# CS324 Windows Compilation

Jamie Bayne

December 18, 2015

**Abstract**

This guide is intended for people taking the CS324 Computer Graphics course who wish to compile the labs or their coursework on a Windows machine.

## 1 Introduction

There are two main ways to develop C++ applications on Windows. In this guide, we're only going to look at one: shutting our eyes and pretending very hard that we're using a UNIX-like system. We will compile code with a Windows port of GCC called MINGW, which provides its own independent C++ runtime. MINGW runs in a POSIX *compatibility layer*, provided by a collection of programs called MSYS2.

The other way is to use Microsoft's own C++ runtime, which is distributed as part of the .NET framework, and will typically use their C++ compiler as shipped with the Visual Studio IDE and Visual C++. This is *heavily discouraged*, for two reasons: one, it is actually less convenient to set up than MINGW, with per-project settings buried in opaque Windows forms; and two, the final coursework requires you to submit a Makefile which compiles your code on the DCS machines, which will require additional porting effort on your part.

As a quick note, this guide includes code listings in which long lines are broken with an arrow symbol. It is important that you remember that these do not represent actual newline characters (or hitting an enter key) in the line:

```
This is actually one very long line, which will show up as two
 ↪  lines in the document
```

## 2 MSYS2

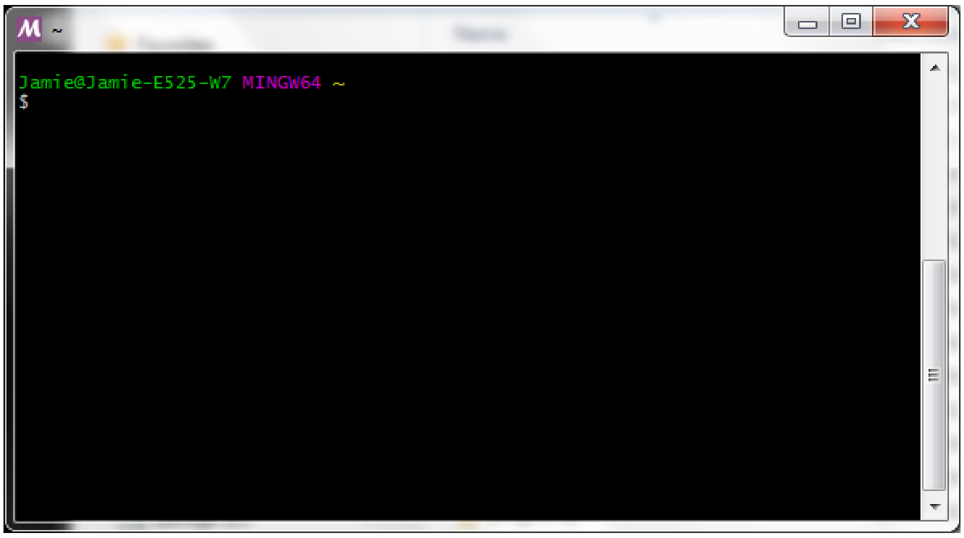
MSYS2 is a POSIX compatibility layer which provides a UNIX/Linux-like environment on Windows. This means that it gives you a command-line interface from which you can run specially-converted Linux programs, the most important of which is the MINGW C/C++ compiler.

This option doesn't permit the use of an IDE to compile your code (without some additional setup) but it does make setting up libraries quite simple.

- Download link and instructions: `http://msys2.github.io/`

This will give you three different shells (command line interfaces): 32 and 64bit MINGW shells, and a general purpose MSYS2 shell. For our purposes, you probably want to run the MINGW64 shell and forget about the others.

Opening it, you should see something like this:



```
Jamie@Jamie-E525-W7 MINGW64 ~
$
```

Figure 1: The MINGW64 Command Prompt

Now, we need to install some libraries and build-tools through the shell to get the CS324 labs working.

