Chapter 7: The Debugger

References:

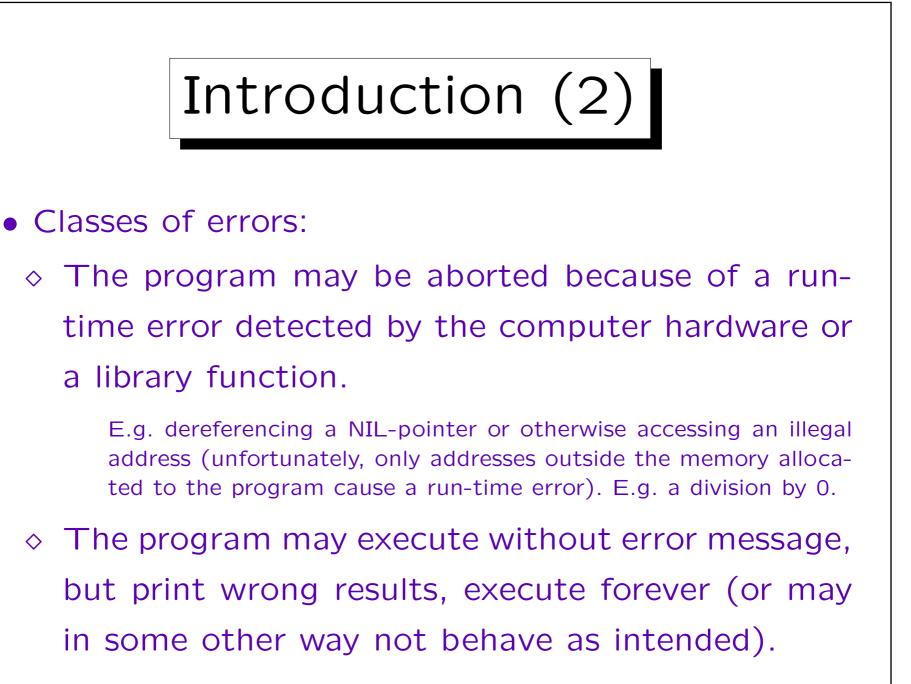
- Online Documentation of Microsoft Visual C++ 6.0 (Standard Edition): MSDN Library: Visual Studio 6.0 release.
- Peter Rechenberg, Gustav Pomberger (Eds.): Informatik-Handbuch (in German). Carl Hanser Verlag, 1997. Kapitel 12: Systemsoftware (H. Mössenböck).
- Brian W. Kernighan / Rob Pike: The Practice of Programming. Addison-Wesley, 1999, ISBN 0-201-61586-X.



- Even if a program runs through compiler/linker without error, it still may not work properly.
- Debugging is the process of finding such errors.

Errors in programs are also called "bugs". In the early days of computers, it might have been possible that small insects got inside the hardware and caused a malfunction. However, already Edison used the word "bug" for a problem in his phonograph (1889), and already at that times the insect was only imaginary.

 "Good programmers know that they spend as much time debugging as writing so they try to learn from their mistakes." [Kernighan/Pike]



Introduction (3)

• "Debugging is hard and can take long and unpredictable amounts of time, so the goal is to avoid having to do much of it. Techiques that help reduce debugging time include good design, good style, boundary condition tests, assertions and sanity checks in the code, defensive programming, well-designed interfaces, limited global data, and checking tools. An ounce of prevention really is worth a pound of cure." [Kernighan/Pike]



• Put assertions in your program (conditions that must be true and cause an error message if not).

E.g. every procedure should check its parameters.

 Procedures should not crash if called with wrong parameters ("Defensive Programming").

This is somewhat similar to assertions. E.g. some versions of printf print "(null)" if a string to be printed is NIL, others crash.

• Consider extreme cases.

This is what Kernighan/Pike call "boundary condition tests". E.g. if you develop a loop, consider the case that it is never executed. If you write data to an array, consider the case that the array is full.



• It might be useful to write procedures that dump data structures in readable form, echo the input read, etc. (debugging output).

