

## Pointers

November 29, 2001  
EECS 1530

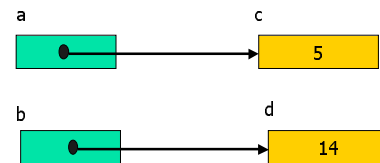
## What are Pointers?

- Pointers give the location of an item rather than the item itself
- Declaration syntax:  
type \* variable\_name;
- To get a pointer to a variable you use the & (address) operator.

## Sample

```
int main (void)
{
    int *a, *b;    // a and b are pointers
    int c, d;     // c and d are normal ints
    c = 5;
    d = 14;
    a = &c;      // a points to c
    b = &d;      // b points to d
    ...
}
```

## What really is happening



## More details

- To get what a pointer points at (Dereferencing a pointer) use an \* in front of the pointer name  
cout << \*a << endl;

## Pointer Problems

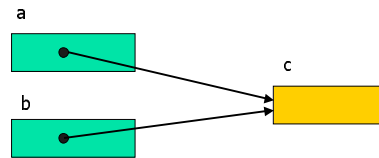
- Pointers do cause a lot of problems
- Pointers do **not** hold data – they must point to something to have any use.
- Multiple pointers can point to the same thing.

## Another sample

```
int main (void)
{
    int *a, *b;
    int c = 5, d = 14;

    a = &c; // Both a and b point to c
    b = &c;
    cout << *a << " " << *b << endl;
    cout << a << " " << b << endl;
    *a = 7; // change the value pointed to by a
    cout << *a << " " << *b << endl;
    cout << a << " " << b << endl;
    return 0;
}
```

## Diagram



## Why use Pointers?

## Why Use Pointers?

- Allow Multiple References to 1 Object
- Necessary for Dynamic allocation
- Used to implement a Call by reference – C++ Passes a pointer rather than a variable.

## Dynamic Allocation

- Returns a pointer to the object
- Uses the "new" keyword
- Example:  
int \*dynint;  
  
dynint = new int;  
\*dynint = 15;

## Getting rid of dynamic variables

- Can deallocate or delete dynamic variables
- Uses delete operator:  
  
delete dynint;



## Dynamic Allocation -- Arrays

```
int *dynarray;    // Yes I know this looks wrong

dynarray = new int [50];    // allocate
dynarray[0] = 34;
dynarray[1] = 31;
...
delete [] dynarray;    // deallocate
```