CSCI 200: Foundational Programming Concepts & Design Lecture 13

Reference

&

Memory

Sign into iClicker

Learning Outcomes For Today

- Explain the difference between pass-by-value and pass-by-reference. Draw a diagram of how each stores its parameters in memory.

On Tap For Today

- Reference
- PBV / PBR / PBP
- Practice

On Tap For Today

- Reference
- PBV / PBR / PBP
- Practice

Binary Operations

- Generally

```
lhs @ rhs
```

- Perform binary operation `@` from the right hand side to the left hand side
- Rephrased

```
lvalue @ rvalue
```

`lvalue` vs. `rvalue`

- `lvalue` (formally locator value) is an object that represents some identifiable location in memory (i.e. has an address)

- `rvalue` is an object that does not represent some identifiable location in memory

Examples

```
int var, var2;    // allocate memory on the stack for two integer variables
var = 5;          // ok, place 5 into memory of var
var2 = var;       // ok, implicit lvalue-to-rvalue conversion for var
4 = var;          // error! Cannot assign var to 4.  4 is not an lvalue
(var + 1) = 4;    // error! (var + 1) is not an lvalue
```

Functions Revisited

```
void f(int bar) {          // pass-by-value
  bar = 3;
}
void g(int* pFoo) {        // pass-by-pointer
  *pFoo = 2;
}


int var;          // allocate memory on the stack
var = 5;          // ok, place 5 into memory of var
f(3);             // pass rvalue to f()
g(&var);          // pass lvalue to f()
```

Program Entry Point: main()

```
int add( int x, int y ) {
    int a;
    a = x + y;
    return a;
}

int main() {
    int a(4), b(3);
    int c = add( a, b );
    int d = add( 5, 8 );
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014			
0x40960018			
0x4096001c			
0x40960020			
0x40960024			
0x40960028			
0x4096002c			
0x40960030			
0x40960034			
0x40960038			
0x4096003c			

Evaluate main()

```
int add( int x, int y ) {
    int a;
    a = x + y;
    return a;
}

int main() {
    int a(4), b(3);
    int c = add( a, b );
    int d = add( 5, 8 );
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014			
0x40960018			
0x4096001c	b	3	
0x40960020			
0x40960024			
0x40960028			
0x4096002c			
0x40960030	a	4	
0x40960034			
0x40960038			b 0x4096001c
0x4096003c			a 0x40960030

Pass by Value

```
int add( int x, int y ) {
    int a;
    a = x + y;
    return a;
}

int main() {
    int a(4), b(3);
    int c = add( a, b );
    int d = add( 5, 8 );
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014			
0x40960018			
0x4096001c	b	3	
0x40960020			
0x40960024	c		
0x40960028			
0x4096002c			
0x40960030	a	4	
0x40960034			c 0x40960024
0x40960038			b 0x4096001c
0x4096003c			a 0x40960030

Pass by Value

```
int add( int x, int y ) {
    int a;
    a = x + y;
    return a;
}

int main() {
    int a(4), b(3);
    int c = add( a, b );
    int d = add( 5, 8 );
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014	y	3	
0x40960018			
0x4096001c	b	3	
0x40960020			
0x40960024	c		
0x40960028			
0x4096002c			y 0x40960014
0x40960030	a	4	x 0x40960038
0x40960034			c 0x40960024
0x40960038	x	4	b 0x4096001c
0x4096003c			a 0x40960030

Evaluate add()

```
int add( int x, int y ) {
    int a;
    a = x + y;
    return a;
}

int main() {
    int a(4), b(3);
    int c = add( a, b );
    int d = add( 5, 8 );
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014	y	3	
0x40960018			
0x4096001c	b	3	
0x40960020			
0x40960024	c		
0x40960028	a		a 0x40960028
0x4096002c			y 0x40960014
0x40960030	a	4	x 0x40960038
0x40960034			c 0x40960024
0x40960038	x	4	b 0x4096001c
0x4096003c			a 0x40960030

Evaluate add()

```
int add( int x, int y ) {
    int a;
    a = x + y;
    return a;
}

int main() {
    int a(4), b(3);
    int c = add( a, b );
    int d = add( 5, 8 );
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014	y	3	
0x40960018			
0x4096001c	b	3	
0x40960020			
0x40960024	c		
0x40960028	a	7	a 0x40960028
0x4096002c			y 0x40960014
0x40960030	a	4	x 0x40960038
0x40960034			c 0x40960024
0x40960038	x	4	b 0x4096001c
0x4096003c			a 0x40960030

