C Syntax

What the Compiler needs to understand your program

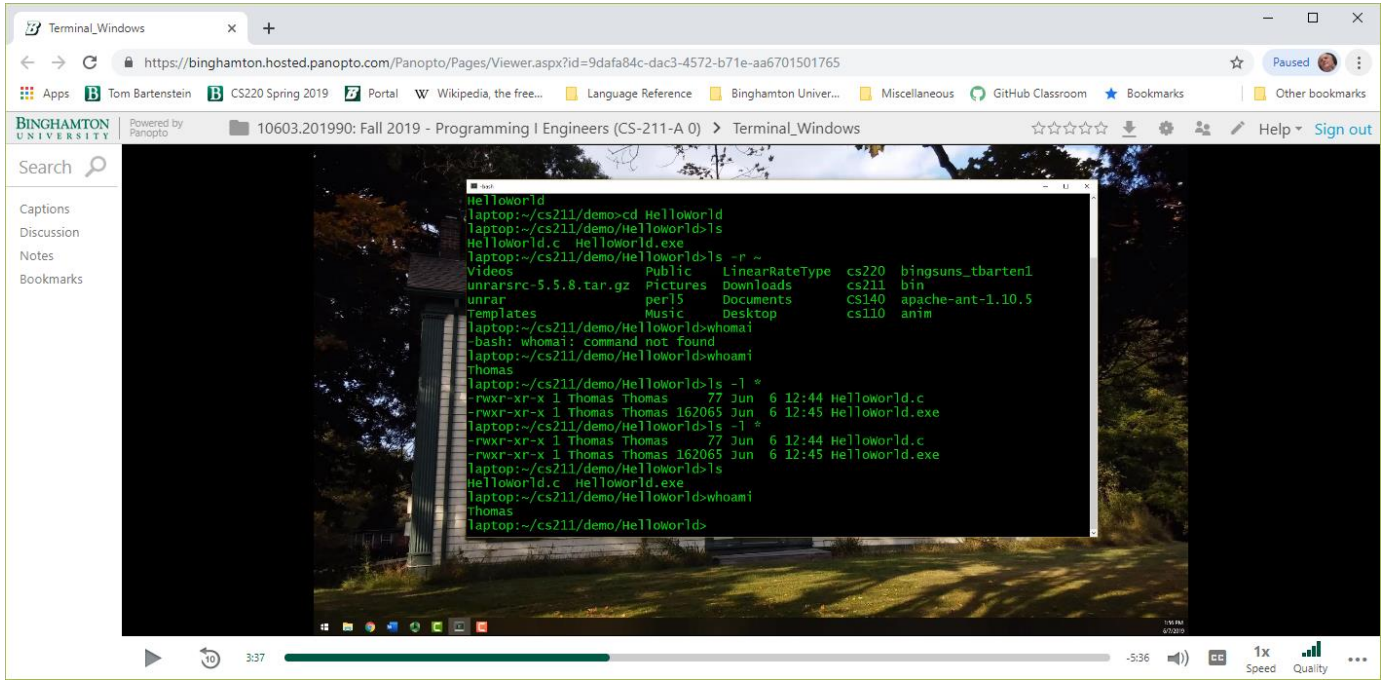
iClicker Attendance

Please click on A if you are here:

