Admin

Homework #1
• Due Feb 6

Reading
• Finish Chapter 1
• Start Chapter 2

TAs
• +1 UTA: Michael Zhou
• Lindsay is Head TA
• Office Hours Posted
  – See Piazza or course webpage
Last time ......

Who can remind us what we covered last time?
Last time …

Who can remind us what we covered last time?

• Representing positive and negative integer
  – Sign Magnitude
  – 1’s Complement
  – 2’s Complement

• 2’s Complement math
  – Addition, subtraction, negation

• Representing “real” numbers
  – Fixed Point
  – Rational
  – IEE Floating Point

• Characters/strings
Computer Memory

Where do we put these numbers?
  • Registers  [more on these later]

  • Memory
Computer Memory

Where do we put these numbers?

• Registers [more on these later]
  – In the processor
  – Compute directly on them
  – Few of them (~16 or 32)

• Memory
Computer Memory

Where do we put these numbers?

- Registers [more on these later]
  - In the processor
  - Compute directly on them
  - Few of them (~16 or 32)

- Memory [Our focus now]
  - External to processor
  - Load/store values to/from registers
  - Very large (multiple GB)
Memory Organization

Memory: billions of numbers…how to get the right one?

• We number our numbers!
• Each memory location has an address
• Processor asks to read or write specific address
  – Memory, please load address 0x123400
  – Memory, please write 0xFE into address 0x8765000
• Kind of like a giant array
Memory Organization

Memory: billions of numbers...how to get the right one?

• We number our numbers!

• Each memory location has an address

• Processor asks to read or write specific address
  – Memory, please load address 0x123400
  – Memory, please write 0xFE into address 0x8765000

• Kind of like a giant array
  – Array of what?
    – Bytes?
    – 32-bit ints?
    – 64-bit ints?
Memory Organization

Most systems: byte (8-bit) addressed
  • Memory is “array of bytes”
    – Each address specifies 1 byte
  • Support to load/store 16, 32, 64 bit quantities
    – Byte ordering varies from system to system

Some systems “word addressed”
  • Memory is “array of words”
    – Word is “int”
    – Smaller operations “faked” in processor
  • Not very common
Word of the Day: Endianess

**Byte Order**

**Big Endian:** byte 0 is 8 most significant bits IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA

**Little Endian:** byte 0 is 8 least significant bits Intel 80x86, DEC Vax, DEC Alpha
Endianness

Example: 12 as 32-bit int at address 0x1234:

<table>
<thead>
<tr>
<th>Address</th>
<th>Little Endian</th>
<th>Big Endian</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1237</td>
<td>0x00</td>
<td>0x0C</td>
</tr>
<tr>
<td>0x1236</td>
<td>0x00</td>
<td>0x00</td>
</tr>
<tr>
<td>0x1235</td>
<td>0x00</td>
<td>0x00</td>
</tr>
<tr>
<td>0x1234</td>
<td>0x0C</td>
<td>0x00</td>
</tr>
</tbody>
</table>

Programming impact: machines of different Endianness

- Network programs
  - htons, htonl, ntohl, ntohs
- Saved data files
Alignment

**Alignment:** require that objects fall on address that is multiple of their size.

32-bit integer

- Aligned if address \( \% 4 = 0 \)

64-bit integer?

- Aligned if ?

Results of un-aligned access? Depends…

- Nothing
- Accepted, but slow (very very slow?)
- Crash your program
Memory Layout

Program has many pieces, all in memory
  • “Thinks” it has all of memory to itself

Pieces of a program:
  • **Instructions**: Called “text”
    – Just numbers! (next few weeks)
  • **Statically allocated data**
    – Global variables
  • **Dynamically allocated data**: Heap
    – new (Java), malloc (C)
    – Grows “up” (to higher addresses)
  • **Local variables, parameters, RA**: Stack
    – Grows “down”
```c
int anumber = 3;

int factorial (int x) {
    if (x == 0) {
        return 1;
    }
    else {
        return x * factorial (x - 1);
    }
}

int main (void) {
    int z = factorial (anumber);
    printf("%d\n", z);
    return 0;
}
```
Let’s do a little Java…

```java
public class Example {
    public static void swap (int x, int y) {
        int temp = x;
        x = y;
        y = temp;
    }
    public static void main (String[] args) {
        int a = 42;
        int b = 100;
        swap (a, b);
        System.out.println("a = " + a + " b = " + b);
    }
}
What does this print? Why?
```
public class Example {
    public static void swap (int x, int y) {
        int temp = x;
        x = y;
        y = temp;
    }
    public static void main (String[] args) {
        int a = 42;
        int b = 100;
        swap (a, b);
        System.out.println("a = " + a + " b = " + b);
    }
}

