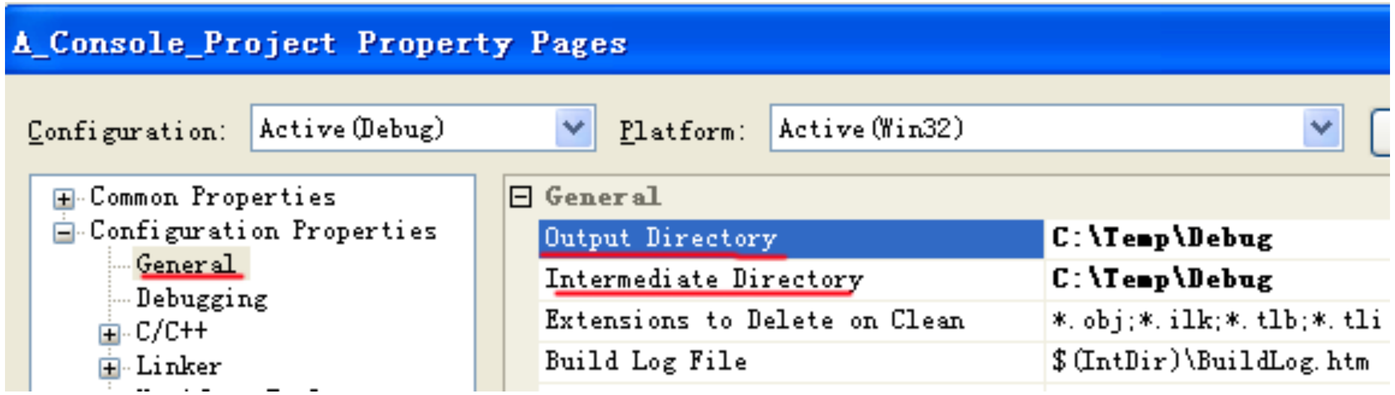
```
Class View
<Search>
MarchingCube
    Macros and Constants
  {} DUT
      IsoSurface<short>
      IsoSurface<T>
      NonobtuseMC<short>
      NonobtuseMC<T>
      TriBox
      Utility
      Edge
      LineMeshIntType
      Point3dId
MeshIO
Test
~IsoSurface()
calculateIntersection(uint x, uint
calculateNormals()
deleteSurface()
generateSurface(const T * scalar
getCellLengthX() const
getCellLengthY() const
getCellLengthZ() const
Solution Explorer   Class View
```



```
View
<Search>
A Console Project
    Global Functions and Variables
    Macros and Constants
main(int argc, char **args)
```

Step 6: Setup Intermediate Directory (Optional)

