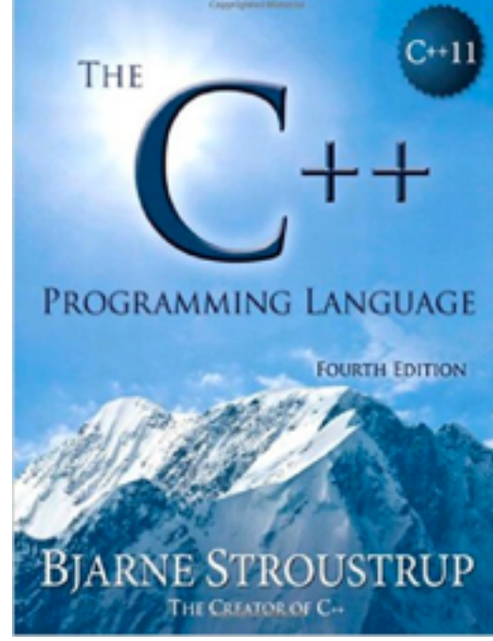
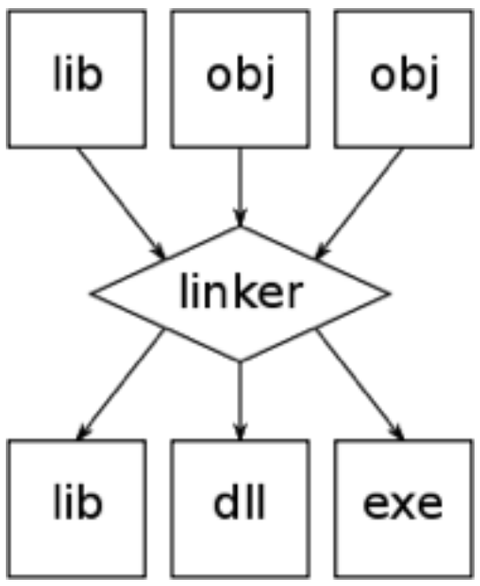
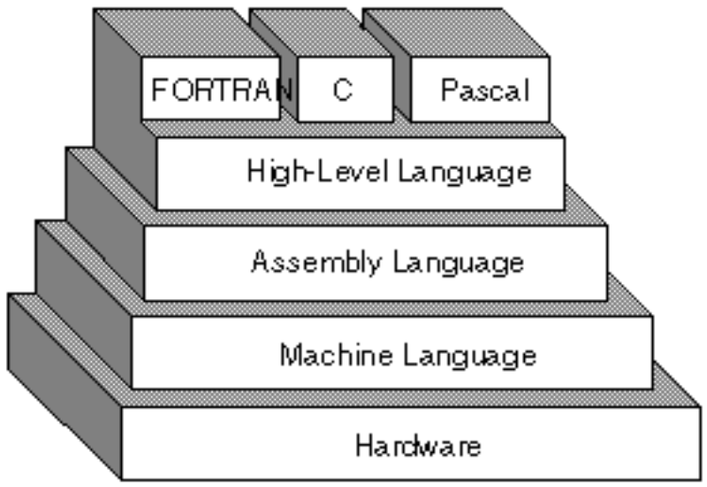


