CSCI 200: Foundational Programming Concepts & Design Lecture 10

Dynamic Memory Allocation & Deallocation

Pass-by-Pointer

Log into iClicker

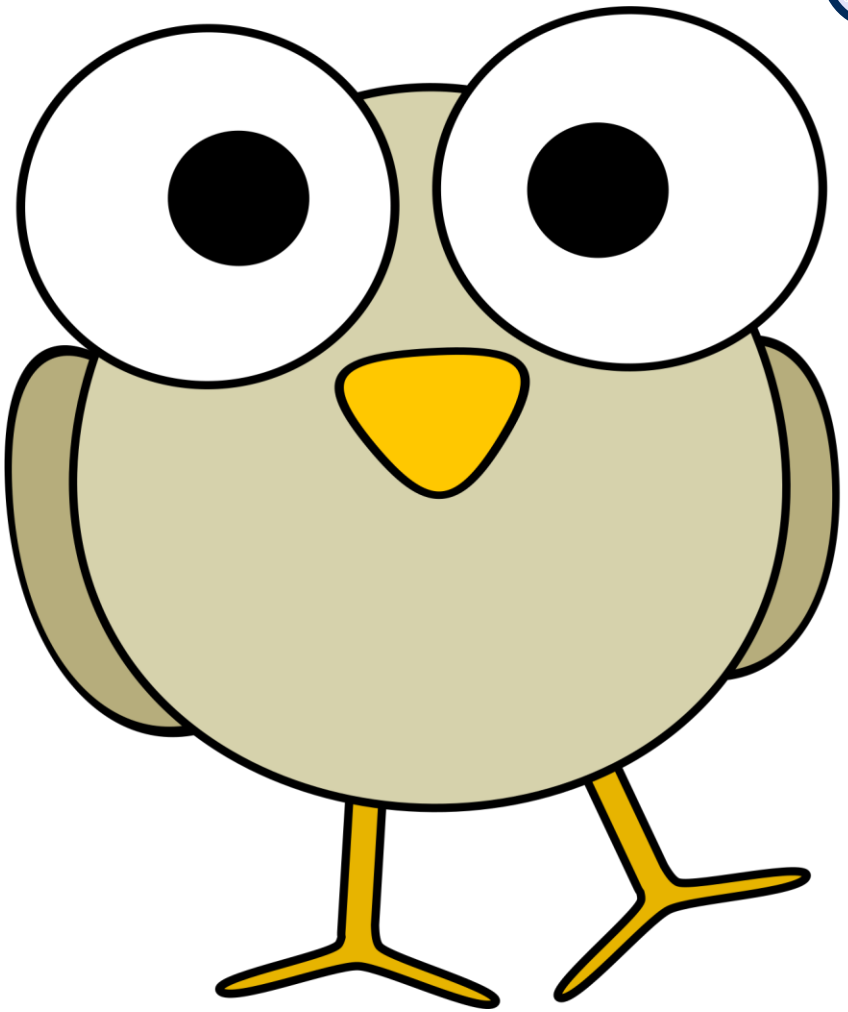
Previously in CSCI 200

- Stack: storage for variables known at compile time
- Free Store: pool of unused memory for dynamic memory
- Pointer points to a value at a memory address
  - Type of pointer is type of value
- Use a pointer to store values on the free store

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

Questions?



Learning Outcomes For Today

- Diagram the memory associated with pointers and where the values lie (either in the stack or the free store).
- Diagram how pass-by-pointer works with pass-by-value and pass-by-reference in functions.
- Discuss causes of & solutions to memory leaks, segmentation faults, dangling pointers, null pointer exceptions, and other pointer related errors.

Important Note

- We're going to be using pointers extensively here on out.
- If anything's unclear today, ask!

On Tap For Today

- Pointers
- Pass-By-Value and Return-By-Value
- Pass-By-Pointer and Return-By-Pointer
- Practice

On Tap For Today

- Pointers
- Pass-By-Value and Return-By-Value
- Pass-By-Pointer and Return-By-Pointer
- Practice

Practice #1: What is a pointer?

