Compiling and Linking Overview

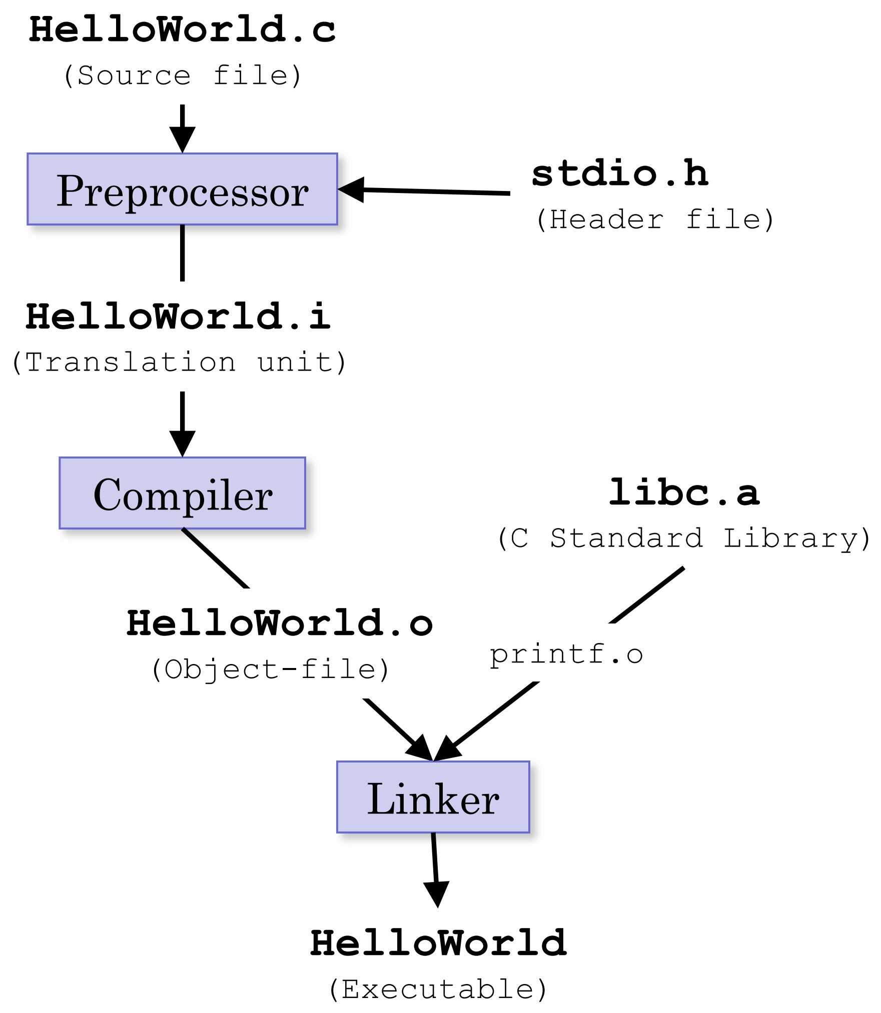
John Penn (CACI/ER7)

HelloWorld.c

```
#include <stdio.h>

int main(int argc, char* argv[]) {

   printf("Hello World\n");

   return 0;

}
```

Building a Program

```
cc      HelloWorld.c    -o HelloWorld
```



Building a Program

The *Preprocessor* is the first thing to run in a C compilation. It processes directives such as `#include`, `#define`, etc. It's just a text-processor.

Building a Program

The *Compiler* takes the output of the preprocessor and generates an object (.o) file which contains:

- Executable machine code.
- Global and static variables
- Constants and string literals
- Symbol table (<name, address>)
- Relocation records (symbolic references to external variables and functions)

Building a Program

The *Linker* combines multiple object files(.o’ s) and/or libraries (.a’ s) to produce an executable file. It does so by:

- Resolving references to external symbols,
- Assigning final addresses to functions and variables,
- Updating code and data to reflect new addresses (a process called relocation).

Object files (.o)

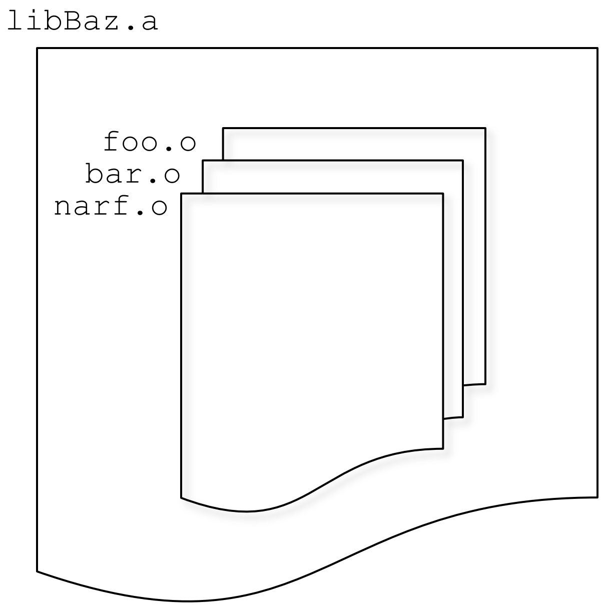
- Executable machine code.
- Global and static variables
- Constants and string literals
- Symbol table
- Relocation records

```
HelloWorld.o

0100101010010101011001001010
10100101010111010101
none
"Hello World\n"
Has    : main, addr of main
Needs  : printf()
```

Relocation records are the “blanks that need to be filled in”.

Libraries (.a files)

Libraries are just files that contain collections of .o files.



Linking

The linker looks at all of the symbol tables and the relocation record tables of a collection of object files and libraries, combines them and fills in the blanks. If all of the relocation records can be resolved then the link is successful.

In order for the resulting linked file to be executable, it' s symbol table must define `main()`.

A difference between C and C++ linking

C++ function-call relocation records need to know whether they are calling a C-functions or a C++ functions. The default is to assume that a call is to a C++ function. If a function is in fact a C-function then the user needs to indicate that it ‘s a C-function using “extern C”.
For example:

```
#ifdef __cplusplus
extern "C" {
#endif

My_awesome_C_func( void* all_the_knowledge );

#ifdef __cplusplus
}
#endif
```

Surrounding your C-function prototypes with “#ifdef __cplusplus” stuff is a good practice so that it can be used from both C or C++.

The End