

Why?

1

You seek performance

1

You seek performance

“zero-overhead principle”

2

You seek to interface
directly with hardware

3

That's kinda it

C

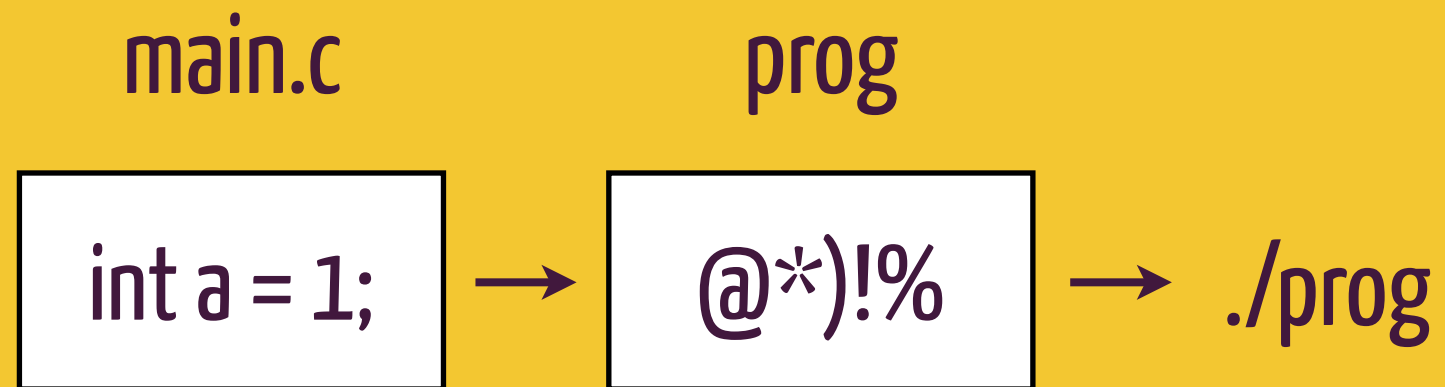
a nice way to avoid writing
assembly language directly

