Getting started with C

Adapted from materials by Dr. Carrier

Our first C program!

Our first C program!

```c
#include <stdio.h>

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

Our first C program!

```c
#include <stdio.h>

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

To compile:

```
gcc hello_world.c
```

Our first C program!

```c
#include <stdio.h>

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

To compile:

```
gcc hello_world.c
```

To run:

```
./a.out
```

Breaking it down

```c
#include <stdio.h>

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

Breaking it down

```c
#include <stdio.h>

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

#include <stdio.h> is a ***preprocessor directive***

Breaking it down

```c
#include <stdio.h>

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

#include <stdio.h> is a ***preprocessor directive***

The C processor does “simple” text-based changes <u>before compilation</u>

Breaking it down

```c
#include <stdio.h>

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

#include <stdio.h> is a ***preprocessor directive***

The C processor does “simple” text-based changes before compilation

Here, #include replaces that line with the stdio header file

Breaking it down

```c
#include <stdio.h>

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

#include <stdio.h> is a ***preprocessor directive***

The C processor does “simple” text-based changes before compilation

Here, #include replaces that line with the stdio header file

(We’ll talk about header files later)

Breaking it down

```c
#include <stdio.h>

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

#include <stdio.h> is a ***preprocessor directive***

The C processor does “simple” text-based changes before compilation

Here, #include replaces that line with the stdio header file

(We’ll talk about header files later)

You can think of this like importing in Python!

Breaking it down

```c
#include <stdio.h>

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

```
int main() {
   …
}
```

Is our “main function”.

Breaking it down

```c
#include <stdio.h>

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

```
int main() {
   …
}
```

Is our “main function”.

Execution always starts in the “main” function!

Breaking it down

```c
#include <stdio.h>

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

`printf(“...”);`

Will print a formatted string to stdout

But in this case, it’s just a typical string.

Breaking it down

```c
#include <stdio.h>

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

`printf(“...”);`

Will print a formatted string to stdout

But in this case, it’s just a typical string.

Note the semicolon!

Breaking it down

```c
#include <stdio.h>

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

`printf(“...”);`

Will print a formatted string to stdout

But in this case, it’s just a typical string.

Note the semicolon!

`return 0;`

Breaking it down

```c
#include <stdio.h>

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

`printf(“...”);`

Will print a formatted string to stdout

But in this case, it’s just a typical string.

Note the semicolon!

`return 0;`

Our main function returns an int (integer), so we actually need to return one!

Compiling `gcc hello_world.c`

Compiling

```
gcc hello_world.c
```

gcc is our compiler

Compiling

```
gcc hello_world.c
```

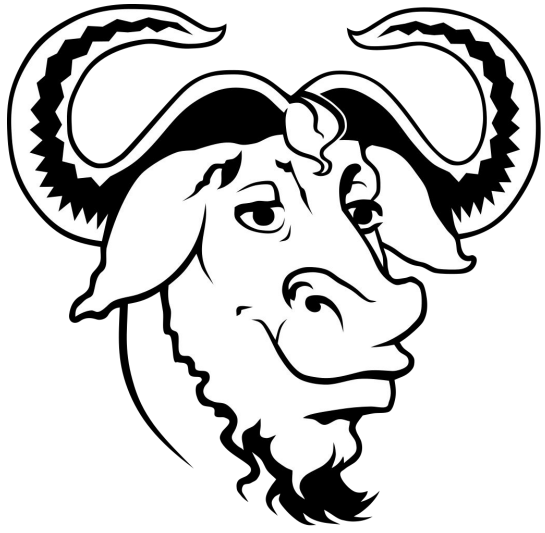
gcc is our compiler

GNU C Compiler

GNU Compiler Collection

Compiling

```
gcc hello_world.c
```

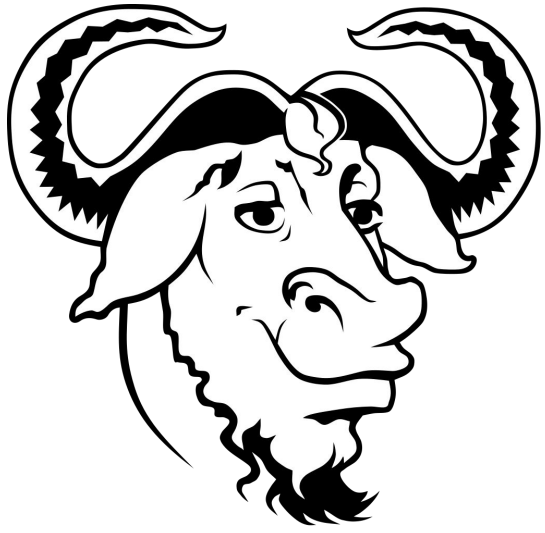


gcc is our compiler

- GNU C Compiler
- GNU Compiler Collection

Compiling