A. I am here today.

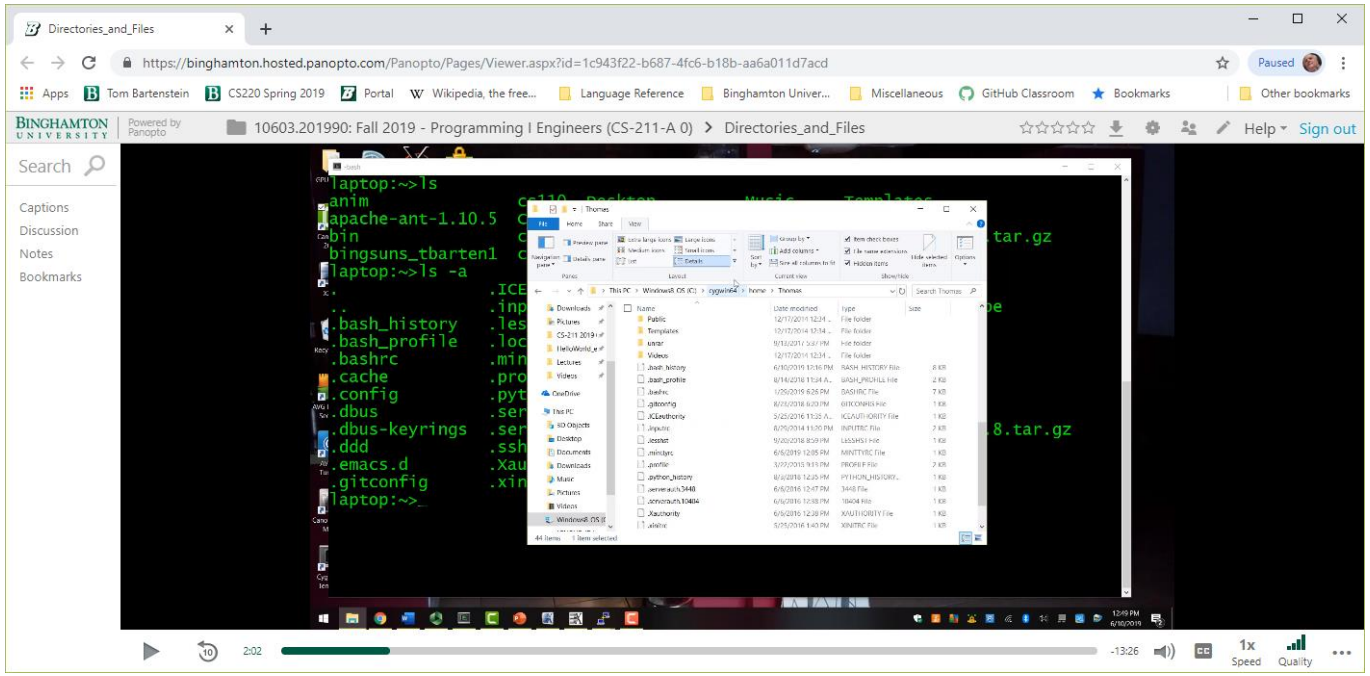
Video Review: Unix Terminal

- History
- Scrolling, Editing, Command History
- Questions?



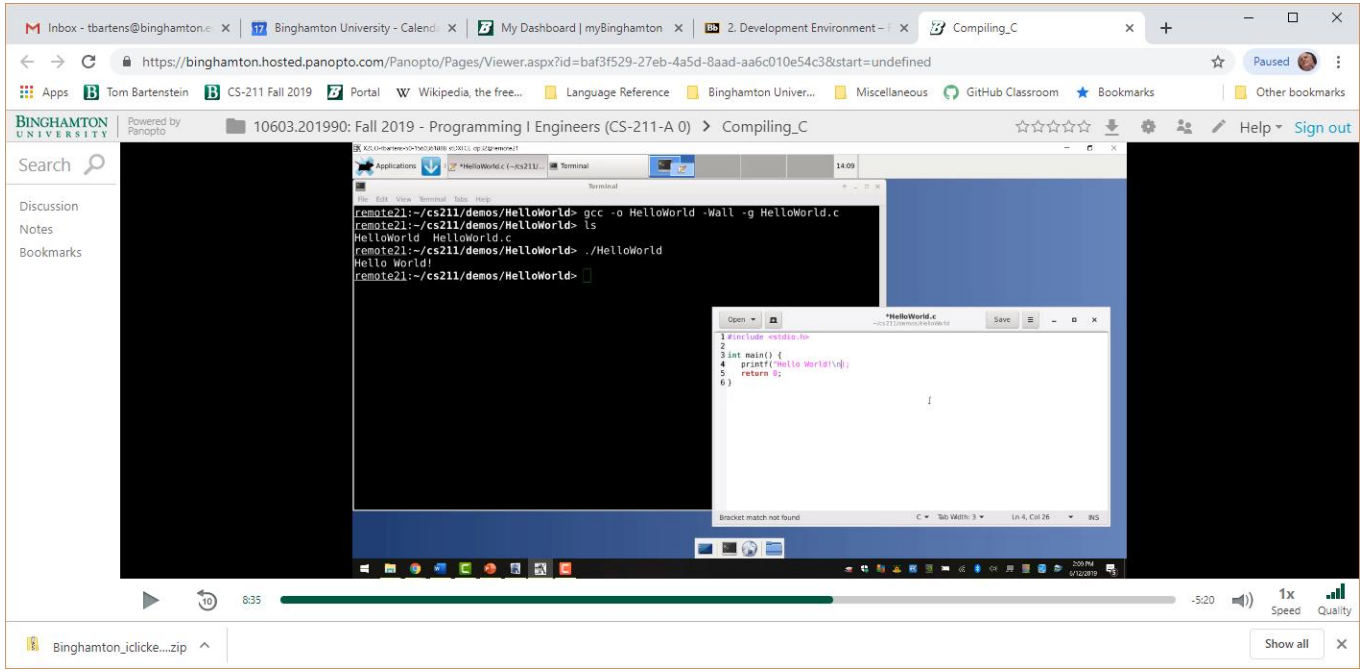
Video Review: Files and Folders

- Home directory
- Current directory
- Unix commands: mkdir, rmdir, ls
- Questions?