What does this print? Why?
Let’s do a little Java…

```java
public class Example {
    public static void swap (int x, int y) {
        int temp = x;
        x = y;
        y = temp;
    }
    public static void main (String[] args) {
        int a = 42;
        int b = 100;
        swap (a, b);
        System.out.println("a = " + a + " b = " + b);
    }
}
```

What does this print? Why?
Let’s do a little Java…

public class Example {
    public static void swap (int x, int y) {
        int temp = x;
        x = y;
        y = temp;
    }
    public static void main (String[] args) {
        int a = 42;
        int b = 100;
        swap (a, b);
        System.out.println("a = " + a + " b = " + b);
    }
}

What does this print? Why?
public class Example {
    public static void swap (int x, int y) {
        int temp = x;
        x = y;
        y = temp;
    }
    public static void main (String[] args) {
        int a = 42;
        int b = 100;
        swap (a, b);
        System.out.println("a = " + a + " b = " + b);
    }
}

What does this print? Why?

Stack:

<table>
<thead>
<tr>
<th>main</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>42</td>
</tr>
<tr>
<td>b</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>swap</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>100</td>
</tr>
<tr>
<td>y</td>
<td>100</td>
</tr>
<tr>
<td>temp</td>
<td>42</td>
</tr>
<tr>
<td>RA</td>
<td>c0</td>
</tr>
</tbody>
</table>

© Alvin R. Lebeck / Andrew Hilton
Let’s do a little Java…

public class Example {
    public static void swap (int x, int y) {
        int temp = x;
        x = y;
        y = temp;
    }
    public static void main (String[] args) {
        int a = 42;
        int b = 100;
        swap (a, b);
        System.out.println("a =" + a + " b = " + b);
    }
}

What does this print? Why?
Let’s do a little Java…

```java
public class Example {
    public static void swap (int x, int y) {
        int temp = x;
        x = y;
        y = temp;
    }
    public static void main (String[] args) {
        int a = 42;
        int b = 100;
        swap (a, b);
        System.out.println("a = " + a + " b = " + b);
    }
}
```

What does this print? Why?
Let’s do some different Java...

```java
public class Ex2 {
    int data;
    public Ex2 (int d) { data = d; }
    public static void swap (Ex2 x, Ex2 y) {
        int temp = x.data;
        x.data = y.data;
        y.data = temp;
    }
    public static void main (String[] args) {
        Example a = new Example (42);
        Example b = new Example (100);
        swap (a, b);
        System.out.println("a = " + a.data + " b = " + b.data);
    }
}
```

What does this print? Why?
Let’s do some different Java...

```java
public class Ex2 {
    int data;
    public Ex2 (int d) { data = d; }
    public static void swap (Ex2 x, Ex2 y) {
        int temp = x.data;
        x.data = y.data;
        y.data = temp;
    }
    public static void main (String[] args) {
        Example a = new Example (42);
        Example b = new Example (100);
        swap (a, b);
        System.out.println("a =" + a.data +
                         " b = " + b.data);
    }
}
```

What does this print? Why?
Let’s do some different Java…

```java
public class Ex2 {
    int data;
    public Ex2 (int d) { data = d; }
    public static void swap (Ex2 x, Ex2 y) {
        int temp = x.data;
        x.data = y.data;
        y.data = temp;
    }
    public static void main (String[] args) {
        Example a = new Example (42);
        Example b = new Example (100);
        swap (a, b);
        System.out.println("a = " + a.data + " b = " + b.data);
    }
}
```

What does this print? Why?
Let’s do some different Java...

```java
public class Ex2 {
    int data;
    public Ex2 (int d) { data = d; }
    public static void swap (Ex2 x, Ex2 y) {
        int temp = x.data;
        x.data = y.data;
        y.data = temp;
    }
    public static void main (String[] args) {
        Example a = new Example (42);
        Example b = new Example (100);
        swap (a, b);
        System.out.println("a = " + a.data + " b = " + b.data);
    }
}
```

What does this print? Why?

© Alvin R. Lebeck / Andrew Hilton
Let’s do some different Java…

```java
public class Ex2 {
    int data;
    public Ex2 (int d) { data = d; }
    public static void swap (Ex2 x, Ex2 y) {
        int temp = x.data;
        x.data = y.data;
        y.data = temp;
    }
    public static void main (String[] args) {
        Example a = new Example (42);
        Example b = new Example (100);
        swap (a, b);
        System.out.println("a = " + a.data + " b = " + b.data);
    }
}
```

What does this print? Why?
public class Ex2 {
    int data;
    public Ex2 (int d) { data = d; }
    public static void swap (Ex2 x, Ex2 y) {
        int temp = x.data;
        x.data = y.data;
        y.data = temp;
    }
    public static void main (String[] args) {
        Example a = new Example (42);
        Example b = new Example (100);
        swap (a, b);
        System.out.println("a = " + a.data +
                           " b = " + b.data);
    }
}