Today:
Compilation
Pipeline

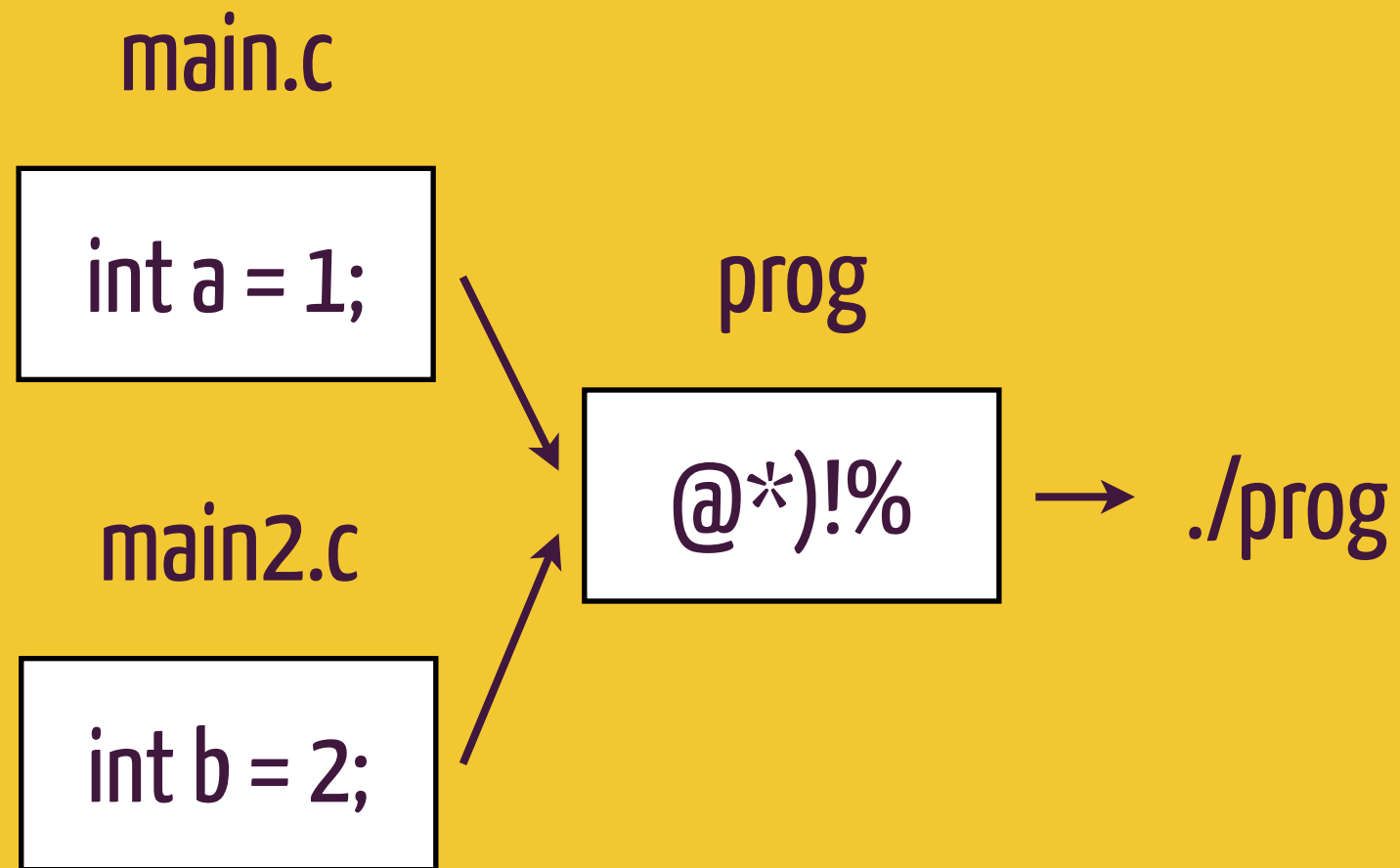
Source Code



Program



```
gcc -o prog main.c
```



```
gcc -o prog main.c main2.c
```

```
$ gcc -o prog main.c
$ ./prog
Hello, World!
$
```

```
$ gcc -o prog main.c
```

```
$ ./prog
```

```
Hello, World!
```

```
$
```

DONE

“To debug the sausage, one must see how it is made.”

—Someone, probably

Main.java

```
int a = 1;
```

Main.class

```
%!(*@
```

java Main

Main.java

```
int a = 1;
```

Main.class

```
%!(*@
```

java Main

main.py

```
a = 1
```

main.pyc

```
%!(*@
```

python main.py

Main.java

```
int a = 1;
```

Main.class

```
%!(*@
```

java Main

main.py

```
a = 1
```

main.pyc

```
%!(*@
```

python main.py

main.c

```
int a = 1;
```

```
int a = 1;
```

main.o

```
%!(*@
```

prog

```
@*)!%
```

./prog

main.c

```
int a = 1;
```



```
int a = 1;
```



main.o

```
%(**@
```



prog

```
@**)!%
```



./prog

Pre-Process

main.c

```
int a = 1;
```



```
int a = 1;
```



main.o

```
%( * @
```



prog

```
@* ) ! %
```



./prog

Pre-Process

Compile

main.c

```
int a = 1;
```



```
int a = 1;
```



main.o

```
%(**@
```



prog

```
@**)!%
```



./prog

Pre-Process

Compile

Link

main.c

```
int a = 1;
```



```
int a = 1;
```



main.o

```
%( *@
```



prog

```
@*)!%
```



./prog

main2.o

```
%( *@
```



Pre-Process

Compile

Link

Pre-Process

Compile

Link

Pre-Process

Compile

Link

`#include`

`#define`

`#ifdef`

rimshot.txt

```
ba-dum chh
```

joke.txt

```
A man walks into  
a bar. Ouch!  
#include "rimshot.txt"
```

```
cpp -P joke.txt
```


output:

```
A man walks into  
a bar. Ouch!  
ba-dum chh
```

```
cpp -P joke.txt
```

double.py

```
#define fosho def
#define kthx return
#define wutz print

fosho double(x):
    kthx x * 2
wutz double(6)
```

