CSCI-UA.0201

Computer Systems Organization

C Programming – Basics

Thomas Wies

wies@cs.nyu.edu

https://cs.nyu.edu/wies

Brian Kernighan

Dennis Ritchie

In 1972 **Dennis Ritchie** at Bell Labs writes C and in 1978 the publication of **The C Programming Language** by Kernighan & Ritchie caused a revolution in the computing world.

Why C?

- Mainly because it produces code that runs nearly as fast as code written in assembly language. Some examples of the use of C might be:
  - Operating Systems
  - Language Compilers
  - Assemblers
  - Text Editors
  - Print Spoolers
  - Network Drivers
  - Language Interpreters
  - Utilities

Interesting Opinion About C

You might never use it professionally, but it contains a lifetime of lessons. And the hardest problems, the ones that the top engineers are asked to solve, will sooner or later hit some foundational C code.

Here are some things that are written in C:

- The Java virtual machine is written in ANSI C
- Linux is written in C (and some assembly, but mostly C)
- Python is written in C
- Mac OS X kernel is written in C
- Windows is written in C and C++
- The Oracle database is written in C and C++
- Cisco routers, those things which connect the Internet, also C

Name anything that is foundational, complex, and performance critical. It was written in C, with a sprinkling of assembly thrown in.

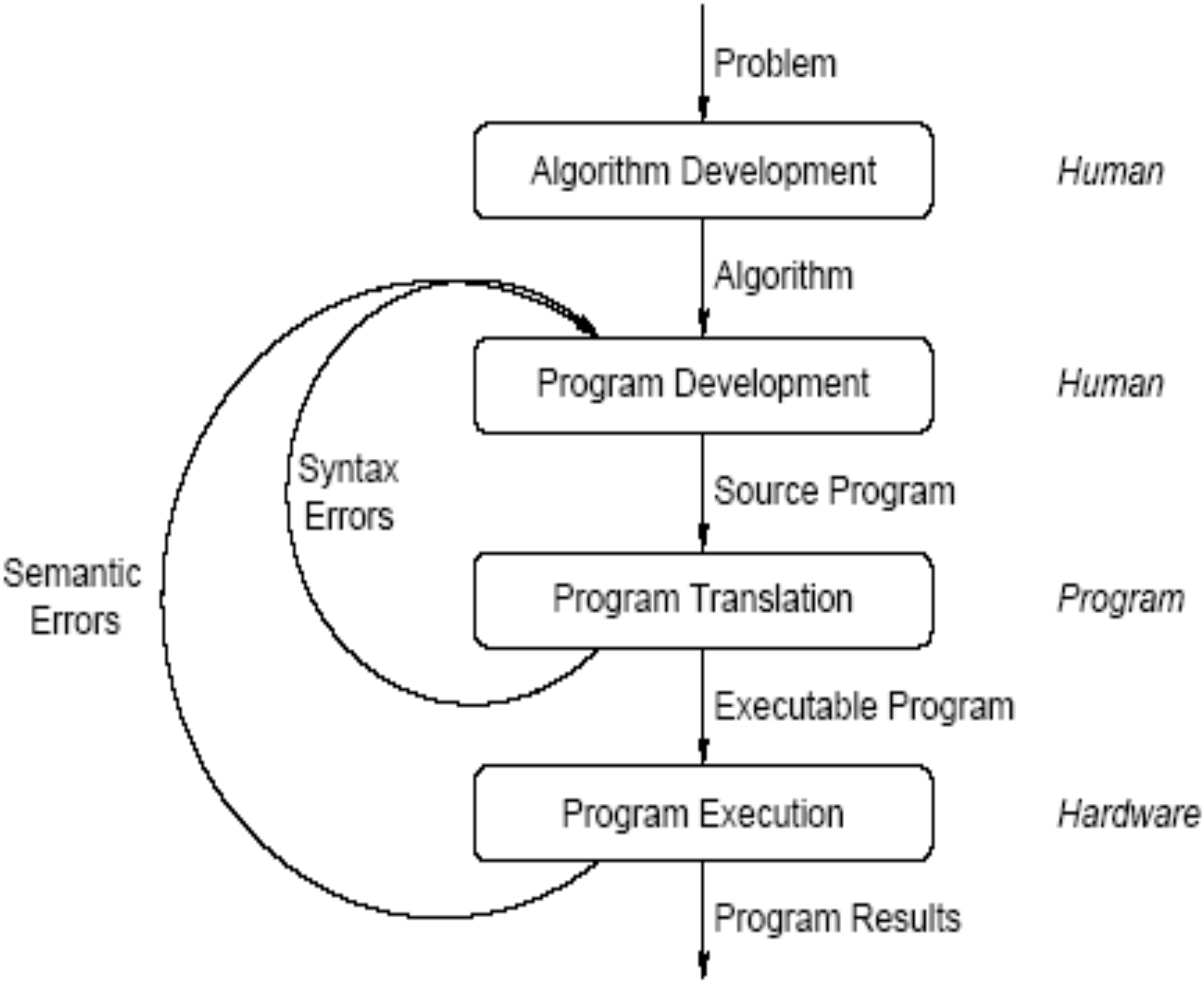
C will make you a better Java programmer. You'll know when the JVM is using the stack and when it's using the heap, and what that means. You'll have a more intuitive sense of what garbage collection does. You'll have a better sense of the relative performance cost of objects versus primitives.