Video Review: Compilers

- Running gcc
- Warning and Error Messages
- Running a program
- Questions?

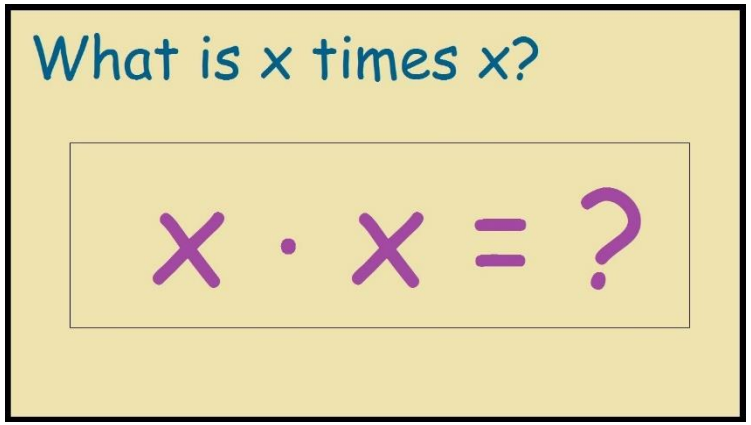


iClicker Question

Please click on the choice that is true:

A. The gcc parameter “-Wall” helps build a wall between your code and library code.
B. The gcc parameter “-g” tells the compiler to run garbage collection.
C. The gcc parameter “-o HelloWorld” tells the compiler to put its output (an executable) in a file called “HelloWorld”.
D. None of the above.

