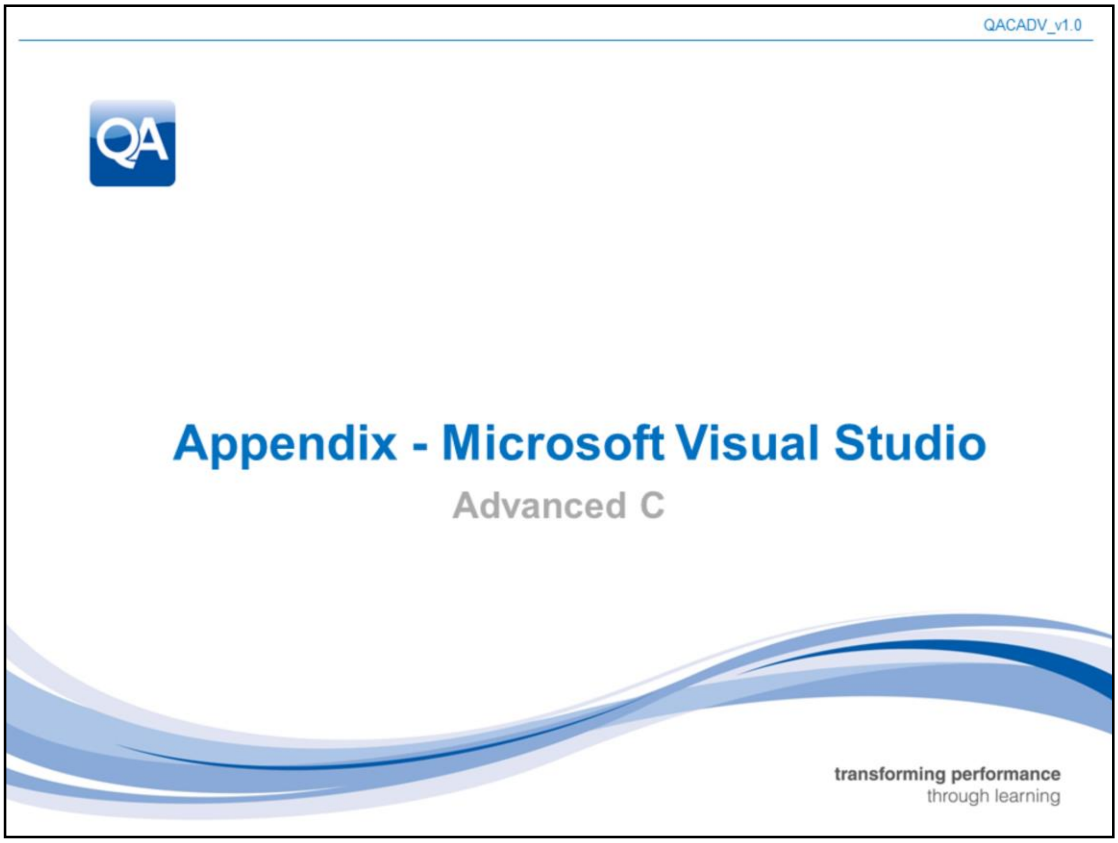
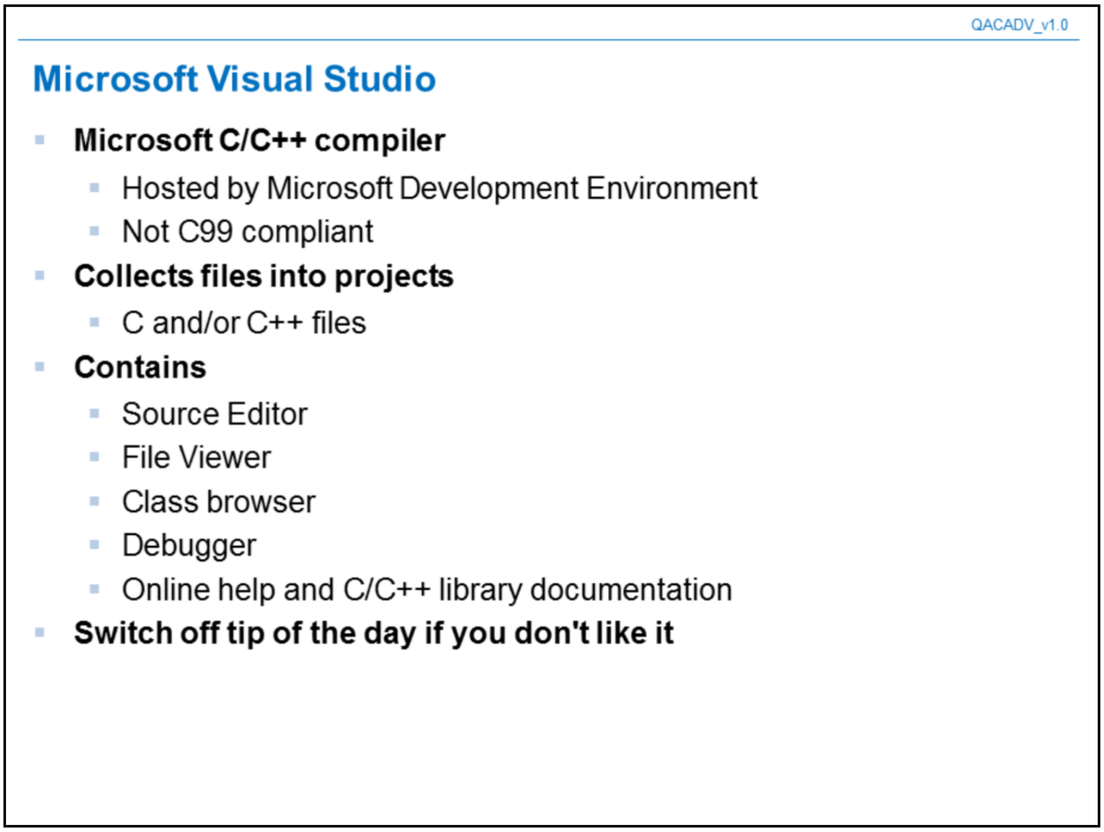