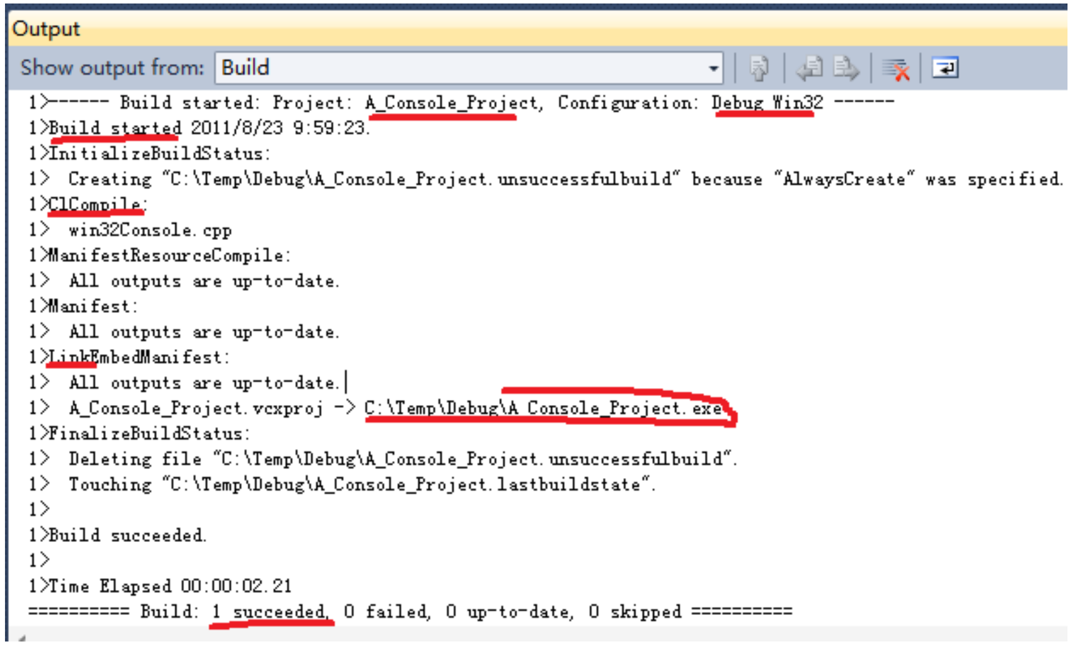
1. Open the Class View window, right click on our project **A_Console_Project** and choose **Properties**. Select **General** from the left hand pane and set the **Output Directory** and **Intermediate Directory** both to **C:\Temp\Debug**.
2. Click OK and we're all set to compile our project.



Output Directory	C:\Temp\Debug
Intermediate Directory	C:\Temp\Debug
Extensions to Delete on Clean	*.obj;*.ilk;*.tlb;*.tli
Build Log File	$(IntDir)\BuildLog.htm

Step 7: Build (compile+link) Project

1. **View->Other Windows->Output** from VS2010 menus. The **Output** window shows the output from the compile and linking process.
2. Chose **Build->Build Solution (F7)** from the VS2010 menus.
3. If you've copied the code correctly, you should see the following output to the right.
4. If the build failed you should see some text explaining why it failed. If the error is a **compiling error**, you should be able to double click on the error and the source window should update to show you the line on which the error occurred.

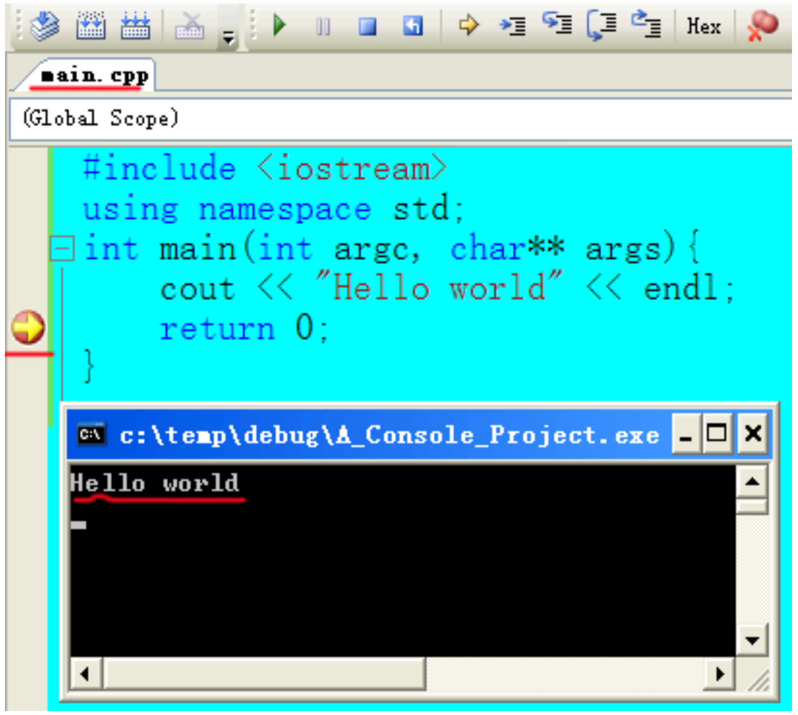


```
1>------ Build started: Project: A_Console_Project, Configuration: Debug Win32 ------
1>Build started 2011/8/23 9:59:23.
1>InitializeBuildStatus:
1>  Creating "C:\Temp\Debug\A_Console_Project.unsuccessfulbuild" because "AlwaysCreate" was specified.
1>ClCompile:
1>  win32Console.cpp
1>ManifestResourceCompile:
1>  All outputs are up-to-date.
1>Manifest:
1>  All outputs are up-to-date.
1>LinkEmbedManifest:
1>  All outputs are up-to-date.
1>  A_Console_Project.vcxproj -> C:\Temp\Debug\A_Console_Project.exe
1>FinalizeBuildStatus:
1>  Deleting file "C:\Temp\Debug\A_Console_Project.unsuccessfulbuild".
1>  Touching "C:\Temp\Debug\A_Console_Project.lastbuildstate".
1>
1>Build succeeded.
1>
1>Time Elapsed 00:00:02.21
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Step 8: Run the Program

- Place a **breakpoint** on the line “return 0” by left clicking on the position of the **big red point** (left click again, it will disappear) and press **F5**.
- A console window appears and the program will break at the line “return 0”.
- Press **F5** again to continue the execution of the program, and the program will exist successfully.

Congratulations!