What does this print? Why?
public class Ex2 {
    int data;
    public Ex2 (int d) { data = d; }
    public static void swap (Ex2 x, Ex2 y) {
        int temp = x.data;
        x.data = y.data;
        y.data = temp;
    }
    public static void main (String[] args) {
        Example a = new Example (42);
        Example b = new Example (100);
        swap (a, b);
        System.out.println("a =" + a.data +
                        " b = " + b.data);
    }
}

What does this print? Why?
Let’s do some different Java...

```java
public class Ex2 {
    int data;
    public Ex2 (int d) { data = d; }
    public static void swap (Ex2 x, Ex2 y) {
        int temp = x.data;
        x.data = y.data;
        y.data = temp;
    }
    public static void main (String[] args) {
        Example a = new Example (42);
        Example b = new Example (100);
        swap (a, b);
        System.out.println("a = " + a.data + " b = " + b.data);
    }
}
```

What does this print? Why?

CPS 104
References and Pointers

Java has **references**:

- Any variable of object type is a reference
- Point at objects (which are all in the heap)
  - Under the hood: is the memory address of the object
- Cannot explicitly manipulate them (e.g., add 4)
References and Pointers

Java has references:
  • Any variable of object type is a reference
  • Point at objects (which are all in the heap)
    – Under the hood: is the memory address of the object
  • Cannot explicitly manipulate them (e.g., add 4)

Some languages (C, assembly) have explicit pointers:
  • Hold the memory address of something
  • Can explicitly compute on them
  • Can de-reference the pointer (*ptr) to get thing-pointed-to
  • Can take the address-of (&x) to get something’s address
  • Can do very unsafe things, shoot yourself in the foot
Pointers

“address of” operator &
  • don’t confuse with bitwise AND operator (later today)

Given
  int x; int *p;
  p = &x;

Then
  *p = 2; and x = 2; produce the same result

What happens for p = 2?;

On 32-bit machine, p is 32-bits

<table>
<thead>
<tr>
<th>x 0x26cf0</th>
</tr>
</thead>
<tbody>
<tr>
<td>⋮</td>
</tr>
<tr>
<td>p 0x26d00</td>
</tr>
</tbody>
</table>

© Alvin R. Lebeck / Andrew Hilton
Arrays

Java:

```java
int [] x = new int [nElems];
```

C:

```c
int data[42]; //if size is known constant
int data = malloc (nElem * sizeof(*data));
```

- malloc takes number of bytes
- sizeof tells how many bytes something takes

C++:

- Has `new int[nElems]` like Java
- And `malloc(size)` like C
Address Calculation

**x** is a pointer, what is **x+33**?

A pointer, but where?
- what does calculation depend on?

Result of adding an int to a pointer depends on size of object pointed to

Result of subtracting two pointers is an int

\[(d + 3) - d = \_\_\_\_\_\_

\[
\begin{array}{cccc}
0 & 1 & 32 & 33 \\
98 & 99
\end{array}
\]

`int * a = new int[100]`

\[
\begin{array}{cccc}
0 & 1 & 32 & 33 \\
98 & 99
\end{array}
\]

`a[33]` is the same as `*(a+33)`

if `a` is `0x00a0`, then `a+1` is `0x00a4`, `a+2` is `0x00a8` (decimal 160, 164, 168)

\[
\begin{array}{cccc}
0 & 1 & 33 & 199
\end{array}
\]

`double * d = new double[200];`

\[
\begin{array}{cccc}
0 & 1 & 33 & 199
\end{array}
\]

`*(d+33)` is the same as `d[33]`

if `d` is `0x00b0`, then `d+1` is `0x00b8`, `d+2` is `0x00c0` (decimal 176, 184, 192)
More Pointer Arithmetic

address one past the end of an array is ok for pointer comparison only

what’s at \(*(\text{begin}+44)\)?

what does \text{begin}++ mean?

how are pointers compared using < and using == ?

what is value of \text{end} - \text{begin}?

```c
char * a = new char[44];
char * begin = a;
char * end = a + 44;

while (begin < end)
{
    *begin = 'z';
    begin++;
}
```
More Pointers & Arrays

```c
int * a = new int[100];

0 1 32 33 98 99
```

a is a pointer
*a is an int
a[0] is an int (same as *a)
a[1] is an int
a+1 is a pointer
a+32 is a pointer
*(a+1) is an int (same as a[1])
*(a+99) is an int
*(a+100) is trouble
Array Example

#include <stdio.h>

main()
{
    int *a = malloc (100 * sizeof(*a));
    int *p = a;
    int k;

    for (k = 0; k < 100; k++)
    {
        *p = k;
        p++;
    }
    printf("entry 3 = %d\n", a[3])
}
malloc / new (C/C++)