Principles of Programming Languages

Kristopher Micinski





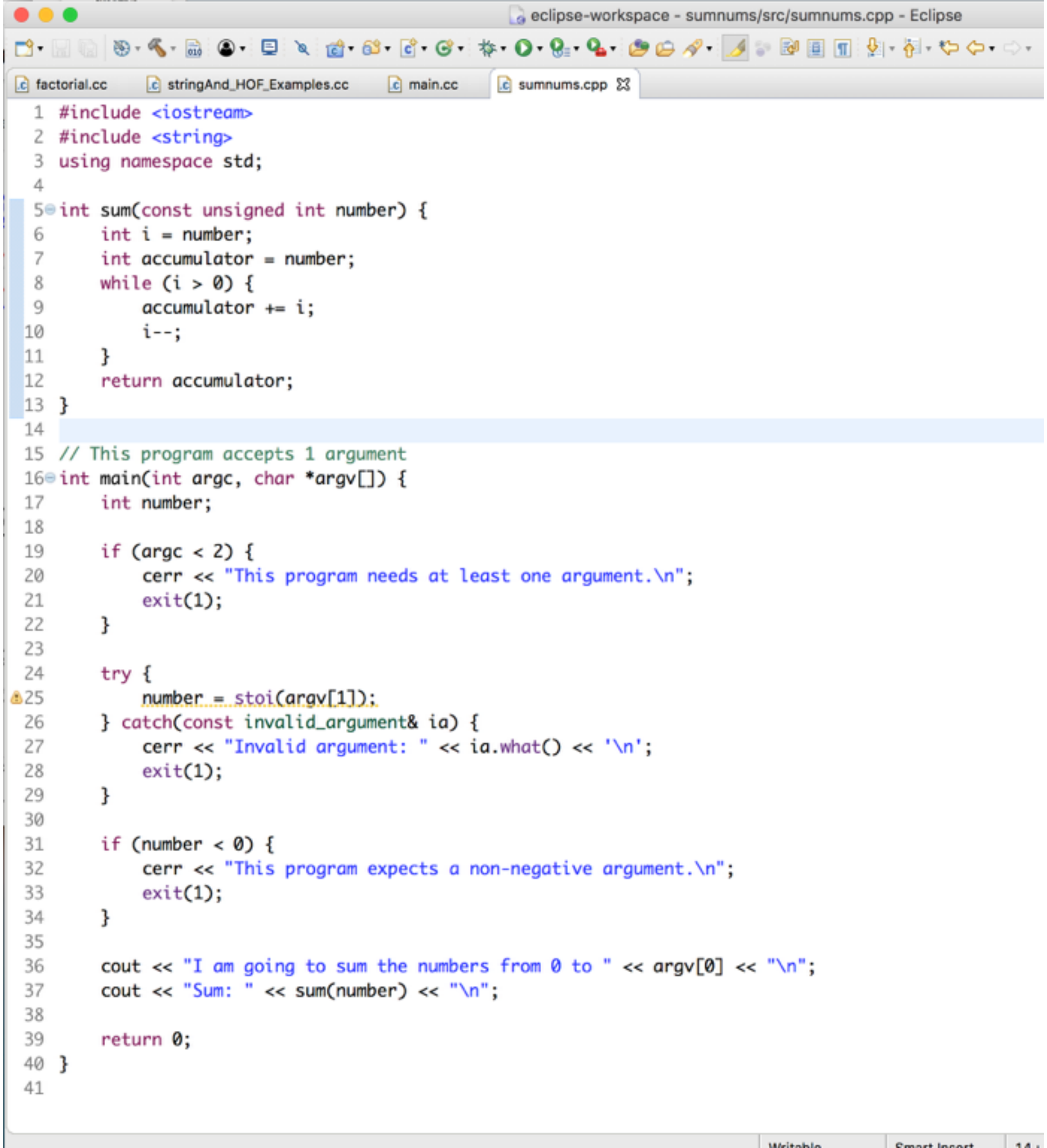
This class is about understanding how programs work