```
#include <iostream>
using namespace std;
int main(int argc, char** args){
    cout << "Hello world" << endl;
    return 0;
}
```

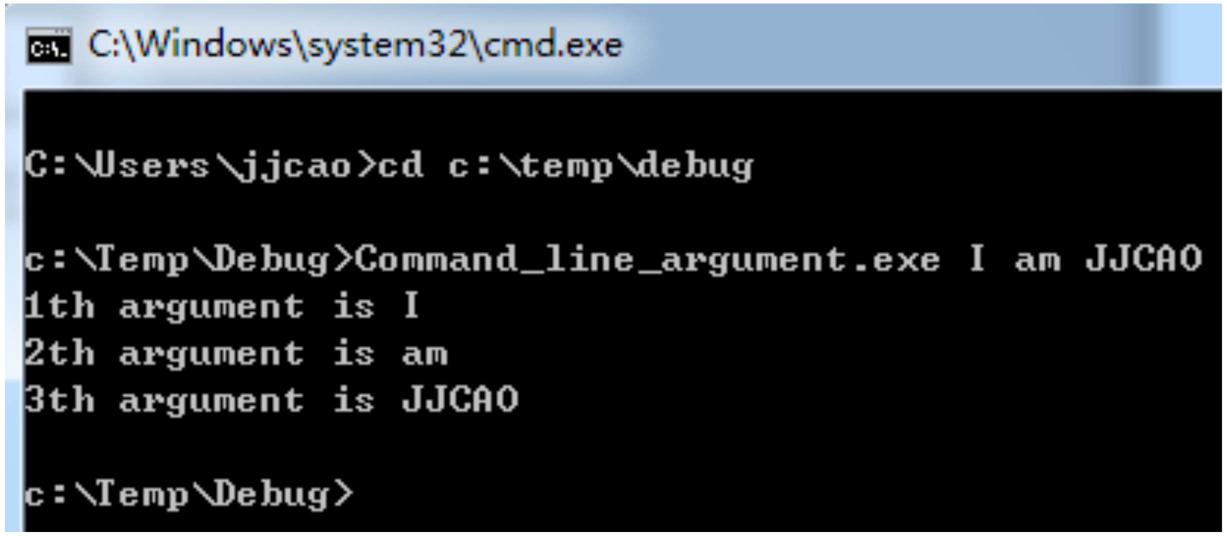
```
Hello world
```

Questions?

2. Command Line Argument

1. In Windows, the GUI will be used to communicate with the user.
2. In DOS, command line arguments are used to parse in users' specified parameters.
3. In the following figure, "**Command_line_argument.exe**" is the command and **I am JJCAO** is the command line arguments being parsed in.

1. Modify Main
2. Modify Project's Command Arguments
3. Build Project
4. Run the program