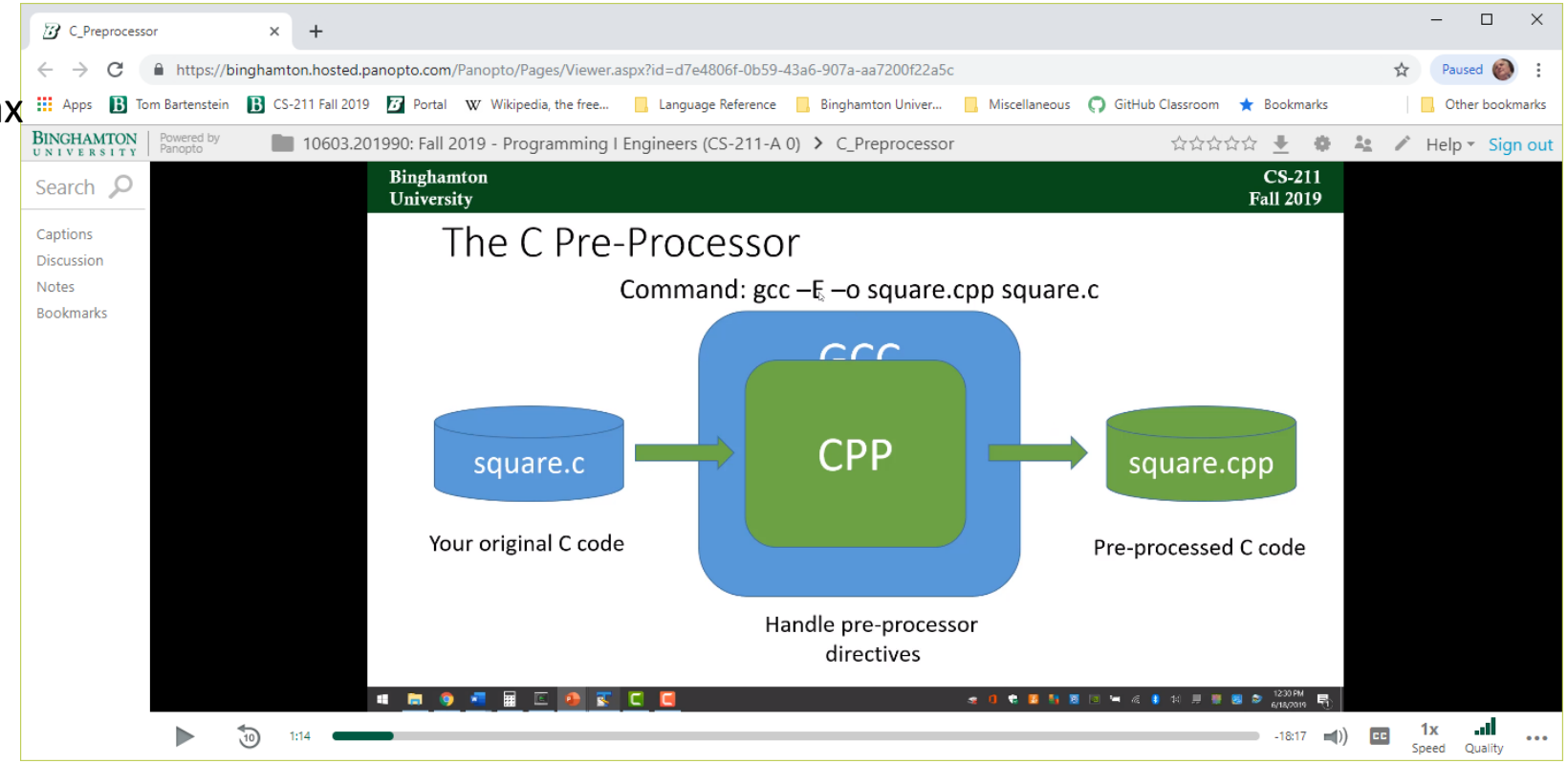
Exercise Review



- Write a computer program that reads a number, and computes the square of that number.

The C Pre-processor

- Watch the video, available on myCourses
  - Content
    - Videos
      - 3. C Syntax



iClicker Question

Please click on the choice that is true:

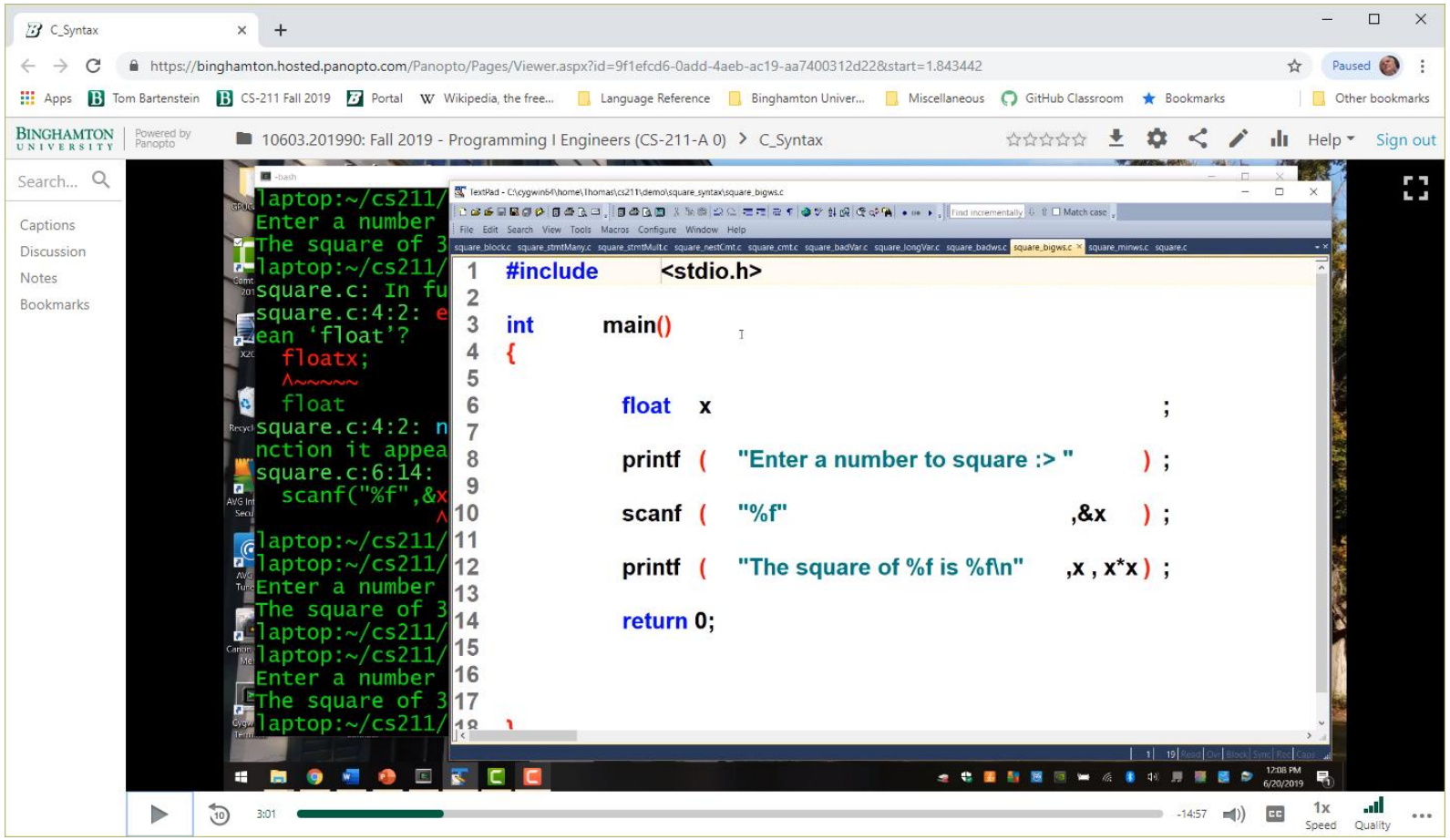
```
#include <stdio.h>

int main() {
    printf("Hello World!\n");
    return 0;
}
```