QACADV_v1.0

Appendix - Microsoft Visual Studio

Advanced C

transforming performance
through learning



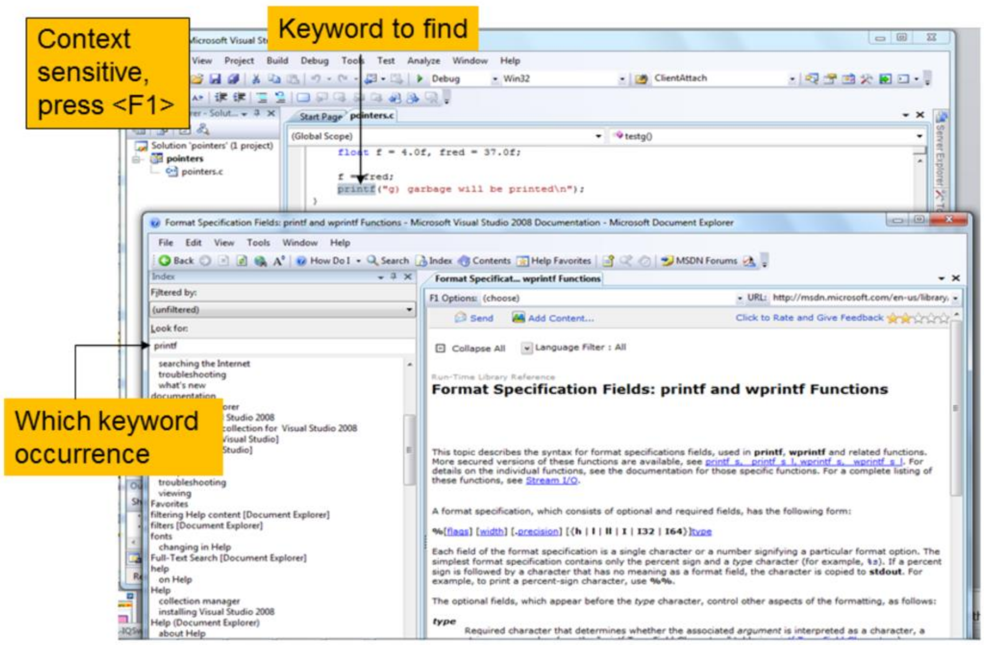
QACADV_v1.0

Microsoft Visual Studio

- **Microsoft C/C++ compiler**
  - Hosted by Microsoft Development Environment
  - Not C99 compliant
- **Collects files into projects**
  - C and/or C++ files
- **Contains**
  - Source Editor
  - File Viewer
  - Class browser
  - Debugger
  - Online help and C/C++ library documentation
- **Switch off tip of the day if you don't like it**

QACADV_v1.0

Getting Help



```
float f = 4.0f, fred = 37.0f;

f = fred;
printf("g) garbage will be printed\n");
}
```

Run-Time Library Reference

Format Specification Fields: printf and wprintf Functions

This topic describes the syntax for format specifications fields, used in **printf**, **wprintf** and related functions. More secured versions of these functions are available, see printf_s, _printf_s_l, wprintf_s, _wprintf_s_l. For details on the individual functions, see the documentation for those specific functions. For a complete listing of these functions, see Stream I/O.

A format specification, which consists of optional and required fields, has the following form:

%[flags] [width] [.precision] [{h | l | ll | I | I32 | I64}]type

Each field of the format specification is a single character or a number signifying a particular format option. The simplest format specification contains only the percent sign and a *type* character (for example, %s). If a percent sign is followed by a character that has no meaning as a format field, the character is copied to **stdout**. For example, to print a percent-sign character, use **%%**.

The optional fields, which appear before the *type* character, control other aspects of the formatting, as follows:

type

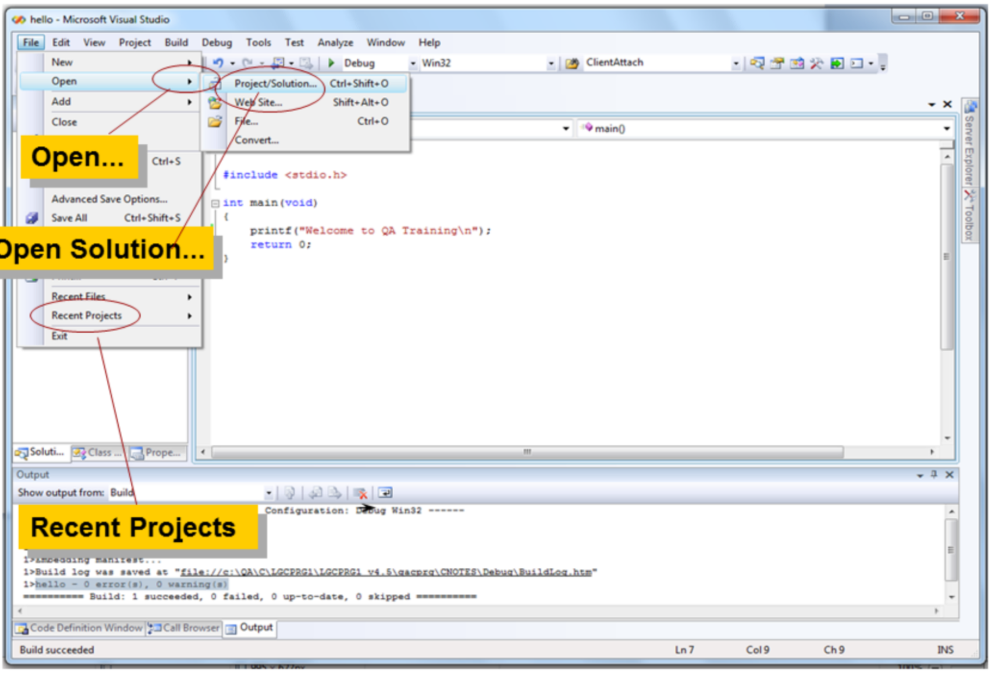
Required character that determines whether the associated *argument* is interpreted as a character, a

Visual Studio provides extensive on-line documentation via the MSDN (Microsoft Developer's Network). This contains the entire Visual C/C++ documentation, reference material from the Win32 SDK and the Knowledge Base. To find specific information, you can either browse through the table of contents, use the Search option, or access the Index. While you are editing source code, you can get immediate help by selecting the item in question and hitting the F1 key.