```
C:\Users\jjcao>cd c:\temp\debug

c:\Temp\Debug>Command_line_argument.exe I am JJCAO
1th argument is I
2th argument is am
3th argument is JJCAO

c:\Temp\Debug>
```

Step 1: Modify Main

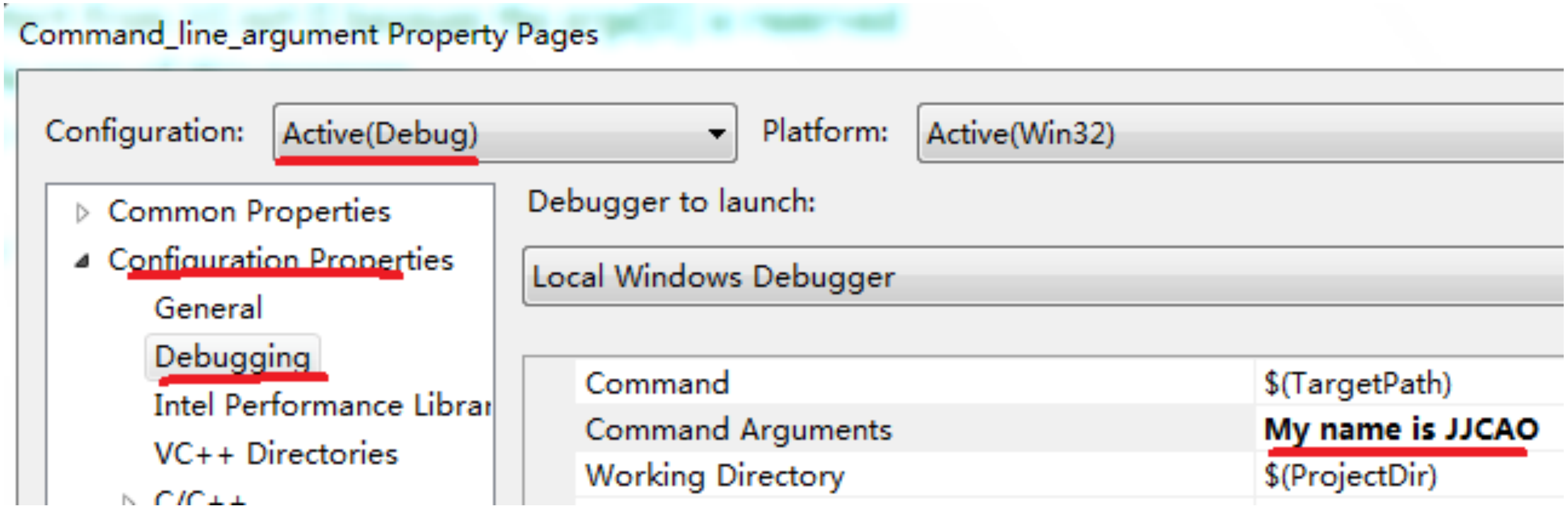
```
#include <iostream>
using namespace std;
int main(int argc, char** args)
{
// Notice I start from i=1 not 0 because the args[0] is reserved
//        for the name of this program.
          for(int i = 1; i < argc; i++)
          {
               cerr << i << "th argument is " << args[i] << "\n";
          }
}
```

Note: The relationship of argc and args:

1. args is **an array of char***
2. argc is the size of the array: args, which is determined when command line arguments are passed to the main() function. So after you change the size of args, argc is not updated automatically.

