**Practical # 01**

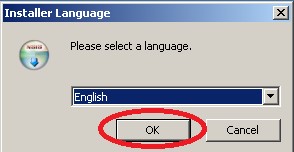
# Introduction to DEV C++ IDE

**Objective: *To understand the DEV C++ IDE(Integrated Development Environment) and implement a simple C Program.* Theory:**

## The Integrated Development Environment (IDE)

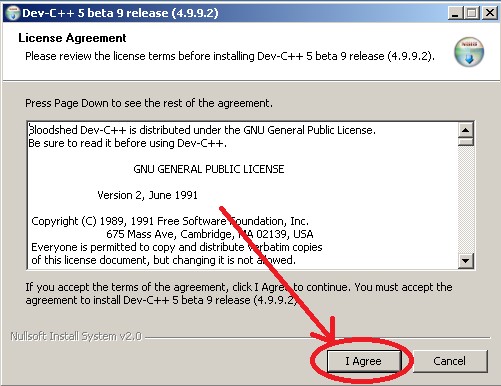
Dev-C++, developed by [Bloodshed Software,](http://www.bloodshed.net/) is a fully featured graphical IDE (Integrated Development Environment), which is able to create Windows or console-based C/C++ programs using the MinGW compiler system. MinGW (Minimalist GNU\* for Windows) uses GCC (the GNU g++ compiler collection), which is essentially the same compiler system that is in Cygwin (the unix environment program for Windows) and most versions of Linux. ***Installation Steps:***

1. Download the installer from the internet. Follow the instructions and install the program. The following screenshots will help you install and run the product:



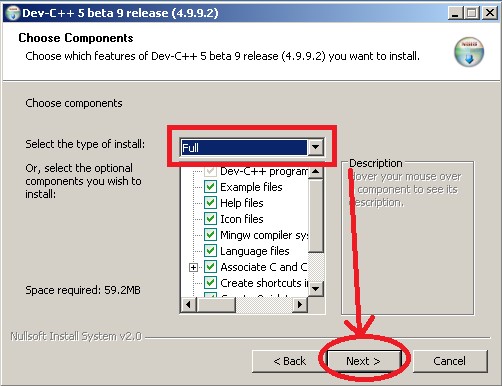
1. License Agreement

Click on the "I Agree" button to continue



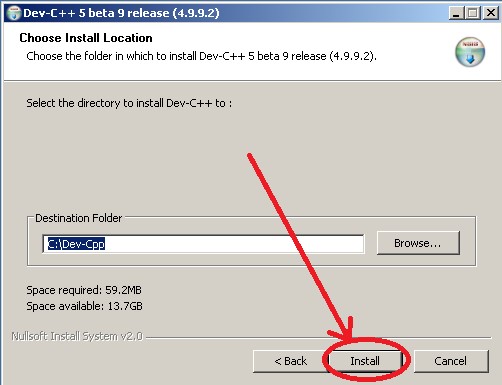
1. Choose Components

Make sure that the type of install is Full and click the Next button to continue

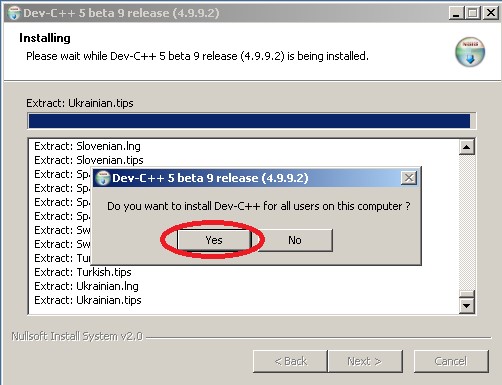


1. Choose Install Location

Click the Install button to continue

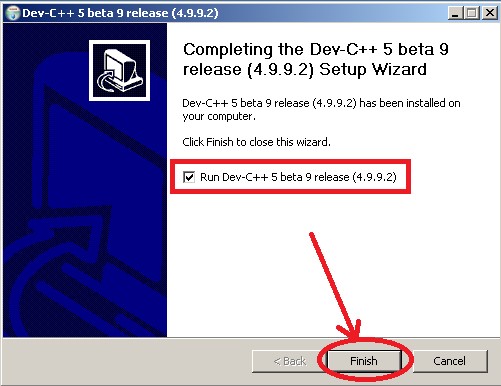


1. Installing Click the Yes button



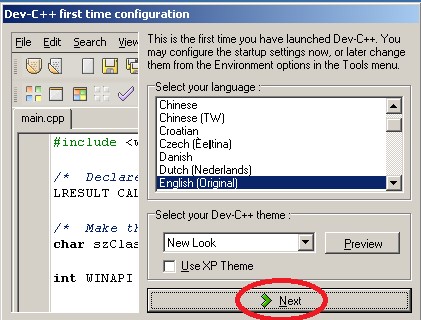
1. Finished

Click the Finish button to finalize the installation and run the program.



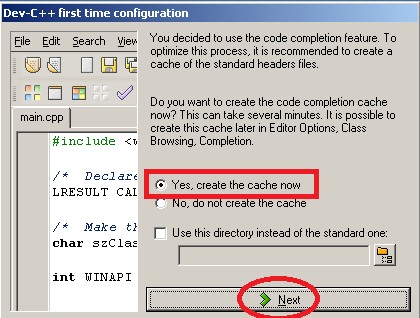
1. First Time Configuration

Click the Next button to continue



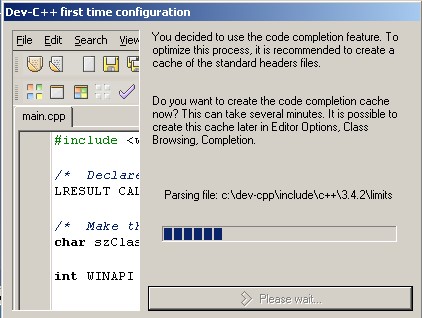
1. First Time Configuration

Click the Next button to continue



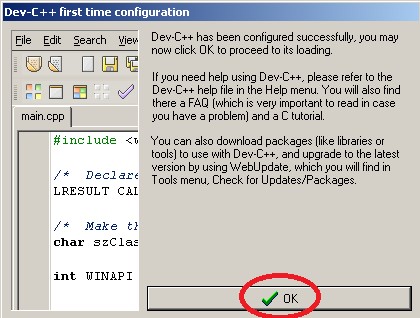
1. First Time Configuration

Wait for the Progress Bar to Complete



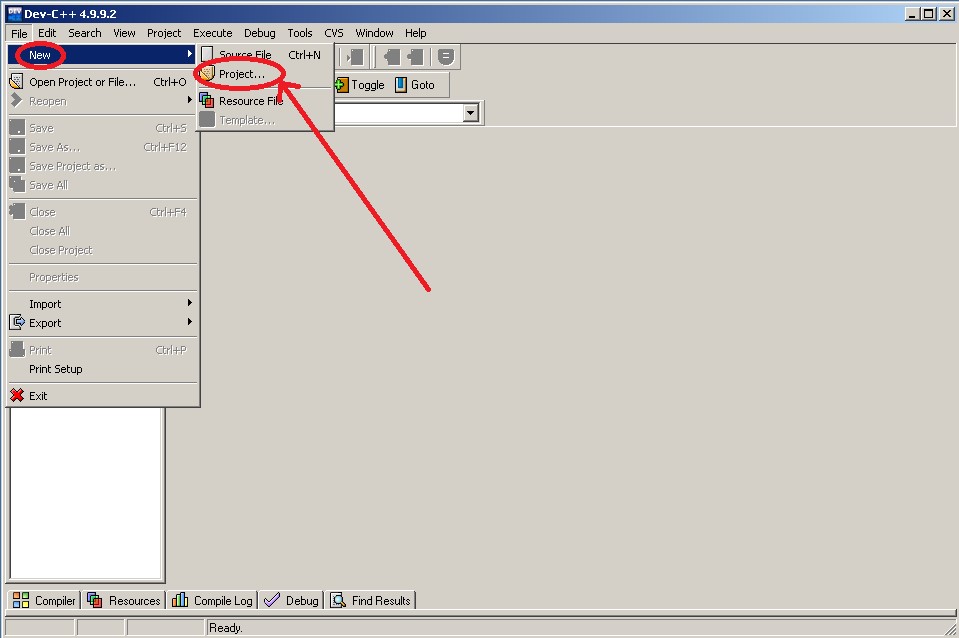
1. First Time Configuration

Click the OK button to Finalize



**New Project Menu**

Click the File menu, then select the New menu item and click the Project menu item.



