Development offices all over Silicon Valley, NYC, etc.

Pointer/reference/abstract diagrams are drawn on whiteboards in software

and finishing Lab 2.

CSI 310: Lecture 6 (No Lab assignment. Use Lab time for TA help with Prog. I.)
my obj = new Thing (...); // C++ equivalent in Java
my obj = new Thing(...); // C++ equivalent in Java
}
Tip for the Trendy: The following in Java:
What is a pointer value (synonyms: address, locator, reference in Java)?

class Thing ... {
...
}

...Going back to pointers, again...
(2) Java "Reference type" which are like C/++, pointers (to class objects independent)

(1) Primitive type (roughly like C/++, just implementation

Java has only 2:

(3) Pointer type (int*, char*, float*, etc.)
(2) Struct/class type (printf, scanf, malloc, etc.)
(1) Primitive type (int, char, float, etc.)

C/++ has 3 kinds of named variable and array types:

called Java reference variables, to hold and copy Java reference values.

Java hides numerical pointer values/addresses and provides variables values/addresses.

C/++/assembly/machine language EXPOSE numerical pointer
The declaration of a pointer variable specifies BOTH:

1) that it IS a pointer variable, and
2) the TYPE of those variables it can point to.

The name of the newly declared pointer variable can hold the address of the type of variable that the pointer variable can point to (which, by coincidence, is also the C/C++ dereferencing operator...hmm...). Important for USING pointers!

float * MYFirstPointer;
Why? 2003 can change while MyP still points to MyInt.

MyP does NOT point to 2003!!

Now we can say "MyP points to MyInt."

\[
\begin{array}{c}
\text{MyInt} \\
2003
\end{array}
\]

The value of MyP is the address of the int variable MyInt

The value of MyInt points to int variable MyInt

\[
\begin{array}{c}
\text{MyP} \\
\text{int* MyP};
\end{array}
\]

computer's memory: produces in the

MyP = &MyInt;
MyInt = 2003;
MyP = MyInt;
int *MyP;
int &MyP;

MEANS:

pointer variable MyP points to int variable MyInt.
That means: Access the variable whose address is in the pointer (or is the address value).

The * operator dereferences a pointer (or address value).

Important for USING Pointers!

Huhh? One variable
has two different
names (different kinds).

ALIASES.

General problem:

count >> MyP

count >> *MyP

count >> MyInt

count >> *MyInt

MyInt

MyP

0xbe0fef04

2003

2003
Before Multiplication

After Multiplication

\[ \text{MyInt} \times 2 = \text{MyInt} \]

0xbe0fef04

4006

count >> MyP

4006

count >> MyP

4006

count >> MyP

MyInt

4006

MyInt

2003

MyInt

0xbe0fef04
so it could print the int value located there.

This is a "CRASH" computer tried to read memory at the illegal address ox0,

Segmentation Fault

```
cout >> *myP >> endl;
0x0 0xbe0fef04
myP
```

4006

```
cout >> myInt;  // "  " >> myP >> "  " >> myInt;  // "  " >> myP
```

4006

After Assignment:

```
4006
myInt
myP
```

Before Assignment:

```
4006
myInt
myP
```

myP = NULL;

© S. Chakravarthy 2005

University at Albany Computer Science Dept.
(changed).

Features of the variable ‘is VALUE’ (synonym: state) can be

“clone,” referring to the variable itself. A variable has a unique identity even after it is

renamed in a new context. The variable is the object itself. Like a

variable

4. (This is new.) A NAME can be the VALUE of a

conceptual entities (3 different things).

3. The value, the name, and the variable itself are three DIFFERENT

2. Each variable holds a value (or state).

1. Each variable (synonym: object) has (some kind of) name.

The main points...
Brandon = 98;
The value of this integer variable points to a 4-byte address or pointer value.
The above integer variable whose address is 992 and value is 57.