https://www.quora.com/Financially-speaking-which-computer-languages-can-earn-the-most-for-the-programmer/answer/Carter-Page-1

Your first goal: Learn C!

- Resources
  - KR book: “The C Programming Language”
  - These lectures
  - Additional online resources (some links on the course website)

- Learning a Programming Language
  - The best way to learn is to write programs


Problem
Algorithm Development
Human
Algorithm
Program Development
Human
Syntax Errors
Source Program
Semantic Errors
Program Translation
Program
Executable Program
Program Execution
Hardware
Program Results

Writing and Running Programs

```
#include <stdio.h>
/* The simplest C Program */
int main(int argc, char **argv)
{
  printf(“Hello World\n”);
  return 0;
}
```

1. Write text of program (source code) using a text editor, save as text file e.g. my_program.c

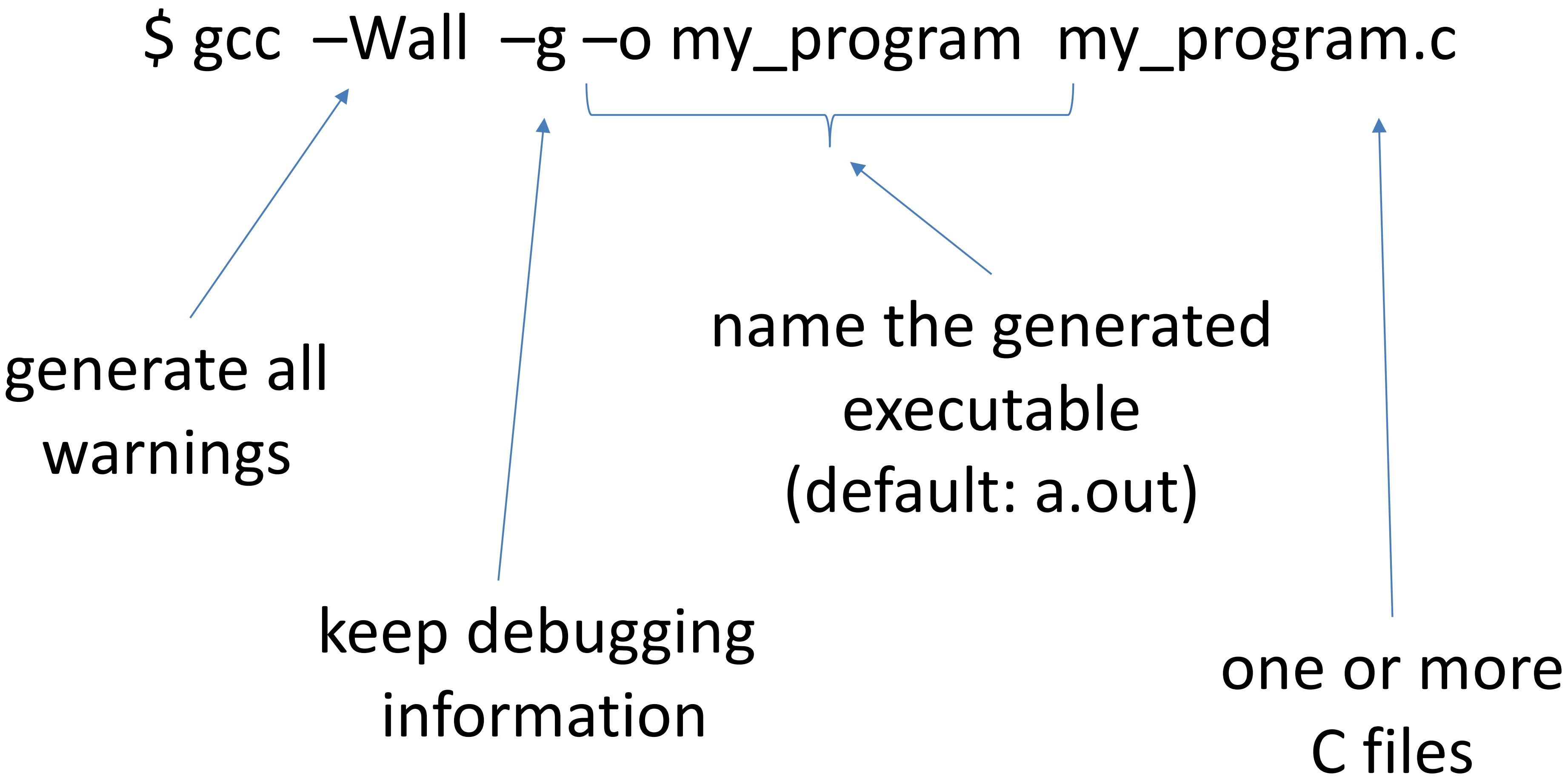
2. Run the compiler to convert program from source to an “executable” or “binary”:

```
$ gcc  –Wall  –g –o my_program  my_program.c
```