To do this, we're going to have to learn how a computer works

Here's a program in a new language, C++

C++ is a *compiled* language

A translator (compiler) turns C++ into binary code


eclipse-workspace - sumnums/src/sumnums.cpp - Eclipse
factorial.cc
stringAnd_HOF_Examples.cc
main.cc
sumnums.cpp
#include <iostream>
#include <string>
using namespace std;
int sum(const unsigned int number) {
int i = number;
int accumulator = number;
while (i > 0) {
accumulator += i;
i--;
}
return accumulator;
}
// This program accepts 1 argument
int main(int argc, char *argv[]) {
int number;
if (argc < 2) {
cerr << "This program needs at least one argument.\n";
exit(1);
}
try {
number = stoi(argv[1]);
} catch(const invalid_argument& ia) {
cerr << "Invalid argument: " << ia.what() << '\n';
exit(1);
}
if (number < 0) {
cerr << "This program expects a non-negative argument.\n";
exit(1);
}
cout << "I am going to sum the numbers from 0 to " << argv[0] << "\n";
cout << "Sum: " << sum(number) << "\n";
return 0;
}

eclipse-workspace - sumnums/src/sumnums.cpp - Eclipse

factorial.cc | stringAnd_HOF_Examples.cc | main.cc | sumnums.cpp

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int sum(const unsigned int number) {
6      int i = number;
7      int accumulator = number;
8      while (i > 0) {
9          accumulator += i;
10         i--;
11     }
12     return accumulator;
13 }
14
15 // This program accepts 1 argument
16 int main(int argc, char *argv[]) {
17     int number;
18
19     if (argc < 2) {
20         cerr << "This program needs at least one argument.\n";
21         exit(1);
22     }
23
24     try {
25         number = stoi(argv[1]);
26     } catch(const invalid_argument& ia) {
27         cerr << "Invalid argument: " << ia.what() << '\n';
28         exit(1);
29     }
30
31     if (number < 0) {
32         cerr << "This program expects a non-negative argument.\n";
33         exit(1);
34     }
35
36     cout << "I am going to sum the numbers from 0 to " << argv[0] << "\n";
37     cout << "Sum: " << sum(number) << "\n";
38
39     return 0;
40 }
41
```

Main procedure

Program starts here

eclipse-workspace - sumnums/src/sumnums.cpp - Eclipse

factorial.cc | stringAnd_HOF_Examples.cc | main.cc | sumnums.cpp

```cpp
#include <iostream>
#include <string>
using namespace std;

int sum(const unsigned int number) {
    int i = number;
    int accumulator = number;
    while (i > 0) {
        accumulator += i;
        i--;
    }
    return accumulator;
}

// This program accepts 1 argument
int main(int argc, char *argv[]) {
    int number;

    if (argc < 2) {
        cerr << "This program needs at least one argument.\n";
        exit(1);
    }

    try {
        number = stoi(argv[1]);
    } catch(const invalid_argument& ia) {
        cerr << "Invalid argument: " << ia.what() << '\n';
        exit(1);
    }

    if (number < 0) {
        cerr << "This program expects a non-negative argument.\n";
        exit(1);
    }

    cout << "I am going to sum the numbers from 0 to " << argv[0] << "\n";
    cout << "Sum: " << sum(number) << "\n";