If you put these procedures inside an **#ifdef**, you can easily activate them if something went wrong and you want to see a bit more what is going on internally.

• You should test each piece/module of the program after it is developed, not only the complete program at the end.

After every change, tests must be repeated. If might be possible to collect test input files and the corresponding output files and write a shellskript/batcg file that runs all the tests.

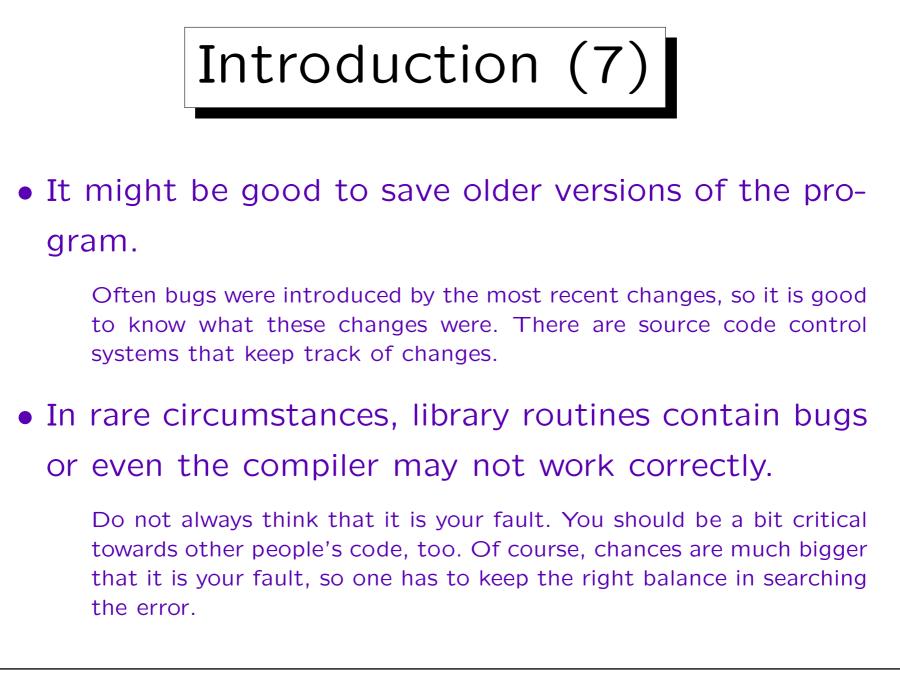


• If you detect a bug, immediately save the current version of the program (source files and binary), the input data, and any other information that might be needed to reproduce the unwanted behaviour.

If a bug cannot be reproduced, it is very difficult to find it.

• Really clarify the source of each error!

Sometimes it might be possible to change the program so that the bug no longer appears, but without really understanding why the program did behave in the way it did. From this you don't learn anything and chances are that the bug will reappear in some other form or that the changes you did created a new bug.



Debuggers (1)

- Debuggers are programs that support the task of debugging.
- One can use a debugger in two ways:
 - ◊ One can analyse the main memory after the program crashed ("Post mortem debugging").
 - One can run the program step by step under the control of the debugger ("dynamic debugger" / "tracing debugger").