Return by Value

```
int add( int x, int y ) {
    int a;
    a = x + y;
    return a;
}

int main() {
    int a(4), b(3);
    int c = add( a, b );
    int d = add( 5, 8 );
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014	y	3	
0x40960018			
0x4096001c	b	3	
0x40960020			
0x40960024	c		
0x40960028	a	7	a 0x40960028
0x4096002c			y 0x40960014
0x40960030	a	4	x 0x40960038
0x40960034			c 0x40960024
0x40960038	x	4	b 0x4096001c
0x4096003c			a 0x40960030

Return by Value

```
int add( int x, int y ) {
    int a;
    a = x + y;
    return a;
}

int main() {
    int a(4), b(3);
    int c = add( a, b );
    int d = add( 5, 8 );
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014		3	
0x40960018			
0x4096001c	b	3	
0x40960020			
0x40960024	c	7	
0x40960028		7	
0x4096002c			
0x40960030	a	4	
0x40960034			c 0x40960024
0x40960038		4	b 0x4096001c
0x4096003c			a 0x40960030

Pass by Value

```
int add( int x, int y ) {
    int a;
    a = x + y;
    return a;
}

int main() {
    int a(4), b(3);
    int c = add( a, b );
    int d = add( 5, 8 );
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014		3	
0x40960018			
0x4096001c	b	3	
0x40960020			
0x40960024	c	7	
0x40960028	d	7	
0x4096002c			
0x40960030	a	4	d 0x40960028
0x40960034			c 0x40960024
0x40960038		4	b 0x4096001c
0x4096003c			a 0x40960030

Pass by Value

```
int add( int x, int y ) {
    int a;
    a = x + y;
    return a;
}

int main() {
    int a(4), b(3);
    int c = add( a, b );
    int d = add( 5, 8 );
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014	x	5	
0x40960018			
0x4096001c	b	3	
0x40960020			
0x40960024	c	7	
0x40960028	d	7	y 0x4096003c
0x4096002c			x 0x40960014
0x40960030	a	4	d 0x40960028
0x40960034			c 0x40960024
0x40960038		4	b 0x4096001c
0x4096003c	y	8	a 0x40960030

Evaluate add()

```
int add( int x, int y ) {
    int a;
    a = x + y;
    return a;
}

int main() {
    int a(4), b(3);
    int c = add( a, b );
    int d = add( 5, 8 );
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014	x	5	
0x40960018			
0x4096001c	b	3	
0x40960020			
0x40960024	c	7	a 0x40960034
0x40960028	d	7	y 0x4096003c
0x4096002c			x 0x40960014
0x40960030	a	4	d 0x40960028
0x40960034	a		c 0x40960024
0x40960038		4	b 0x4096001c
0x4096003c	y	8	a 0x40960030

Evaluate add()

```
int add( int x, int y ) {
    int a;
    a = x + y;
    return a;
}

int main() {
    int a(4), b(3);
    int c = add( a, b );
    int d = add( 5, 8 );
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014	x	5	
0x40960018			
0x4096001c	b	3	
0x40960020			
0x40960024	c	7	a 0x40960034
0x40960028	d	7	y 0x4096003c
0x4096002c			x 0x40960014
0x40960030	a	4	d 0x40960028
0x40960034	a	13	c 0x40960024
0x40960038		4	b 0x4096001c
0x4096003c	y	8	a 0x40960030

Return by Value

```
int add( int x, int y ) {
    int a;
    a = x + y;
    return a;
}

int main() {
    int a(4), b(3);
    int c = add( a, b );
    int d = add( 5, 8 );
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014	x	5	
0x40960018			
0x4096001c	b	3	
0x40960020			
0x40960024	c	7	a 0x40960034
0x40960028	d	7	y 0x4096003c
0x4096002c			x 0x40960014
0x40960030	a	4	d 0x40960028
0x40960034	a	13	c 0x40960024
0x40960038		4	b 0x4096001c
0x4096003c	y	8	a 0x40960030

Return by Value

```
int add( int x, int y ) {
    int a;
    a = x + y;
    return a;
}

int main() {
    int a(4), b(3);
    int c = add( a, b );
    int d = add( 5, 8 );
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014		5	
0x40960018			
0x4096001c	b	3	
0x40960020			
0x40960024	c	7	
0x40960028	d	13	
0x4096002c			
0x40960030	a	4	d 0x40960028
0x40960034		13	c 0x40960024
0x40960038		4	b 0x4096001c
0x4096003c		8	a 0x40960030

