Tutorial 1

NWEN 241
Systems Programming

Alvin C. Valera

alvin.valera@ecs.vuw.ac.nz

Reminder

- Week 2 Computer Sessions Sign-Up: https://ecs.victoria.ac.nz/Courses/NWEN241_2019T1/LabSignUp

Content

- Development environment and process
- Compilation process
- Standard C library
- Standard C++ library

Development Environment

- Lab: Systems and Network Lab (CO246)
  - ID access cards (Swipe Cards)
  - Should work if you are registered in NWEN 241

- PC workstations
  - Linux operating system, KDE as graphical user interface
  - Network file system: you can access your files from any of the PCs
  - Compilers & debuggers: gcc, g++, gdb, and more
  - Text editors: kate, gedit, emacs, vi, vim, and more

- Text editor vs IDE: Text editor

- Remote access: https://ecs.victoria.ac.nz/Support/TechNoteWorkingFromHome

Reference Compilers

The source code for your assignments will be compiled and marked using the compilers installed in the ECS lab computers and servers

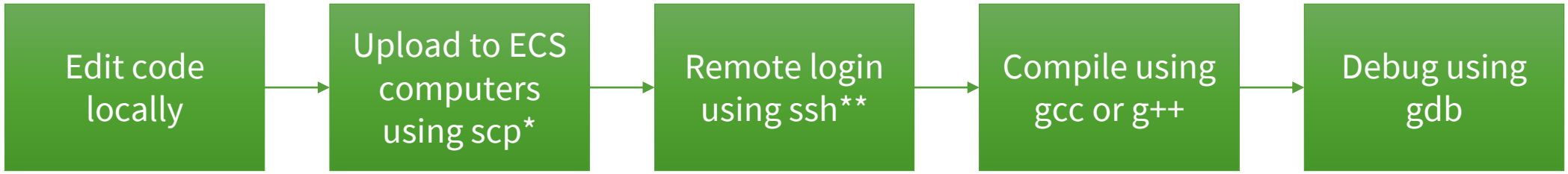
- C Compiler: **gcc version 8.2.1***
- C++ Compiler: **g++ version 8.2.1***

*As of 5 March 2019

Development Process

- You do not need to be in CO246 to write, compile and debug your code

- You can follow this remote development workflow:



* Winscp in Windows

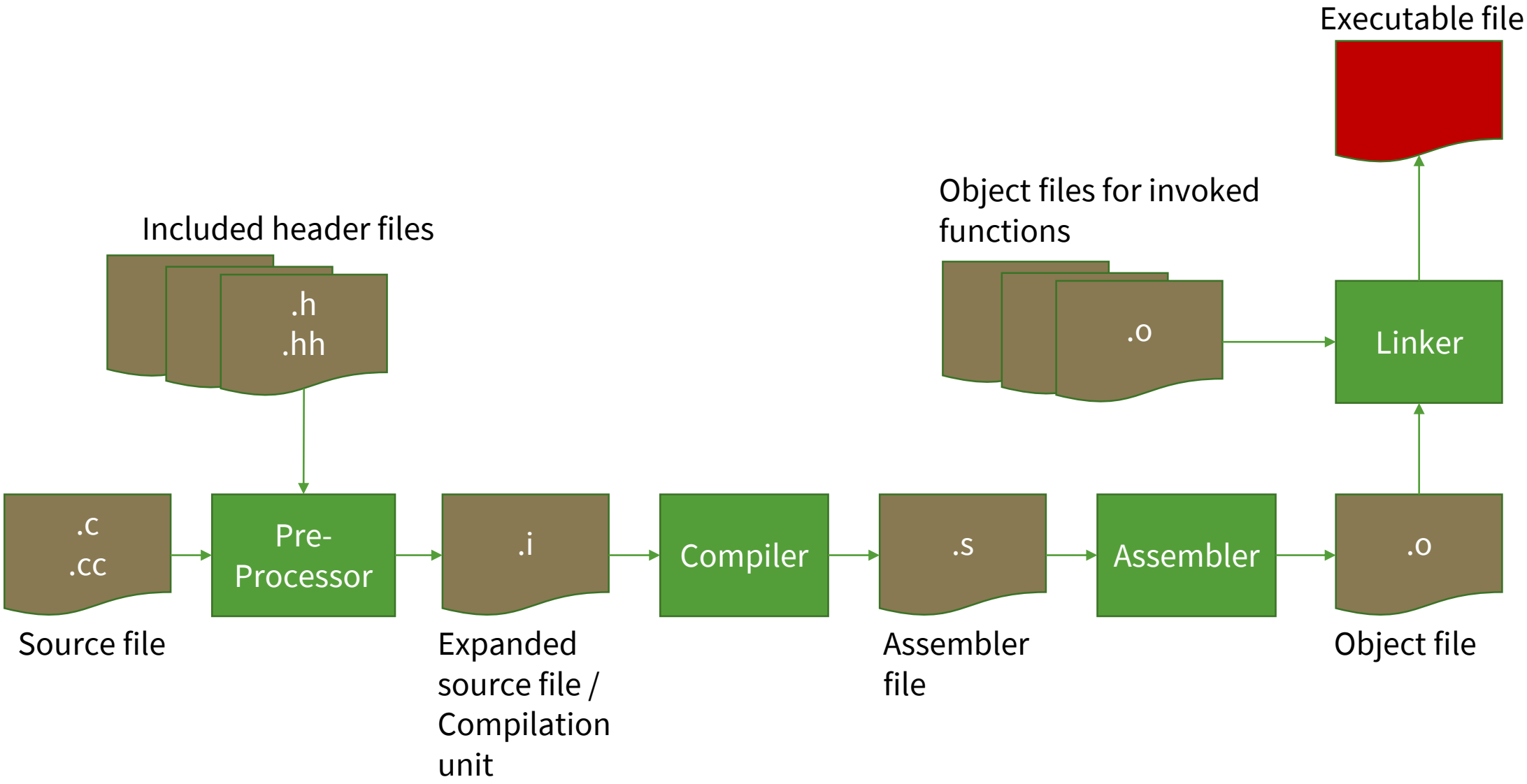
** Putty in Windows

ECS Computers for Working Remotely

Servers that you can login into to work remotely:

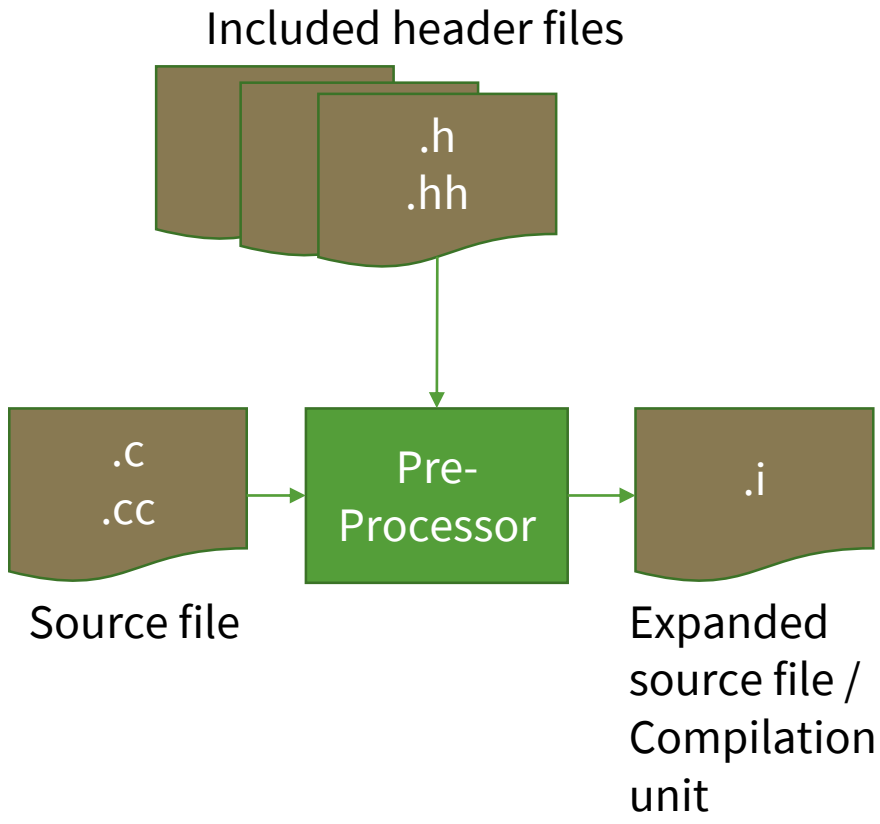
- `barretts.ecs.vuw.ac.nz`
- `embassy.ecs.vuw.ac.nz`
- `greta-pt.ecs.vuw.ac.nz`
- `regent.ecs.vuw.ac.nz`