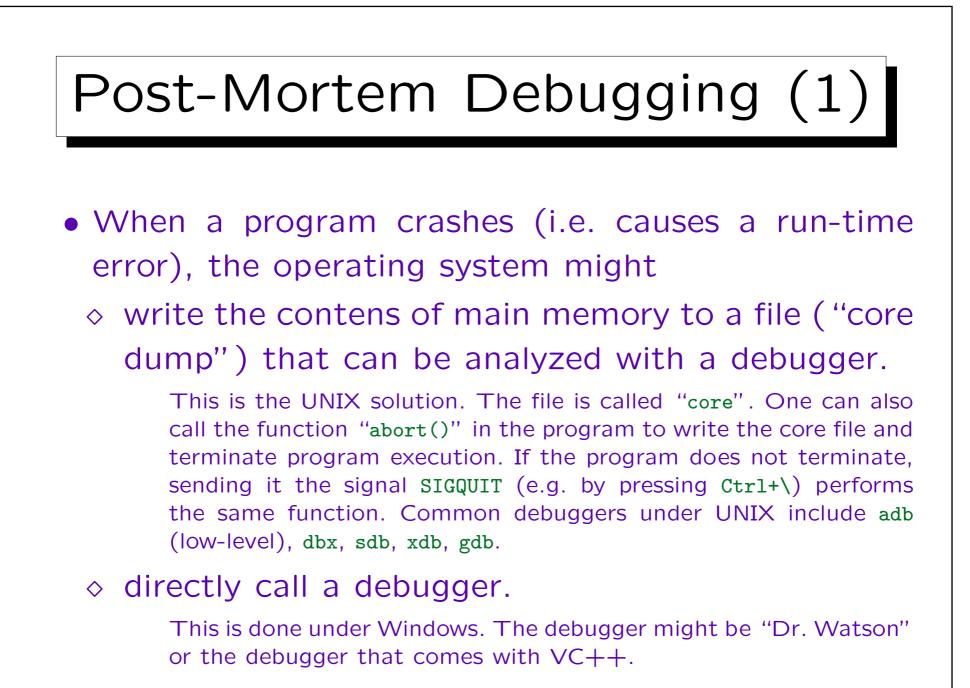


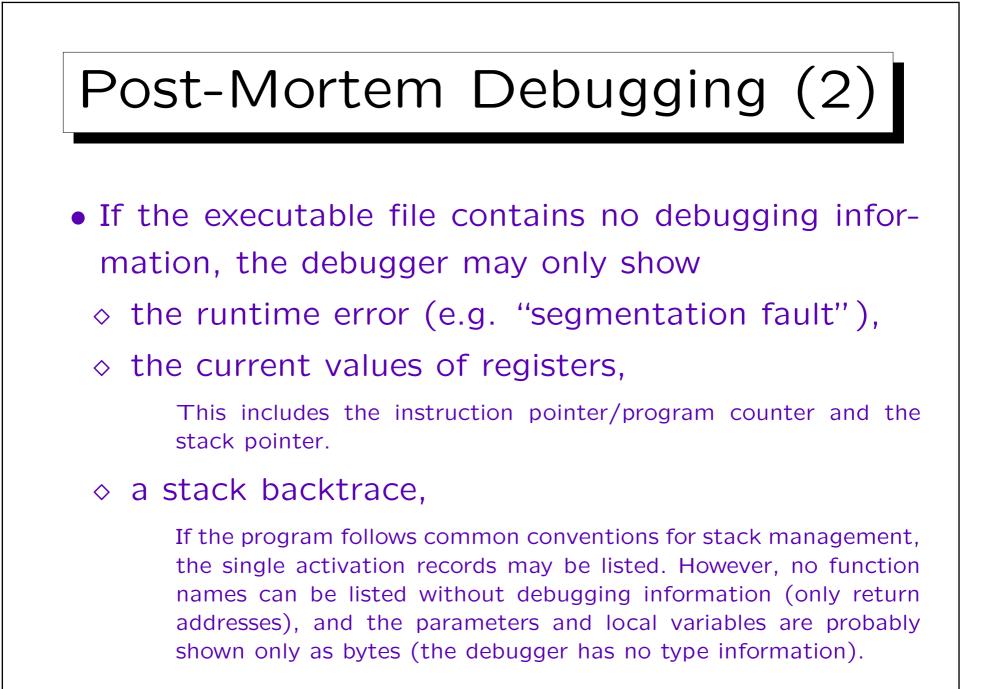
Whereas the compiler translates e.g. C into machine code, an important task of the debugger is to relate the main memory contents (machine code, data bytes) back to the original program.