The pointer variable whose address is 987 is pointing to

Here's a pointer variable whose address is 987.

Here's the integer variable whose address is 992.

You will see hex in Lab2.

Real programmers write their addresses in hexadecimal.

(You will see hex in Lab2.) Because hex to binary (base 2) conversion is very easy.

Real programmers write their addresses in hexadecimal.

I will denote pointer values by black dots because the numeric value is usually boring.

Pointer values, like any other numbers, can be stored in variables, like

Here's a pointer variable whose value is 992.

The above integer variable whose address is 992 is pointing to

The pointer variable whose address is 987 is pointing to

Real programmers write their addresses in hexadecimal.

(You will see hex in Lab2.) Because hex to binary (base 2) conversion is very easy.

Real programmers write their addresses in hexadecimal.

You will see hex in Lab2.

Here's a pointer variable whose address is 987.

Here's the integer variable whose address is 992.

You will see hex in Lab2.

Real programmers write their addresses in hexadecimal.

(You will see hex in Lab2.) Because hex to binary (base 2) conversion is very easy.

Real programmers write their addresses in hexadecimal.

You will see hex in Lab2.

Here's a pointer variable whose value is 992.

The above integer variable whose address is 992 is pointing to

The pointer variable whose address is 987 is pointing to

Real programmers write their addresses in hexadecimal.

(You will see hex in Lab2.) Because hex to binary (base 2) conversion is very easy.

Real programmers write their addresses in hexadecimal.
try {xx: function() {
  x = 3;
}
xx = 1;
xx = 2;
xx = 3;

using namespace std;

#include<iostream>

Expressions like (x+3), are not variables, so:

\$ \text{ \textbackslash e +} \text{ \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e \textbackslash e

@S. Chakken 2005
University at Albany Computer Science Dept.
A POINTER VARIABLE can store a pointer value.
That means: Access the variable whose address is in the dereferenced pointer.

The * operator dereferences a pointer (or address value).

(or: Access the variable whose address is the dereferenced address value).

```cpp
0x0e0f04
cout >> MyP >> endl;
0x0e0f04
cout >> *MyP >> endl;
0x0e0f04
cout >> MyInt >> endl;
```

MyInt

MyP

2003

2003
Before Multiplication:

After Multiplication:

\[ \text{MYP} \times 2 \]

\[ \text{MYP} \]

\[ \text{MYP} \times 2 \]

\[ \text{MYP} \]

\[ \text{MyInt} \]

\[ \text{MyInt} \]

\[ \text{MyInt} \]

\[ \text{MyInt} \]

\[ 0xbe0fef04 \]

\[ 0xbe0fef04 \]
so it could print the int value located there.

This is a "CRASH": Computer tried to read memory at the illegal address 0x0,

Segmentation Fault

cout << *MyP << endl;

cout << MyP << " null" << endl;

cout << MyP << endl;

After Assignment:

Before Assignment:

© S. Chakken 2005

University at Albany Computer Science Dept.
...the 3 variables, before and after each step...

Please simulate the execution of this by writing the values currently in each of

\[
\text{temp}';
\text{p2} = \text{temp}';
\text{p1} = \text{p2}';
\text{temp} = \text{p1}';
\]

\[
\text{int} \text{temp}';
\text{p2} = \text{a}';
\text{p1} = \text{a}';
\text{int} \text{p1}'; \text{int} \text{p2}';
\]

How can you program swapping the values of I and J using pointers?

\[
\text{i} = 1928; \text{j} = 2003;
\text{int} \text{i}'; \text{int} \text{j}';
\]
assert (p1 == p2) (p2 == p1);

1928 2003

cout >> I >> J >> end

2003 1928

cout >> *p1 >> *p2 >> end

p2 = TEMP
p1 = p2;
TEMP = p1;
int *TEMP
int *p1; int *p2;
int j; int I; I = 1928; j = 2003;

Now, how is swapping the values of the pointer variables different?