    return 0;
}
```

Writable | Smart Insert | 14 :

sum function
(calculates sum(0 to number))

Here's a program in a new language, C++

C++ is a *compiled* language

A translator (compiler) turns C++ into binary code

Binary: The native language of the processor

- Modern processors are *very* fast
- (m/b)illions of *instructions* per sec

Processors execute a small number of *very basic* instructions

MOV r1, r2 ADD r1, r2,r3

IFZERO r1, +20

These instructions written in a binary encoding (**Why**?)

Binary: The native language of the processor

- Modern processors are *very* fast
- (m/b)illions of *instructions* per sec

Processors execute a small number of *very basic* instructions

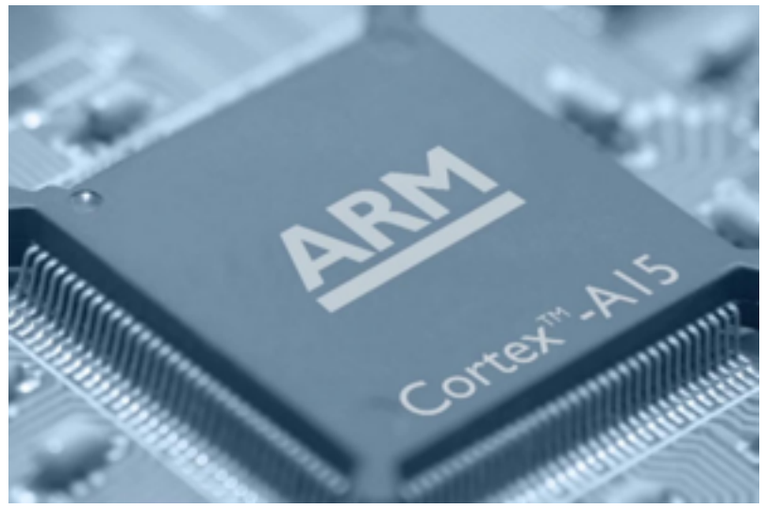
MOV r1, r2 ADD r1, r2,r3

IFZERO r1, +20

These instructions written in a binary encoding (Why?)

Compact representation Quick to decode and execute

Thousands of different processors



Each speaks a different language

Called its *architecture*

Different versions of architecture add features, etc..

eclipse-workspace - sumnums/src/sumnums.cpp - Eclipse

factorial.cc | stringAnd_HOF_Examples.cc | main.cc | sumnums.cpp

```cpp
#include <iostream>
#include <string>
using namespace std;

int sum(const unsigned int number) {
    int i = number;
    int accumulator = number;
    while (i > 0) {
        accumulator += i;
        i--;
    }
    return accumulator;
}

// This program accepts 1 argument
int main(int argc, char *argv[]) {
    int number;

    if (argc < 2) {
        cerr << "This program needs at least one argument.\n";
        exit(1);
    }

    try {
        number = stoi(argv[1]);
    } catch(const invalid_argument& ia) {
        cerr << "Invalid argument: " << ia.what() << '\n';
        exit(1);
    }

    if (number < 0) {
        cerr << "This program expects a non-negative argument.\n";
        exit(1);
    }

    cout << "I am going to sum the numbers from 0 to " << argv[0] << "\n";
    cout << "Sum: " << sum(number) << "\n";

    return 0;
}
```

So I need to turn this into something my i7 speaks…

To do that, I use a *compiler*

"Compile a file named sumnums.cpp, and output an executable file named sumnums"

```
g++ sumnums.cpp -o sumnums
```

"Compile a file named sumnums.cpp, and output an executable file named sumnums"

```
g++ sumnums.cpp -o sumnums
```

(*Ton* of options here, especially for large projects with complex configs / multifiles)

eclipse-workspace - sumnums/src/sumnums.cpp - Eclipse

factorial.cc | stringAnd_HOF_Examples.cc | main.cc | sumnums.cpp

```cpp
#include <iostream>
#include <string>
using namespace std;

int sum(const unsigned int number) {
    int i = number;
    int accumulator = number;
    while (i > 0) {
        accumulator += i;
        i--;
    }
    return accumulator;
}

// This program accepts 1 argument
int main(int argc, char *argv[]) {
    int number;

    if (argc < 2) {
        cerr << "This program needs at least one argument.\n";
        exit(1);
    }

    try {
        number = stoi(argv[1]);
    } catch(const invalid_argument& ia) {
        cerr << "Invalid argument: " << ia.what() << '\n';
        exit(1);
    }

    if (number < 0) {
        cerr << "This program expects a non-negative argument.\n";
        exit(1);
    }

    cout << "I am going to sum the numbers from 0 to " << argv[0] << "\n";
    cout << "Sum: " << sum(number) << "\n";

