CS:35 Data Structures and Algorithms

Copies in C++

5/1/2013

```
int foo(int x) {
    return x;
}

int a = 2;
a = foo(a);
```

How many copies of a are made?


```
class C {
int *arr;
    // ...
};

C foo(C x) {
    return x;
}

C a;
a = foo(a);
```

Needlessly costly if C is large...

Mental Model of Computation

```
int mystery(int x, int *px) {
    int y = x + *px;
    x = *px;
    return y;
}

int x = 1;
int *px = new int(2);
x = mystery(x, px);
```

```
int mystery(int x, int *px) {
    int y = x + *px;
    x = *px;
    return y;
}

int x = 1;
int *px = new int(2);
x = mystery(x, px);
```

Stack Heap


```
int mystery(int x, int *px) {
    int y = x + *px;
    x = *px;
    return y;
}

int x = 1;
int *px = new int(2);
x = mystery(x, px);
```

Stack Heap

x 1


```
int mystery(int x, int *px) {
    int y = x + *px;
    x = *px;
    return y;
}

int x = 1;
int *px = new int(2);
x = mystery(x, px);
```

Stack Heap

x 1

px 2


```
int mystery(int x, int *px) {
    int y = x + *px;
    x = *px;
    return y;
}

int x = 1;
int *px = new int(2);
x = mystery(x, px);
```

Stack Heap

x 1

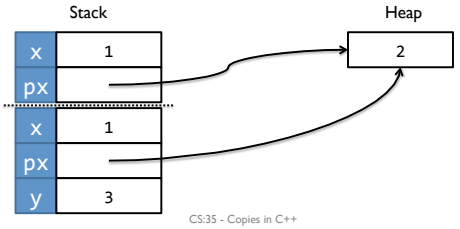
px 2

x 1

px

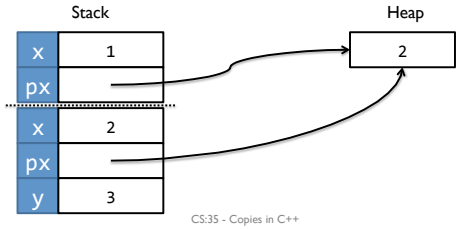
```
int mystery(int x, int *px) {
    int y = x + *px;
    x = *px;
    return y;
}

int x = 1;
int *px = new int(2);
x = mystery(x, px);
```



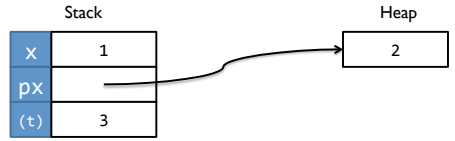

```
int mystery(int x, int *px) {
    int y = x + *px;
    x = *px;
    return y;
}

int x = 1;
int *px = new int(2);
x = mystery(x, px);
```




```
int mystery(int x, int *px) {
    int y = x + *px;
    x = *px;
    return y;
}

int x = 1;
int *px = new int(2);
x = mystery(x, px);
```




```
int mystery(int x, int *px) {
    int y = x + *px;
    x = *px;
    return y;
}

int x = 1;
int *px = new int(2);
x = mystery(x, px);
```

Exercise Your Own Mental Model

```
int foo(int x, int *y, int *z) {
    int a = x + *y;
    *y = a + 1;
    *z = *y + *z;
    y = &x;
    // POINT (B)
    return *y;
}
int a = 3;
int *b = new int(2);
int c = a + 1;

// POINT (A)
int d = foo(a, b, &c);
// POINT (C)
```


Exercise Your Own Mental Model

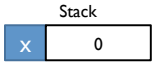
```
int foo(int x, int *y, int *z) {
    int a = x + *y;
    *y = a + 1;
    *z = *y + *z;
    y = &x;
    // POINT (B)
    return *y;
}

int a = 1;
// POINT (D)
int d = foo(a, &a, &a);
// POINT (F)
```


When Do Copies Occur?

1. Parameter passing.
2. Returns.
3. Assignment.

(A)



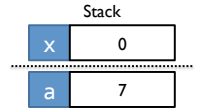
(B)

f(x);

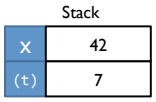
When Do Copies Occur?

1. Parameter passing.
2. Returns.
3. Assignment.

(A)



(B)

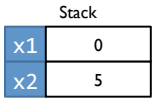


return a;

When Do Copies Occur?

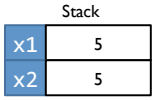
1. Parameter passing.

(A)



2. Returns.

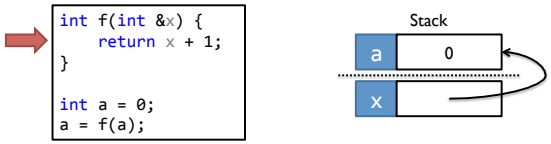
(B)



3. Assignment.

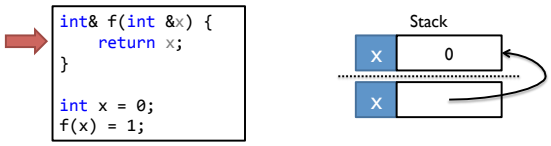
```
x1 = x2;
```


The Mighty Reference (Type)



For any type T, T& is a *reference to T*.

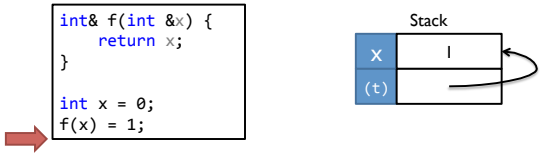
Dereferencing a reference is *implicit*.



Reference types avoid copying values.

```
int& f(int &x) {
    return x;
}

int x = 0;
f(x) = 1;
```



Reference types avoid copying values.


```
int& f(int x) {
    return x;
}
```

But, like pointers, be wary...

References vs. Pointers

If references are just implicit pointers, what use do they have?

Memory Management with Classes

```
class C {
int *arr;
    // ...
};

C foo(C x) {
    return x;
}

C a;
a = foo(a);
```

What dictates how C is copied?

1. Parameter Passing.
2. Returns

The *Copy Constructor*.

```
class C {
    public:
    C(const C& rhs) {
        // Copy fields of rhs to this new C
    }
};
```

(By default, performs a *shallow copy*.)

3. Assignment

The *Copy Assignment Operator*.

```
class C {
public:
    C& operator=(const C& rhs) {
        // Assign fields of rhs into this existing C.
    }
};
```

(By default, performs a *shallow copy*.)

The *Rule of Three*

If any one of the *copy constructor*, *copy assignment operator*, or *destructor* are implemented, then all three must be implemented (coherently) to ensure memory safety.

Exercise!

`debt-backend.cpp` contains code that implements a back-end for a debt-collection agency.

Run it, test it, break it (if possible), then fix it!!

(Any questions or comments? posera@cis.upenn.edu or http://www.cis.upenn.edu/~posera thanks!)