A. There are no pre-processor directives in "HelloWorld.c"
B. There is only one pre-processor directive in "HelloWorld.c"
C. There are more than one pre-processor directives in "HelloWorld.c"
D. I have no clue what a pre-processor directive is

C Syntax - in General

- Watch the video, available on myCourses
  - Content
    - Videos
      - 3. C Syntax



iClicker Question

Please click on the choice that is true:

```
#include <stdio.h>

int main() {
    printf("Hello World!\n");
    return 0;
}
```

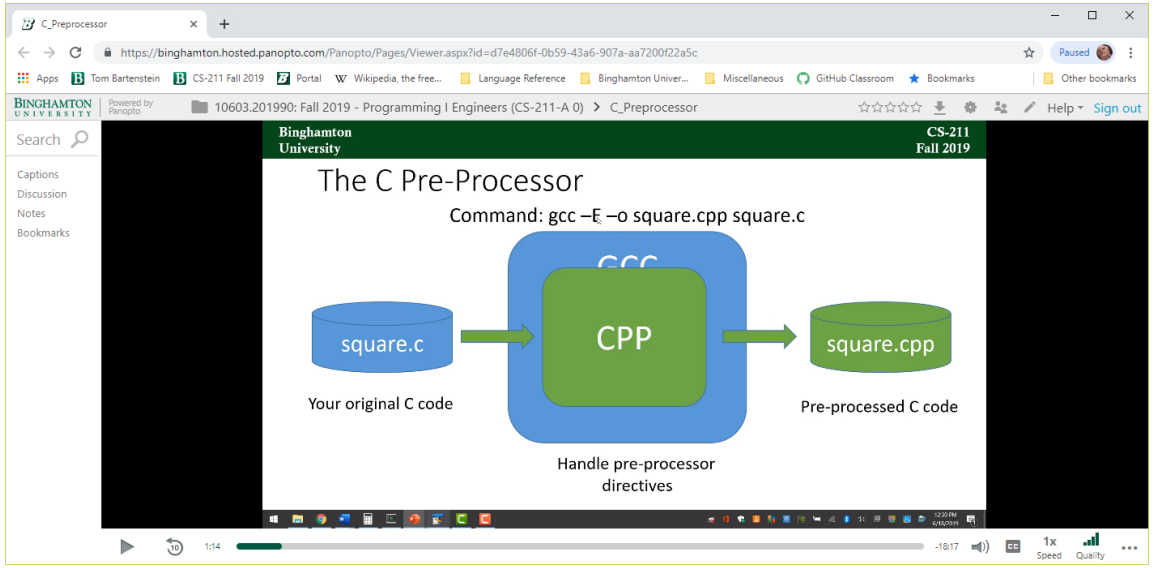
A. There is one block with two statements in "HelloWorld.c"
B. There is one block with three statements in "HelloWorld.c"
C. There are no blocks in "HelloWorld.c"
D. I have no clue what a block is.

Exercise: interesting.c

Write a C program, based on “Hello World” that prints out something interesting about yourself. For instance, what is your favorite thing to do in your spare time, or what is the most interesting thing that has happened to you recently? A single line of output is OK, but you may want to write several lines… just no War and Peace please.