    return 0;
}
```

Writable | Smart Insert | 14 :

Compiler

```
87654321  0011 2233 4455 6677 8899 aabb ccdd eeff  0123456789abcdef
00000000: cffa edfe 0700 0001 0300 0000 0100 0000  ................
00000010: 0400 0000 5002 0000 0020 0000 0000 0000  ....P.... ......
00000020: 1900 0000 d801 0000 0000 0000 0000 0000  ................
00000030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000040: 6812 0000 0000 0000 7002 0000 0000 0000  h.......p.......
00000050: 6812 0000 0000 0000 0700 0000 0700 0000  h...............
00000060: 0500 0000 0000 0000 5f5f 7465 7874 0000  ........__text..
00000070: 0000 0000 0000 0000 5f5f 5445 5854 0000  ........__TEXT..
00000080: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000090: 0b0e 0000 0000 0000 7002 0000 0400 0000  ........p.......
000000a0: d814 0000 3a00 0000 0004 0080 0000 0000  ....:...........
000000b0: 0000 0000 0000 0000 5f5f 6763 635f 6578  ........__gcc_ex
000000c0: 6365 7074 5f74 6162 5f5f 5445 5854 0000  cept_tab__TEXT..
000000d0: 0000 0000 0000 0000 0c0e 0000 0000 0000  ................
000000e0: 3c01 0000 0000 0000 7c10 0000 0200 0000  <.......|.......
000000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000100: 0000 0000 0000 0000 5f5f 6373 7472 696e  ........__cstrin
00000110: 6700 0000 0000 0000 5f5f 5445 5854 0000  g....._TEXT..
00000120: 0000 0000 0000 0000 480f 0000 0000 0000  ........H.......
00000130: 8b00 0000 0000 0000 b811 0000 0000 0000  ................
00000140: 0000 0000 0000 0000 0200 0000 0000 0000  ................
00000150: 0000 0000 0000 0000 5f5f 636f 6d70 6163  ........__compac
00000160: 745f 756e 7769 6e64 5f5f 4c44 0000 0000  t_unwind__LD....
00000170: 0000 0000 0000 0000 d80f 0000 0000 0000  ................
00000180: 0001 0000 0000 0000 4812 0000 0300 0000  ........H.......
00000190: a816 0000 0e00 0000 0000 0002 0000 0000  ................
000001a0: 0000 0000 0000 0000 5f5f 6568 5f66 7261  ........__eh_fra
000001b0: 6d65 0000 0000 0000 5f5f 5445 5854 0000  me......__TEXT..
000001c0: 0000 0000 0000 0000 d810 0000 0000 0000  ................
000001d0: 9001 0000 0000 0000 4813 0000 0300 0000  ........H.......
000001e0: 1817 0000 0100 0000 0b00 0068 0000 0000  ...........h....
000001f0: 0000 0000 0000 0000 2400 0000 1000 0000  ........$.......
00000200: 000c 0a00 0000 0000 0200 0000 1800 0000  ................
00000210: 2017 0000 2300 0000 5019 0000 2005 0000   ...#...P... ...
00000220: 0b00 0000 5000 0000 0000 0000 0300 0000  ....P...........
00000230: 0300 0000 0900 0000 0c00 0000 1700 0000  ................