Practice #2: What is printed?

```
01 double a = 12.0;
02 double b = 23.0;

03 double *ptr = &a;
04 *ptr = b;
05 ptr = &b;
06 *ptr = 13.0;

07 cout << a << " " << b;
```

Practice #2: What is printed?

```
01 double a = 12.0;
02 double b = 23.0;

03 double *ptr = &a;
04 *ptr = b;
05 ptr = &b;
06 *ptr = 13.0;

07 cout << a << " " << b; // 23 13
```

Practice #3: What is printed?

```
01 double a;
02 double b = 5.0;

03 cout << a << " " << b;
```

Practice #3: What is printed?

```
01 double a;
02 double b = 5.0;

03 cout << a << " " << b; // undefined 5
```

Practice #4: What is printed?

```
01 double *ptr = new double;

02 cout << ptr << endl;
03 cout << *ptr << endl;

04 *ptr = 2.25;

05 cout << ptr << endl;
06 cout << *ptr << endl;

07 delete ptr;

08 cout << ptr << endl;
```

Practice #4: What is printed?

```
01 double *ptr = new double;

02 cout << ptr << endl;  // an address
03 cout << *ptr << endl; // 0

04 *ptr = 2.25;

05 cout << ptr << endl;  // same address
06 cout << *ptr << endl; // 2.25

07 delete ptr;

08 cout << ptr << endl;  // same address
```

Practice #5: What error occurs?

```
01 double *ptr = nullptr;

02 *ptr = 22;

03 cout << *ptr;

04 delete ptr;
```

A) Memory Leak
B) Double Deallocation
C) Dangling Pointer
D) Null Pointer Exception
E) There is no error

Practice #5: What error occurs?

```
01 double *ptr = nullptr;

02 *ptr = 22;

03 cout << *ptr;

04 delete ptr;
```

A) Memory Leak
B) Double Deallocation
C) Dangling Pointer
D) Null Pointer Exception
E) There is no error

Practice #6: What error occurs?

```
01 double *ptr = new double;

02 *ptr = 22;

03 delete ptr;

04 cout << *ptr;
```

A) Memory Leak
B) Double Deallocation
C) Dangling Pointer
D) Null Pointer Exception
E) There is no error

Practice #6: What error occurs?

```
01 double *ptr = new double;

02 *ptr = 22;

03 delete ptr;

04 cout << *ptr;
```

A) Memory Leak
B) Double Deallocation
C) Dangling Pointer
D) Null Pointer Exception
E) There is no error

Practice #7: What error occurs?

```
01 void foo() {
02    double *ptr = new double;
03    *ptr = 22;
04    cout << *ptr << endl;
05 }
06 int main() {
07    foo();
08    foo();
09 }
```

A) Memory Leak
B) Double Deallocation
C) Dangling Pointer
D) Null Pointer Exception
E) There is no error

Practice #7: What error occurs?

```
01 void foo() {
02    double *ptr = new double;
03    *ptr = 22;
04    cout << *ptr << endl;
05 }
06 int main() {
07    foo();
08    foo();
09 }
```

A) **Memory Leak**
B) Double Deallocation
C) Dangling Pointer
D) Null Pointer Exception
E) There is no error

Practice #8: What error occurs?

```
01 double *ptr = new double;

02 *ptr = 22;


03 delete ptr;
04 delete ptr;
```

A) Memory Leak
B) Double Deallocation
C) Dangling Pointer
D) Null Pointer Exception
E) There is no error

Practice #8: What error occurs?

```
01 double *ptr = new double;

02 *ptr = 22;


03 delete ptr;
04 delete ptr;
```

A) Memory Leak

B) Double Deallocation

C) Dangling Pointer

D) Null Pointer Exception

E) There is no error

Practice #9: What error occurs?

```
01 double *ptr = new double;

02 *ptr = 22;

03 delete ptr;

04 ptr = nullptr;
05 delete ptr;
```

A) Memory Leak
B) Double Deallocation
C) Dangling Pointer
D) Null Pointer Exception
E) There is no error

Practice #9: What error occurs?

```
01 double *ptr = new double;

02 *ptr = 22;

03 delete ptr;

04 ptr = nullptr;
05 delete ptr;
```

A) Memory Leak
B) Double Deallocation
C) Dangling Pointer
D) Null Pointer Exception
E) There is no error

Common Errors

- Dereferencing a pointer that doesn't point to anything anymore (seg fault due to dangling pointers!!)
- Not returning dynamic memory when done (memory leak!!)
- Dereferencing a null pointer (seg fault due to null pointer exception!!)
- Using `delete` on a variable not created with `new`
  - E.g. trying to delete from the stack
- Using `delete` on a pointer that's already deallocated
- Thinking pointer points to `x` when it actually points to `y`

On Tap For Today

- Pointers
- Pass-By-Value and Return-By-Value
- Pass-By-Pointer and Return-By-Pointer
- Practice

Practice #10: PBV - What is the output?

```
01 void enter_coordinate(int x, int y) {
02   cout << "Enter (x, y) coordinate: ";
03   cin >> x >> y;
04 }

05 ...

06 int x = 1, y = 1;

07 enter_coordinate(x, y); // user enters 4 5

08 cout << x << " " << y;
```