The MSDN is a fully functional HTML browser. Help is context sensitive and can even hot-link to up-to-date help pages on Microsoft's web site if you have a TCP/IP link online.

QACADV_v1.0

Solutions



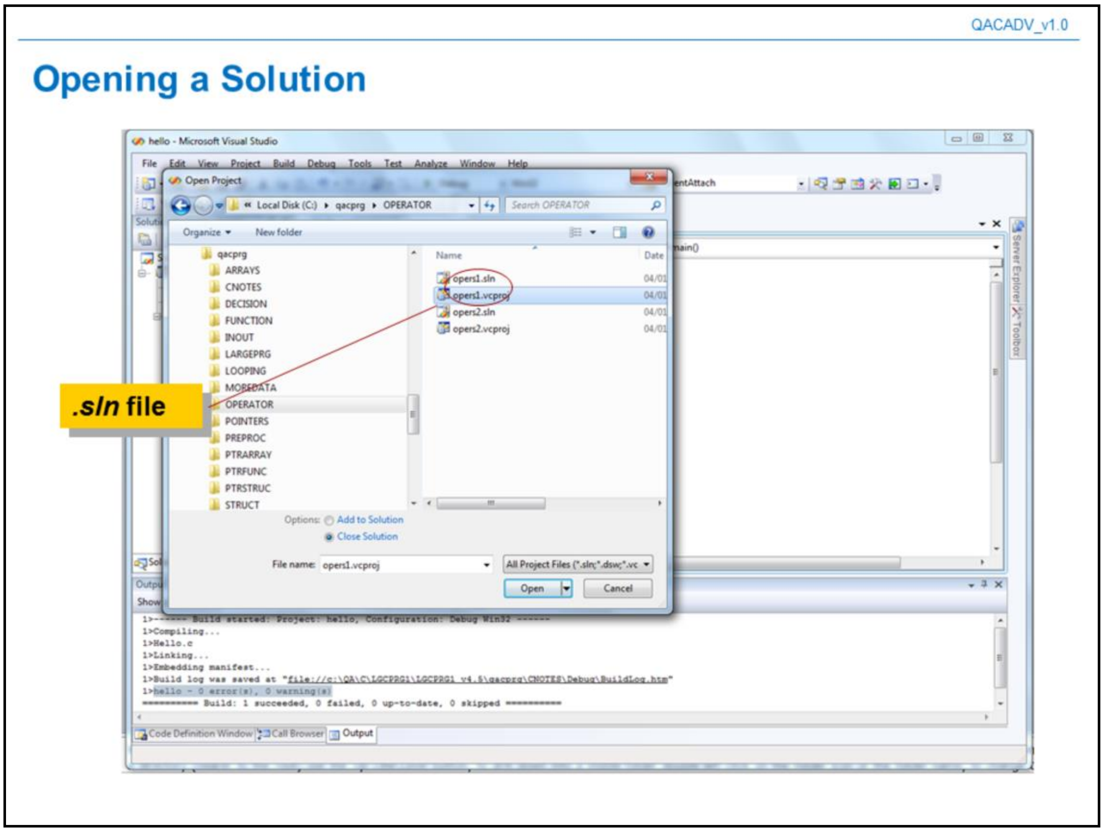
```c
#include <stdio.h>

int main(void)
{
    printf("Welcome to QA Training\n");
    return 0;
}
```

```
Configuration: Debug Win32 ------
1>Embedding manifest...
1>Build log was saved at "file://c:\QA\C\LGCPRG1\LGCPRG1 v4.5\qacprg\CNOTES\Debug\BuildLog.htm"
1>hello - 0 error(s), 0 warning(s)
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

The Integrated Development Environment (IDE) uses so called 'Solutions'. An unfortunate phrase when doing course exercises, since it implies something completely different! A Solution contains one or more **projects**, even projects of different types (for example C, C++ and Java projects).

To create a new project from the *File* menu, click *New* and then click the *Project…* tab. For C, select a project type of 'Empty Project'. Specify the project *Name* and *Location*. Click *OK*.



Opening a Solution

Options: ○ Add to Solution [✓ Close Solution]

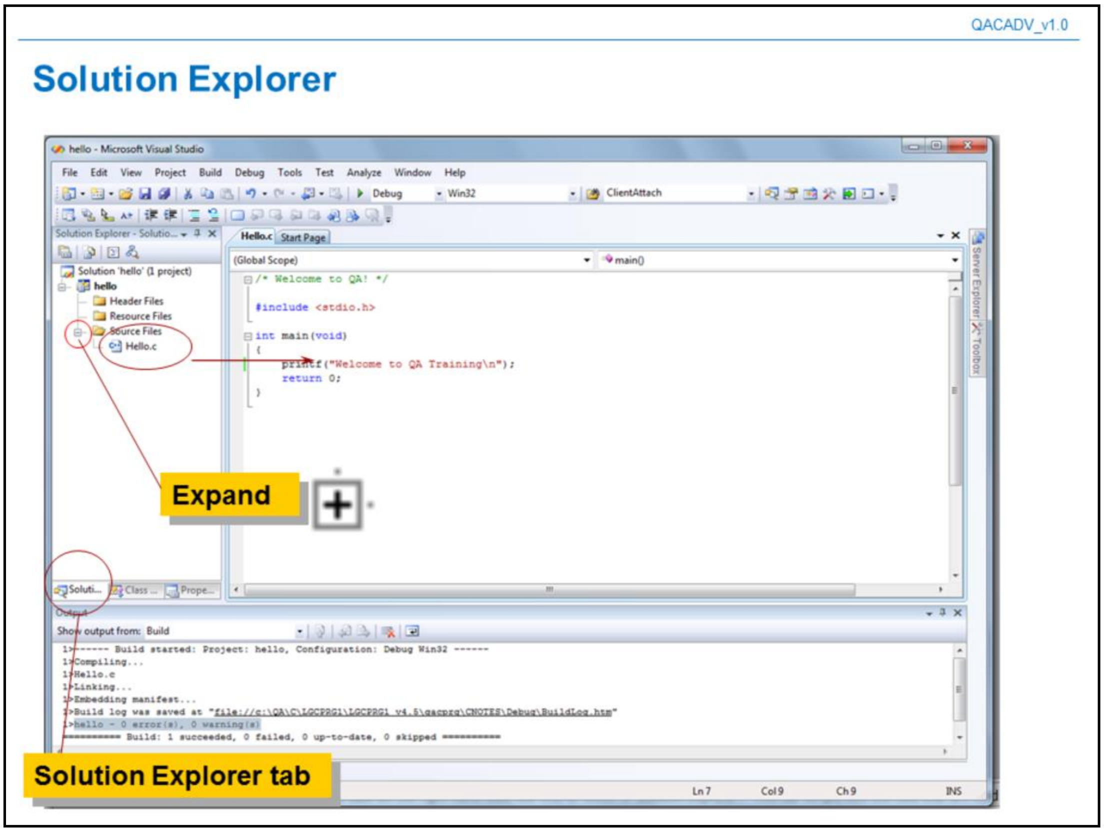
File name: opers1.vcproj

All Project Files (*.sln;*.dsw;*.vc

