MSVC Compiler Quirks

Visual Studio by default uses Microsoft's proprietary compiler MSVC, which often does not compile programs that work fine on Linux compilers such as `g++` or `clang++`. Therefore, assignments that require submitting source code should either find a way to make Visual Studio use an open-source compiler that can be tested on Linux without any change from teachers (if you do, please edit this page!), or make sure you test your code on MSVC!

Common causes

`windows.h` defines

Calls to functions whose names get clobbered by `#define`s in `windows.h` will likely break. Sometimes globally defining `WIN32_LEAN_AND_MEAN` and `NOMINMAX` helps, otherwise you will have to manually do this for all macros that affect your code:

```
#ifdef min
#undef min
#endif

#ifdef max
#undef max
#endif

// and so on...
```

`#include` extensions

MSVC allows backslashes in `#include` paths, which is nonstandard. You can pass `-fms-extensions` to clang to bypass this, but ideally `#include`s should be corrected.

MSVC also does not check the casing of filenames, as this is not relevant on Windows. When building on Linux however, this is very important. Make sure the path in the `#include` directive matches exactly that of the file, down to the casing!

Transitive includes

Some standard library headers may include other headers, such as `string` including `vector` implicitly. This is not defined by the standard (in fact, it is permitted!), so may break your code when switching compilers if you unknowingly depend on them.

The tool include-what-you-use can be used to solve this problem.

If using Clang, you can also reduce the number of these transitive includes by defining `_LIBCPP_REMOVE_TRANSITIVE_INCLUDES` (see the docs for more info).

C++ Standard extensions

Don't use extensions to the C++ standard, as these may not be available on MSVC. On modern CMake versions, you can set the target property `CXX_EXTENSIONS` to `OFF`.

C++ features

MSVC is often behind other compilers in implementing newer C++ features, so stick to older standard versions if you can. In older versions, MSVC usually has about the same support as GCC, and more than Clang, so you should be fine. You can see the current status on cppreference.

Solutions

MSVC on WINE

There are packages in the AUR that do this. It would be good to explore using them, and update this page!

Virtual machine

Install Visual Studio on a VM, and make sure the project builds and runs.

Friends

Ask a friend to check if it builds on their Windows machine!

CI/CD

It is possible to use CI/CD systems to check that a project builds. If it's not a graphical app, you can even test running it in the VM! An example setup for building .sln projects on push with GitHub Actions can be found here.

Do note that, while GitHub offers 2,000 minutes for free, using a Windows VM **counts as double minutes**! You only have 1,000 real minutes per month, which should still be largely enough.

Revision #4
Created 4 February 2024 23:57:47 by Mat
Updated 20 February 2024 16:06:46 by Mat