New Project

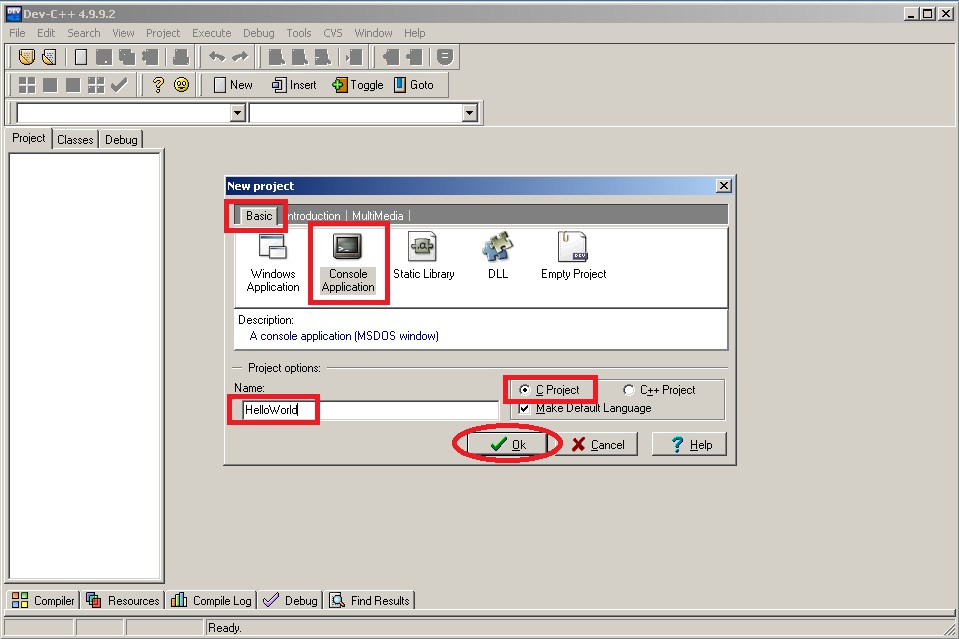
On the top, make sure that Basic tab is selected and under the Basic tab, select “Console

Application”

Give a name to your project using the Name text box, For instance, “Hello World”.

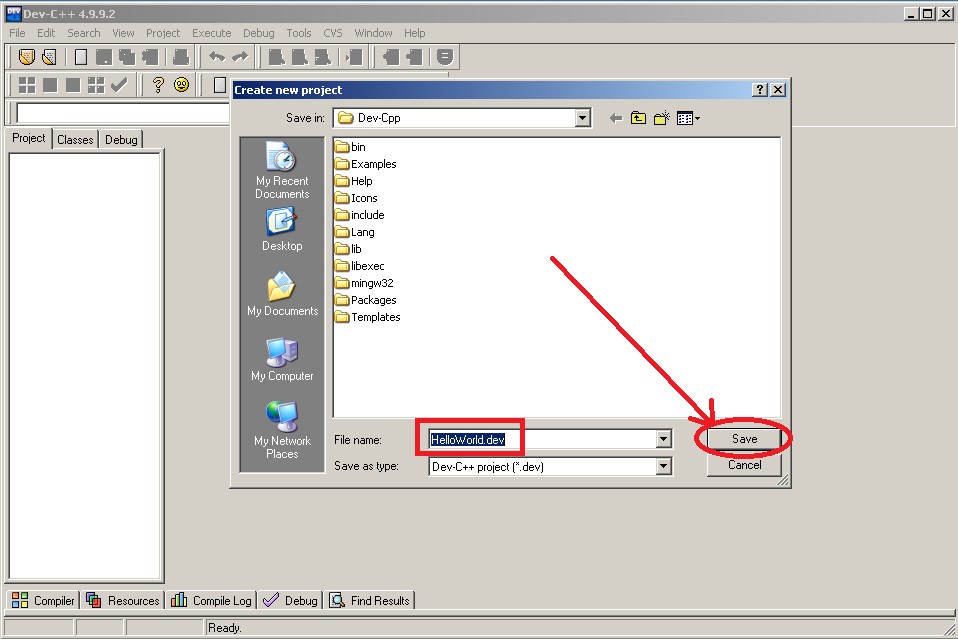
Important: Choose “C Project” under “Project Options”, on the left