```
1>------ Build started: Project: hello, Configuration: Debug Win32 ------
1>Compiling...
1>Hello.c
1>Linking...
1>Embedding manifest...
1>Build log was saved at "file://c:\QA\C\LGCPRG1\LGCPRG1 v4.5\qacprg\CNOTES\Debug\BuildLog.htm"
1>hello - 0 error(s), 0 warning(s)
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

When using *File-Open Solution…* a dialog box will appear allowing you to select the required solution. Simply navigate to the required folder (directory); to move up a directory (nearer to the root) use the *Up One Level* button, to drill down into a visible folder double left-click on the folder icon or the folder name, to change to a different disk use the drop down menu just to the left of the *Up One Level* button.

When the specific `.sln` file is visible, simply double left-click it to open the Solution. It is possible that the file suffixes may not be visible. This might be because of a Microsoft Explorer setting. Open Explorer and from its *View* menu select *Options…*, and then select the *View* tab and look at the check box labelled *Hide file extensions for known file types*.



```
/* Welcome to QA! */

#include <stdio.h>

int main(void)
{
    printf("Welcome to QA Training\n");
    return 0;
}
```

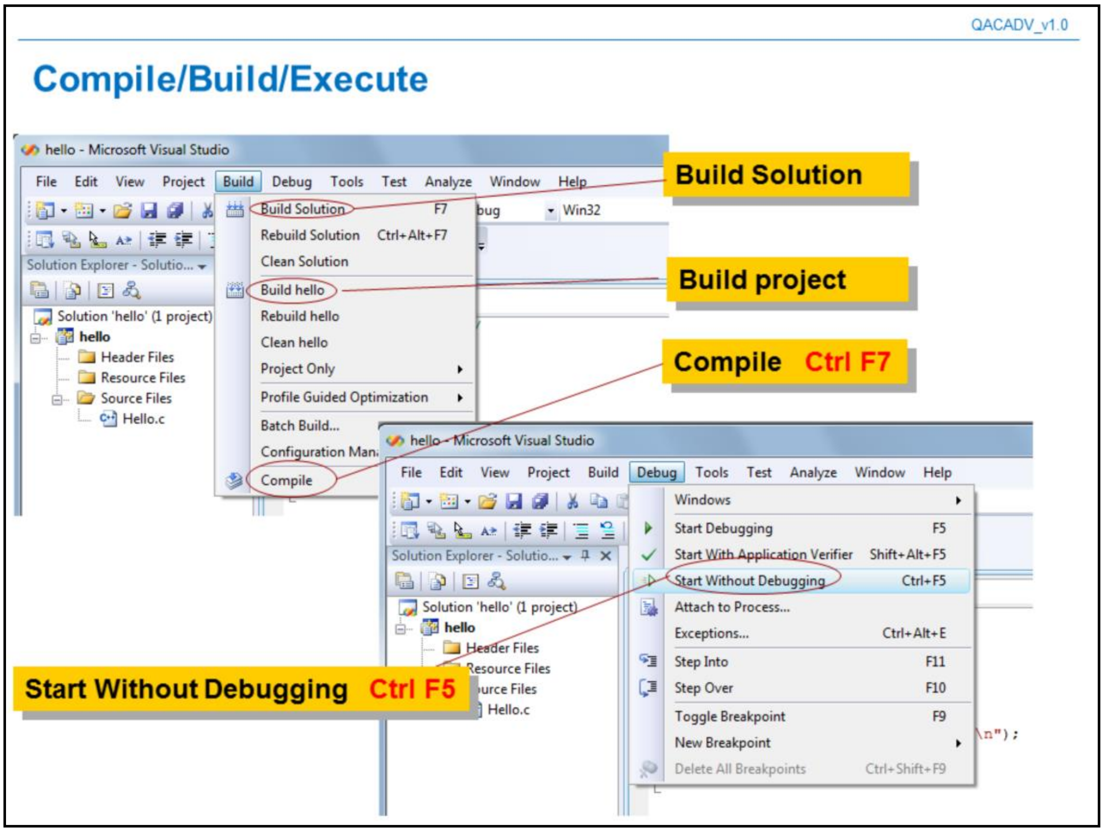
```
1>------ Build started: Project: hello, Configuration: Debug Win32 ------
1>Compiling...
1>Hello.c
1>Linking...
1>Embedding manifest...
1>Build log was saved at "file://c:\QA\C\LGCPRG1\LGCPRG1 v4.5\qacprg\CNOTES\Debug\BuildLog.htm"
1>hello - 0 error(s), 0 warning(s)
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