Compilation Process



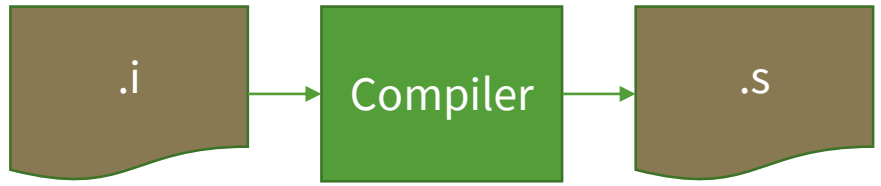
Preprocessing Phase

- The **preprocessor** modifies the original C/C++ program according to directives that begin with the '#' character
  - Example: #include <stdio.h> command tells the preprocessor to read the contents of the system header file **stdio.h** and insert it directly into the program text.
- The result is another C/C++ program, typically with the **.i** suffix.



Compilation Phase

- The **compiler** translates the text file (**.i**) into the text file (**.s**), which contains an assembly-language program.



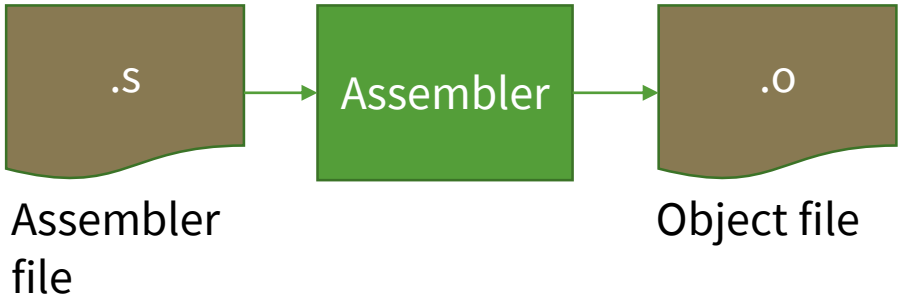
Expanded source file / Compilation unit

Assembler file

Assembly Phase

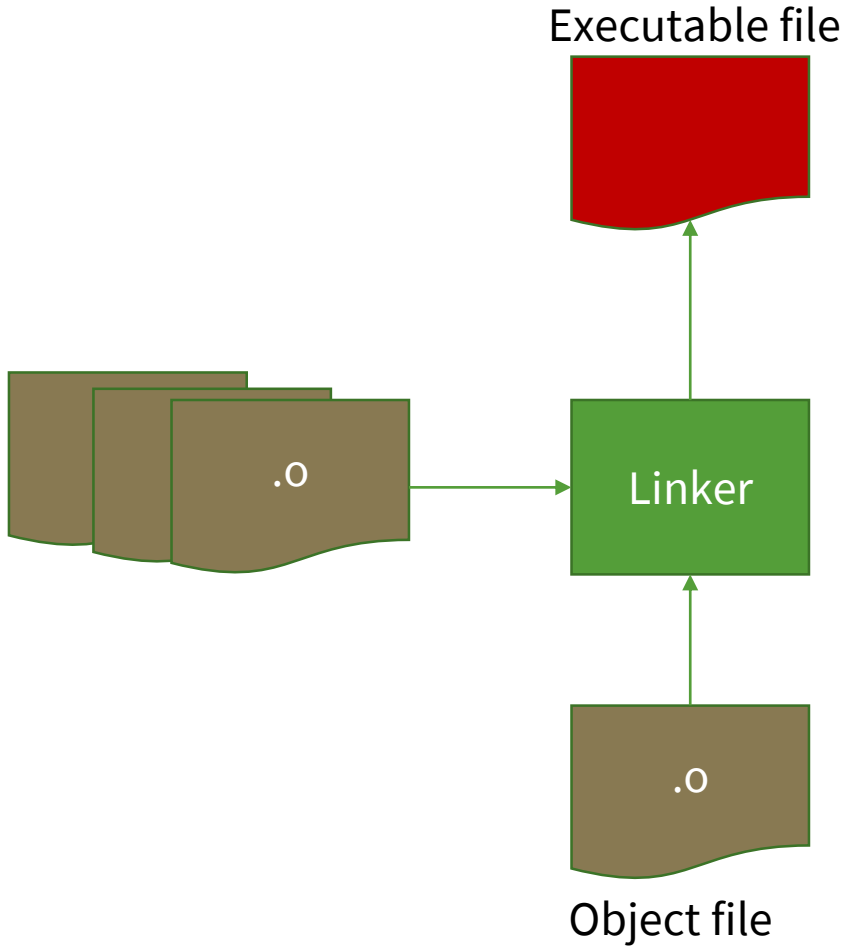
- The **assembler** translates assembler file (**.s**) into machine-language instructions, packages them in a form known as a *relocatable object program*, and stores the result in the *object file* (**.o**).

- Object files are binary - if you try to open one with a text editor, it would appear to be gibberish.



Linking Phase

- The **linker** looks for external object files needed by the program and merges these with the object file generated in the assembly phase, creating an executable object file (or simply *executable*) that is ready to be loaded into memory and executed by the system.



Compilation Process in Action

Using the GNU C/C++ Compilers

Programs

```c
/* Program to calculate the area of a circle */