### 2.1 Pacman

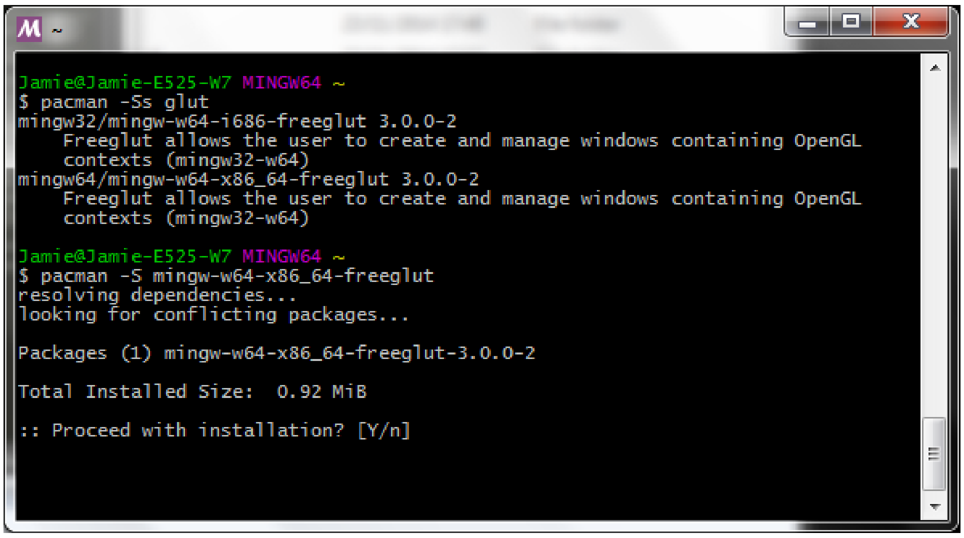
MSYS2 uses a package manager called `pacman`, which has some simple commands.

If you want to install something, *e.g.* GLUT, you can search the package repository using the -Ss flag:

```
pacman -Ss glut
```

In this case, you will see an entry for both MINGW32 and MINGW64. You want to install (-S) the version corresponding to your shell (MINGW64):

```
pacman -S mingw-w64-x86_64-freeglut
```



```
Jamie@Jamie-E525-W7 MINGW64 ~
$ pacman -Ss glut
mingw32/mingw-w64-i686-freeglut 3.0.0-2
    Freeglut allows the user to create and manage windows containing OpenGL
    contexts (mingw32-w64)
mingw64/mingw-w64-x86_64-freeglut 3.0.0-2
    Freeglut allows the user to create and manage windows containing OpenGL
    contexts (mingw32-w64)

Jamie@Jamie-E525-W7 MINGW64 ~
$ pacman -S mingw-w64-x86_64-freeglut
resolving dependencies...
looking for conflicting packages...

Packages (1) mingw-w64-x86_64-freeglut-3.0.0-2

Total Installed Size:  0.92 MiB

:: Proceed with installation? [Y/n]
```

Figure 2: Installing a package with Pacman

Over time, you may need to update the local package list and your installed packages. To do so, you can use the -Sy and -Su flags:

```
pacman -Sy    # update package list
pacman -Su    # update installed packages
pacman -Syu   # combine both operations
```

Finally, you may want to remove a package. To do this, use the -Rs flags:

```
pacman -Rs mingw-w64-x86_64-freeglut
```

## 2.2 Required Packages

You want to install the following programs with `pacman`:

```
base-devel
openssh # for copying files from joshua
unzip # for extracting the labs
mingw-w64-x86_64-gcc # gcc
```

and libraries

```
mingw-w64-x86_64-freeglut # freeglut
mingw-w64-x86_64-glew # glew
```

You can combine this into one `-S` command:

```
pacman -S base-devel openssh unzip
 ↪  mingw-w64-x86_64-{gcc,freeglut,glew}
```

There are a number of other packages you may find useful, such as `cmake`, `git`, `glm` and `zsh`. To read about a command, *e.g.* `gcc`, you can do `man gcc` just as in Linux.

# 3 Compiling the CS324 Labs

Once you have everything installed, the next task is to compile the labs. Whether or not you intend to do the labs on your computer, this is a good test to see if everything is set up correctly. First, download the labs from the DCS network and extract them

```
scp DCS_USERNAME@joshua.dcs.warwick.ac.uk:/modules/cs324/cs324-
 ↪  labs.zip
 ↪  .
unzip cs324-labs.zip
```

You can also use `scp` to copy files from your DCS home directory. For example, to transfer my personal copy of the labs, I can do

```
scp -r phulgm@joshua.dcs.warwick.ac.uk:~/cs324-labs my-cs324-labs
```

Now, you want to compile the labs. As MSYS2 emulates a UNIX-like system, you should compile with the `Makefile.linux` makefiles.