3-Compiler gives errors and warnings; edit source file, fix it, and re-compile

Run it and see if it works ☺

```
$ ./my_program
Hello World
$ █
```



```
$ gcc  –Wall  –g –o my_program  my_program.c
```

generate all warnings

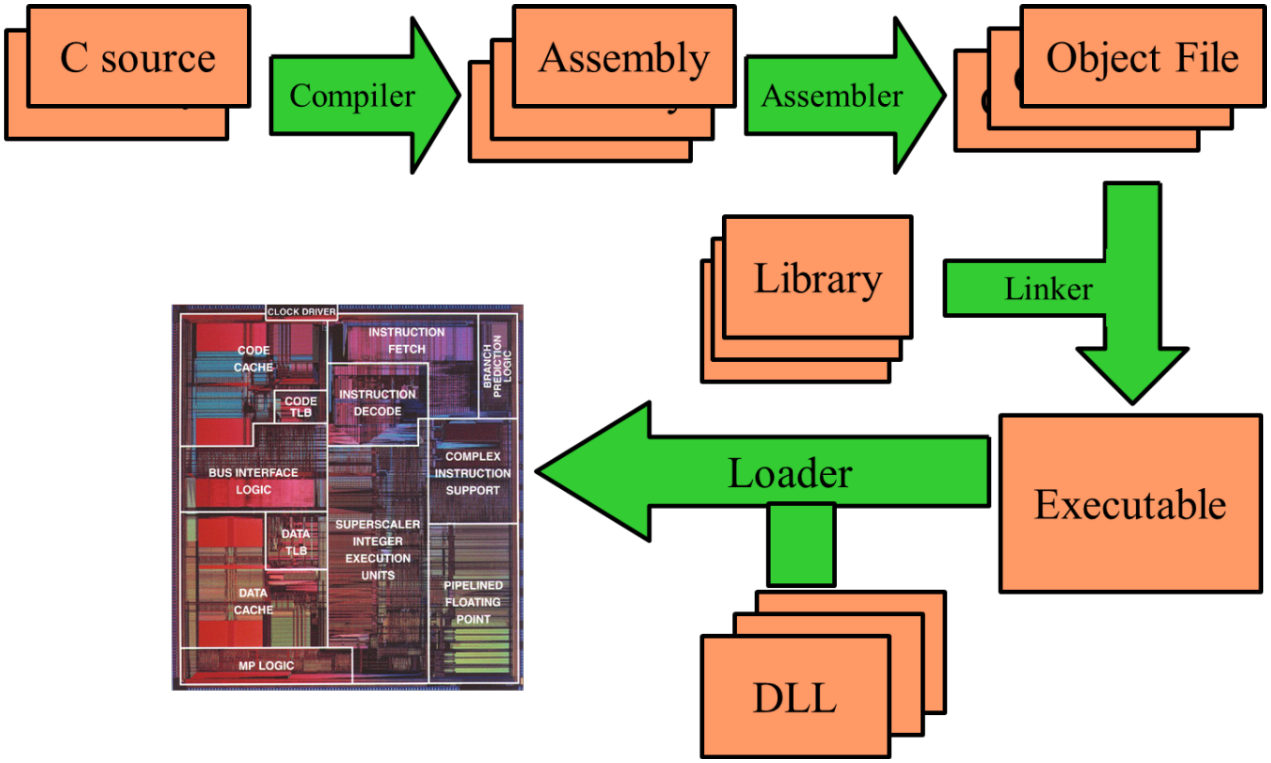
keep debugging information

name the generated executable (default: a.out)