Return by Value

```
int add( int x, int y ) {
    int a;
    a = x + y;
    return a;
}

int main() {
    int a(4), b(3);
    int c = add( a, b );
    int d = add( 5, 8 );
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014		5	
0x40960018			
0x4096001c	b	3	
0x40960020			
0x40960024	c	7	
0x40960028	d	13	
0x4096002c			
0x40960030	a	4	d 0x40960028
0x40960034		13	c 0x40960024
0x40960038		4	b 0x4096001c
0x4096003c		8	a 0x40960030

Program Terminates

```
int add( int x, int y ) {
    int a;
    a = x + y;
    return a;
}

int main() {
    int a(4), b(3);
    int c = add( a, b );
    int d = add( 5, 8 );
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014		5	
0x40960018			
0x4096001c		3	
0x40960020			
0x40960024		7	
0x40960028		13	
0x4096002c			
0x40960030		4	
0x40960034		13	
0x40960038		4	
0x4096003c		8	

Program Entry Point: main()

```
void add_five( int* pX ) {
    *pX += 5;
}

int main() {
    int a(4);
    cout << a << endl;
    add_five( &a );
    cout << a << endl;
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014			
0x40960018			
0x4096001c			
0x40960020			
0x40960024			

Evaluate main()

```
void add_five( int* pX ) {
    *pX += 5;
}

int main() {
    int a(4);
    cout << a << endl;
    add_five( &a );
    cout << a << endl;
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014			
0x40960018			
0x4096001c	a	4	
0x40960020			
0x40960024			a 0x4096001c

Evaluate main()

```
void add_five( int* pX ) {
    *pX += 5;
}

int main() {
    int a(4);
    cout << a << endl;
    add_five( &a );
    cout << a << endl;
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014			
0x40960018			
0x4096001c	a	4	
0x40960020			
0x40960024			a 0x4096001c

Pass by Pointer

```
void add_five( int* pX ) {
    *pX += 5;
}

int main() {
    int a(4);
    cout << a << endl;
    add_five( &a );
    cout << a << endl;
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014			
0x40960018			
0x4096001c	a	4	
0x40960020			
0x40960024			a 0x4096001c

Pass by Pointer

```
void add_five( int* pX ) {
    *pX += 5;
}

int main() {
    int a(4);
    cout << a << endl;
    add_five( &a );
    cout << a << endl;
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014			
0x40960018			
0x4096001c	a	4	
0x40960020	pX	0x4096001c	pX 0x40960020
0x40960024			a 0x4096001c

Evaluate add_five()

```
void add_five( int* pX ) {
    *pX += 5;
}

int main() {
    int a(4);
    cout << a << endl;
    add_five( &a );
    cout << a << endl;
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014			
0x40960018			
0x4096001c	a	9	
0x40960020	pX	0x4096001c	pX 0x40960020
0x40960024			a 0x4096001c

Evaluate main()

```
void add_five( int* pX ) {
    *pX += 5;
}

int main() {
    int a(4);
    cout << a << endl;
    add_five( &a );
    cout << a << endl;
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014			
0x40960018			
0x4096001c	a	9	
0x40960020			
0x40960024			a 0x4096001c

Return by Value

```
void add_five( int* pX ) {
    *pX += 5;
}

int main() {
    int a(4);
    cout << a << endl;
    add_five( &a );
    cout << a << endl;
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014			
0x40960018			
0x4096001c	a	9	
0x40960020			
0x40960024			a 0x4096001c

Program Terminates

```
void add_five( int* pX ) {
    *pX += 5;
}

int main() {
    int a(4);
    cout << a << endl;
    add_five( &a );
    cout << a << endl;
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014			
0x40960018			
0x4096001c		9	
0x40960020			
0x40960024			

Create Explicit `lvalue`

```
int var;
var = 5;
int& var2 = var;          // now assign lvalue to var2
                          // var2 backed by same memory as var
var2 = 6;                 // changes var as well!
```

Reference

- & - reference operator

```
int main() {

    int x = 4;
    int y = x;              // assign the value of x
    int& z = x;             // assign the reference of x

    cout << x << endl;      // print value of x - 4
    cout << &x << endl;     // print address of x - 0x4ab338cc

    cout << y << endl;      // print value of y - 4
    cout << &y << endl;     // print address of y - 0x5a23bbdf

    cout << z << endl;      // print value of z - 4
    cout << &z << endl;     // print address of z - 0x4ab338cc

    z = 5;
    cout << x << " "        // prints 5
         << y << " "        // prints 4
         << z << endl;      // prints 5

    return 0;

