CSCI 200: Foundational Programming Concepts & Design Lecture 10

Dynamic Memory Allocation & Deallocation

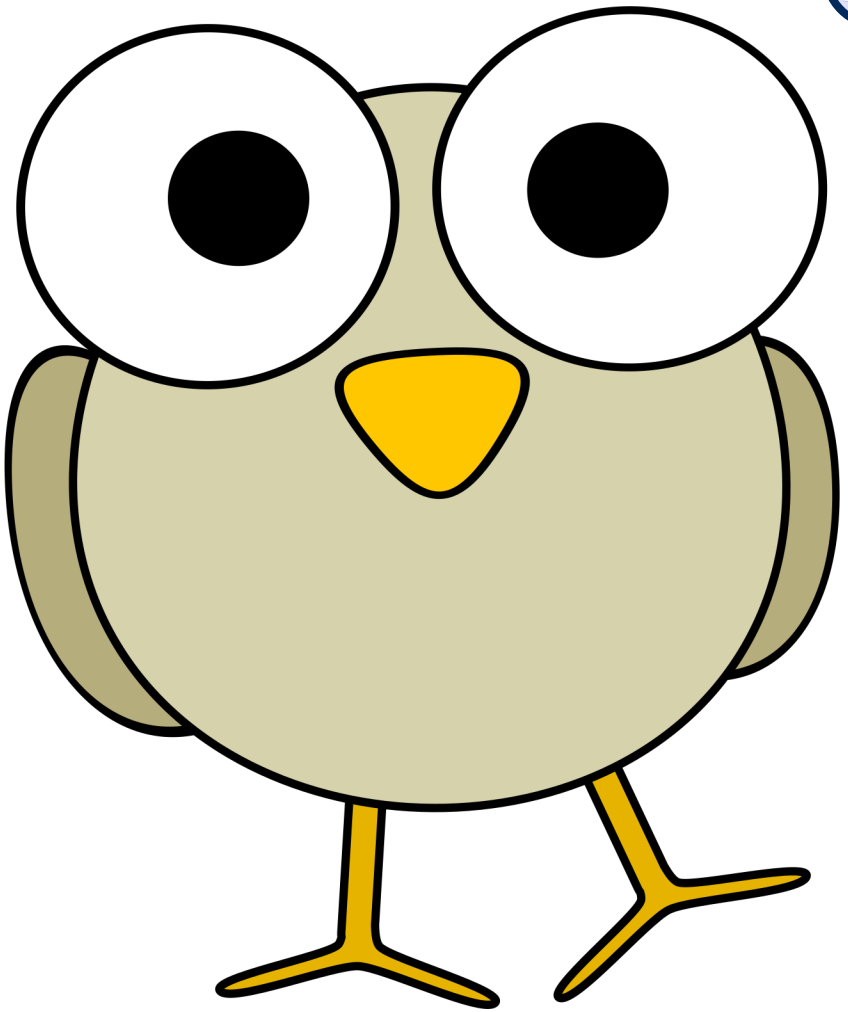
Pass-by-Pointer

Have Canvas open and have paper on hand

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

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

Canvas Survey

- 9/15 In Class Survey
  - Access code: **englishpointer**

- Answer question 1 to start

Practice #2: What is printed?

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

double *ptr = &a;
*ptr = b;
ptr = &b;
*ptr = 13.0;

cout << a << endl;
cout << b << endl;
```

Practice #3: What is printed?

```
double a;
double b = 5.0;

cout << a << endl;
cout << b << endl;
```

Practice #4: What is printed?

```
double *ptr = new double;

cout << ptr << endl;
cout << *ptr << endl;

*ptr = 2.25;

cout << ptr << endl;
cout << *ptr << endl;

delete ptr;

cout << ptr << endl;
```

Practice #5: What error occurs?

```
double *ptr = nullptr;

*ptr = 22;

cout << *ptr << endl;


delete ptr;
```

Null Pointers

- A pointer that doesn't point to anything, set to null

```
double *ptr = nullptr;  // 0x0
```

- Requires compiling against C++11 or newer

```
g++ -std=c++17 -o main.o -c main.cpp
```

- Put it in the Makefile!

```
# in Makefile
CXXVERSION = -std=c++17
```

Practice #6: What error occurs?

```
double *ptr = new double;

*ptr = 22;


delete ptr;

cout << *ptr << endl;
```

Practice #7: What error occurs?

```
void foo() {
   double *ptr = new double;
   *ptr = 22;
   cout << *ptr << endl;
}
int main() {
   foo();
   foo();
}
```

Practice #8: What error occurs?

```
double *ptr = new double;

*ptr = 22;


delete ptr;
delete ptr;
```

Practice #9: What error occurs?

```
double *ptr = new double;

*ptr = 22;


delete ptr;


ptr = nullptr;
delete ptr;
```

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 happens?

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

...

int x = 1, y = 1;

enter_coordinate(x, y);   // user types 4 5

cout << x << endl;
cout << y << endl;
```

Practice #11: RBV - What happens?

```
int enter_coordinate_x() {
  int x;
  cout << "Enter (x, y) X coordinate: ";
  cin >> x;
  return x;
}

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

...

int x = 1, y = 1;

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

cout << x << endl;
cout << y << endl;
```

On Tap For Today

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

Practice #12: PBP - What happens?

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

...

int x = 1, y = 1;

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

cout << x << endl;
cout << y << endl;
```

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);
f3(&z);
cout << x << endl;
cout << z << endl;
```

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?

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?
```

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?
```

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