When the Solution initially opens there are several tabs on the right-hand side, the first of which is 'Solution Explorer'. This tab simply lists the files that are held in the opened Solution. These files are arranged in a hierarchy just like the folders in Windows Explorer. The top level headings are: Source Files, Header Files and Resource Files. These headings can be expanded and contracted by left-clicking on the plus sign (+) and minus sign (-) just like in InfoView.

Double left-clicking on a filename will open that file ready for editing. You can also right-click on a filename. This will bring up a shortcut menu with entries such as *Open* and *Compile*.

In C it is a good idea to use files in a modular way. A good rule of thumb is that each module should have two associated files, one `.h` file (the header file) and one `.c` file (the source file).

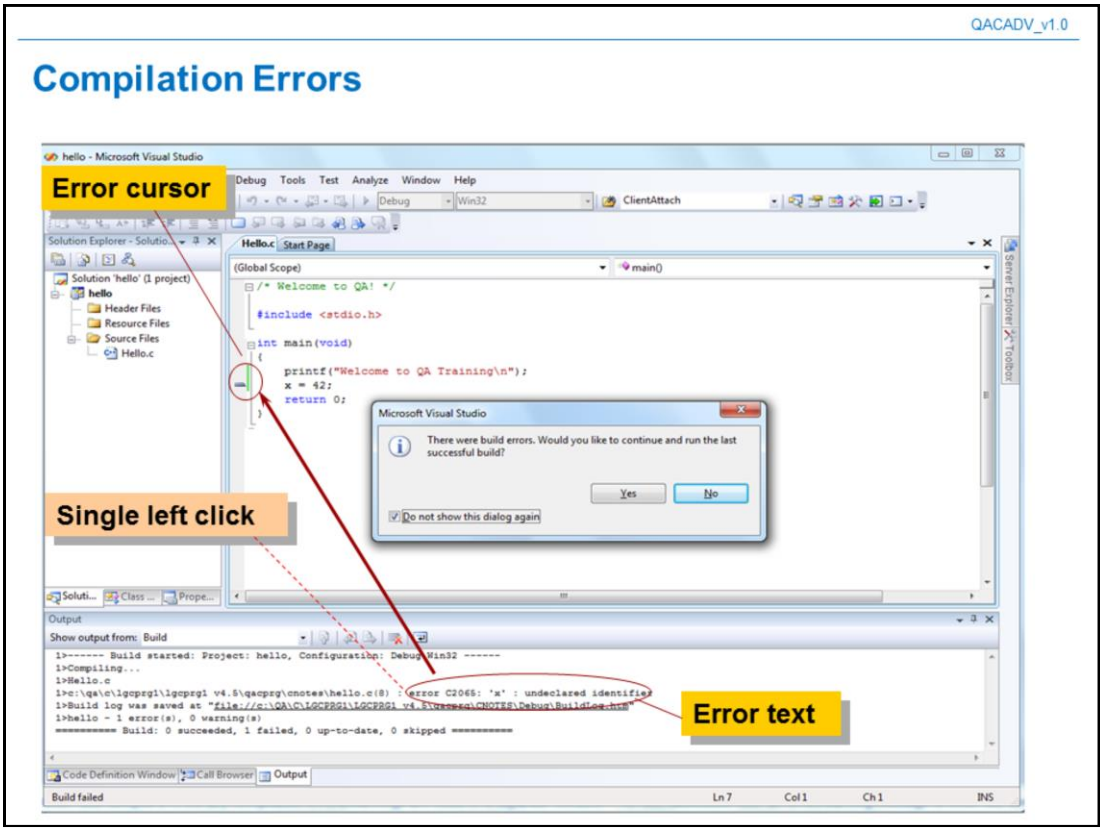
The file listing in Solution Explorer is an ordered listing of files in a project and does not necessarily reflect the physical organisation of the files on your hard disk.



To compile a `.c` file make sure its window is selected and hit Ctrl+F7, or select *Compile* from the *Build* menu.

To build an executable file (or a `.DLL`), select *Build* from the *Build* menu. During a build the environment compiles only those `.c` files whose associated `.obj` files are out of date, and then attempts to link. The *Rebuild* option ignores the date stamps of `.c` and associated `.obj` files and recompiles all `.c` files before attempting to link.

To execute/run the application hit Ctrl+F5, or select *Start without Debugging* from the *Debug* menu. You probably do not want to run the application using *Start* (F5) as this option does not pause the application after it has exited.


QACADV_v1.0

Compilation Errors