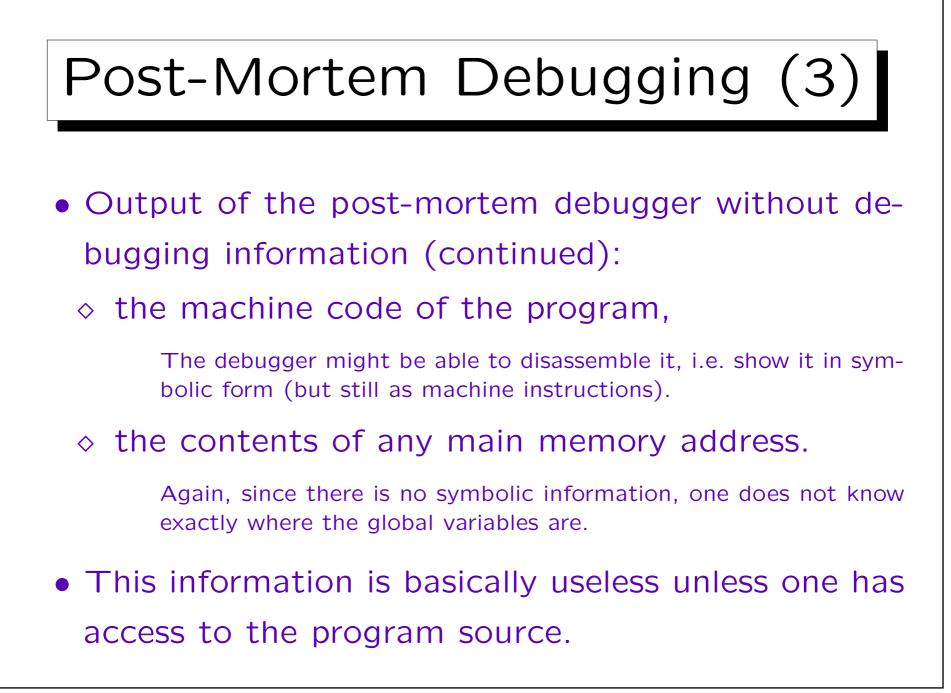
It needs the help of the compiler/linker for this task: They might make debugging information available.

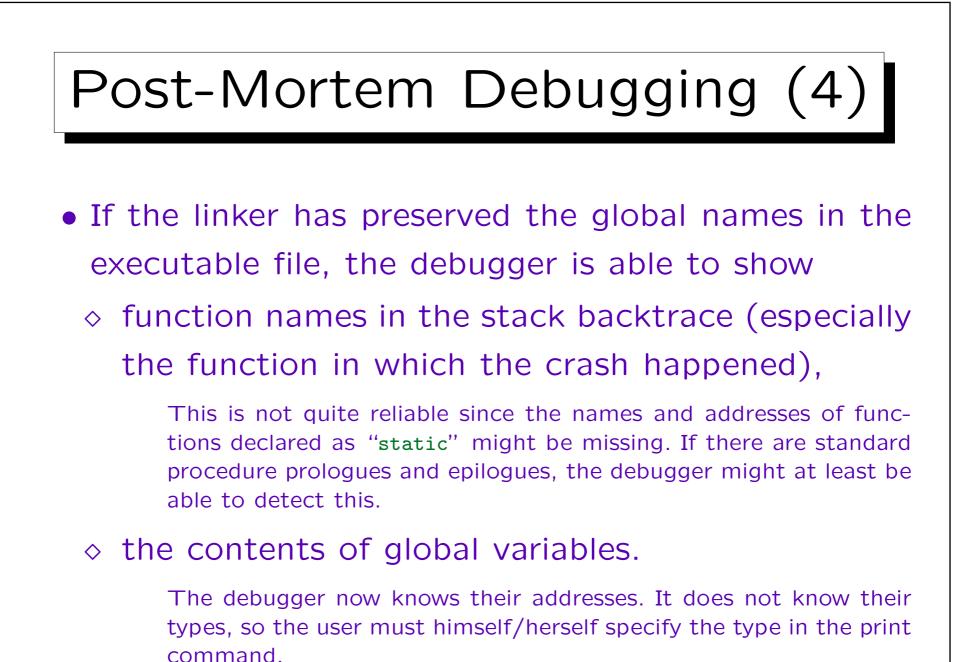
• Program development environments usually contain editor, compiler, linker, make, and a debugger.