Step 2: Modify Project's Command Arguments Setting

Open the **Class View** window, right click on our project (**Command_line_argument**), and choose **Properties**. Choose **Debugging** from the left hand pane. Add **My name is JJCAO** to the **Command Arguments.** Click **OK**



Command_line_argument Property Pages

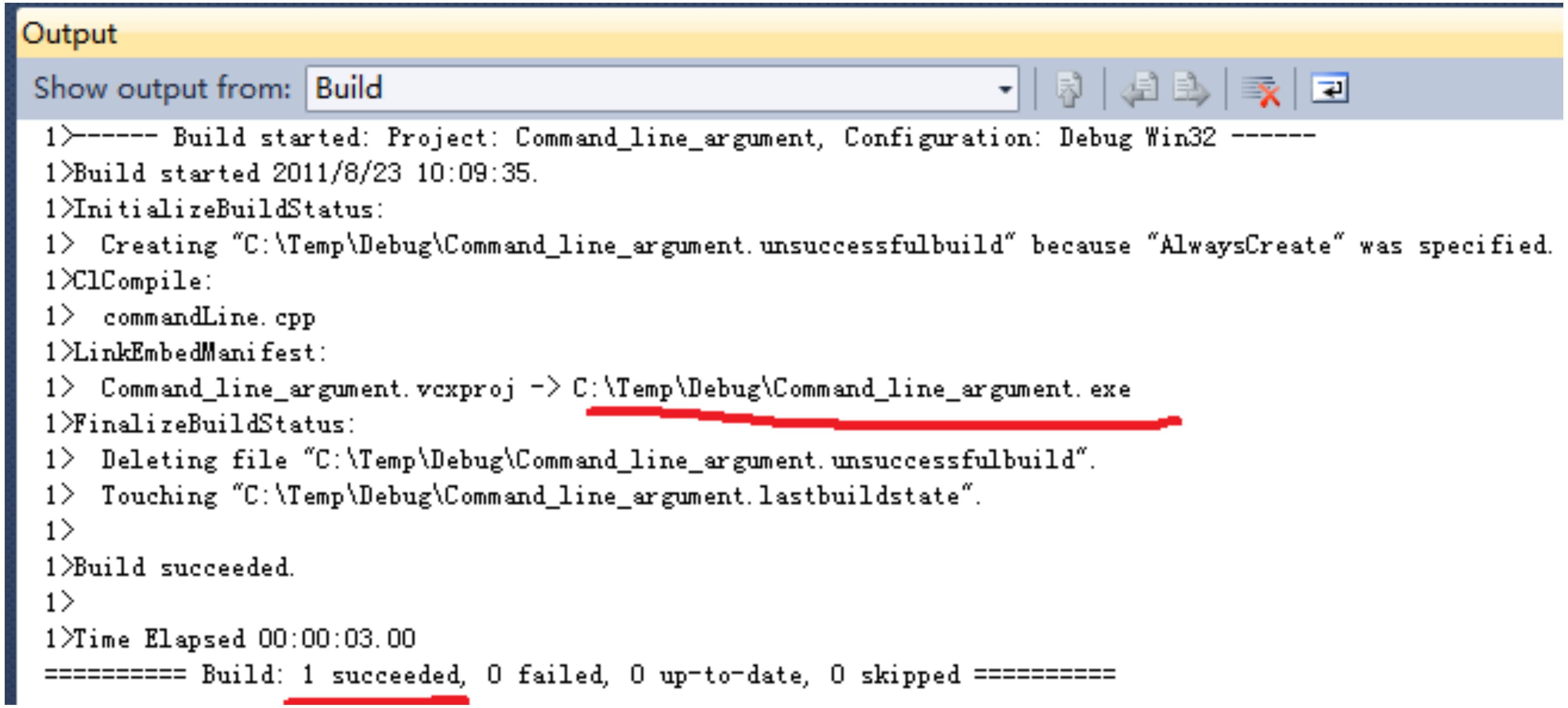
Configuration: Active(Debug)
Platform: Active(Win32)

Debugger to launch: Local Windows Debugger

Command	$(TargetPath)
Command Arguments	**My name is JJCAO**
Working Directory	$(ProjectDir)

Step 3: Build Project

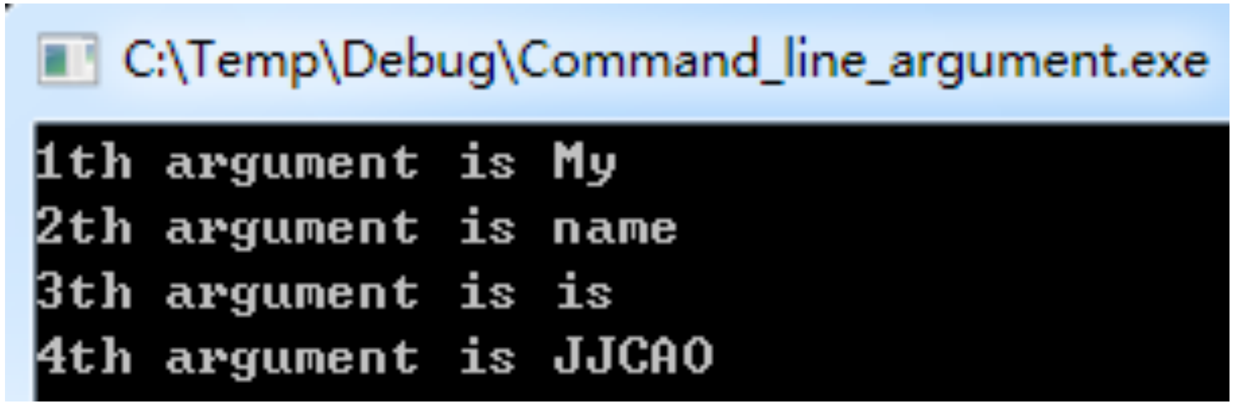
- Press **F7** to build the solution.