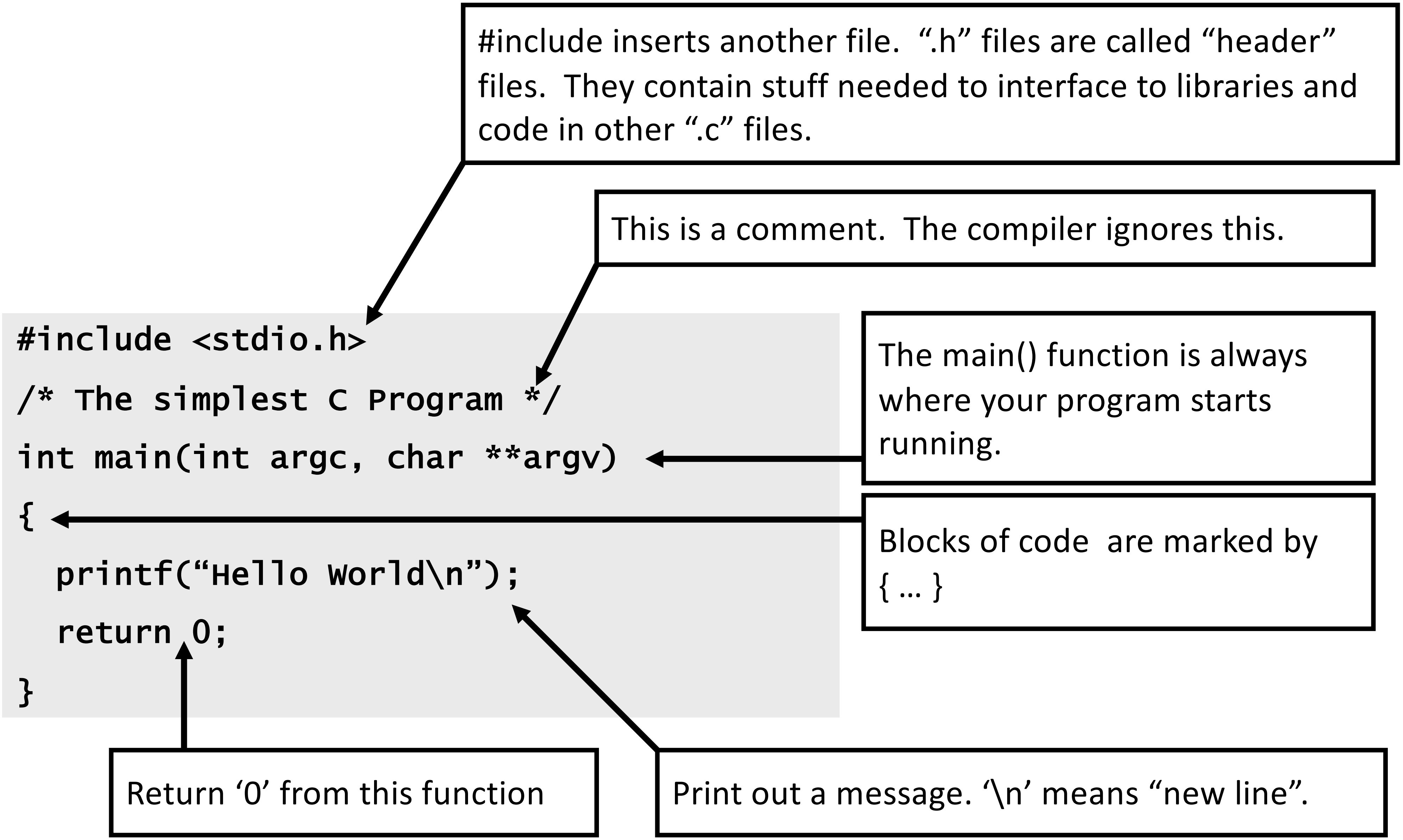
one or more C files

About C

- **Procedural language**
  - Functions calling each other, starting with main().
- **Case-sensitive**



C Syntax and Hello World



#include inserts another file. “.h” files are called “header” files. They contain stuff needed to interface to libraries and code in other “.c” files.