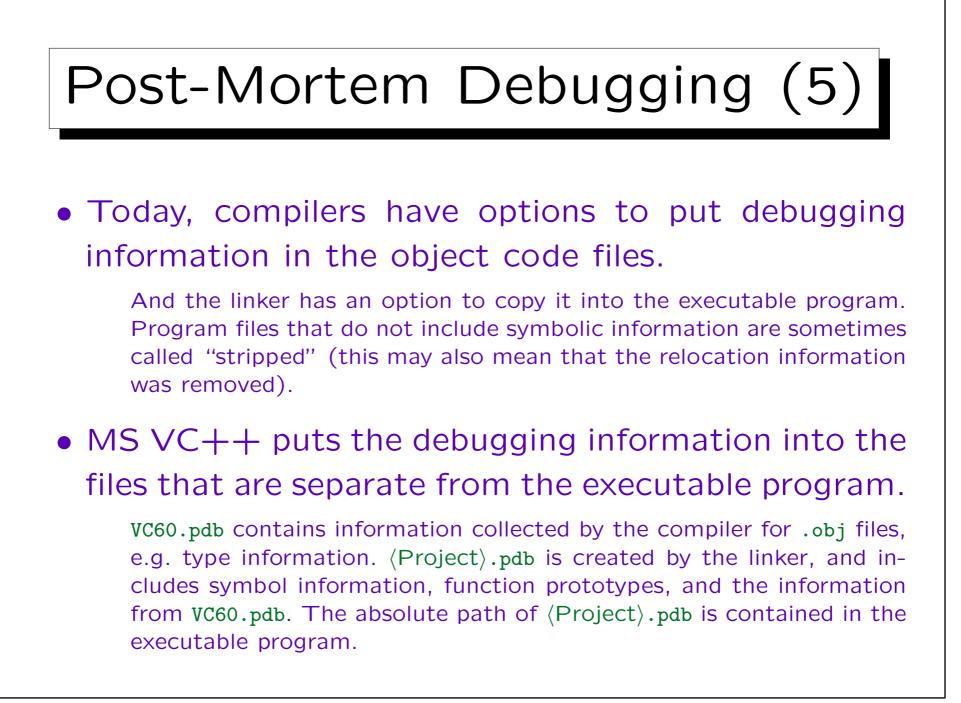
Then the debugging information might only be understandable by this debugger. However, there are also standards for debugging information (e.g. in COFF, the common object file format).