#include <stdio.h>
#define PI 3.14

float sq(float);

int main(void)
{
   float radius, area;

   /* Ask user to input */
   printf("Radius = ");
   scanf("%f", &radius);

   area = PI * sq(radius);
   printf("Area = %f\n", area);
   return 0;
}

float sq(float r)
{
   return (r * r);
}
```

circle.c

```cpp
/* Program to calculate the area of a circle */
#include <iostream>
#define PI 3.14

float sq(float);

int main(void)
{
   float radius, area;

   /* Ask user to input */
   std::cout << "Radius = ";
   std::cin >> radius;

   area = PI * sq(radius);
   std::cout << "Area = " << area << "\n";
   return 0;
}

float sq(float r)
{
   return (r * r);
}
```

circle.cc

GNU C Compiler (gcc)

- gcc does:
  - pre-processing,
  - compilation,
  - assembly, and
  - linking
- Normally all done together, but you can get gcc to stop after each stage

```
gcc circle.c
```

Generates executable file **`a.out`**

```
gcc circle.c -o circle
```

Generates executable file **`circle`**

GNU C++ Compiler (g++)

- g++ does:
  - preprocessing,
  - compilation,
  - assembly, and
  - linking
- Normally all done together, but you can get g++ to stop after each stage

```
g++ circle.cc
```

Generates executable file **`a.out`**

```
g++ circle.cc -o circle
```

Generates executable file **`circle`**

Preprocessing

- Preprocessor directives begin with a #
  - macro substitution
  - inclusion of named files, and
  - conditional inclusion

```
#define PI 3.14
```