00000240: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000250: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000260: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000270: 5548 89e5 897d fc8b 7dfc 897d f88b 7dfc  UH...}..}..}..}.
00000280: 897d f483 7df8 000f 8d17 0000 008b 45f8  .}..}.........E.
00000290: 8345 f489 45f4 8b45 f883 c0ff 8945 f8e9  .E..E..E.....E..
000002a0: dfff ffff 8b45 f45d c30f 1f80 0000 0000  .....E.]........
000002b0: 5548 89e5 4881 ecd0 0000 00c7 45bc 0000  UH..H.......E...
000002c0: 0000 897d b848 8975 b883 7db8 010f 8d24  ...}.H.u..}....$
000002d0: 0000 0048 8b3d 0000 0000 488d 35d7 0e00  ...H.=....H.5...
000002e0: 00e8 0000 0000 bf01 0000 0048 8985 78ff  ...........H..x.
000002f0: ffff e800 0000 0048 8b45 b048 8b00 488d  .......H.E.H..H.
00000300: 4d90 4889 4dc8 4889 45c0 488b 55c8 4889  M.H.M.H.E.H.U.H.
00000310: 55d8 4889 45d0 488b 45d8 4889 45e0 4889  U.H.E.H.E.H.E.H.
00000320: 45e8 4889 45f0 4889 45f8 48c7 4010 0000  E.H.E.H.E.H.@...
00000330: 0000 48c7 4008 0000 0000 48c7 0000 0000  ..H.@.....H.....
00000340: 0048 8b55 d048 89d7 4889 8d70 ffff ff48  .H.U.H..H..p...H
00000350: 8985 68ff ffff 4889 9560 ffff ffe8 0000  ..h...H..`......
00000360: 0000 488b bd68 ffff ff48 8bb5 60ff ffff  ..H..h...H..`...
00000370: 4889 c2e8 0000 0000 4531 c044 89c6 ba0a  H.......E1.D....
00000380: 0000 0048 8bbd 70ff ffff e800 0000 0089  ...H..p.........
00000390: 855c ffff ffe9 0000 0000 488d 7d90 e800  .\........H.}...
000003a0: 0000 008b 855c ffff ff89 45ac 837d ac00  .....\....E..}..
000003b0: 0f8d 4000 0000 488b 3d00 0000 0048 8d35  ..@...H.=....H.5
000003c0: 1f0e 0000 e800 0000 00bf 0100 0000 4889  ..............H.
000003d0: 8550 ffff ffe8 0000 0000 89d1 4889 4588  .P..........H.E.
000003e0: 894d 8448 8d7d 90e8 0000 0000 e900 0000  .M.H.}..........
000003f0: 00e9 8b00 0000 488b 3d00 0000 0048 8d35  ......H.=....H.5
00000400: 0e0e 0000 e800 0000 0048 8b75 b048 8b36  .........H.u.H.6
00000410: 4889 c7e8 0000 0000 488d 351c 0e00 0048  H.......H.5....H
00000420: 89c7 e800 0000 0048 8b3d 0000 0000 488d  .......H.=....H.
00000430: 3508 0e00 0048 8985 48ff ffff e800 0000  5....H..H.......
00000440: 008b 7dac 4889 8540 ffff ffe8 0000 0000  ..}.H..@........
00000450: 488b bd40 ffff ff89 c6e8 0000 0000 488d  H..@..........H.
00000460: 35d6 0d00 0048 89c7 e800 0000 0031 c948  5....H.......1.H
-UU=:----F1  sumnums.o      Top L1     (Hexl company)
```

So, the compiler turns C++ into a giant list of these instructions…

So, the compiler turns C++ into a giant list of these instructions…

These are written in *assembly*
(Human-readable binary)

Let's see what assembly the compiler generates…