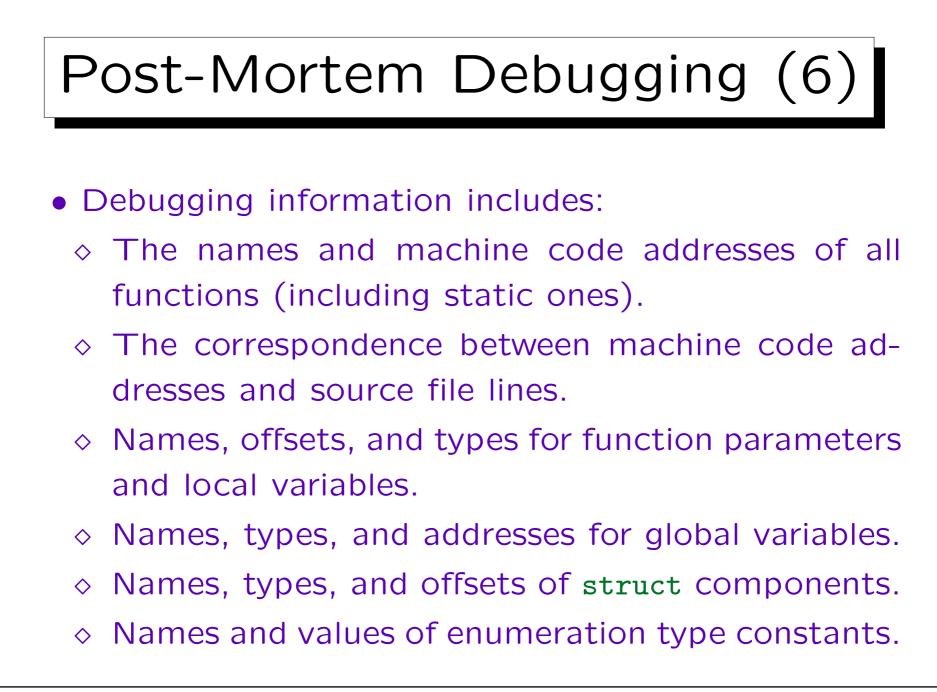




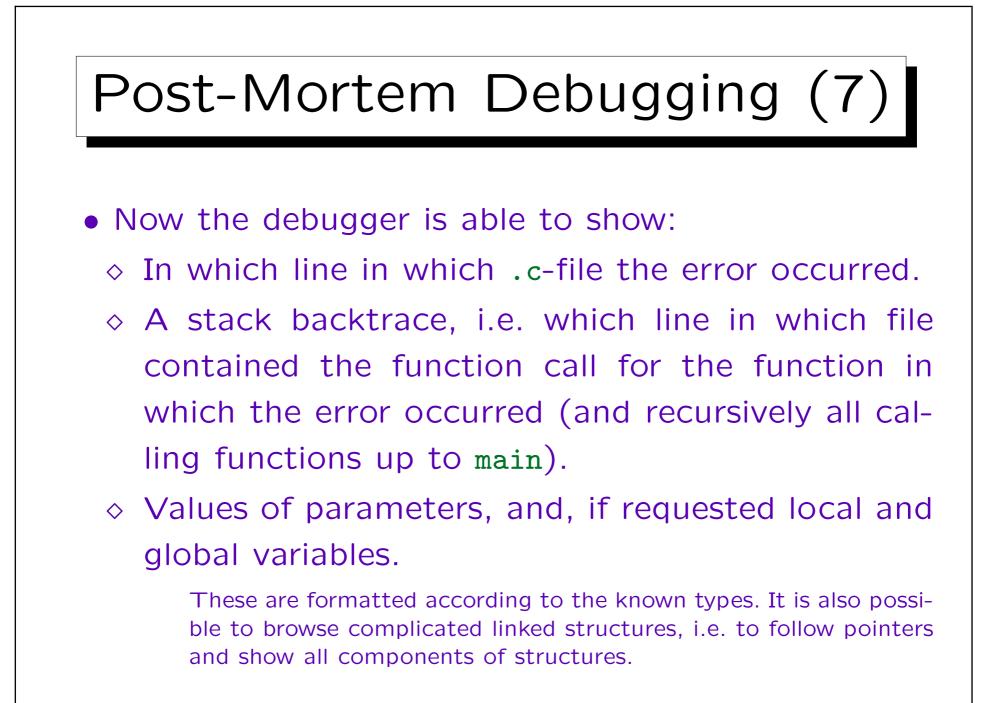


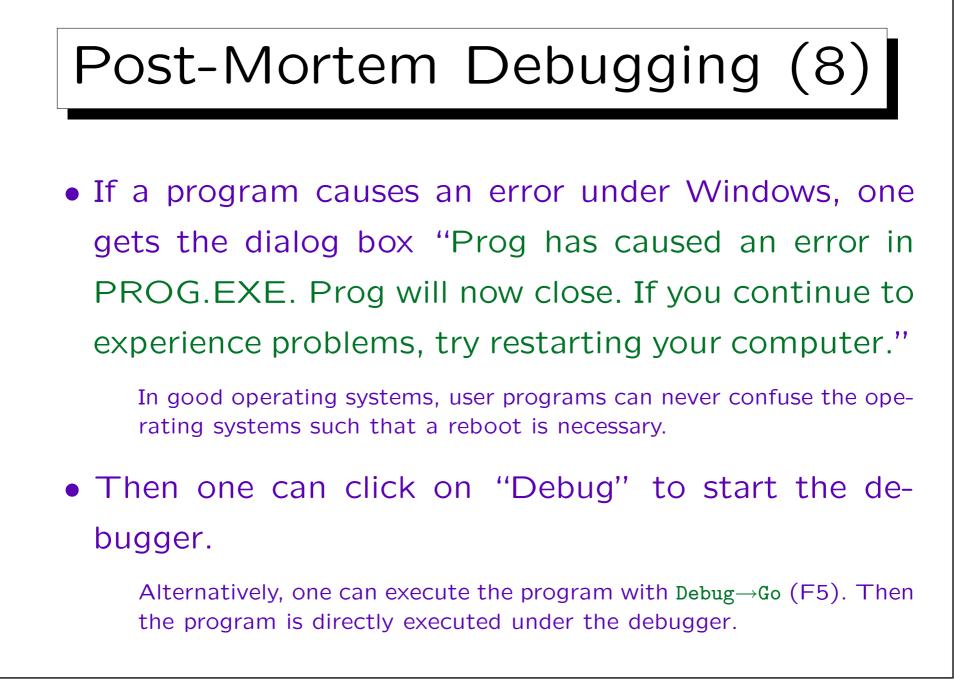


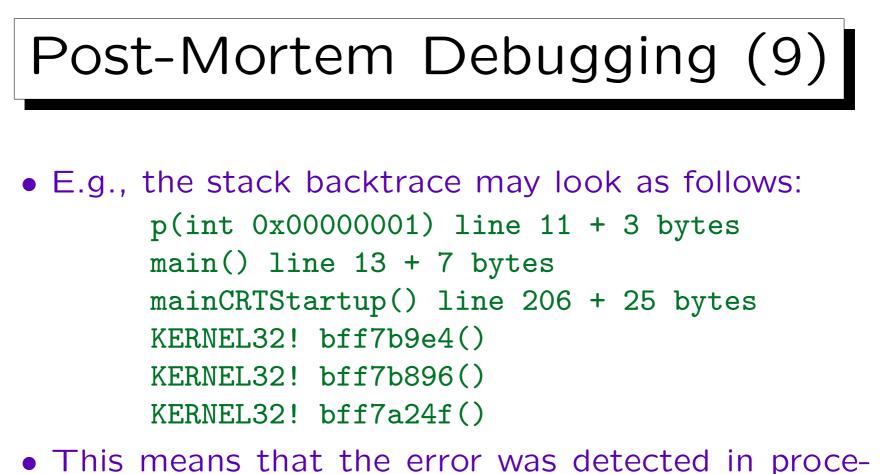




7. The Debugger

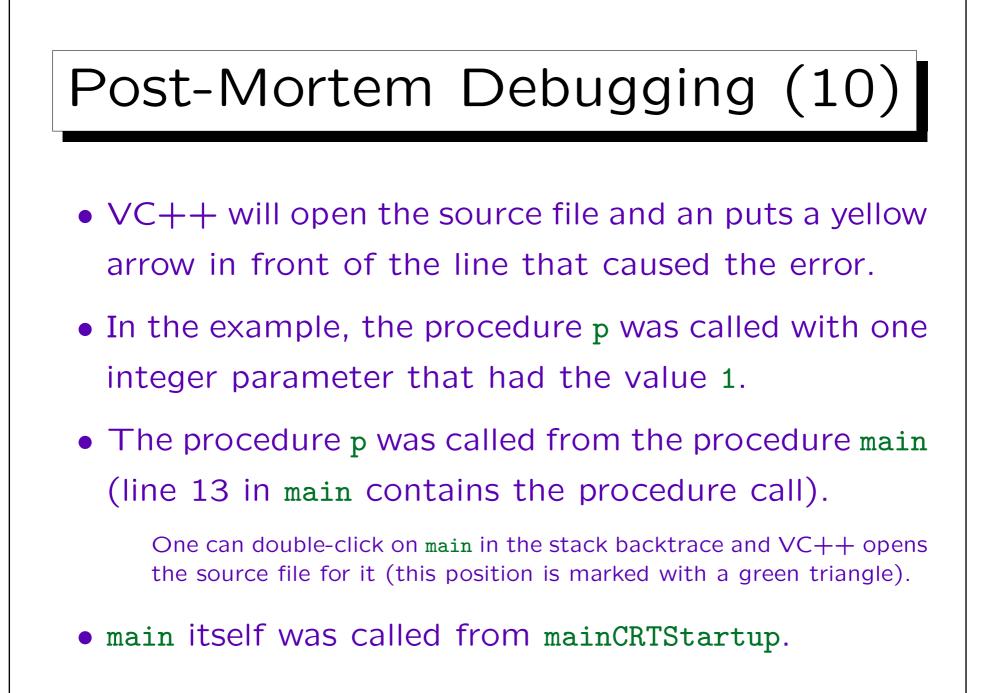


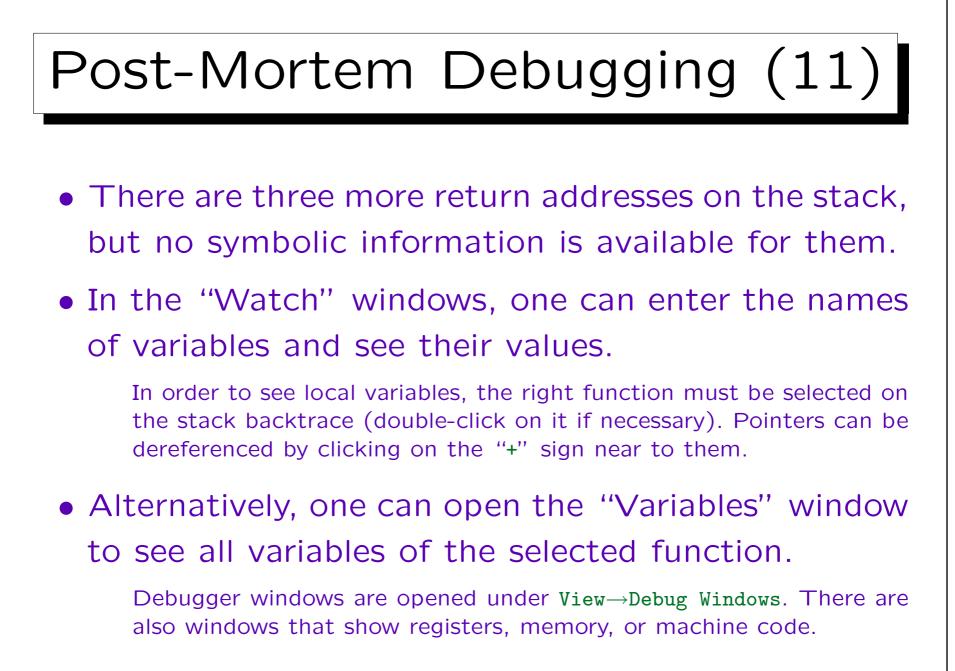




 This means that the error was detected in procedure p at line 11.

The line was compiled into several machine instructions and it probably happend in the second one. This explains the offset of 3 bytes.





Dynamic Debugger (1)

 A dynamic debugger (or tracing debugger) can execute the program step by step. In each step, it is possible to view variable values.

In MS VC++, one can start the program under the debugger with Build \rightarrow Start Debug \rightarrow Step Into. (the "Build" menu then changes to "Debug"). By repeatedly selecting this (or pressing F10) one can execute every line of the program. However, the debugger will also "step into" library procedures. If one wants to execute an entire procedure call without stopping, one selects "Step Over". If one wants to execute the rest of the current procedure and stop only at the next statement of the calling function, one selects "Step Out".



- One can also execute longer portions of the program without stopping, e.g. everything until the critical point.
- One does this by setting a breakpoint on the line where one wants execution to be temporarily suspended.

In MS VC++, one clicks with the right mouse button in front of the line where one wants the breakpoint to be set. The context menu contains "Insert/Remove Breakpoint". The program is started with Build \rightarrow Start Debug \rightarrow Go (then the Build menue changes to Debug, which also contains Stop Debug).

7. The Debugger