```
1>------ Build started: Project: Command_line_argument, Configuration: Debug Win32 ------
1>Build started 2011/8/23 10:09:35.
1>InitializeBuildStatus:
1>  Creating "C:\Temp\Debug\Command_line_argument.unsuccessfulbuild" because "AlwaysCreate" was specified.
1>ClCompile:
1>  commandLine.cpp
1>LinkEmbedManifest:
1>  Command_line_argument.vcxproj -> C:\Temp\Debug\Command_line_argument.exe
1>FinalizeBuildStatus:
1>  Deleting file "C:\Temp\Debug\Command_line_argument.unsuccessfulbuild".
1>  Touching "C:\Temp\Debug\Command_line_argument.lastbuildstate".
1>
1>Build succeeded.
1>
1>Time Elapsed 00:00:03.00
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Step 4: Run the Program

- Press **F5** to run the program..
- Congratulations! You've got your 2nd successful VS2010 project



C:\Temp\Debug\Command_line_argument.exe

```
1th argument is My
2th argument is name
3th argument is is
4th argument is JJCAO
```

Note: When debugging a c++ program from the IDE (F5), the **current path is the path where the project file (*.vcproj) is located**. When running a c++ program from the IDE (Ctrl+F5), the current path is the path where the excutable file (*.exe) is located (commonly is the path **Debug**).

Questions

Input

- Let user input values to the program (line 6)