```
/* Welcome to QA! */

#include <stdio.h>

int main(void)
{
    printf("Welcome to QA Training\n");
    x = 42;
    return 0;
}
```

Microsoft Visual Studio

There were build errors. Would you like to continue and run the last successful build?

Yes No

[✓] Do not show this dialog again

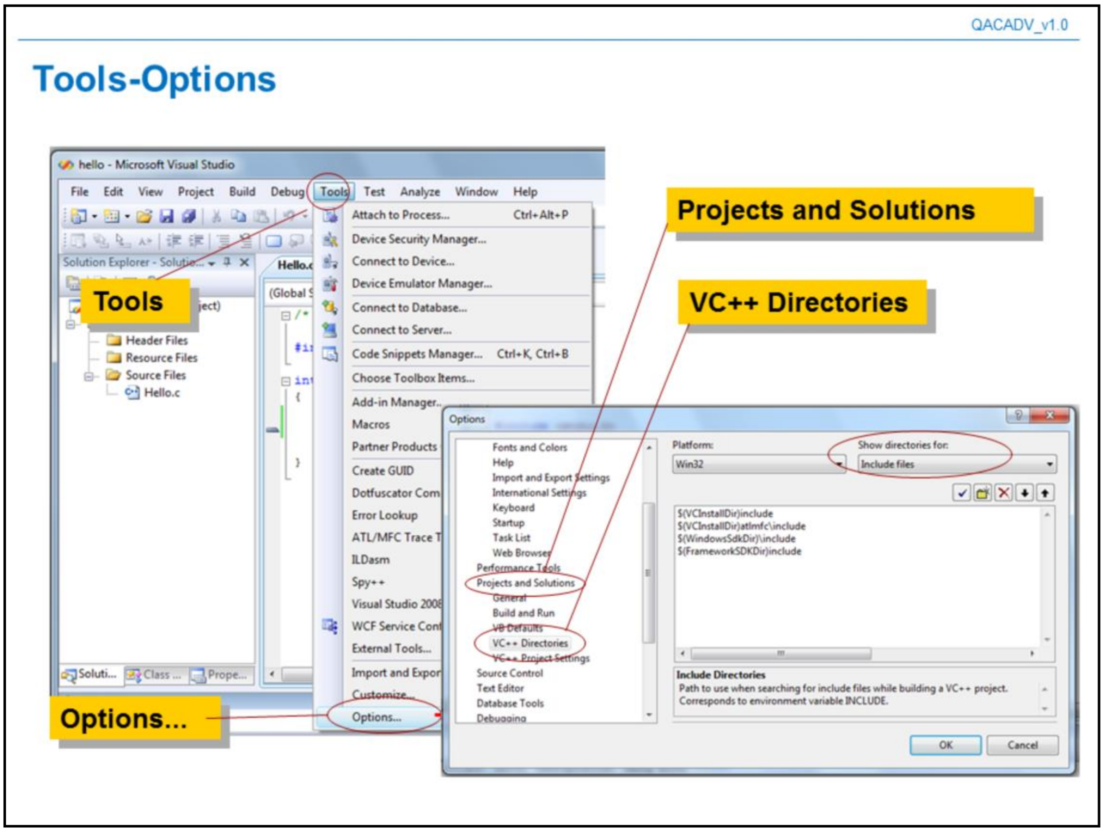
```
1>------ Build started: Project: hello, Configuration: Debug Win32 ------
1>Compiling...
1>Hello.c
1>c:\qa\c\lgcprg1\lgcprg1 v4.5\qacprg\cnotes\hello.c(8) : error C2065: 'x' : undeclared identifier
1>Build log was saved at "file://c:\QA\C\LGCPRG1\LGCPRG1 v4.5\qacprg\CNOTES\Debug\BuildLog.htm"
1>hello - 1 error(s), 0 warning(s)
========== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped ==========
```

When you compile a `.c` file any compilation errors/warnings are listed in a separate window. The text associated with an error/warning message is often too long to fit on screen. The scroll bar can help in this situation, but a really useful tip is to double left-click on the error text. This will move the error cursor to the offending line and display the error text in smaller font on the status bar.

Always try to correct the first error first. Subsequent errors often vanish after the first error is corrected.

If the error message contains the letters `LNK` then there is a link error. In other words you have promised to create something (in a declaration) and you have not honoured that promise (in a definition) and the linker is rightly complaining. Linker errors can be subtle!

Sometimes the build-link mechanism inside Visual Studio doesn't quite work. If you are really struggling with an error it is worth verifying that there is indeed an error in your code. To do this simply force a full rebuild by using the *Rebuild* option rather than the *Build* option.



$(VCInstallDir)include
$(VCInstallDir)atlmfc\include
$(WindowsSdkDir)\include
$(FrameworkSDKDir)include

Include Directories
Path to use when searching for include files while building a VC++ project.
Corresponds to environment variable INCLUDE.

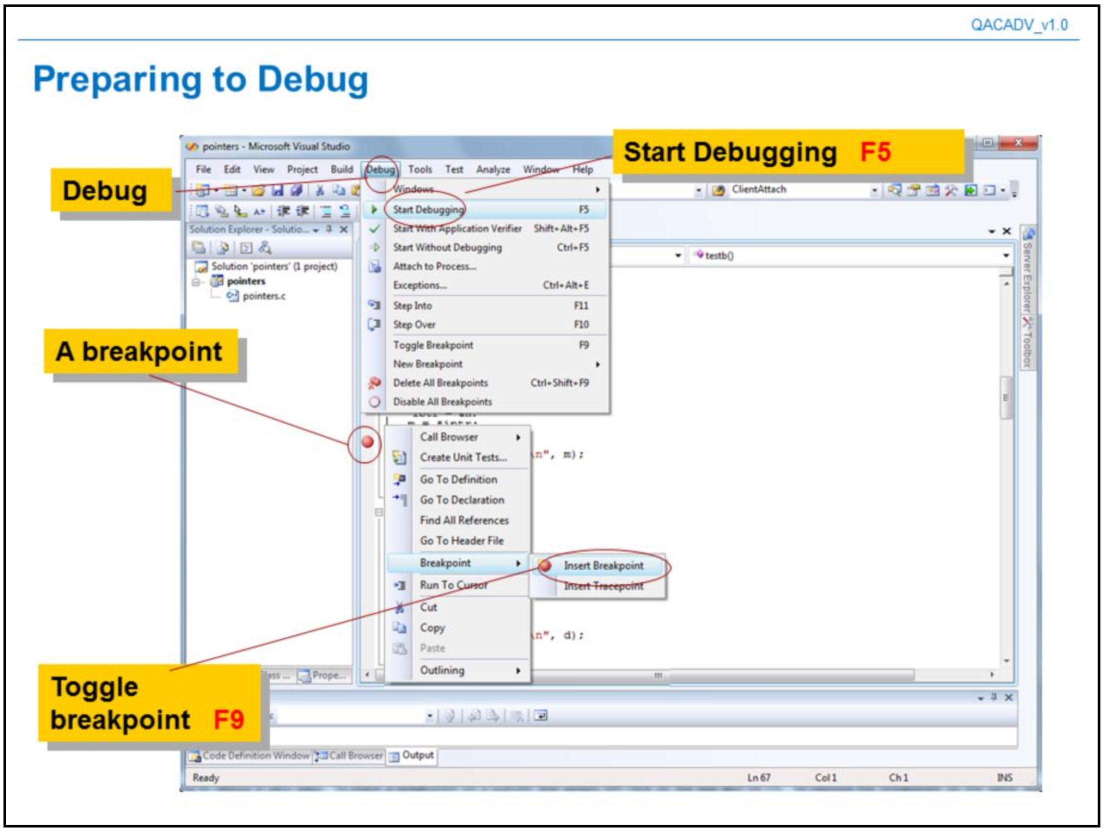
The *Tools* menu contains an entry labelled *Options…* A few of the most useful options that can be set from this menu entry are under the tabs labelled *Projects and Solutions / VC++ Directories* and *Text Editor / All Languages*.

