CSE 351: The Hw/Sw Interface

Tom Bergan, TA
Week #1
Why take 351?

- It’s required 😊

- My pitch:
  - This will (hopefully!) be an eye-opening look “under the hood”
Java

```java
void MatrixMultiply(int[][] A, int[][] B) {
    int[][] Result = new double[8][8];
    for (int i = 0; i < 8; i++)
        for (int j = 0; j < 8; j++)
            for (int k = 0; k < 8; k++)
                Result[i][j] += A[i][k] * B[k][j];
}
```

• How are numbers represented?
• How are data structures represented?
• How is memory allocated?
• What happens on a function call?
• ...
Assembly Language (x86)

```
cmpl $0, -4(%ebp)
je .L2
movl -12(%ebp), %eax
movl -8(%ebp), %edx
leal (%edx, %eax),
```
What is this section for?

• Labs

• Questions!
  - please bring questions!

• Some extensions of the lectures / textbook

• Other resources:
  - discussion board (see course webpage)
  - office hours
Today

• Introduction

• C overview

• Lab 1 quickstart
  - how to get started
  - how to compile and debug C code
Why learn C?

• For this class:
  - assignments are in C
  - C is very close to assembly language

• For yourself:
  - C code is everywhere
Hello, world!

Java

class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}

C

#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("Hello, world!\n");
    return 0;
}
C vs Java

Java

import java.xyz;

class Point {
  public int x;
  public int y;
  public int foo(int a) {
    while (x < y)
      ...
    return 42;
  }
}

C

#include “xyz.h”

struct Point {
  int x;
  int y;
};

int foo(int a) {
  while (x < y)
    ...
  return 42;
}
C: three common confusions

• Pointers

• Arrays

• The syntax for types (it can be weird...)
int a = 5;
int b = 6;
int *pa = &a; // declares a pointer to a
    // with value as the
    // address of a

“address of” operator

As a box diagram

In memory

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>100</td>
<td>5</td>
</tr>
<tr>
<td>b</td>
<td>104</td>
<td>6</td>
</tr>
<tr>
<td>pa</td>
<td>108</td>
<td>100</td>
</tr>
</tbody>
</table>

96  112
Pointers

```c
int a = 5;
int b = 6;
int *pa = &a;
```

As a box diagram

```
  a  b  pa
  5  6
```

In memory

```
   a | 100 |  5  
   b | 104 |  6  
   pa| 108 | 100 
```

Each `int` takes up 4 bytes
Pointers

```c
int a = 5;
int b = 6;
int *pa = &a;
```

Size of a pointer depends on the machine.
- Here, we assume 4 bytes (a 32-bit cpu)
- For your labs, pointers are 8 bytes! (a 64-bit cpu)
Useful tip ...

// This program will print the size of
// various data types. Try it!

#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("sizeof char: %d\n", sizeof(char));
    printf("sizeof int: %d\n", sizeof(int));
    printf("sizeof int*: %d\n", sizeof(int*));
    printf("sizeof char*: %d\n", sizeof(char*));
    return 0;
}

// Output on a 64-bit machine

sizeof char: 1
sizeof int: 4
sizeof int*: 8
sizeof char*: 8
Pointers

```c
int a = 5;
int b = 6;
int *pa = &a;

*pa = 7;  // changes value of `a` to 7
// (a == 7)
```

As a box diagram

In memory

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>100</td>
<td>7</td>
</tr>
<tr>
<td>b</td>
<td>104</td>
<td>6</td>
</tr>
<tr>
<td>pa</td>
<td>108</td>
<td>100</td>
</tr>
</tbody>
</table>
```

112
Pointers

```c
int a = 5;
int b = 6;
int *pa = &a;

*pa = 7;
b = *pa;
```

As a box diagram

```
  a  6  pa
    ^  |
    |  7
```

In memory

```
<table>
<thead>
<tr>
<th>pa</th>
<th>108</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>100</td>
<td>7</td>
</tr>
<tr>
<td>b</td>
<td>104</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>96</td>
<td></td>
</tr>
</tbody>
</table>
```
**Pointers**

```c
int a = 5;
int b = 6;
int *pa = &a;

*pa = 7;
b = *pa;  // changes value of b to 7
```

As a box diagram

```
\[
\begin{array}{ccc}
\text{a} &\rightarrow& \text{pa} \\
7 & & 7 \\
\end{array}
\]
```

In memory

```
\[
\begin{array}{ccc}
\text{a} & 100 & 7 \\
\text{b} & 104 & 7 \\
\text{pa} & 108 & 100 \\
\end{array}
\]
```
Pointers

```c
int a = 5;
int b = 6;
int *pa = &a;

*pa = 7;
b = *pa;
pa = &b;
```

As a box diagram

In memory

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>100</td>
<td>7</td>
</tr>
<tr>
<td>b</td>
<td>104</td>
<td>7</td>
</tr>
<tr>
<td>pa</td>
<td>108</td>
<td>100</td>
</tr>
</tbody>
</table>
```
Pointers

```c
int a = 5;
int b = 6;
int *pa = &a;

*pa = 7;
b = *pa;
pa = &b;  // changes pa to point at b
```
Null pointers

```c
int *p = NULL;  // points at address 0
               // like `null` in Java
int x = *p;
```

As a box diagram

In memory

```
0
1
100
104
108
112

p

X
```
Null pointers

```c
int *p = NULL;

int x = *p; // this will crash your program!
            // 0 is an invalid address
            // you get a “segmentation fault”, aka SIGSEGV
```

As a box diagram

```
X
   p
```

In memory

```
invalid address
0  4
100 104
108 112
   p
```
What are pointers good for?