These are called
“macros”

cpp -P double.py

output:

```
def double(x):  
    return x * 2  
print double(6)
```

cpp -P double.py

output:

```
def double(x):  
    return x * 2  
print double(6)
```

cpp -P double.py | python

12

AWESOME

cpp -P double.py

beer.txt

```
#define beer(x) x bottles of \  
beer on the wall...
```

```
beer(99)
```

```
beer(98)
```

```
beer(97)
```

```
...
```

```
cpp -P beer.txt
```

beer.txt

```
#define beer(x) x bottles of \  
beer on the wall...
```

```
beer(99)
```

```
beer(98)
```

```
beer(97)
```

```
...
```

```
cpp -P beer.txt
```

output:

```
99 bottles of beer on the wall...
98 bottles of beer on the wall...
97 bottles of beer on the wall...
...
```

```
cpp -P beer.txt
```

answer.txt

```
What's 7 times 6?  
#ifdef REVEAL  
42  
#endif
```

```
cpp -P answer.txt
```


output:

```
What's 7 times 6?
```

```
cpp -P answer.txt
```

output:

What's 7 times 6?

???

```
cpp -P answer.txt
```

```
#define REVEAL
```

```
cpp -P answer.txt
```

```
#define REVEAL
```

or:

```
cpp -P -D REVEAL answer.txt
```

```
cpp -P answer.txt
```

output:

```
What's 7 times 6?  
42
```

```
cpp -P -D REVEAL answer.txt
```

answer.txt

```
What's 7 times 6?  
#ifndef REVEAL  
42  
#endif
```

```
cpp -P answer.txt
```

(Fancy)

String Substitution

How is this used in C?

hello.c

```
#include <stdio.h>

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

gcc -E hello.c

hello.c

* angle brackets -> use the system search path

```
#include <stdio.h>

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

```
gcc -E hello.c
```

hello.c

* angle brackets -> use the system search path

```
#include <stdio.h>

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



```
gcc -E hello.c
```

output:

```
int printf(const char * , ...)  
__attribute__((__format__(  
__printf__, 1, 2)));  
  
int main() {  
    printf("Hello, World!\n");  
}
```

* pretending printf is all that's defined in stdio.h

```
gcc -E hello.c
```

output:

```
int printf(const char * , ...);

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

* pretending printf is all that's defined in stdio.h

```
gcc -E hello.c
```

fib.c

```
#define MAX_FIB 20
int fib[MAX_FIB];

int main() {
    fib[0] = 0;
    fib[1] = 1;
    for(int i = 2; i < MAX_FIB; i++)
        fib[i] = fib[i-1] + fib[i-2];
    return 0;
}
```

```
gcc -E fib.c
```

output:

```
int fib[20];

int main() {
    fib[0] = 0;
    fib[1] = 1;
    for(int i = 2; i < 20; i++)
        fib[i] = fib[i-1] + fib[i-2];
}
```

```
gcc -E fib.c
```

debug.c

```
#include <stdio.h>

int main() {
#ifdef DEBUG
    printf("Hello, World!\n");
#endif
    return 0;
}
```

```
gcc -DDEBUG debug.c -o debug
```


debug.c

```
#include <stdio.h>

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

```
gcc -DDEBUG debug.c -o debug
```

debug.c

```
#include <stdio.h>

int main() {
    return 0;
}
```

```
gcc debug.c -o debug
```

Pre-Process

Compile

Link

main.c

```
int a = 1;
```



```
int a = 1;
```



main.o

```
%!(*@
```



prog

```
@*)!%
```

./prog

Compile

Compile

Type-checking

Linear processing

```
def foo(a, b):  
    return a + b
```

```
foo(2, 3)  
foo("2", "3")
```


Type-checking

```
def vegetable(day):  
    if day != "Tuesday":  
        return "tomato"  
    else:  
        return 1000
```

Python says: no problem

Everything has
a single, fixed type