Under *VC++ Directories* you can specify paths for various types of files. It contains a label on the right hand side called *Show Directories for*. The drop down menu under this label allows you to select the file type: *Executable files*, *Include files*, Reference files, *Library files* or *Source Files*. The *Include Files* option contains directories used by the preprocessor when expanding `#include` directives. The *Library Files* option contains directories used by the linker when searching for `.lib` files.

The number of spaces per tab is controlled in an option that can be found under *Text Editor / All Languages* (or *C/C++*) / *Tabs*. A common setting for this is four. A setting of eight or more is not recommended as this can quickly makes the code trundle off the right hand side of the window. Keeping the code closer to the left hand side of the window simply makes the code easier to read.

Environment / *Format and Colors* leads to various controls used to set colour syntax highlighting options.

Another dialog box worth knowing about can be found from the *Customize* entry also found under the *Tools* menu. One of the tabs is labelled *Toolbars* which allows you to control the visibility of individual toolbars.



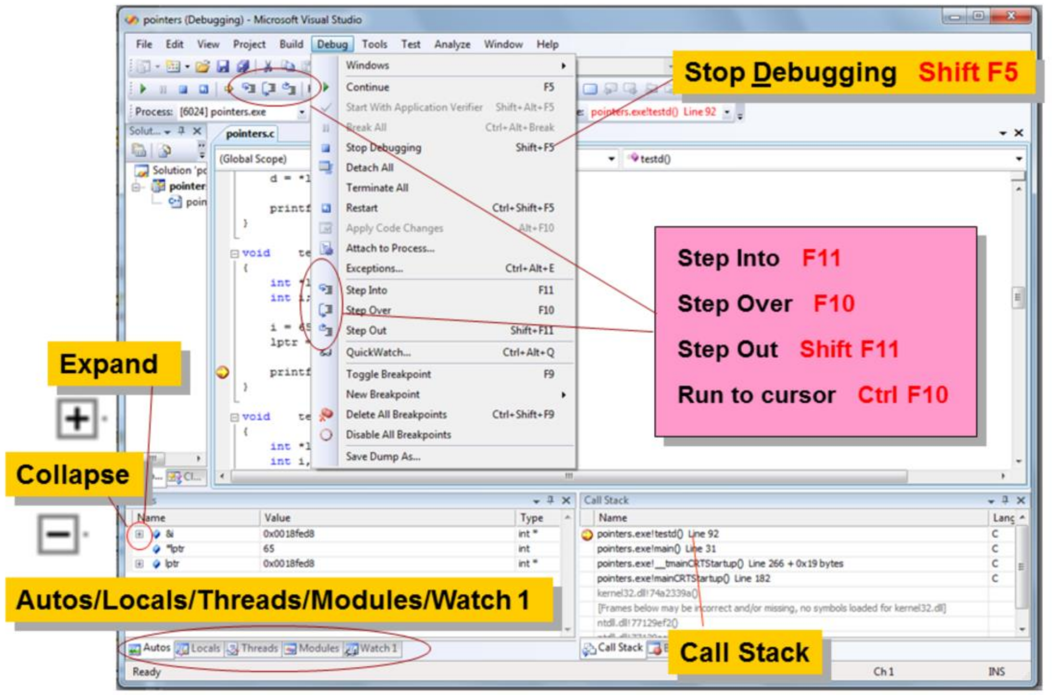
QACADV_v1.0

Preparing to Debug

Preparing to debug is very easy. To place a breakpoint on a specific line simply left-click on the line and hit F9. To remove the breakpoint do the same again. To run the program under the control of the debugger simply hit F5 rather than Ctrl+F5.

QACADV_v1.0

Inside the Debugger

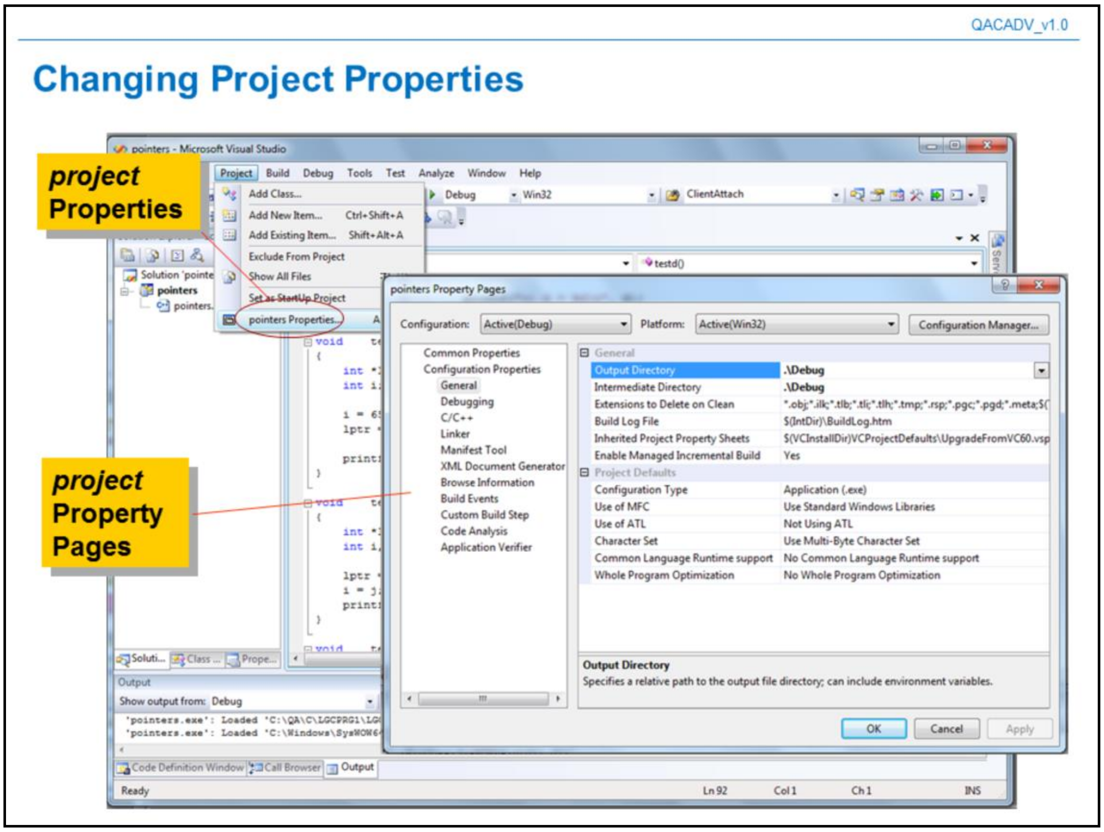


Variables are displayed using the familiar plus sign (+) and minus sign (-) symbols. A variable with a plus sign (+) is in collapsed form, and left-clicking on the plus sign (+) will expand the view. A variable with a minus sign (-) is in expanded form, left-clicking on the minus sign (-) will contract the view. The Variables window contains three tabs:

- The *Auto* tab displays information about variables used in the current statement and the previous statement
- The *Locals* tab displays information about variables that are local to the current function
- The *Watch 1* tab displays any expression and its value that you which to keep any eye on. Typically these could be globals, for example *errno*