• **Data structures!**
  Here’s a binary tree:

```c
struct Tree {
    int x;
    struct Tree *left;
    struct Tree *right;
};
```

```plaintext
22
7
4
2
0
0
...  
10
6
2
0
0
...  
...  
...  
```
Pass-by-value  vs. Pass-by-pointer

```c
int foo(int x) {
    return x + 1;
}

void bar(int* x) {
    *x += 1;
}

void main() {
    int x = 5;
    int y = foo(x);
    // x==5
    // y==6
    bar(&x);
    // x==6
    // y==6
}
```
C: three common confusions

• Pointers

• Arrays

• The syntax for types (it can be weird...)
### Arrays

```cpp
int a[4]; // declares an array of 4 ints

a[0] = 7; // assigns to the first element
a[1] = 8; // assigns to the second element
 // this is just like Java
```

---

**As a box diagram**

In memory

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

In memory

<table>
<thead>
<tr>
<th></th>
<th>96</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[0]</td>
<td>100</td>
</tr>
<tr>
<td>a[1]</td>
<td>104</td>
</tr>
<tr>
<td>a[2]</td>
<td>108</td>
</tr>
<tr>
<td>a[3]</td>
<td>112</td>
</tr>
</tbody>
</table>

...
Arrays are pointers!

```c
int a[4];
int *p = a; // pointer to the first element
// of the array

int *p = &a[0]; // another way to write the
// same declaration
```

As a box diagram

```plaintext
As a box diagram

In memory

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
<td></td>
<td>100</td>
<td>104</td>
<td>108</td>
</tr>
<tr>
<td>112</td>
<td>116</td>
<td>100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

26
```
Arrays are pointers!

```c
int a[4];
int *p = a;  // pointer to the first element of the array

*p = 7;  
```

**As a box diagram**

```
    a
   /   
  /     
     
    p
```

**In memory**

```
<table>
<thead>
<tr>
<th></th>
<th>96</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[0]</td>
<td>100</td>
</tr>
<tr>
<td>a[1]</td>
<td>104</td>
</tr>
<tr>
<td>a[2]</td>
<td>108</td>
</tr>
<tr>
<td>a[3]</td>
<td>112</td>
</tr>
<tr>
<td>p</td>
<td>116</td>
</tr>
</tbody>
</table>
```
Arrays are pointers!

```c
int a[4];
int *p = a; // pointer to the first element of the array

a[0] = 7;   // these statements have the
*p = 7;     // same effect
```

As a box diagram

[Box diagram showing array `a` and pointer `p`]

In memory

```
<table>
<thead>
<tr>
<th>addr</th>
<th>96</th>
<th>100</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[0]</td>
<td>100</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>a[1]</td>
<td>104</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a[2]</td>
<td>108</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a[3]</td>
<td>112</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>116</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Pointer arithmetic

```c
int a[4];
int *p = a;
*(p+2) = 9;
```

As a box diagram

```
```

```
| a     | p    |
```

In memory

```
| a[0]  | 100  |
| a[1]  | 104  |
| a[2]  | 108  |
| a[3]  | 112  |
| p     | 116  |
|       | 100  |
```
Pointer arithmetic

```c
int a[4];
ext *p = a;

a[2] = 9;  // these statements have the same effect
*(p+2) = 9;  // same effect
```

As a box diagram

- Pointer arithmetic adds 2*sizeof(int) to the value of `p`

In memory

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[0]</td>
<td>100</td>
</tr>
<tr>
<td>a[1]</td>
<td>104</td>
</tr>
<tr>
<td>a[2]</td>
<td>108</td>
</tr>
<tr>
<td>a[3]</td>
<td>112</td>
</tr>
<tr>
<td>p</td>
<td>116</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>96</td>
</tr>
<tr>
<td></td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>100</td>
</tr>
</tbody>
</table>
Pointer arithmetic

```c
int a[4];
int *p = a;
a[4] = 0;
*(p+4) = 0;
```

As a box diagram:

```
a
    ^
    p
```

In memory:

```
a[0]  100
a[1]  104
a[2]  108
a[3]  112
p     116
```
int a[4];
int *p = a;

a[4] = 0;    // this statements overwrites
             // p with 0!

*(p+4) = 42; // this will crash!

As a box diagram

In memory

| a[0] | 100 |
| a[1] | 104 |
| a[2] | 108 |
| a[3] | 112 |
| p    | 116 |
| 96   | 9   |
| 96   | 0   |
int *p = 103406;
*p = 42;

This code is evil.

It might overwrite a data structure!

It might even overwrite code!

Memory

this could be anything

103405
103406
103407
103408
C: three common confusions

• Pointers

• Arrays

• The syntax for types (it can be weird...)

Some Pointer/Array Types

int *p
int* p
// declares a pointer to an integer
[note: whitespace doesn’t matter]

int p[10]
// declares an array of 10 integers

p 7 7 7 7 7 7 7 7 7 7
Some Pointer/Array Types

int *p[10]
    // declares an array of 10 pointers which
    // each point to an integer

    ![Diagram of int *p[10]]

int **p
    // declares a pointer to a pointer to
    // an integer

    ![Diagram of int **p]
Some Pointer/Array Types

char *argv[]
// declares an array of pointers-to-chars
// the array has unknown length [used in main()]

int (*p)[10]
// declares a pointer to array of 10 integers [you probably won't use this in this class]
Today

• Introduction

• C overview

• Lab 1 quickstart
  - how to get started
  - how to compile and debug C code

DEMO!