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5    int x;
6    cin >> x;
7
8    cout << x / 3 << ' ' << x * 2;
9
10   return 0;
11 }
```

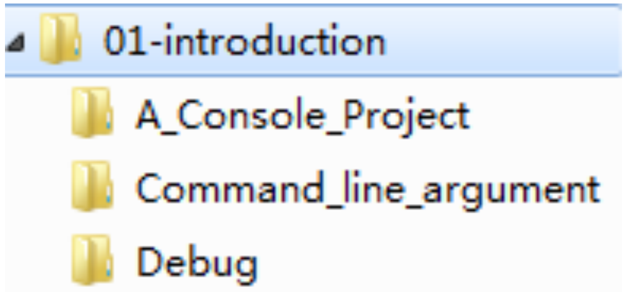
- ctrl+z: cancel input from cin

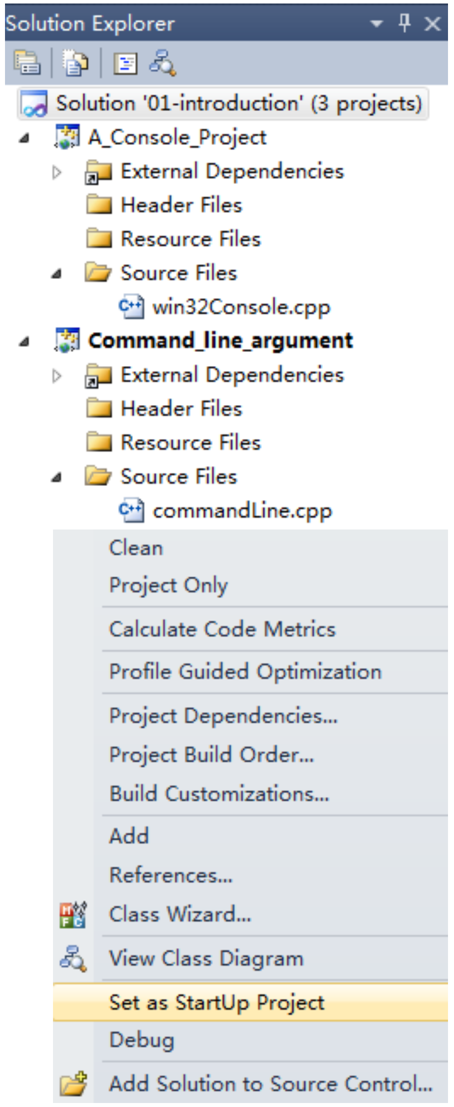
iostream

- cin
- cout
- cerr
- clog
- Ordinarily, sys associates them with the console window.
- They can be redirected to files.

Default current directory of VC

- **Solution**: 01-Introduction
  - Project: A_Console_Project
  - Project: Command_line_argument





- **Current Project**: Command_line_argument
- The **current directory** of the current project
  - The dir where the Command_line_argument.vcxproj is
  - Where is win32Console.cpp?
    ../ A_Console_Project/

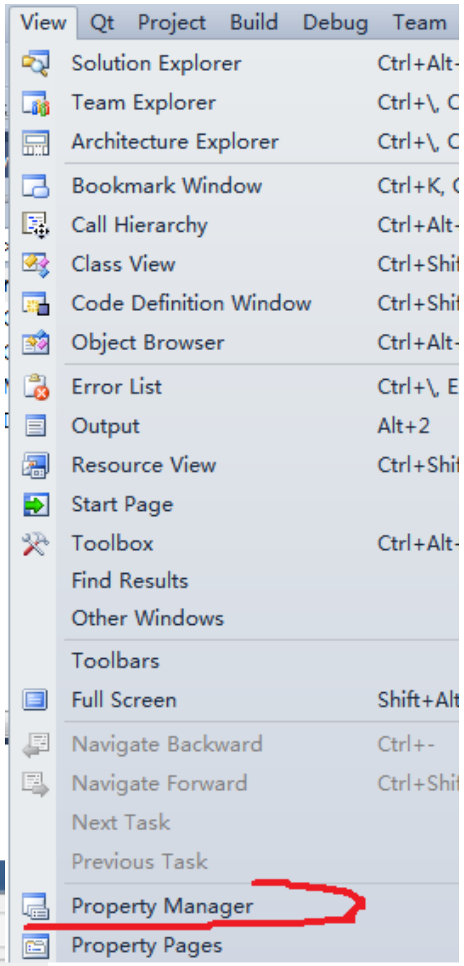
Debugging

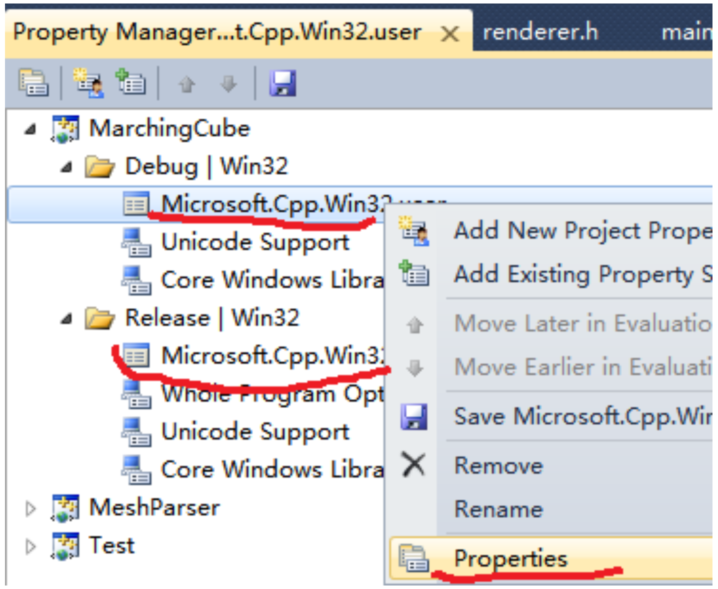
- Compilation error
  - Violations of the syntax rules
  - Misuse of types

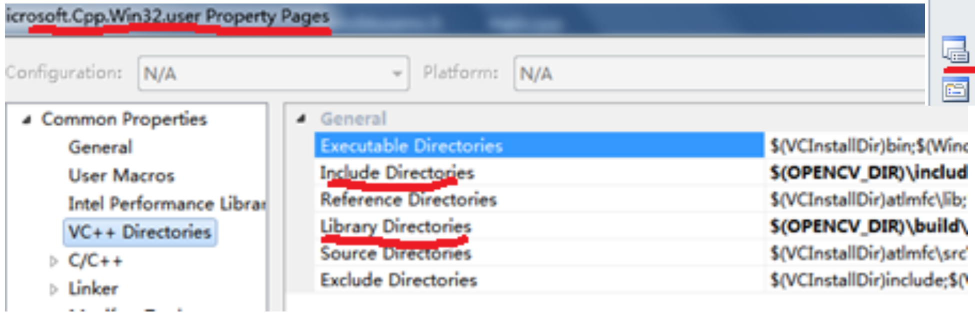
- Runtime error
  - Need debugging

