Introduction to C and C++
Why?
You seek performance
You seek performance

“zero-overhead principle”
You seek to interface directly with hardware
That’s kinda it
C

a nice way to avoid writing assembly language directly
C++ responds to the demands of maintaining large C projects
C++11 responds to the demands of maintaining large C++ projects
Maintain power and flexibility of what came before
Today:
Compilation Pipeline
Source Code

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

```
int a = 1;
```
```c
int a = 1;

int b = 2;
```

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

```
prog
@*)!%

./prog
```
$ gcc -o prog main.c
$ ./prog
Hello, World!
$
$ gcc -o prog main.c
$ ./prog
Hello, World!
$
“To debug the sausage, one must see how it is made.”

—Someone, probably
int a = 1;
Pre-Process
Compile
Link
#include
#define
#if
ifdef
#endif
rimshot.txt

ba–dum chh

joke.txt

A man walks into a bar. Ouch!

```cpp
#include "rimshot.txt"
```

cpp -P joke.txt
A man walks into a bar. Ouch!
ba-dum chh
double.py

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

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

These are called “macros”
def double(x):
    return x * 2
print double(6)
def double(x):
    return x * 2
print double(6)

cpp -P double.py | python

12
```cpp
#define beer(x) x bottles of \ beer on the wall...

beer(99)
beer(98)
beer(97)
...
```
#define beer(x) x bottles of \ beer on the wall...

beer(99)
beer(98)
beer(97)
...

cpp -P beer.txt
99 bottles of beer on the wall...
98 bottles of beer on the wall...
97 bottles of beer on the wall...
...

output:

cpp -P beer.txt
What’s 7 times 6?
#ifdef REVEAL
42
#endif
output:

What’s 7 times 6?

cpp -P answer.txt
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
What's 7 times 6?

```cpp
#include REVEAL
42
#endif
```

cpp -P answer.txt
(Fancy)  
String Substitution
How is this used in C?
#include <stdio.h>

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

gcc -E hello.c
#include <stdio.h>

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

gcc -E hello.c
#include <stdio.h>

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

gcc -E hello.c
int printf(const char *, ...)  
__attribute__((__printf__, 1, 2)));

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

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

gcc -E hello.c
output:

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

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

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

gcc -E hello.c
#include is not

import pickle

import java.io.*;
```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;
}
```

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];
}
```c
#include <stdio.h>

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

gcc -DDEBUG debug.c -o debug
#include <stdio.h>

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

gcc -DDDEBUG debug.c -o debug
debug.c

#include <stdio.h>

int main() {
    return 0;
}

gcc debug.c -o debug
Pre-Process

Compile

Link
int a = 1;

Compile

main.c

int a = 1;

main.o

%!(*)@

prog

@*)(!%

./prog
Compile

Type-checking

Linear processing
int reptile() {
    return "frog";
}

Type-checking
Type-checking

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

`reptile.c: In function ‘reptile’: reptile.c:2:5: warning: return makes integer from pointer without a cast`
def vegetable(day):
    if day != "Tuesday":
        return "tomato"
    else:
        return 1000
def vegetable(day):
    if day != "Tuesday":
        return "tomato"
    else:
        return 1000

Python says: no problem
int reptile() {
    return "frog";
}
int () {
    return char*;
}

int vegetable(char *day) {
    if (strcmp(day, "Tuesday") != 0) {
        return "tomato";
    } else {
        return 1000;
    }
}
int (char*) {
    if (int){
        return char*;
    } else {
        return int;
    }
}
```c
int (char*) {
    if (int){
        return char*;
    } else {
        return int;
    }
}
```
int (char*) {
    if (int){
        return char*;
    } else {
        return int;
    }
}
Everything has a single, fixed type
def foo(a, b):
    return a + b

foo(2, 3)
foo("2", "3")
Variable Declarations

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

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

double fmin(double, double);
Function Declarations

void exit(int);

returns nothing

int rand(void);

takes no arguments
int foo(int a, int b){
    return a + b;
}
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;
all theoretical casts
allowed casts

```
int
float
double
char
int[]
int*
void (*)(int)
struct X
```

```
int
float
double
char
int[]
int*
void (*)(int)
struct X
```
allowed casts

int float double char int[] int*
void (*f)(int) struct X

int float double char int[] int*
void (*f)(int) struct X
implicit casts

int      int
float    float
double   double
char     char
int[]    int[]
int*     int*
void (*f)(int) void (*f)(int)
struct X struct X
implicit casts

int → int
float → float
double → double
char → char
int[] → int[]
int* → int*
void (*f)(int) → void (*f)(int)
struct X → struct X
Implicit casts:

- `int` to `int`
- `float` to `float`
- `double` to `double`
- `char` to `char`
- `int[]` to `int[]`
- `int*` to `int*`
- `void (*f)(int)` to `void (*f)(int)`
- `struct X` to `struct X`
implicit casts

int → int
float → float
double → double
char → char
int[] → int[]
int* → int*
void (*f)(int) → void (*f)(int)
struct X → struct X
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();

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

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

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

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

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

int answer() {  // definition
    return 1337;
}
```c
int answer();

#include "answer.h"

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

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

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

int answer() {
    return 1337;
}
```
gcc -o prog main.c answer.c

prog

int main()

int 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;
}

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
Pre-Process

Compile

Link