Library routines that handle memory management for heap (allocation / deallocation)

Java has garbage collection (reclaim memory of unreferenced objects)

C/C++ must use free/delete, else memory leak
Strings as Arrays

A string is an array of characters with ‘\0’ at the end
Each element is one byte, ASCII code
‘\0’ is null (ASCII code 0)
**Strlen()**

`strlen()` returns the # of characters in a string

- same as # elements in char array?

```c
int strlen(char * s)
// pre: ‘\0’ terminated
// post: returns # chars
{
    int count=0;
    while (*s++)
        count++;
    return count;
}
```
Vector Class vs. Arrays

Vector Class
- insulates programmers
- array bounds checking
- automagically growing/shrinking when more items are added/deleted

How are Vectors implemented?
- Arrays, re-allocated as needed

Arrays can be more efficient
- but be leery of claims that C-style arrays required for efficiency
Outline

Memory
  • Memory holds numbers
  • Everything (floats, colors, sound, letters) is a number

Bit Manipulations
Bit Manipulation: Motivating Example

Suppose I want to track

- 20 Million people
- 30 properties per person (e.g., IS_MALE, IS_STUDENT, ...)
  - Each may or may not apply to any given person
- How could I represent this?
Bad Idea 1:

class Person {
    static final int IS_MALE = 0;
    static final int IS_STUDENT = 1;
    ...
    vector<Integer> props;
    ...
    boolean hasProperty(int p) {
        for (int I =0; I < props.size(); i++){
            if (props.at(i).intValue() == p) return true;
        }
        return false;
    }
}
P.S. Undergrad friend of mine did this, code was very slow
Okish Idea

class Person {
    bool is_male;
    bool is_student;
    ...
}

Why is this not so great?
class Person {
    bool is_male;
    bool is_student;
    ...
}

Why is this not so great?
• Byte is smallest addressable size: typically bool = 1 byte
• Wastes space
• As we will see later, wasting space can hurt performance (caches)
class Person {
    static final int IS_MALE = 0x1; //0000...0001
    static final int IS_STUDENT = 0x2; //0000...0010
    static final int KNOWS_JAVA = 0x4; //0000...0100
    static final int KNOWS_C = 0x8; //0000...1000
    ...
    int props;
    bool hasProperty(int p) {
        return (props & p) != 0;
    }
}

Use one int, and manipulate/test individual bits
Fast, and only need 1 32-bit int (4 bytes) for 32 properties!
Bit-wise operations: AND

Previous slide used bit-wise & (AND):

• Do not confuse AND (a & b) with address-of (& a)
• Do not confuse logical AND (a && b)
• Each result bit will be 1 iff corresponding bit in a AND b is 1:

\[
\begin{array}{c}
1101010 \\
& \& 0100100 \\
\hline
0100000
\end{array}
\]

• Using bits as sets, corresponds to intersection
Bit-wise operations: OR

### Bit-wise OR |

- Each result bit will be 1 iff corresponding bit in a OR in b is 1:

```
  1101010
| 0100100
  1101110
```

- Using bits as sets, corresponds to union
  - Add p to props: `props = props | p;`
Bit-wise operations: XOR

Bit-wise XOR ^

• OR is inclusive (either or both)
• XOR is exclusive-or (either, but not both):

\[
\begin{array}{c}
1101010 \\
^ \ 0100100 \\
1001110
\end{array}
\]
Bit-wise operations: NOT

Bit-wise negation ~ (NOT)

Flip each bit

\[
\sim 1101010 \\
0010101
\]

- Can combine NOT and AND for removal:
  - Remove \( p \) from \( \text{props} \): \( \text{props} = \text{props} \& \sim p; \)
  
  Suppose \( \text{props} \) is 11011101 and \( p \) is 00010000:

  \[
  \sim p = 11101111 \\
  \text{Props} \& \sim p: \\
  11011101 \\
  11101111 \\
  11001101
  \]
Bit-wise operations: SHIFT

Left shift (<<)
- Moves left, bringing in 0s at right, excess bits “fall off”
- $10010001 \ll 2 = 0100100$
- $x \ll k$ corresponds to $x \times 2^k$

Logical (or unsigned) right shift (>>)
- Moves bits right, bringing in 0s at left, excess bits “fall off”
- $10010001 \gg 3 = 00010010$
- $x \gg k$ corresponds to $x / 2^k$ for unsigned $x$

Arithmetic (or signed) right shift (>>)
- Moves bits right, bringing in (sign bit) at left
- $10010001 \gg 3 = 11110010$
- $x \gg k$ corresponds to $x / 2^k$ for signed $x$
Summary

Homework #1 Feb 6

Computer memory is linear array of bytes
Pointers contain address of other data
Bitwise operations

Next Time
Dive into C
Reading
Finish Ch 1, Start on Ch 2