This is a comment. The compiler ignores this.

```
#include <stdio.h>
/* The simplest C Program */
int main(int argc, char **argv)
{
  printf(“Hello World\n”);
  return 0;
}
```

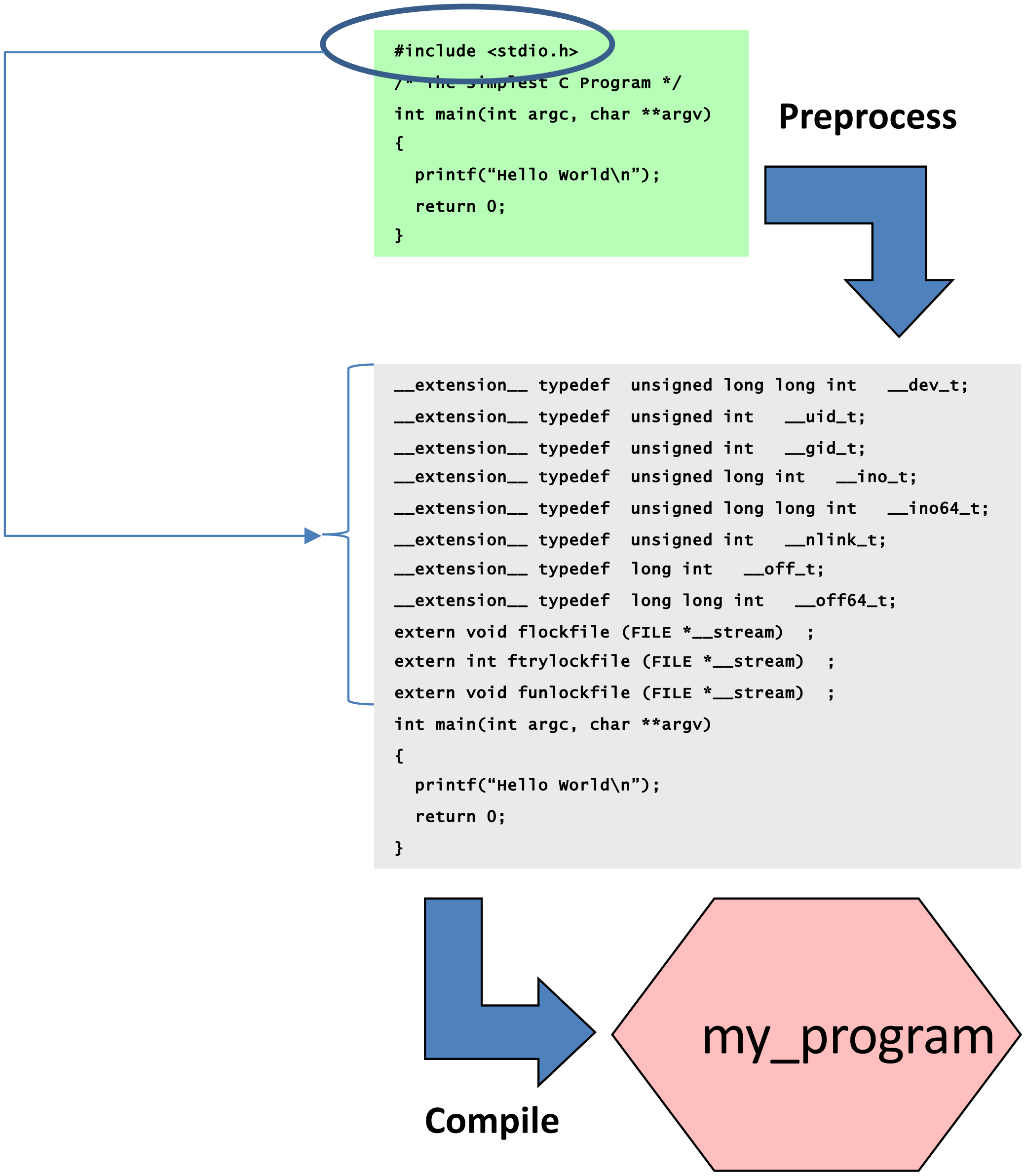
The main() function is always where your program starts running.

Blocks of code are marked by { … }

Return ‘0’ from this function

Print out a message. ‘\n’ means “new line”.