Click the OK button to create your project



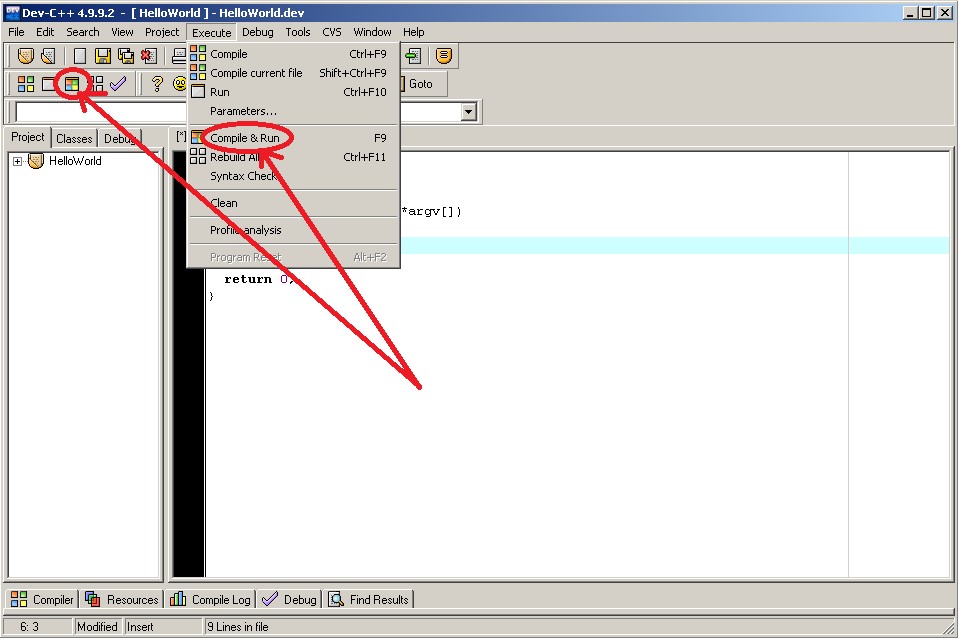
**Create New Project**

Give a name to your project file and click the Save button to continue



**Compile & Run:**

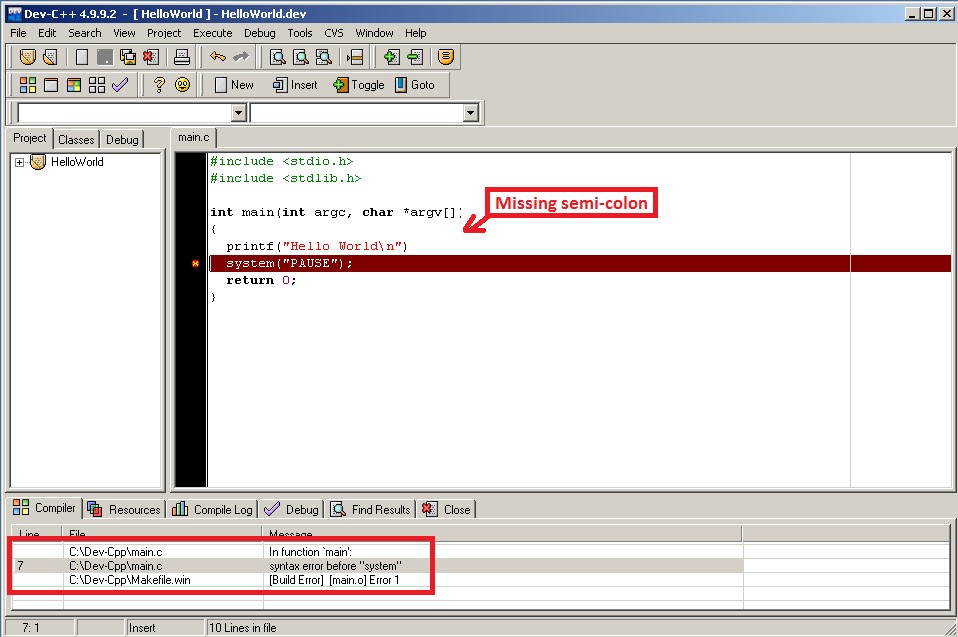
Click “Compile & Run” menu item or the icon displayed in the below screenshot or just Press F9 to compile and run your program.