```
int vegetable(char *day) {  
    if (strcmp(day, "Tuesday") != 0) {  
        return "tomato";  
    } else {  
        return 1000;  
    }  
}
```



```
int (char*) {  
    if (int) {  
        return char*; ←  
    } else {  
        return int;  
    }  
}
```

Type-checking

```
int reptile() {  
    return "frog";  
}
```

reptile.c: In function 'reptile':
reptile.c:2:5: warning: **return makes integer from pointer without a cast**

Variable Declarations

```
int foo;  
float foo;  
double foo;  
char foo;
```

```
int foo[42];  
int *foo;  
struct Bar foo;
```

```
reptile.c: In function 'reptile':  
reptile.c:2:5: warning: return makes  
integer from pointer without a cast
```

```
int a = 4;  
float b = (float)a;
```

Function Declarations

```
double fmin(double, double);
```



return type



argument types



Function Declarations

```
void exit(int);
```



returns nothing

```
int rand(void);
```



takes no arguments

```
int foo(int a, int b) {  
    return a + b;  
}
```



Compile

Type-checking

Linear processing

Linear processing


(just a small note)



You can only use
what's declared above

```
int main() {  
    printf("%d\n", answer());  
    return 0;  
}
```

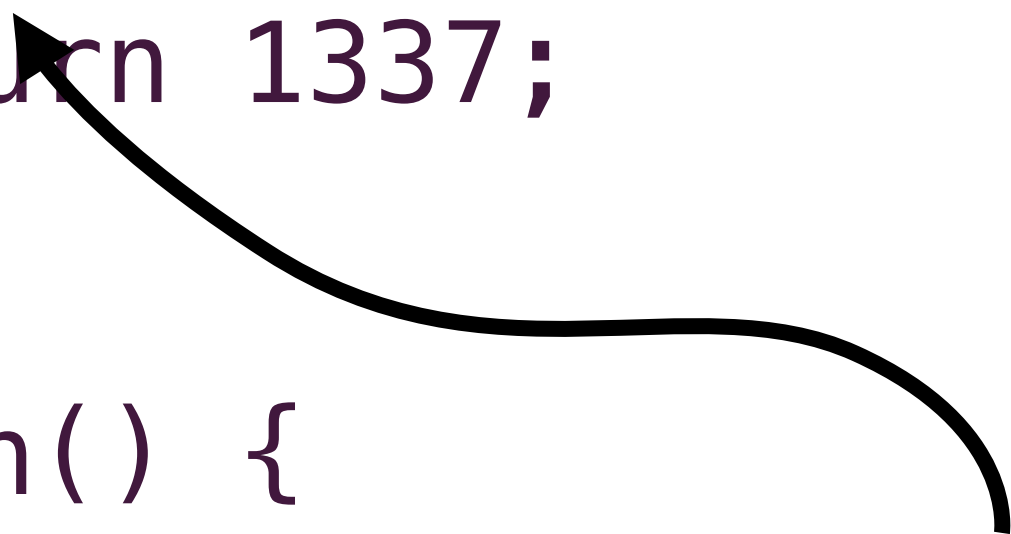
```
int answer() {  
    return 1337;  
}
```



```
int main() {  
    printf("%d\n", answer());  
    return 0;  
}  
  
int answer() {  
    return 1337;  
}
```

```
int answer() {  
    return 1337;  
}
```

```
int main() {  
    printf("%d\n", answer());  
    return 0;  
}
```




```
int answer(); ← declaration
```

```
int main() {  
    printf("%d\n", answer());  
    return 0;  
}
```

```
int answer() { ← definition  
    return 1337;  
}
```

```
int answer(); ← declaration
```

```
int main() {  
    printf("%d\n", answer());  
    return 0;  
}
```



```
int answer() { ← definition  
    return 1337;  
}
```



```
int answer(); ← declaration
```

```
#include "answer.h"
```

```
int main() {  
    printf("%d\n", answer());  
    return 0;  
}
```

```
int answer() { ← definition  
    return 1337;  
}
```

Pre-Process

Compile

Link

main.o