Preprocessing



```
#include <stdio.h>
/* The Simplest C Program */
int main(int argc, char **argv)
{
  printf(“Hello World\n”);
  return 0;
}
```

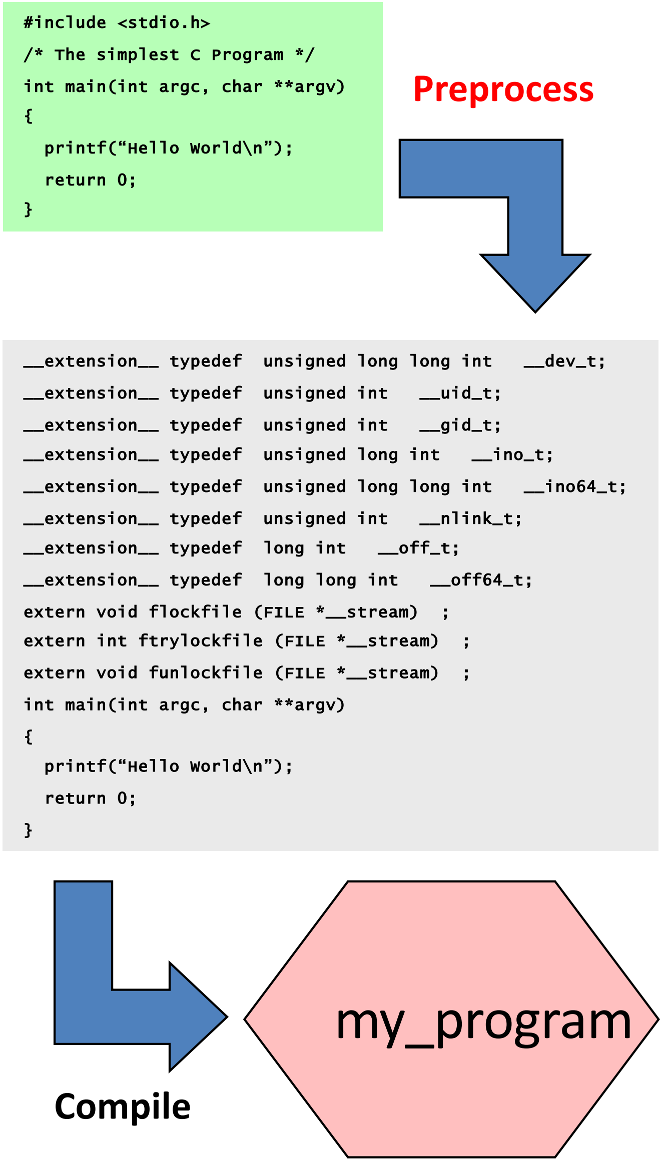
Preprocess

```
__extension__ typedef  unsigned long long int   __dev_t;
__extension__ typedef  unsigned int   __uid_t;
__extension__ typedef  unsigned int   __gid_t;
__extension__ typedef  unsigned long int   __ino_t;
__extension__ typedef  unsigned long long int   __ino64_t;
__extension__ typedef  unsigned int   __nlink_t;
__extension__ typedef  long int   __off_t;
__extension__ typedef  long long int   __off64_t;
extern void flockfile (FILE *__stream)  ;
extern int ftrylockfile (FILE *__stream)  ;
extern void funlockfile (FILE *__stream)  ;
int main(int argc, char **argv)
{
  printf(“Hello World\n”);
  return 0;
}
```