However, there are a few small changes required before this will compile. The OpenGL libraries used have slightly different names, and some libraries aren't available. You can correct this yourself, but to speed up the process I've written a small script to do it for you. Navigate to the `cs324-labs` directory

```
cd cs324-labs
```

and paste (`Shift+Insert`) this code directly into your MINGW64 terminal:

```
for f in */*.linux; do
    sed -i '
        s/lglut/lfreeglut/
        s/lGL/lopengl32/
        s/lGLU/lglu32/
        s/-lX11//
        s/lGLEW/lglew32/
        s|-L/modules/cs324/glew-1.11.0/lib||
        s|-L/usr/X11R6/lib||
        s|-I/modules/cs324/glew-1.11.0/include||
        /INCDIRS *= *$/d
        /LIBDIRS *= *$/d
        s/CPPFLAGS/CXXFLAGS/
        s/CXXFLAGS *=.*$/& -std=c++11/' $f
    rename .linux "" $f
done
```

The last line of the script renames `Makefile.linux` to `Makefile` for your convenience. Note that the instructions given in Lab 1 (Section 2.1 of the lab-script),

```
ln -fs Makefile.linux Makefile
```

won't work, as Windows does not understand the concept of soft links.

You should now be able to compile the labs according to the instructions in the lab script:

```
cd lab-1
make simple
make double
```

## 3.1 `drand48` and `srand48`

As of the current set of labs (2015), there is a problem with labs 8 and 10 because they use the UNIX methods `drand48` and `srand48`, which are not available on Windows. Attempting to compile the programs in these labs will result in the following error:

```
$ cd lab-8
$ make particles
g++ -O3 -std=c++11  -O3 -std=c++11 -std=c++11  particles.cpp
 ↪  -lfreeglut -lopengl32 -lglu32  -lm  -o particles
particles.cpp: In function 'void make_particles(size_t)':
particles.cpp:238:20: error: 'srand48' was not declared in this
 ↪  scope
  srand48(time(NULL));
                    ^
particles.cpp:246:32: error: 'drand48' was not declared in this
 ↪  scope
   float rand_r = float(drand48());
                                ^
<builtin>: recipe for target 'particles' failed
make: *** [particles] Error 1
```

To work around this, paste the following code after the `#include` blocks in the `.cpp` file (in this case, `lab-8/particles.cpp`):

```
#ifdef __WIN32__
#include <random>
void srand48(long) {};
double drand48() {
    static std::ranlux48 source(std::random_device{}());
    return std::uniform_real_distribution<double>(0,1)(source);
}
#endif
```

## 4 The CS324 Coursework

### 4.1 Building on Windows

For the coursework, you have to provide a Makefile to build your project. If you are unfamiliar with Makefiles, you can use the existing lab Makefiles a a skeleton. However, if you wish to split your code into many files, you can use this simple example as a skeleton instead (*be careful to preserve indentation*):

```
# the name of the executable created
PROGRAM_NAME = cs324_coursework

# Modify this variable as appropriate as you add .cpp files to
# your project
SRCS = main.cpp

# you shouldn't need to modify below here, but you can
# if you know what you're doing
CXXFLAGS= -O3 -std=c++11
LDFLAGS= $(CXXFLAGS) $(LIBDIRS) -std=c++11
LDLIBS = -lfreeglut -lopengl32 -lglu32 -lglew32 -lm
OBJS=$(SRCS:%.cpp=%.o)

default: $(PROGRAM_NAME)

$(PROGRAM_NAME): $(OBJS)
    $(CXX) $(LDFLAGS) $^ $(LDLIBS) -o $@

clean:
    -@rm $(OBJS) $(PROGRAM_NAME).exe

.PHONY: default clean
```

## 4.2 Converting for Linux

While you can develop on Windows, we require that your submission provides a Makefile which builds on Linux. If your Windows Makefile is called `Makefile.windows`, this script will create a file called `Makefile.linux`:

```
sed '
    s/lfreeglut/lglut/
    s/lopengl32/lGL/
    s/lglu32/lGLU/
    s/lglew32/lGLEW/
    s/LDLIBS *=.*$/& -lX11/
    s|LDFLAGS *=.*$|& -L/usr/X11R6/lib|
    s|LDFLAGS *=.*$|& -L/modules/cs324/glew-1.11.0/lib|
    s|LDFLAGS *=.*$|& -Wl,-rpath=/modules/cs324/glew-1.11.0/lib|
    s|CXXFLAGS *=.*$|& -I/modules/cs324/glew-1.11.0/include|
    s/\.exe *//
    ' Makefile.windows > Makefile.linux
```

To see if your coursework does indeed compile on Linux, you can transfer it to joshua and test it there:

```
scp -r my_coursework_folder
 ↪  DCS_USERNAME@joshua.dcs.warwick.ac.uk:~/remote_coursework
ssh DCS_USERNAME@joshua.dcs.warwick.ac.uk
cd remote_coursework
make -f Makefile.linux
```

## 5 Contact

If you have any problems, suggestions or corrections, please feel free to drop me an email at `j.bayne@warwick.ac.uk`.