Resources

- Programming in C, Chapter 3
- C FAQ: C Preprocessor (http://c-faq.com/cpp/index.html)
- Wikipedia: C Preprocessor (https://en.wikipedia.org/wiki/C_preprocessor),
- Wikipedia: C Syntax (https://en.wikipedia.org/wiki/C_syntax)
- C FAQ: Style (http://c-faq.com/style/index.html)
- C Syntax Tutorial (https://www.tutorialspoint.com/cprogramming/c_basic_syntax.htm) or (https://www.studytonight.com/c/c-syntax.php)



The C Pre-processor

Summary Notes

The C Compiler

Command: gcc -g –Wall –o square square.c

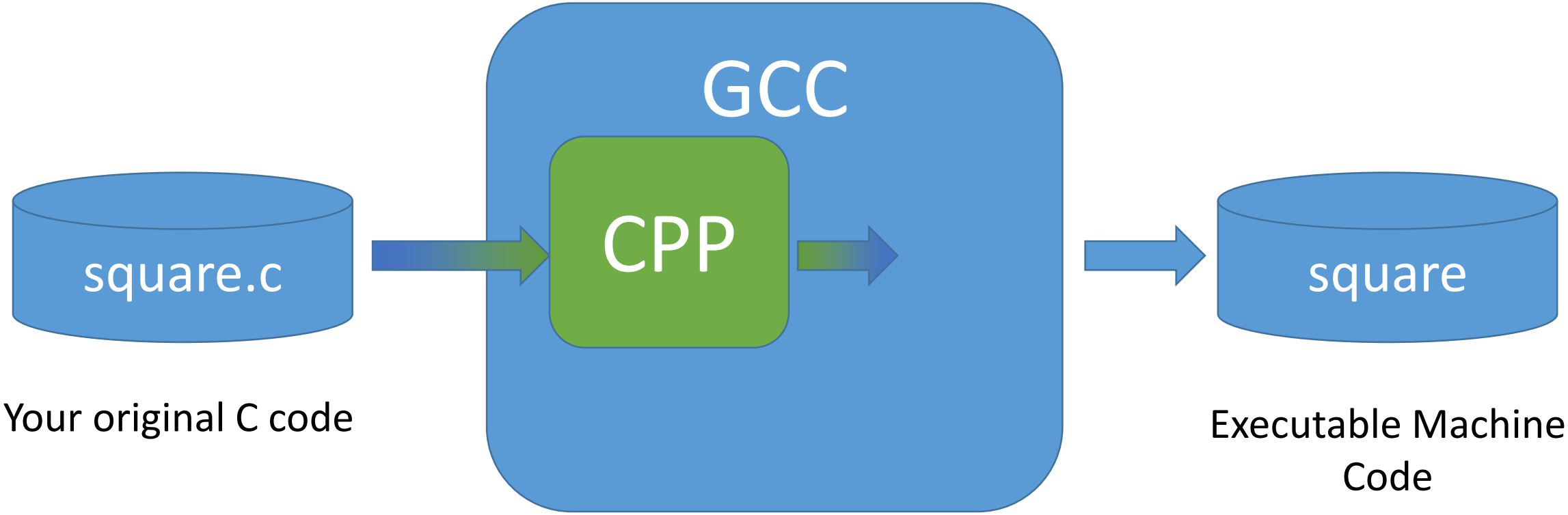
Your original C code

Compiler

Executable Machine Code

The C Compiler

Command: gcc -g –Wall –o square square.c



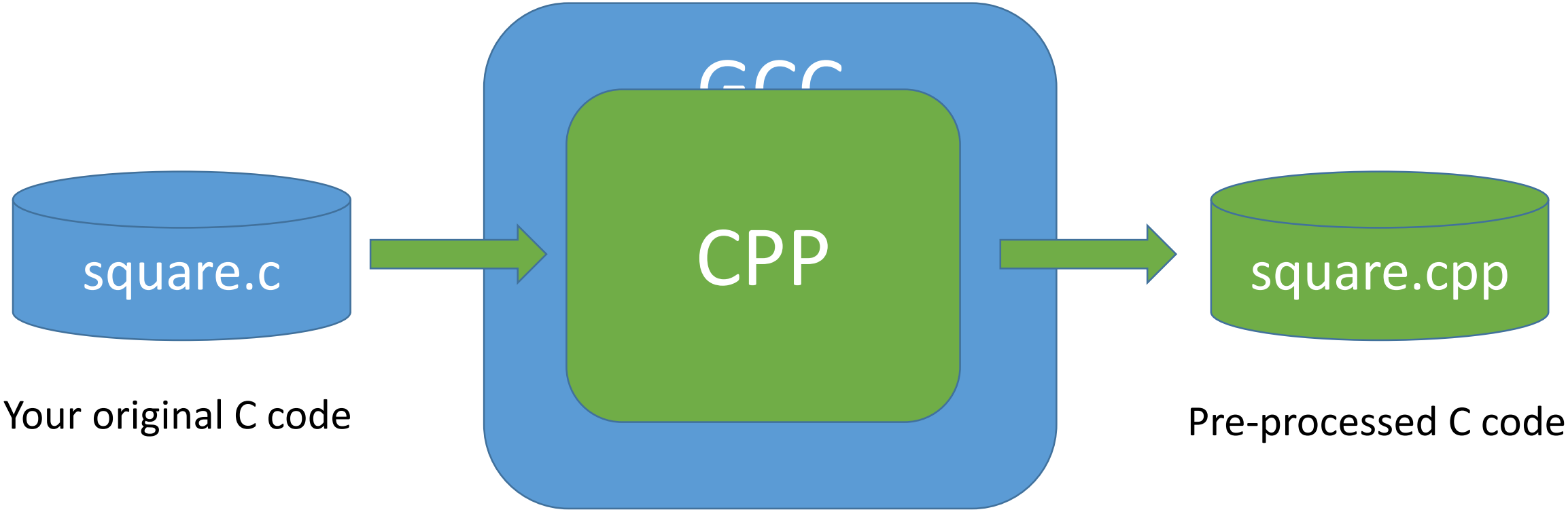
Your original C code

Executable Machine Code

Compiler

The C Pre-Processor

Command: gcc –E –o square.cpp square.c



Your original C code

Handle pre-processor directives

Pre-processed C code

Pre-Processing Directives

- Any line that starts with “#” is a “pre-processor directive”
- Pre-processor consumes that entire line
  - Possibly replacing it with other C code

#include directive

- #include <*filename*>
  - Replace directive with contents of *filename*, as found in the system library