Set include & lib path independent with solutions

Set it in Property Manager (You have to open a project first.) If you set it in the Context Menu of a solution or project, it will be dependent on specified projections.







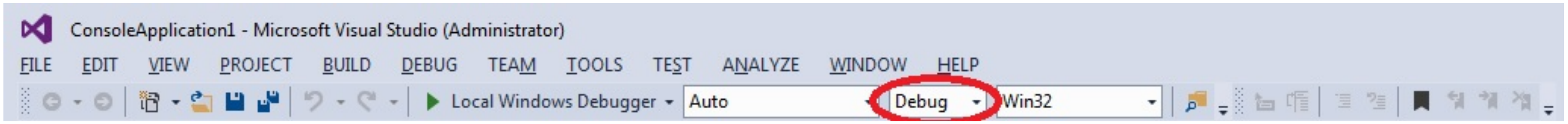
Configuration: N/A Platform: N/A

General

Executable Directories	$(VCInstallDir)bin;$(Winc
Include Directories	$(OPENCV_DIR)\includ
Reference Directories	$(VCInstallDir)atlmfc\lib;
Library Directories	$(OPENCV_DIR)\build\
Source Directories	$(VCInstallDir)atlmfc\src
Exclude Directories	$(VCInstallDir)include;$(

Build configurations

- **Switching between debug and release in Visual Studio**


ConsoleApplication1 - Microsoft Visual Studio (Administrator)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Local Windows Debugger
Auto
Debug
Win32

Pack Your Solution
-- before sending it to others

1. Delete *.ncb || *.sdf
2. Delete debug, release && ipch directories
3. Compress all into a *.rar or *.zip
4. Sent it by email || …