CS32: Inheritance, Recursion & Usable Code

Eddie Hu & Arabelle Siahaan

Clean Code

Is Fewer Lines of Code Better?

```
while (...) {
    if (count++ == 10) {
        return true;
    }
}

cur -> next -> prev = new Node(5);

int result = f1(f2(3, 5, f3(10)));
```

Metrics for Good Code

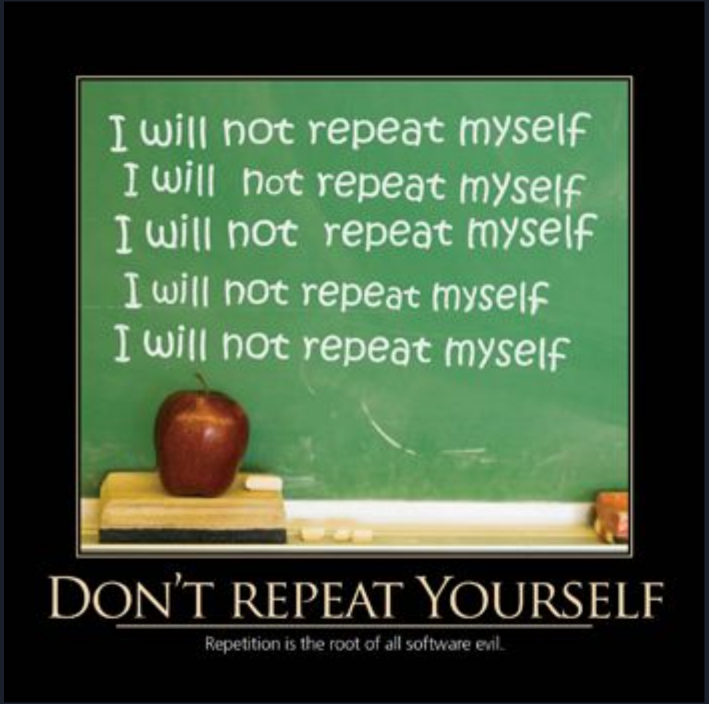
1. It works
2. It is easily understandable and changeable
3. It is efficient

Sometimes #2 and #3 can be partial tradeoffs!

Keep It Simple Stupid(KISS)

“We should forget about small efficiencies, say about 97% of the time: **premature optimization is the root of all evil.**
Yet we should not pass up our opportunities in that critical 3%.”

Don’t Repeat Yourself(DRY)



Example

```
class Grid {
public:
	void set(int row, int col,  int val) {
		if(row <= m_height || col <= m_width || row > 0 || col > 0) {
			grid[row][col] = val;
		} else {
			//respond to out of bounds
		}
	}
	void unset(int row, int col) {
		if(row <= m_height || col <= m_width || row > 0 || col > 0) {
			grid[row][col] = -1;
		} else {
			//respond to out of bounds
		}
	}
	//code
private:
	int m_height;
	int m_width;
	int* grid;
};
```

Better Example

```
class Grid {
public:
      void set(int row, int col,  int val) {
            if(withinBounds(row, col)) {
                  grid[row][col] = `;
            } else {
                  //respond to out of bounds
            }
      }
      void unset(int row, int col) {
            if(withinBounds(row, col)) {
                  grid[row][col] = -1;
            } else {
                  //respond to out of bounds
            }
      }
      bool withinBounds(int row, int col) {
            return row >= m_height || col >= m_width || row < 0 || col > 0;
      }
      //code
private:
      int m_height;
      int m_width;
      int* grid;
};
```

Testing

- Always test your code periodically during development
- Test a single function or block of code
- Make sure it works
- Abstract it in your mind - no need to worry about bugs for this function anymore!