  - May contain *subdir/filename* to search in subdirectory of the system library
- #include "*filename*"
  - Replace directive with contents of *filename*, as found in current directory
  - May contain *subdir/filename* to search in subdirectory of the current directory

- Contents of replaced files are pre-processed as well

gcc –E output

- C code, 
- pseudo “comment” lines of the form:

 # lno “filename” flags

- Where:
  - *lno* is the line number in…
  - *filename*
  - *flags*: 1=“start”, 2=“end”, 3=“library”, 4=“extern”
- See: https://gcc.gnu.org/onlinedocs/cpp/ for complete details

#define directive

#define *name value*

- From now on, every instance of *name* is replaced by *value*
- *name* is usually all uppercase
- *Value* is everything up to final white space on line
- *Value* can cross lines if ended with a backslash (\)
- Used for compiler constants, e.g.

```
#define DIME_VALUE 10
value = DIME_VALUE * number_of_dimes;
```

- Complex #defines for pre-processor macros

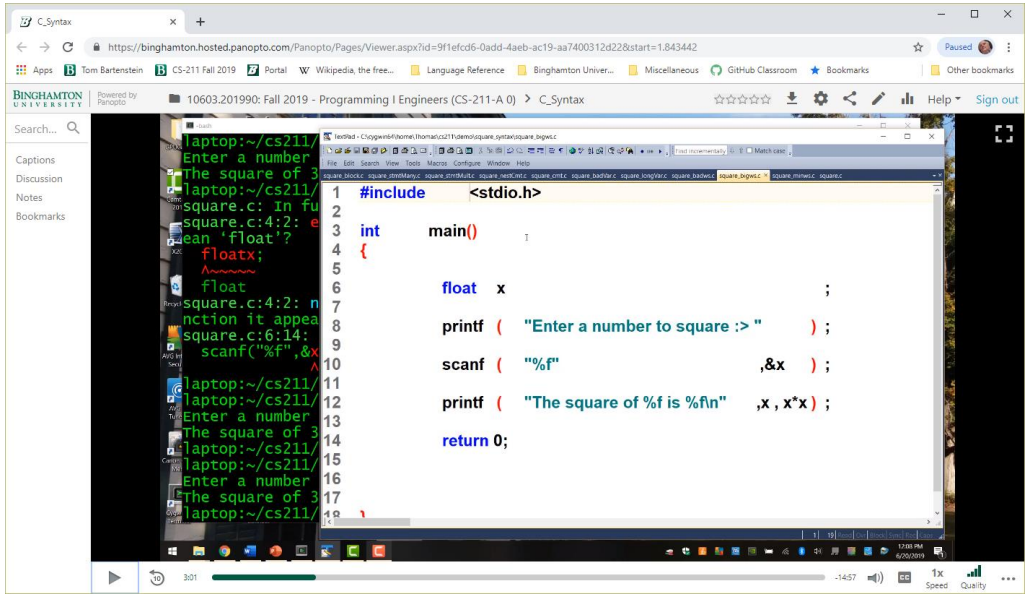
#ifdef directive

#ifdef *name* or #ifndef *name*
#endif

- Only include lines between #ifdef and #endif if *name* is #defined
- Allows multiple implementations e.g.

```
#ifdef method1
#endif
#ifdef method2
#endif
```