The Debug toolbar contains several commands:

- *Step Into* single steps through instructions in the program and enters each function call that is encountered
- *Step Over* single steps through instructions in the program but function calls are executed without single stepping through the function instructions
- *Step Out* executes the program out of a function call and stops on the instruction immediately following the call to the function
- *Run to Cursor* executes the program as far as the line that contains the cursor



Changing Project Properties

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

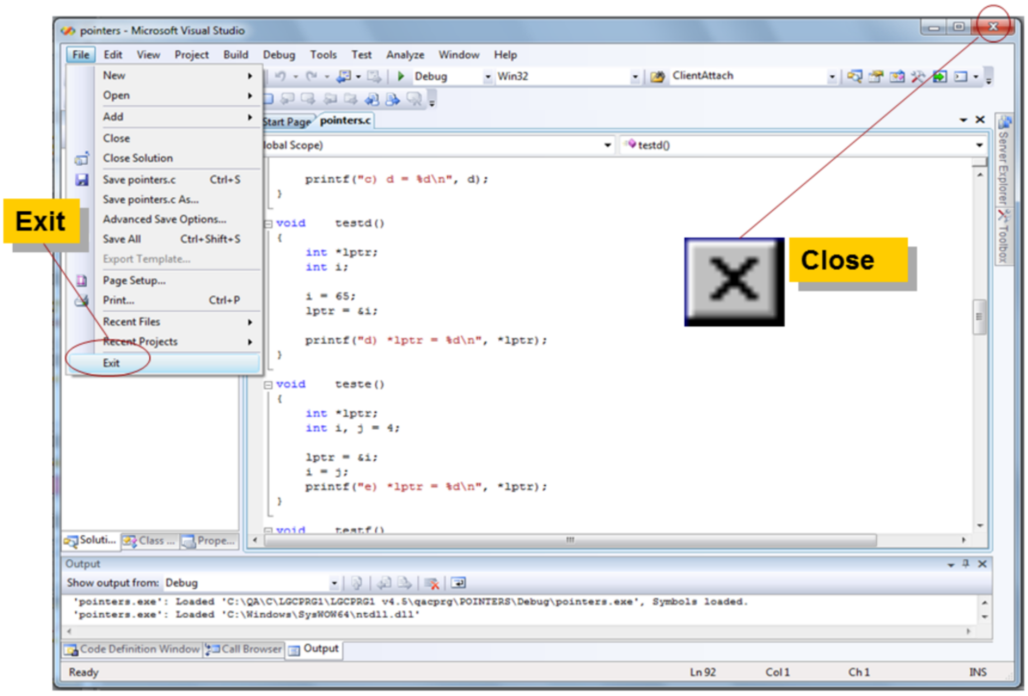
General	
Output Directory	.\Debug
Intermediate Directory	.\Debug
Extensions to Delete on Clean	*.obj;*.ilk;*.tlb;*.tli;*.tlh;*.tmp;*.rsp;*.pgc;*.pgd;*.meta;$(
Build Log File	$(IntDir)\BuildLog.htm
Inherited Project Property Sheets	$(VCInstallDir)VCProjectDefaults\UpgradeFromVC60.vsp
Enable Managed Incremental Build	Yes
Project Defaults	
Configuration Type	Application (.exe)
Use of MFC	Use Standard Windows Libraries
Use of ATL	Not Using ATL
Character Set	Use Multi-Byte Character Set
Common Language Runtime support	No Common Language Runtime support
Whole Program Optimization	No Whole Program Optimization

Output Directory
Specifies a relative path to the output file directory; can include environment variables.

The Project Property pages contain items that control the behaviour of the compiler and linker, such as optimisation options, and libraries to be linked. The Property pages are extensive, and require some practice in browsing through them. For starters, expand the *C/C++* 'folder'

QACADV_v1.0

Exiting



```
    printf("c) d = %d\n", d);
}

void    testd()
{
    int *lptr;
    int i;

    i = 65;
    lptr = &i;

    printf("d) *lptr = %d\n", *lptr);
}

void    teste()
{
    int *lptr;
    int i, j = 4;

    lptr = &i;
    i = j;
    printf("e) *lptr = %d\n", *lptr);
}

void    testf()
```

```
'pointers.exe': Loaded 'C:\QA\C\LGCPRG1\LGCPRG1 v4.5\qacprg\POINTERS\Debug\pointers.exe', Symbols loaded.
'pointers.exe': Loaded 'C:\Windows\SysWOW64\ntdll.dll'
```

To exit the Integrated Development Environment select *Exit* from the *File* menu or left-click on the close button at the top right of the window. If any files are unsaved you will be prompted to save them before shutdown.