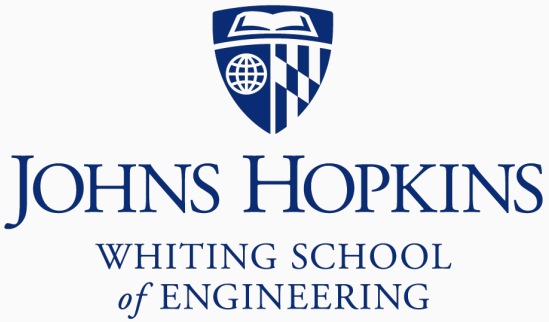
Stages of compilation & Hello, world

Ben Langmead

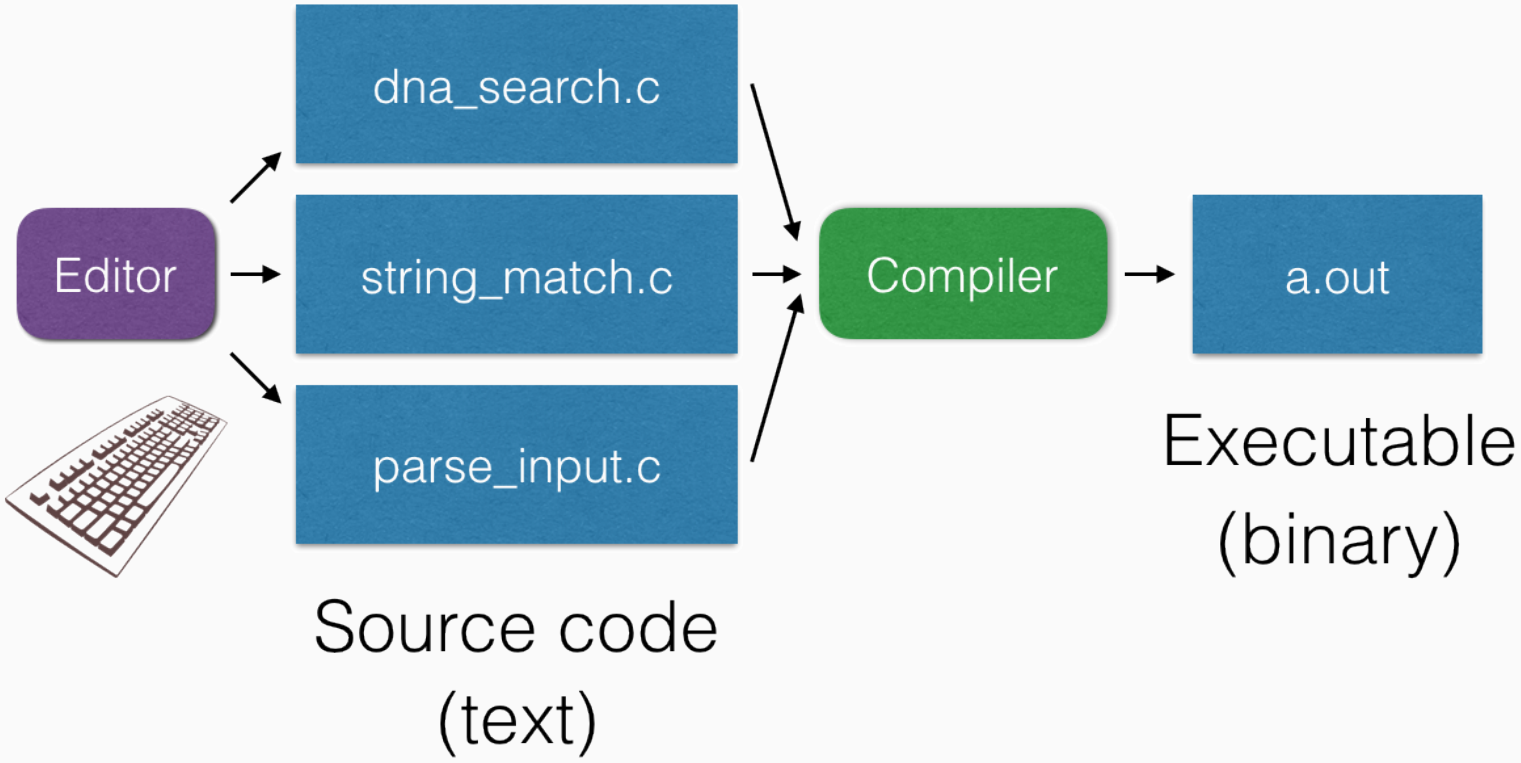
ben.langmead@gmail.com

www.langmead-lab.org

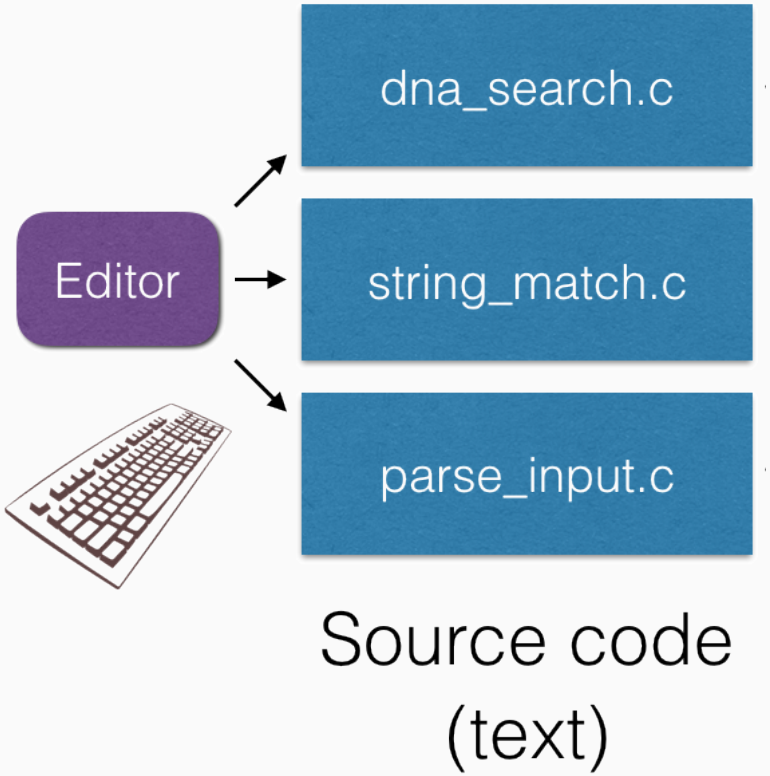


Source markdown available at github.com/BenLangmead/c-cpp-notes

Stages of compilation & Hello, world

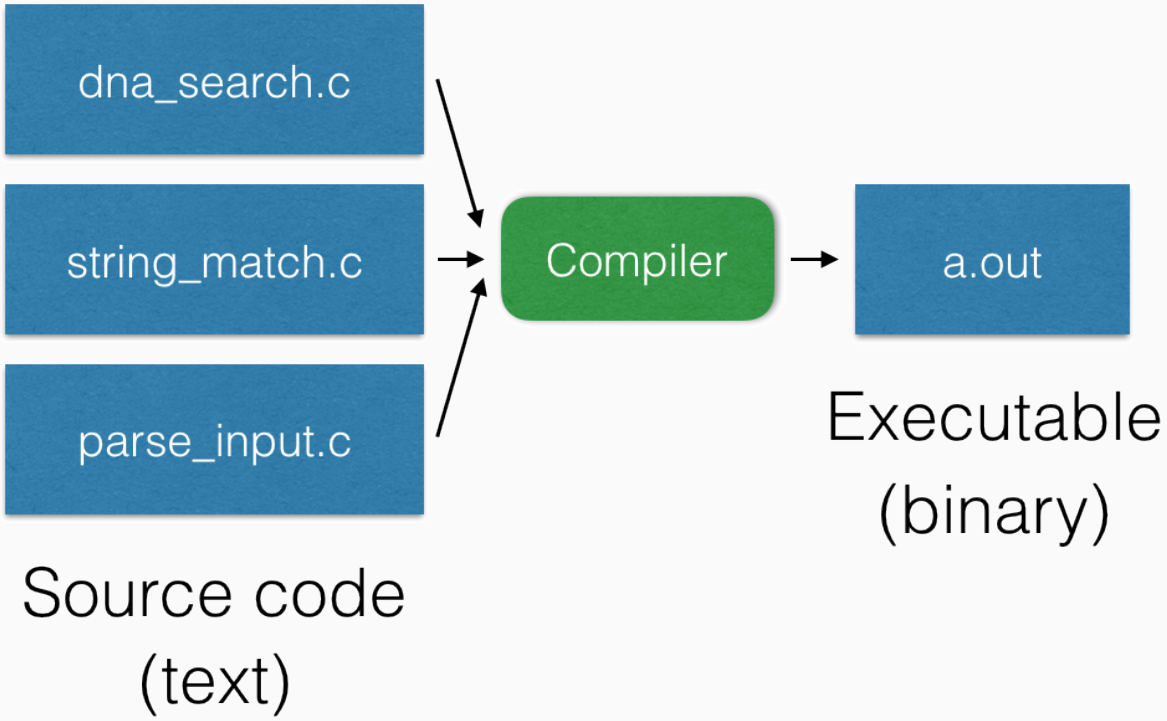


Basic C/C++ programming workflow



- `emacs dna_search.c` (then edit, save, exit)
- `emacs string_match.c` (then edit, save, exit)
- `emacs parse_input.c` (then edit, save, exit)

Basic C/C++ programming workflow



- `gcc dna_search.c string_match.c, parse_input.c -std=c99 -pedantic -Wall -Wextra`

Inside the compiler

Step 1: preprocessor

- Gather relevant source code
  - Could be spread across source files that "include" & depend on each other
- Process the directives that start with #
  - We'll see `#define`, `#include`

Step 2: compiler

- Turn human-readable *source code* into *object code*
- Might yield warnings & errors if your code has mistakes

Inside the compiler

Step 3: linker

- Gather relevant *object code* into a single executable file
- Might yield warnings & errors if relevant code is missing, there's a naming conflict, etc

Inside the compiler

Between source code and machine code is a human-readable version of the machine code called “assembly language”