PI will be replaced by **3.14**

```
#include <stdio.h>
```

Contents of file **stdio.h** will be copied

Preprocessing

- To make gcc/g++ stop after preprocessing, use –E

```
gcc -E circle.c
```

Output goes to standard output

```
gcc -E circle.c -o circle.i
```

Generates **`circle.i`**

```
g++ -E circle.cc
```

Output goes to standard output

```
g++ -E circle.cc -o circle.i
```

Generates **`circle.i`**

Compilation

- Output from this stage is *assembly (*or *assembler) code* (symbolic representation of numeric machine code)
- To make gcc/g++ stop after compilation, use -S

```
gcc -S circle.i
```

Generates **`circle.s`**

```
gcc -S circle.c -o circleC.s
```

- Generates **`circleC.s`**
- .c and .i files become .s files

```
g++ -S circle.i
```

Generates **`circle.s`**

```
g++ -S circle.cc -o circleCC.s
```

- Generates **`circleCC.s`**
- .c and .i files become .s files

Assembly

- Output from this stage is *object code*
- To make gcc/g++ stop after assembly, use -c

```
gcc -c circle.s
```

Generates object file **`circle.o`**

```
gcc -c circle.c -o circleC.o
```

- Generates object file **`circleC.o`**
- .c, .i and .s files become .o files

```
g++ -c circle.s
```

Generates object file **`circle.o`**

```
g++ -c circle.cc -o circleCC.o
```

- Generates object file **`circleCC.o`**
- .c, .i and .s files become .o files

Linking

- Bring together multiple pieces of object code and arrange them into one executable

```
gcc circle.o
```

Generates executable file **`a.out`**

```
gcc circle.o -o circle
```

Generates executable file **`circle`**

```
g++ circle.o
```

Generates executable file **`a.out`**

```
g++ circle.o -o circle
```

Generates executable file **`circle`**

Running

- To run the executable file

```
./a.out
```

```
./circle
```

Source Code in Multiple Files

- A C/C++ source code usually consists of more than 1 source files
- Using an editor of your choice, modify `circle.c` by removing the function definition of `sq()`
- Create another source file `sq.c` that only contains the function definition of `sq()`
- Save the remaining part as `main.c`

Source Code in Multiple Files

```
/* main.c: Program to calculate the area of a circle */

#include <stdio.h>                /* library file access */
#define PI 3.1415926              /* macro definition - symbolic constant */

float sq(float);                  /* square function - function prototype */

int main(void)                    /* function heading  */
{
   float radius, area;            /* variable declarations */

   printf("Radius = ");           /* output statement (prompt) */
   scanf("%f", &radius);          /* input statement */

   area = PI * sq(radius);        /* use square function */
   printf("Area = %f\n", area);   /* output statement */
   return 0;                      /* return statement */
}
```

```
/* sq.c: Function to square a number */

float sq(float r)
{  return (r * r);}               /* square function - function definition*/
```

Source Code in Multiple Files

- Generate executable file `circle` in 1 step:

```
gcc main.c sq.c –o circle
```

- In general, the compilation process for a multi-file source code involves
  - Generating the object file for each source file
  - Merging all the generated object files, as well external object files used by the program

Standard C/C++ Libraries

Libraries to Use in Assignments

- **Standard C Library** – for writing **Pure C** code

- **Standard C++ Library** – for writing **C++** code

Standard C Library

C provides a standard library* which consists of the following headers:

`assert.h`	`float.h`	`math.h`	`stdarg.h`	`stdlib.h`
`ctype.h`	`limits.h`	`setjmp.h`	`stddef.h`	`string.h`
`errno.h`	`locale.h`	`signal.h`	`stdio.h`	`time.h`

*C99 and C11 standards added more header files.

- To know more about the C standard library, visit
  `https://www.tutorialspoint.com/c_standard_library/index.htm`

Standard C++ Library

- Consists of too many header files to list here!
- See https://en.cppreference.com/w/cpp/header for details
- Header files from the standard C library are part of standard, but they are referred by a different name: c is prepended and .h is dropped from the file name
  - Example: `stdio.h` becomes `cstdio`