}
```

Precedence Table

Category	Precedence	Operator	Associativity
Parenthesis	1	( )	Innermost First
Postfix Unary Operators	2	a++ a-- f()	Left to Right
Prefix Unary Operators	3	++a --a +a -a !a ~a (type)a &a *p new delete	Right to Left
Binary Operators	4	a*b a/b a%b	Left to Right
	5	a+b a-b	
Relational Operators	6	a<b a>b a<=b a>=b	
	7	a==b a!=b	
Bitwise Operators	8	a&b	
	9	a^b	
	10	a|b	
Logical Operators	11	a&&b	
	12	a||b	
Assignment Operators	13	a=b a+=b a-=b a*=b a/=b a%=b a&=b a^=b a|=b	Right to Left

Pass by Reference with &

- Instead of passing the value of an argument to the function, pass the argument's memory address to the function

```
void add_five( int& x ) { // int &x
    x += 5;
}
int main() {
    int a(4);
    cout << a << endl;    // 4
    add_five( a );
    cout << a << endl;    // 9
    return 0;
}
```

Program Entry Point: main()

```
void add_five( int& x ) {
    x += 5;
}

int main() {
    int a(4);
    cout << a << endl;
    add_five( a );
    cout << a << endl;
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014			
0x40960018			
0x4096001c			
0x40960020			
0x40960024			

Evaluate main()

```
void add_five( int& x ) {
    x += 5;
}

int main() {
    int a(4);
    cout << a << endl;
    add_five( a );
    cout << a << endl;
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014			
0x40960018			
0x4096001c	a	4	
0x40960020			
0x40960024			a 0x4096001c

Evaluate main()

```
void add_five( int& x ) {
    x += 5;
}

int main() {
    int a(4);
    cout << a << endl;
    add_five( a );
    cout << a << endl;
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014			
0x40960018			
0x4096001c	a	4	
0x40960020			
0x40960024			a 0x4096001c

Pass by Reference

```
void add_five( int& x ) {
    x += 5;
}

int main() {
    int a(4);
    cout << a << endl;
    add_five( a );
    cout << a << endl;
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014			
0x40960018			
0x4096001c	a	4	
0x40960020			
0x40960024			a 0x4096001c

Pass by Reference

```
void add_five( int& x ) {
    x += 5;
}

int main() {
    int a(4);
    cout << a << endl;
    add_five( a );
    cout << a << endl;
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014			
0x40960018			
0x4096001c	a x	4	
0x40960020			x 0x4096001c
0x40960024			a 0x4096001c

Evaluate add_five()

```
void add_five( int& x ) {
    x += 5;
}

int main() {
    int a(4);
    cout << a << endl;
    add_five( a );
    cout << a << endl;
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014			
0x40960018			
0x4096001c	a x	9	
0x40960020			x 0x4096001c
0x40960024			a 0x4096001c

Evaluate main()

```
void add_five( int& x ) {
    x += 5;
}

int main() {
    int a(4);
    cout << a << endl;
    add_five( a );
    cout << a << endl;
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014			
0x40960018			
0x4096001c	a	9	
0x40960020			
0x40960024			a 0x4096001c

Return by Value

```
void add_five( int& x ) {
    x += 5;
}

int main() {
    int a(4);
    cout << a << endl;
    add_five( a );
    cout << a << endl;
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014			
0x40960018			
0x4096001c	a	9	
0x40960020			
0x40960024			a 0x4096001c

Program Terminates

```
void add_five( int& x ) {
    x += 5;
}

int main() {
    int a(4);
    cout << a << endl;
    add_five( a );
    cout << a << endl;
    return 0;
}
```

Address	Identifier	Value	Stack
0x40960014			
0x40960018			
0x4096001c		9	
0x40960020			
0x40960024			

On Tap For Today

- Reference
- PBV / PBR / PBP
- Practice

Practice #1: PBV / PBR / PBP

- What's the difference?

```
void f1(int x)   { x   = 3; }
void f2(int* pZ) { *pZ = 3; }


...
int x = 1, z = 1;
f1(x);
f2(&z);
cout << x << " " << z;
```

Practice #1: PBV / PBR / PBP

- What's the difference?

```
void f1(int x)   { x   = 3; }
void f2(int* pZ) { *pZ = 3; }


...
int x = 1, z = 1;
f1(x);
f2(&z);
cout << x << " " << z;    // 1 3
```

Practice #2: PBV / PBR / PBP

- What's the difference?

```
void g2(int* pY) { pY = new int; }

void g3(int** ppZ) { *ppZ = new int; }


...

int *ptr = nullptr;
int *ptr2 = nullptr;

g2(ptr);
g3(&ptr2);

cout << *ptr << " " << *ptr2;
```

- What also happens with each of these?

Practice #2: PBV / PBR / PBP

- What's the difference?