```
_main:                                  ## @main
    .cfi_startproc
## BB#0:
    pushq   %rbp
Ltmp0:
    .cfi_def_cfa_offset 16
Ltmp1:
    .cfi_offset %rbp, -16
    movq    %rsp, %rbp
Ltmp2:
    .cfi_def_cfa_register %rbp
    subq    $16, %rsp
    leaq    L_.str(%rip), %rdi
    movl    $0, -4(%rbp)
    callq   _puts
```

Hello world

Create a directory where you will store your work for today

Go to that directory and type `emacs hello_world.c`

- Or use your favorite `emacs` alternative

Hello world

```c
#include <stdio.h>

// Print "Hello world!" followed by newline and exit
int main() {
    printf("Hello world!\n");
    return 0;
}
```

After `emacs hello_world.c`, type above into the editor

`Ctrl-x Ctrl-s` to save

`Ctrl-x Ctrl-c` to quit

Hello world

Compile:

```
$ gcc hello_world.c -std=c99 -pedantic -Wall -Wextra
$ ls -l a.out
-rwxr-xr-x 1 root root 8176 May  9 11:50 a.out
```

The result is an executable program called `a.out`

(Later we'll see how to give it a better name)

Hello world

If `a.out` is in our current directory, we run it by typing `./a.out`

```
$ ./a.out
Hello world!
```

Hello world

```c
#include <stdio.h>

// Print "Hello world!" followed by newline and exit
int main() {
    printf("Hello world!\n");