```
int main()
```

```
int answer()
```



main.o

```
int main()
```

```
answer.c: In function 'main':
answer.c:4: warning: implicit declaration of
function 'answer'
Undefined symbols for architecture x86_64:
  "_answer", referenced from:
      _main in ccuzmRrm.o
ld: symbol(s) not found for architecture x86_64
collect2: ld returned 1 exit status
```

```
answer.c: In function 'main':
answer.c:4: warning: implicit declaration of
function 'answer'
Undefined symbols for architecture x86_64:
  "_answer", referenced from:
      _main in ccuzmRrm.o
ld: symbol(s) not found for architecture x86_64
collect2: ld returned 1 exit status
```

```
answer.c: In function 'main':
answer.c:4: warning: implicit declaration of
function 'answer'
Undefined symbols for architecture x86_64:
  "_answer", referenced from:
      _main in ccuzmRrm.o
ld: symbol(s) not found for architecture x86_64
collect2: ld returned 1 exit status
```

**Compiler: “I don’t know what answer is.
I’ll assume it returns an int.”**

```
answer.c: In function 'main':  
answer.c:4: warning: implicit declaration of  
function 'answer'
```

```
Undefined symbols for architecture x86_64:
```

```
  "_answer", referenced from:
```

```
    _main in ccuzmRrm.o
```

```
ld: symbol(s) not found for architecture x86_64
```

```
collect2: ld returned 1 exit status
```



```
answer.c: In function 'main':  
answer.c:4: warning: implicit declaration of  
function 'answer'
```

```
Undefined symbols for architecture x86_64:
```

```
  "_answer", referenced from:
```

```
    _main in ccuzmRrm.o
```

```
ld: symbol(s) not found for architecture x86_64
```

```
collect2: ld returned 1 exit status
```

**Linker: “I looked in all the object files,
but I couldn’t find answer.”**

main.o

```
int main()
```

main.o

```
int main()
```

answer.o

```
int answer()
```

main.o

```
int main()
```

```
int main() {  
    printf("%d\n", answer());  
    return 0;  
}
```

answer.o

```
int answer()
```

```
int answer() {  
    return 1337;  
}
```

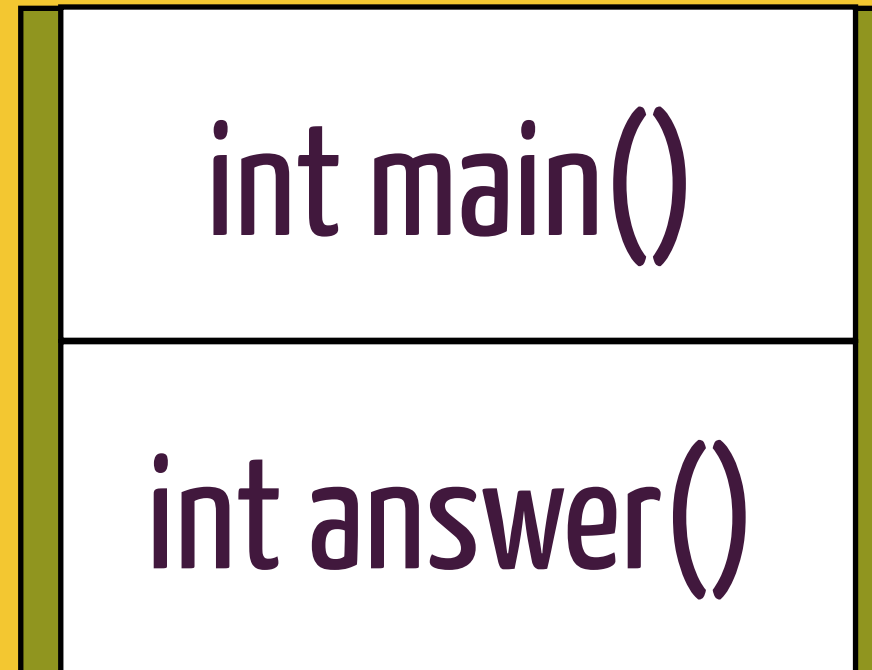
main.o