```
gcc hello_world.c
```



gcc is our compiler

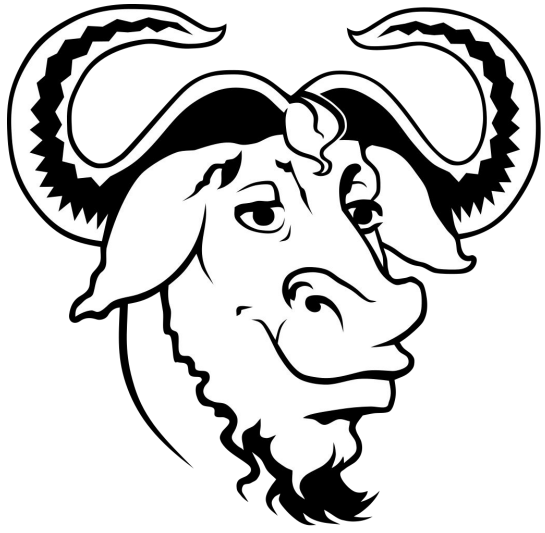
GNU C Compiler

GNU Compiler Collection

This compiles our source code into executable machine code (binary)

Compiling

```
gcc hello_world.c
```



gcc is our compiler

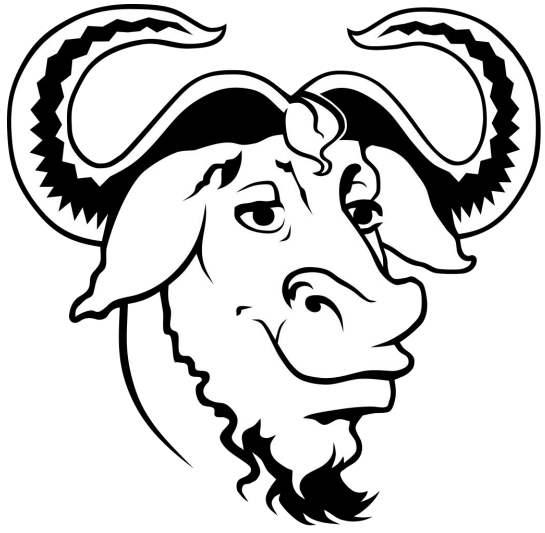
- GNU C Compiler
- GNU Compiler Collection

This compiles our source code into executable machine code (binary)

- Stored in the `a.out` file by default

Compiling

```
gcc hello_world.c
```



gcc is our compiler

- GNU C Compiler
- GNU Compiler Collection

This compiles our source code into executable machine code (binary)

- Stored in the `a.out` file by default

Compiler flags

Specify the output file:

```
gcc hello_world.c -o output_name
```

Compiler flags

Specify the output file:

```
gcc hello_world.c -o output_name
```

WARNING, don’t do this:

```
gcc hello_world.c -o hello_world.c
```

Compiler flags (continued)

Compiler flags (continued)

Optimization flags (capital letter O):

Compiler flags (continued)

Optimization flags (capital letter O):

- `-O` or `-O1` means “optimize some”

Compiler flags (continued)

Optimization flags (capital letter O):

- -O or -O1 means “optimize some”
- -O2 means “optimize more”

Compiler flags (continued)

Optimization flags (capital letter O):

- -O or -O1 means “optimize some”
- -O2 means “optimize more”
- -O3 means “optimize as much as possible”

Compiler flags (continued)

Optimization flags (capital letter O):

- `-O` or `-O1` means “optimize some”
- `-O2` means “optimize more”
- `-O3` means “optimize as much as possible”
- `-Os` and `-Oz` mean “optimize for space”

Compiler flags (continued)

Optimization flags (capital letter O):

- `-O` or `-O1` means “optimize some”
- `-O2` means “optimize more”
- `-O3` means “optimize as much as possible”
- `-Os` and `-Oz` mean “optimize for space”

Debug flag: `-g` (we’ll discuss later)

Compiler flags (continued)

Optimization flags (capital letter O):

- `-O` or `-O1` means “optimize some”
- `-O2` means “optimize more”
- `-O3` means “optimize as much as possible”
- `-Os` and `-Oz` mean “optimize for space”

Debug flag: `-g` (we’ll discuss later)

Warning flags:

- `-Wall` to enable all (can also enable some)