Practice #10: PBV - What is the output?

```
01 void enter_coordinate(int x, int y) {
02   cout << "Enter (x, y) coordinate: ";
03   cin >> x >> y;
04 }

05 ...

06 int x = 1, y = 1;

07 enter_coordinate(x, y); // user enters 4 5

08 cout << x << " " << y;  // 1 1
```

Practice #11: RBV - What is the output?

```
01 int enter_coordinate_x() {
02   int x;
03   cout << "Enter (x, y) X coordinate: ";
04   cin >> x;
05   return x;
06 }

07 int enter_coordinate_y() {
08   // same as above but for y
09 }

10 ...

11 int x = 1, y = 1;

12 x = enter_coordinate_x();  // user enters 4
13 y = enter_coordinate_y();  // user enters 5

14 cout << x << " " << y;
```

Practice #11: RBV - What is the output?

```
01 int enter_coordinate_x() {
02   int x;
03   cout << "Enter (x, y) X coordinate: ";
04   cin >> x;
05   return x;
06 }

07 int enter_coordinate_y() {
08   // same as above but for y
09 }

10 ...

11 int x = 1, y = 1;

12 x = enter_coordinate_x();  // user enters 4
13 y = enter_coordinate_y();  // user enters 5

14 cout << x << " " << y;     // 4 5
```

On Tap For Today

- Pointers
- Pass-By-Value and Return-By-Value
- Pass-By-Pointer and Return-By-Pointer
- Practice

Practice #12: PBP - What is the output?

```
01 void enter_coordinate(int *pX, int *pY) {
02   cout << "Enter (x, y) coordinate: ";
03   cin >> *pX >> *pY;
04 }

05 ...

06 int x = 1, y = 1;

07 enter_coordinate(&x, &y);  // user enters 4 5

08 cout << x << " " << y;
```

Practice #12: PBP - What is the output?

```
01 void enter_coordinate(int *pX, int *pY) {
02   cout << "Enter (x, y) coordinate: ";
03   cin >> *pX >> *pY;
04 }

05 ...

06 int x = 1, y = 1;

07 enter_coordinate(&x, &y);  // user enters 4 5

08 cout << x << " " << y;     // 4 5
```

Functions & Pointers V1

- Pass By Pointer

```
void pointer_setter(int * const P_value, const int VALUE) {
  *P_value = VALUE;
}
int main() {
  int *pX = new int(0);
  pointer_setter(pX, 5);
  cout << *pX << endl;
  return 0;
}
```

Functions & Pointers V2

- Pass By Pointer

```
void pointer_setter(int * const P_value, const int VALUE) {
  *P_value = VALUE;
}
int main() {
  int x = 0;
  pointer_setter(&x, 5);
  cout << x << endl;
  return 0;
}
```

Practice #13: PBV / 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 #13: PBV / 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 #14: PBV / 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 << endl;
cout << *ptr2 << endl;
```

- What also happens with each of these?

Practice #14: PBV / 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 << endl;  // NPE
cout << *ptr2 << endl; // 0
```

- What also happens with each of these?
  - Memory Leak

PBV / 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);
```

PBV / 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
g2(ptr2); // *ptr2 is 7
```

Practice #15: Other Concerns

- What's happens?

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


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

Practice #15: Other Concerns

- What's happens?

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


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

Passing Pointers

- Pass a Pointer By Value when needing to manipulate _____
- Pass a Pointer By Pointer when needing to manipulate _____

Practice #16: Return a Pointer from Function

- Return the pointer to a value

```
int* f() {
  int localStackVariable = 5;
  return &localStackVariable;
}

int* g() {
  int *localFreeStorePointer = new int(5); // be careful of memory leaks
  return localFreeStorePointer;
}


int *ptr = f();
int *ptr2 = g();


cout << *ptr << endl;                    // what does this print?
cout << *ptr2 << endl;                   // what does this print?

delete ptr;                              // what happens?
delete ptr2;                             // what happens?
```

Practice #16: Return a Pointer from Function

- Return the pointer to a value

```
int* f() {
  int localStackVariable = 5;
  return &localStackVariable;
}

int* g() {
  int *localFreeStorePointer = new int(5); // be careful of memory leaks
  return localFreeStorePointer;
}


int *ptr = f();
int *ptr2 = g();


cout << *ptr << endl;                    // undefined
cout << *ptr2 << endl;                   // 5

delete ptr;                              // error - pointer not allocated
delete ptr2;                             // succeeds - dangling pointer
```

On Tap For Today

- Pointers
- Pass-By-Value and Return-By-Value
- Pass-By-Pointer and Return-By-Pointer
- Practice

To Do For Next Time

- Procedural Programming Quiz on Monday
  - No pointers
- Exam I extra credit due Monday
- Exam I in class on Wednesday
  - No pointers