```
int main()
```

answer.o

```
int answer()
```

prog



```
gcc -o prog main.c answer.c
```

```
answer.c: In function 'main':  
answer.c:4: warning: implicit declaration of  
function 'answer'
```

```
answer.c: In function 'main':  
answer.c:4: warning: implicit declaration of  
function 'answer'
```

**Compiler: “I don’t know what answer is.
I’ll assume it returns an int.”**

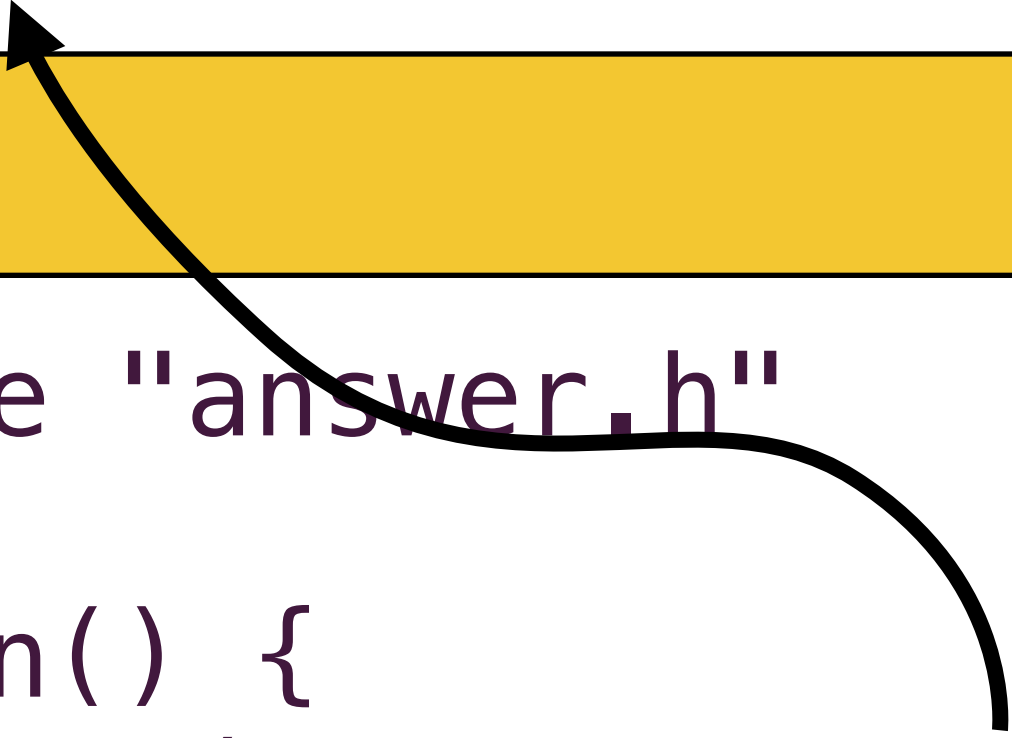
answer.h

```
int answer();
```

main.c

```
#include "answer.h"

int main() {
    printf("%d\n", answer());
    return 0;
}
```



answer.h

```
int answer();
```

answer.c

```
#include "answer.h"

int answer() {
    return 1337;
}
```

Summary

answer.h

```
int answer();
```

main.c

```
#include "answer.h"

int main() {
    printf("%d\n", answer());
    return 0;
}
```

Preprocess: gcc -E main.c

```
int answer();  
  
int main() {  
    printf("%d\n", answer());  
    return 0;  
}
```

Compile: gcc -c main.c main.c

main.o

```
%! (*@
```

answer.o

```
%! (*@
```

Link: gcc -o prog main.o main.o

prog

%! (*@

Pre-Process

Compile

Link

MIT OpenCourseWare
<http://ocw.mit.edu>

6.S096 Introduction to C and C++
IAP 2013

For information about citing these materials or our Terms of Use, visit: <http://ocw.mit.edu/terms>.