Tools of the Trade

hannes@ru.is

Version Control (source control)

- Why Use Version Control Systems?
  - Shared repository
  - History of changes
  - Tagged versions
  - Branches possible

Version Control (source control)

- Locking vs. Automatic Merge
- Central vs. Distributed Repositories
- Individual Files vs. Logical Units
- Text vs. Binary Files
- Examples
  - CVS, Subversion, Git, Mercurial, Perforce, Alienbrain

Source Files

- Regular source files

  .c .cc .cxx .cpp

- Header files

  .h .hpp

- Source is Built through a series of processing steps: Preprocessing, Compliation and Linking

Step 1 Preprocessor

- Generates translation units from source files
  - These units get translated into machine code
- Preprocessor includes header files, replaces macros and follows special directives

Step 2 Compiler

- Translates each translation unit into machine code
- The machine code is placed in object files

  .o .obj
- External references have not been resolved
- Multiple object files can be stored in libraries

  .lib .a

Step 3 Linker

- Object files and libraries are linked into executables
- Relative memory addresses for functions and data are included
- Executables contain fully resolved machine code
  - Except for calls to dynamic link or shared libraries .dll .so
  - Linkers can create such „loadable“ libraries

Projects and Solutions (in VS)

- Projects are collections of source files which, when built, produce a library, an executable, or a DLL
- Solutions are collections of projects

Build Configurations

- A way to manage the multitude of command line options for the preprocessor, compiler and linker
- A set of command line options are stored under differently named build configurations
- Common configurations include options for Debug and Release builds
- Production and Tool configurations are also common

Debugger

- Allows you to inspect the execution of your program with breaks and stepping
- Break Points
  - Line break point, Data break point, Hardware break point, Conditional break point
- The Call Stack displays how you got to the function you are currently inspecting
- The Watch displays the contents of memory as well as results of evaluating debug expressions

Profilers

- Help you understand where execution time is being spent in order to optimize your code
- Statistical Profilers sample the CPU program counter register without tampering with your code
- Instrumenting Profilers use the preprocessing step to add code that generates logging information