```
g++ -S sumnums.cpp
```

(Note I really used:
g++ -S sumnums -fno-asynchronous-unwind-tables
This is because otherwise extra debugging overhead is inserted.)

```
        .section        __TEXT,__text,regular,pure_instructions
        .macosx_version_min 10, 12
        .globl  __Z3sumj
        .p2align        4, 0x90
__Z3sumj:                               ## @_Z3sumj
## BB#0:
        pushq   %rbp
        movq    %rsp, %rbp
        movl    %edi, -4(%rbp)
        movl    -4(%rbp), %edi
        movl    %edi, -8(%rbp)
        movl    -4(%rbp), %edi
        movl    %edi, -12(%rbp)
LBB0_1:                                 ## =>This Inner Loop Header: Depth=1
        cmpl    $0, -8(%rbp)
        jle     LBB0_3
## BB#2:                                ##   in Loop: Header=BB0_1 Depth=1
        movl    -8(%rbp), %eax
        addl    -12(%rbp), %eax
        movl    %eax, -12(%rbp)
        movl    -8(%rbp), %eax
        addl    $-1, %eax
        movl    %eax, -8(%rbp)
        jmp     LBB0_1
LBB0_3:
        movl    -12(%rbp), %eax
        popq    %rbp
        retq

        .globl  _main
        .p2align        4, 0x90
_main:                                  ## @main
Lfunc_begin0:
        .cfi_startproc
        .cfi_personality 155, ___gxx_personality_v0
        .cfi_lsda 16, Lexception0
## BB#0:
        pushq   %rbp
Ltmp24:
        .cfi_def_cfa_offset 16
Ltmp25:
        .cfi_offset %rbp, -16
        movq    %rsp, %rbp
Ltmp26:
        .cfi_def_cfa_register %rbp
        subq    $240, %rsp
        movl    $0, -68(%rbp)
        movl    %edi, -72(%rbp)
        movq    %rsi, -80(%rbp)
        cmpl    $2, -72(%rbp)
        jge     LBB1_2
## BB#1:
```

Divided up by function

```
        .section        __TEXT,__text,regular,pure_instructions
        .macosx_version_min 10, 12
        .globl  __Z3sumj
        .p2align        4, 0x90
__Z3sumj:                               ## @_Z3sumj
## BB#0:
        pushq   %rbp
        movq    %rsp, %rbp
        movl    %edi, -4(%rbp)
        movl    -4(%rbp), %edi
        movl    %edi, -8(%rbp)
        movl    -4(%rbp), %edi
        movl    %edi, -12(%rbp)
LBB0_1:                                 ## =>This Inner Loop Header: Depth=1
        cmpl    $0, -8(%rbp)
        jle     LBB0_3
## BB#2:                                ##   in Loop: Header=BB0_1 Depth=1
        movl    -8(%rbp), %eax
        addl    -12(%rbp), %eax
        movl    %eax, -12(%rbp)
        movl    -8(%rbp), %eax
        addl    $-1, %eax
        movl    %eax, -8(%rbp)
        jmp     LBB0_1
LBB0_3:
        movl    -12(%rbp), %eax
        popq    %rbp
        retq

        .globl  _main
        .p2align        4, 0x90
_main:                                  ## @main
Lfunc_begin0:
        .cfi_startproc
        .cfi_personality 155, ___gxx_personality_v0
        .cfi_lsda 16, Lexception0
## BB#0:
        pushq   %rbp
Ltmp24:
        .cfi_def_cfa_offset 16
Ltmp25:
        .cfi_offset %rbp, -16
        movq    %rsp, %rbp
Ltmp26:
        .cfi_def_cfa_register %rbp
        subq    $240, %rsp
        movl    $0, -68(%rbp)
        movl    %edi, -72(%rbp)
        movq    %rsi, -80(%rbp)
        cmpl    $2, -72(%rbp)
        jge     LBB1_2
## BB#1:
```

Divided up by function

Implementation of sum

```
        .section        __TEXT,__text,regular,pure_instructions
        .macosx_version_min 10, 12
        .globl  __Z3sumj
        .p2align        4, 0x90
__Z3sumj:                               ## @_Z3sumj
## BB#0:
        pushq   %rbp
        movq    %rsp, %rbp
        movl    %edi, -4(%rbp)
        movl    -4(%rbp), %edi
        movl    %edi, -8(%rbp)
        movl    -4(%rbp), %edi
        movl    %edi, -12(%rbp)
LBB0_1:                                 ## =>This Inner Loop Header: Depth=1
        cmpl    $0, -8(%rbp)
        jle     LBB0_3
## BB#2:                                ##   in Loop: Header=BB0_1 Depth=1
        movl    -8(%rbp), %eax
        addl    -12(%rbp), %eax
        movl    %eax, -12(%rbp)
        movl    -8(%rbp), %eax
        addl    $-1, %eax
        movl    %eax, -8(%rbp)
        jmp     LBB0_1
LBB0_3:
        movl    -12(%rbp), %eax
        popq    %rbp
        retq