    return 0;
}
```

`#include` is a preprocessor directive

- Indicates a library to use, like `import` in Java or Python

`main` is a function, every program has exactly one `main`

`int` is its return type

`()` says that `main` takes no parameters

Hello world

```c
#include <stdio.h>

// Print "Hello world!" followed by newline and exit
int main() {
    printf("Hello world!\n");
    return 0;
}
```

`printf` prints a string to the terminal

- `\n` indicates a "newline" – goes to next line

`return 0` means program completed with no errors

Explanatory commenting (`// Print ...`) is good practice

Hello world

What if we omit #include <stdio.h>?:

```
// Print "Hello world!" followed by newline and exit
int main() {
    printf("Hello world!\n");
    return 0;
}
```

Hello world

Compiler prints a *compiler error* and does not generate `a.out`

```
$ gcc hello_world_err.c -std=c99 -pedantic -Wall -Wextra
hello_world_err.c: In function 'main':
hello_world_err.c:3:5: warning: implicit declaration of function 'printf'
[-Wimplicit-function-declaration]
     printf("Hello world!\n");
     ^~~~~~
hello_world_err.c:3:5: warning: incompatible implicit declaration of built-in
function 'printf'
hello_world_err.c:3:5: note: include '<stdio.h>' or provide a declaration of
'printf'
```