Compile

my_program

Preprocessing



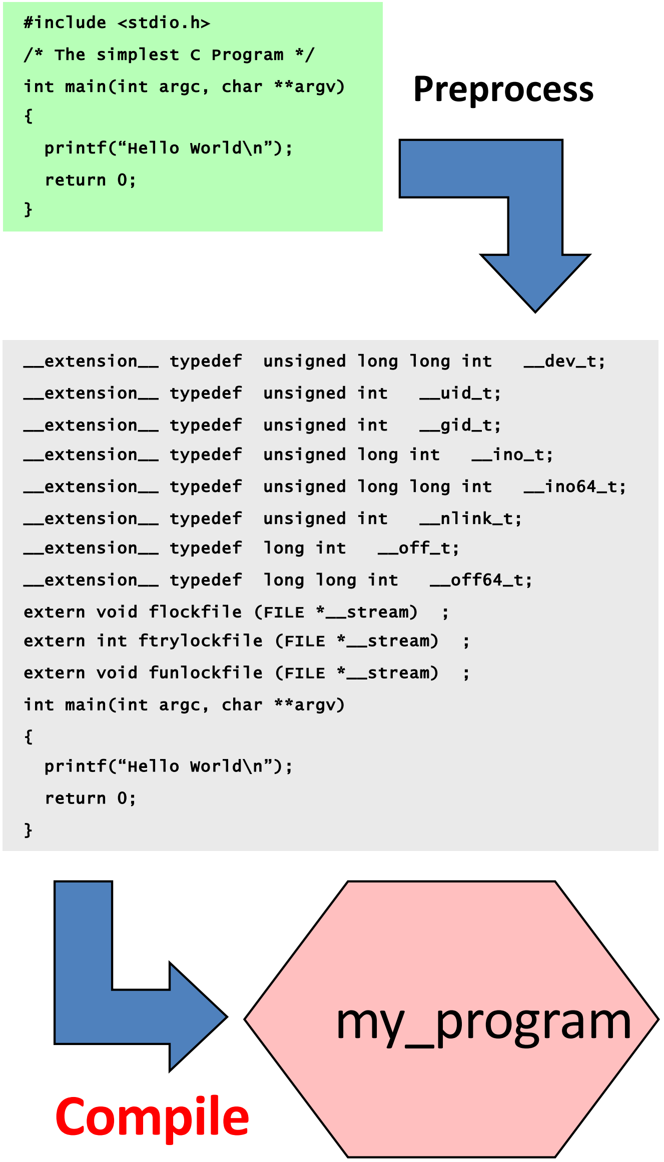
```
#include <stdio.h>
/* The simplest C Program */
int main(int argc, char **argv)
{
  printf(“Hello World\n”);
  return 0;
}
```

```
__extension__ typedef  unsigned long long int   __dev_t;
__extension__ typedef  unsigned int   __uid_t;
__extension__ typedef  unsigned int   __gid_t;
__extension__ typedef  unsigned long int   __ino_t;
__extension__ typedef  unsigned long long int   __ino64_t;
__extension__ typedef  unsigned int   __nlink_t;
__extension__ typedef  long int   __off_t;
__extension__ typedef  long long int   __off64_t;
extern void flockfile (FILE *__stream)  ;
extern int ftrylockfile (FILE *__stream)  ;
extern void funlockfile (FILE *__stream)  ;
int main(int argc, char **argv)
{
  printf(“Hello World\n”);
  return 0;
}
```

In Preprocessing, source code is “expanded” into a larger form that is simpler for the compiler to understand. Any line that starts with ‘#’ is a line that is interpreted by the Preprocessor.

- Include files are “pasted in” (#include)
- Macros are “expanded” (#define)
- Comments are stripped out (/*   */ , //)
- Continued lines (i.e. very long lines) are joined (\ )

Compiling



```
#include <stdio.h>
/* The simplest C Program */
int main(int argc, char **argv)
{
  printf(“Hello World\n”);
  return 0;
}
```

Preprocess

```
__extension__ typedef  unsigned long long int   __dev_t;
__extension__ typedef  unsigned int   __uid_t;
__extension__ typedef  unsigned int   __gid_t;
__extension__ typedef  unsigned long int   __ino_t;
__extension__ typedef  unsigned long long int   __ino64_t;
__extension__ typedef  unsigned int   __nlink_t;
__extension__ typedef  long int   __off_t;
__extension__ typedef  long long int   __off64_t;
extern void flockfile (FILE *__stream)  ;
extern int ftrylockfile (FILE *__stream)  ;
extern void funlockfile (FILE *__stream)  ;
int main(int argc, char **argv)
{
  printf(“Hello World\n”);
  return 0;
}
```

Compile

my_program

- The compiler then converts the resulting text into binary code the CPU can run directly.
- The compilation process involves really several steps:
    - **Compiler**: high level language → assembly
    - **Assembler**: assembly → machine code
    - **Linker**: links all machine code files and needed libraries into one executable file.
- When you type *gcc* you really invoke the compiler, assembler, and linker.

What is "Memory"?

- Is like a big table of numbered slots.
- Each slot stores a byte.

- The number of a slot is its Address.
- One byte Value can be stored in each slot.

Some "logical" data values span more than one slot, like the character string "Hello\n"

A Type names a logical meaning to a span of memory. Some simple types are:

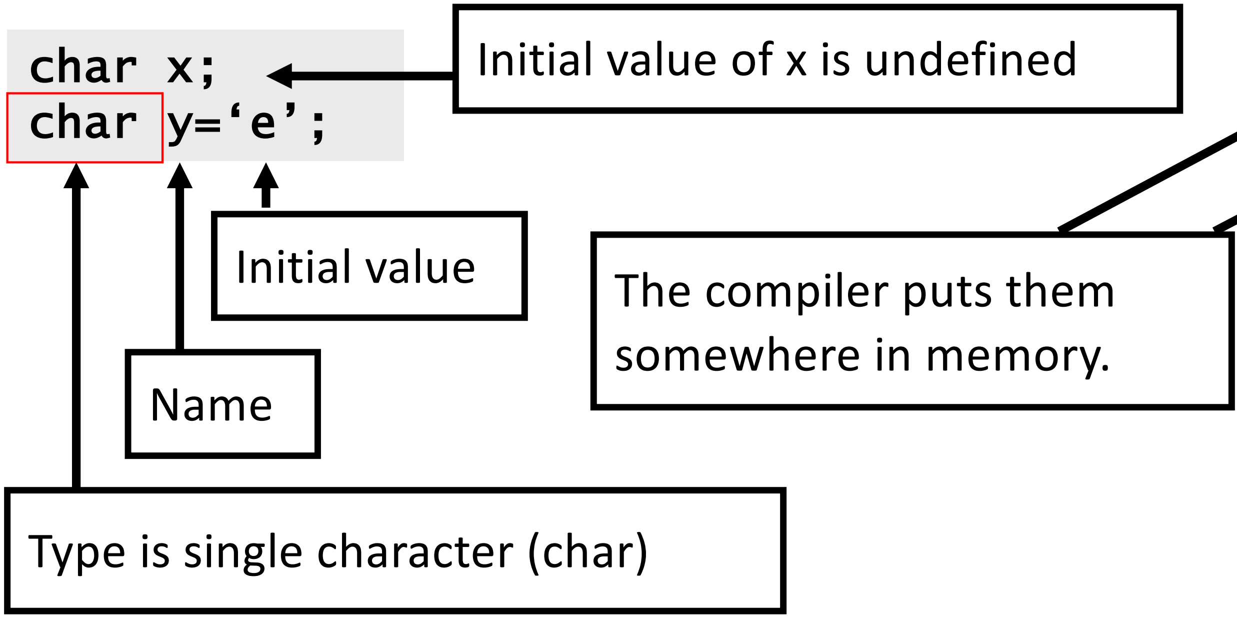
`char`	a single character (1 slot)
`char [10]`	an array of 10 characters
`int`	signed 4 byte integer
`float`	4 byte floating point

Addr	Value
0	
1	
2	
3	
4	'H' (72)
5	'e' (101)
6	'l' (108)
7	'l' (108)
8	'o' (111)
9	'\n' (10)
10	'\0' (0)
11	
12	

What is a Variable?

A Variable names a place in memory where you store a Value of a certain Type.

You first Define a variable by giving it a name and specifying the type, and optionally an initial value



```
char x;
char y='e';
```

Initial value of x is undefined

Initial value

Name

Type is single character (char)

The compiler puts them somewhere in memory.

Symbol	Addr	Value
	0	
	1	
	2	
	3	
x	4	?
y	5	'e' (101)
	6	
	7	
	8	
	9	
	10	
	11	
	12	

Multi-byte Variables

Different types consume different amounts of memory. Most architectures store data on “word boundaries”, or even multiples of the size of a primitive data type (int, char)

```
char x;
char y=‘e’;
int z = 0x01020304;
```

0x means the constant is written in hex

padding

An int consumes 4 bytes

Symbol	Addr	Value
	0	
	1	
	2	
	3	
x	4	?
y	5	‘e’ (101)
	6	
	7	
z	8	4
	9	3
	10	2
	11	1
	12	

Scope

Every Variable is Declared within some scope. A Variable cannot be referenced from outside of that scope.

Scopes are defined with curly braces { }.

The scope of Function Arguments is the complete body of the function.

The scope of Variables defined inside a function starts at the definition and ends at the closing brace of the containing block

The scope of Variables defined outside a function starts at the definition and ends at the end of the file. Called Global Vars.

```
void p(char x)
{

  char y;

  char z;

}

char z;


void q(char a)
{
  char b;


    {
      char c;

    }

  char d;

}
```

Now that we know about variables, let’s combine them to form expressions!

$$\underbrace{X = \overbrace{2 * Y + Z}^{\text{Expression}};}_{\text{Statement}}$$

How Expressions Are Evaluated?

Expressions combine Values using Operators, according to precedence.

```
1 + 2 * 2        → 1 + 4       → 5
(1 + 2) * 2      → 3 * 2       → 6
```

Comparison operators are used to compare values.
In C: 0 means “false”, and *any other value* means “true”.

```
int x=4;
(x < 5)                  → (4 < 5)                 → <true>
(x < 4)                  → (4 < 4)                 → 0
((x < 5) || (x < 4))     → (<true> || (x < 4))     → <true>
```

Not evaluated because
first clause was true

Precedence

- **Highest to lowest**
  - ()
  - *, /, %
  - +, -

When in doubt, use parenthesis.