- Use gcc –Dmethod2 … as an alternative to #define



C Syntax - in General

Summary Notes

White Space

- Needed to separate tokens
  - e.g. "then break" are two contiguous keywords, but "thenbreak" is a single identifier
- Otherwise, white space is ignored
  - "then break" is the same as "then break" is the same as "then break"
  - Enables programmer to choose formatting preferences

Keywords

asm auto break case char const continue default do double else enum extern float for fortran goto if int long register return short signed sizeof static struct switch typedef union unsigned void volatile while

Keywords have special meaning to the compiler, so you can't use them for other names.

Type Keywords

auto char const double enum extern float int long register short signed static struct typedef union unsigned void volatile

These keywords are used to tell the compiler what kind of data it is working on

Flow Control Keywords

break case continue default do else for goto if return switch while

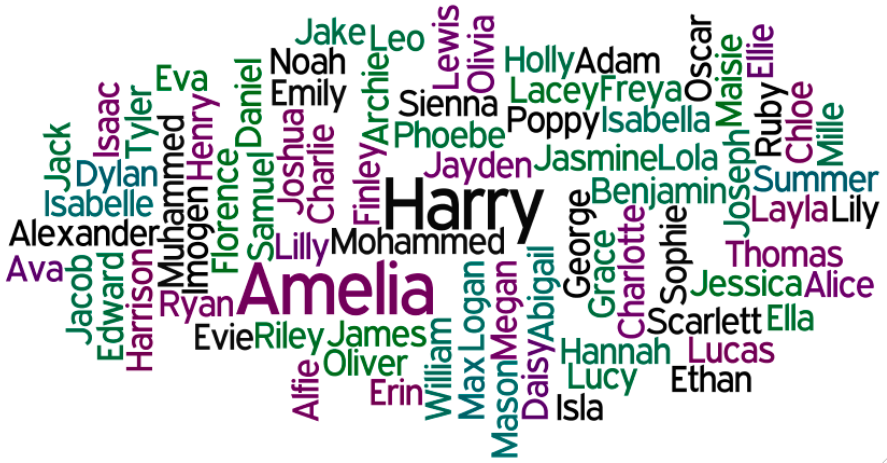
These keywords tell the compiler what order to execute instructions.

Miscellaneous Keywords

asm fortran sizeof

These keywords tell the compiler to switch languages, or look up a size.

Identifiers



- Function, parameter, and variable names
- Must start with a letter or an underscore
  - Underscores usually avoided
- May not contain white space
- After the first letter, can be any number, letter, or underscore
- Identifiers are case sensitive
  - “polyArea” is different from “PolyArea”
- Choose names that are descriptive, and easy to type
  - “Be4aTgh9_fr37200aBy” is probably not a good choice

Comments

- Anything starting with /* up to the next */ is a comment
  - /* comments may span lines
    and continue on to the next line */
  - Comments do NOT nest

```
x=3; /* reset x to 3 */
/* x=3; /* reset x to 3 */  this was a mistake */
```

^ syntax error… this not declared

Oops… x shouldn't be reset here!
Let's comment it out

- // causes a comment to the end of the line (white space matters)
- Use comments to help reader understand what the code does!
  - No need to comment the obvious: a=a+3; // add three to a
- Comments ignored by compiler

Block Comments

```
/* ----------------------------------------

I like this style of comment because I can add or remove
lines from the comment without special comment reformat
intervention.

Besides, it looks clean.

----------------------------------------*/
```

C Statements

- A statement is a list of identifiers, keywords and/or operators that ends with a semi-colon
  - a=a+3;
  - int c=7;
  - int cent_to_far(int c);
- A C statement may span more than one line in the file
  - int MyLongFunctionNamedFunctionThatTakesLotsOfArguments(
    int arg1, int arg2, int arg3, int arg4,
    int arg5, int arg6, int arg7, int arg8);
- There may be more than one statement on one line in the file
  - int a=5; a=a+6; int b; b=cent_to_far(a);

C Blocks

- A block of statements is a list of statements, surrounded by { and }
  - { - Left curly brace
  - } – Right curly brace
- A block can be used anywhere a statement can be used
- Blocks of statements can be nested