```
void g2(int* pY) { pY = new int; }

void g3(int** ppZ) { *ppZ = new int; }


...

int *ptr = nullptr;
int *ptr2 = nullptr;

g2(ptr);                          // memory leak
g3(&ptr2);

cout << *ptr << " " << *ptr2;     // NPE 0
```

- What also happens with each of these?

Practice #3: PBV / PBR / PBP

- What's the difference?

```
void f1(int x)   { x   = 3; }
void f2(int* pZ) { *pZ = 4; }
void g2(int* pY) { pY = new int(5); }

...

int x = 1, z = 1;
int *ptr = new int(6);
int *ptr2 = new int(7);

f1(x);
f1(*ptr);

f2(&z);
f2(ptr);

g2(&z);
g2(ptr2);
```

Practice #3: PBV / PBR / PBP

- What's the difference?

```
void f1(int x)   { x   = 3; }
void f2(int* pZ) { *pZ = 4; }
void g2(int* pY) { pY = new int(5); }

...

int x = 1, z = 1;
int *ptr = new int(6);
int *ptr2 = new int(7);

f1(x);    // x is 1
f1(*ptr); // *ptr is 6

f2(&z);   // z is 4
f2(ptr);  // *ptr is 4

g2(&z);   // z is 4 + memory leak
g2(ptr2); // *ptr2 is 7 + memory leak
```

Practice #4: PBV / PBR / PBP

- What's happens?

```
void h1(int* pY)  { delete pY; pY = nullptr; }
void h2(int** ppZ) { delete *ppZ; *ppZ = nullptr; }


...
int *p1 = new int(5);
int *p2 = new int(7);
h1(p1);
h2(&p2);
cout << *p1 << " ";        // what happens?
cout << *p2;               // what happens?
```

Practice #4: PBV / PBR / PBP

- What's happens?

```
void h1(int* pY)  { delete pY; pY = nullptr; }
void h2(int** ppZ) { delete *ppZ; *ppZ = nullptr; }


...
int *p1 = new int(5);
int *p2 = new int(7);
h1(p1);
h2(&p2);
cout << *p1 << " ";        // seg fault - dangling pointer
cout << *p2;               // seg fault - NPE
```

Practice #5: PBV / PBR / PBP

- What's the difference?

```
void f1(int x)   { x   = 3; }
void f2(int& y)  { y   = 3; }
void f3(int* pZ) { *pZ = 3; }


...
int x = 1, y = 1, z = 1;
f1(x);   // what is x?
f2(y);   // what is y?
f3(&z);  // what is z?
```

Practice #5: PBV / PBR / PBP

- What's the difference?

```
void f1(int x)   { x   = 3; }
void f2(int& y)  { y   = 3; }
void f3(int* pZ) { *pZ = 3; }


...
int x = 1, y = 1, z = 1;
f1(x);   // 1
f2(y);   // 3
f3(&z);  // 3
```

Practice #6: PBV / PBR / PBP

- What's the difference?

```
void g1(int* pY)  { pY = new int; }
void g2(int*& pZ) { pZ = new int; }


...
int* p1 = nullptr;
int* p2 = nullptr;
g1(p1);   // what does p1 point to?
g2(p2);   // what does p2 point to?
```

- What also happens with each of these?

Practice #6: PBV / PBR / PBP

- What's the difference?

```
void g1(int* pY)  { pY = new int; }
void g2(int*& pZ) { pZ = new int; }


...
int* p1 = nullptr;
int* p2 = nullptr;
g1(p1);   // nullptr + memory leak
g2(p2);   // 0
```

- What also happens with each of these?

Practice #7: PBV / PBR / PBP

- What's happens?

```
void h1(int* pY)  { delete pY; pY = nullptr; }
void h2(int*& pZ) { delete pZ; pZ = nullptr; }


...
int* p1 = new int;
int* p2 = new int;
h1( p1 );
h2( p2 );
cout << p1 << " ";       // value?
cout << p2;              // value?
```

Practice #7: PBV / PBR / PBP

- What's happens?

```
void h1(int* pY)  { delete pY; pY = nullptr; }
void h2(int*& pZ) { delete pZ; pZ = nullptr; }


...
int* p1 = new int;
int* p2 = new int;
h1( p1 );
h2( p2 );
cout << p1 << " ";       // free store address (dangling pointer)
cout << p2;              // 0x0 (null pointer)
```

On Tap For Today

- Reference
- PBV / PBR / PBP
- Practice

To Do For Next Time

- L2C due before class on Monday
- A2 due Tuesday 11:59 pm
- Coming Monday
  - A new programming paradigm!
  - Do readings ahead of time