        .globl  _main
        .p2a
_main:                                  ## @main
Lfunc_begin0:
        .cfi_startproc
        .cfi_personality 155, ___gxx_personality_v0
        .cfi_lsda 16, Lexception0
## BB#0:
        pushq   %rbp
Ltmp24:
        .cfi_def_cfa_offset 16
Ltmp25:
        .cfi_offset %rbp, -16
        movq    %rsp, %rbp
Ltmp26:
        .cfi_def_cfa_register %rbp
        subq    $240, %rsp
        movl    $0, -68(%rbp)
        movl    %edi, -72(%rbp)
        movq    %rsi, -80(%rbp)
        cmpl    $2, -72(%rbp)
        jge     LBB1_2
## BB#1:
```

Don’t worry that this code is hard to understand for now

(It also confuses me..)

Divided up by function

Implementation of main

I can manually transform the assembly to the binary…

```
as sumnums.s
```

```
Kyles-MacBook-Pro-2:src micinski$
Kyles-MacBook-Pro-2:src micinski$ ./sumnums.o
-bash: ./sumnums.o: cannot execute binary file
Kyles-MacBook-Pro-2:src micinski$
```

Crud…

For example: code to print to the screen

Insight: my program needs a lot of other stuff to run…

This is kept in a *library*

(But keep in mind, that's also **just code**. Nothing particularly magical)

Your code

```
	.section	__TEXT,__text,regular,pure_instructions
	.macosx_version_min 10, 12
	.globl	__Z3sumj
	.p2align	4, 0x90
__Z3sumj:                               ## @_Z3sumj
## BB#0:
	pushq	%rbp
	movq	%rsp, %rbp
	movl	%edi, -4(%rbp)
	movl	-4(%rbp), %edi
	movl	%edi, -8(%rbp)
	movl	-4(%rbp), %edi
	movl	%edi, -12(%rbp)
LBB0_1:                                 ## =>This Inner Loop Header: Depth=1
	cmpl	$0, -8(%rbp)
	jle	LBB0_3
## BB#2:                                ##   in Loop: Header=BB0_1 Depth=1
	movl	-8(%rbp), %eax
	addl	-12(%rbp), %eax
	movl	%eax, -12(%rbp)
	movl	-8(%rbp), %eax
	addl	$-1, %eax
	movl	%eax, -8(%rbp)
	jmp	LBB0_1
LBB0_3:
	movl	-12(%rbp), %eax
	popq	%rbp
	retq

	.globl	_main
	.p2align	4, 0x90
_main:                                  ## @main
Lfunc_begin0:
	.cfi_startproc
	.cfi_personality 155, ___gxx_personality_v0
	.cfi_lsda 16, Lexception0
## BB#0:
	pushq	%rbp
Ltmp24:
	.cfi_def_cfa_offset 16
Ltmp25:
	.cfi_offset %rbp, -16
	movq	%rsp, %rbp
Ltmp26:
	.cfi_def_cfa_register %rbp
	subq	$240, %rsp
	movl	$0, -68(%rbp)
	movl	%edi, -72(%rbp)
	movq	%rsi, -80(%rbp)
	cmpl	$2, -72(%rbp)
	jge	LBB1_2
## BB#1:
```

+

lstdc++

lm

etc…

Executable file

=

Question (for next time):

Can I run a program compiled for one architecture and use it on another?

Now: C++ coding

Next time:

- More C++ programming nitty-gritty
- Representing HOFs in C++
- What's your computer doing when you call a function
  - How is stack laid out, what is a StackOverflow?
    - How can we avoid them

C++ is a huge language, don't feel embarrassed if you think you know nothing. I can't think of a single smart person I know who even claims to know "most" of C++

But I do know some people who admit it's a useful and powerful tool when you use the *right* features