**Running**: Assuming you did not make any syntax errors on your code, you should see a similar output window running your program.

**Compile Failed**

If you try to compile a code which has syntax errors, Compiler window lists the errors with their line numbers. You can double click the error and see the error highlighted in the code.



**Review Questions/ Exercise:**

1. **Discuss the steps necessary to produce executable file?**

**Step 1: Writing the Source Code**

- Write the program's source code in a programming language (e.g., C, C++, Java, Python).

- Use a text editor or an Integrated Development Environment (IDE) to create and edit the source code file.

**Step 2: Compilation (Optional)**

- If the chosen programming language is compiled (e.g., C, C++), compile the source code into object code using a compiler (e.g., GCC).

- Compilation checks for syntax errors and translates the source code into machine-specific assembly code.

**Step 3: Assembly (Optional)**

- If the compiler generates assembly code, assemble it into object code using an assembler (e.g., AS).

- Assembly converts the assembly code into machine-specific binary code.

**Step 4: Linking**

- Link the object code files generated in Step 2 or 3 using a linker (e.g., LD).

- Linking resolves external references between object files and libraries, creating a single executable file.

**Step 5: Loading**

- Load the executable file into memory.

- The operating system's loader loads the executable file into memory, preparing it for execution.

**Step 6: Execution**

- Execute the program by running the executable file.

- The operating system's processor executes the machine-specific binary code in the execution.

1. **Discuss the purpose of Compiler & the file needed by compiler?**

**Purpose of a Compiler**:

A compiler is a complex software program that plays a crucial role in the development and execution of computer programs. Its primary purpose is to translate source code written in a high-level programming language (such as C, C++, Java, or Python) into machine code that can be executed directly by the computer's processor.

**Files Needed by a Compiler:**

To compile a program, the compiler requires the following files:

1. Source Code File (e.g., .c, .cpp, .java, .py): This is the file containing the high-level program code written by the programmer.

2. Header Files (e.g., .h, .hpp): These files contain function declarations, macro definitions, and type definitions that are used by the source code.

3. Library Files (e.g., .lib, .so, .dll): These files contain pre-compiled code that provides additional functionality, such as standard libraries or third-party libraries.

4. Configuration Files (e.g., Makefile, build.config): These files specify compilation options, include paths, and library dependencies.

5. Object Files (e.g., .o, .obj): These files are generated by the compiler as intermediate output, containing machine code for each source file.

1. **Discuss the linker & the file needed by the linker?**

**What is a Linker?**

A linker, also known as a linker editor or ld, is a program that takes object files generated by a compiler and combines them into a single executable file that can be run on a computer. The linker resolves external references between object files and libraries, allowing the program to use functions and variables defined elsewhere.

**File Needed by the Linker:**

To perform its task, the linker requires the following files:

1. Object files (.obj, .o): These files contain machine code generated by the compiler. Each object file represents a translation unit, which is typically a single source file (e.g., main.c becomes main.o).

2. Library files (.lib, .a, .so): These files contain compiled code that provides additional functionality. Libraries can be static (linked directly into the executable) or dynamic (loaded at runtime).

3. Header files (.h, .hpp): Although not directly used by the linker, header files declare functions, variables, and types that are used in the object files.

**Name: \_\_\_\_\_**

**Roll #: \_\_\_\_\_**

**Date: \_\_\_\_\_**

**Subject Teacher / LAB Engineer**

**Remarks:**