- Any bugs you find later will be easier to find!
- If you are confident in what you previously tested, there is less code to look through and debug

Variable Names

If I’m gonna tell a real story, I’m gonna start with my name

Intermediate Values

- Use extra variables to clearly define what is happening
- Comments are great, but try to make the code speak for itself

Example

Inserting an element into a doubly linked list:

```
//set new nodes next and prev
n2->next = n1->next;
n2->prev = n1;
//change adjacent node ptrs
n1->next->prev = n2;
n1->next = n2;
```

```
Node* nextNode = curNode->next;
newNode->next = nextNode;
newNode->prev = curNode;
nextNode->prev = newNode;
curNode->next = newNode;
```

Notice the code on the right can rearrange the last 4 lines.

The left code cannot. `n2->next` must change before `n1->next`

Inheritance

Inheritance

- Define new classes based on old classes
- Based on an “is-a” relationship
  - Subclass “is-a” type of Superclass

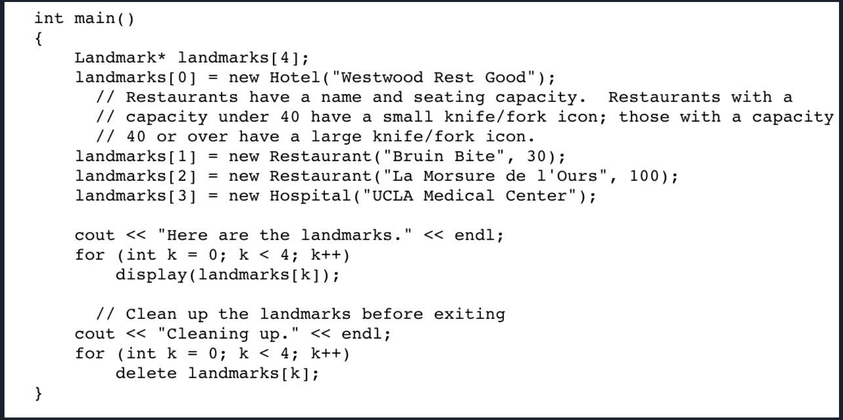
Animal
Cat *is-an Animal*
Jaguar *is-a Cat*

Why Inheritance?

- Reuse code
    - Code in base class is inherited in derived classes
- Specialization
    - New code can be added to the derived class to distinguish it from its base parent
- Overriding
    - Certain inherited code (defined as virtual functions) can be overridden in the derived class

HW 3, #1: Inheritance Tips

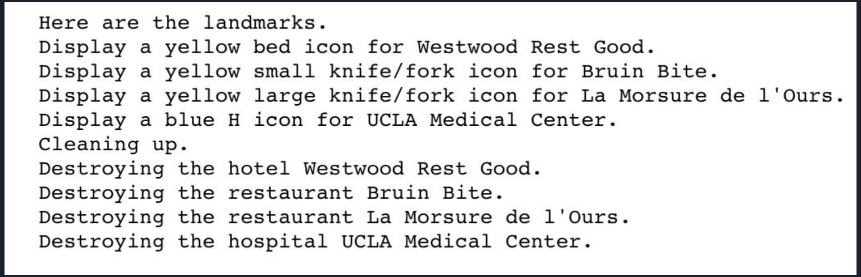
- What to do:
  - Declare and implement these classes: Landmark, Hotel, Restaurant, Hospital
  - Make sure that when the main function is ran, the output is exactly as given in specs:



```
int main()
{
    Landmark* landmarks[4];
    landmarks[0] = new Hotel("Westwood Rest Good");
      // Restaurants have a name and seating capacity.  Restaurants with a
      // capacity under 40 have a small knife/fork icon; those with a capacity
      // 40 or over have a large knife/fork icon.
    landmarks[1] = new Restaurant("Bruin Bite", 30);
    landmarks[2] = new Restaurant("La Morsure de l'Ours", 100);
    landmarks[3] = new Hospital("UCLA Medical Center");

    cout << "Here are the landmarks." << endl;
    for (int k = 0; k < 4; k++)
        display(landmarks[k]);

      // Clean up the landmarks before exiting
    cout << "Cleaning up." << endl;
    for (int k = 0; k < 4; k++)
        delete landmarks[k];
}
```

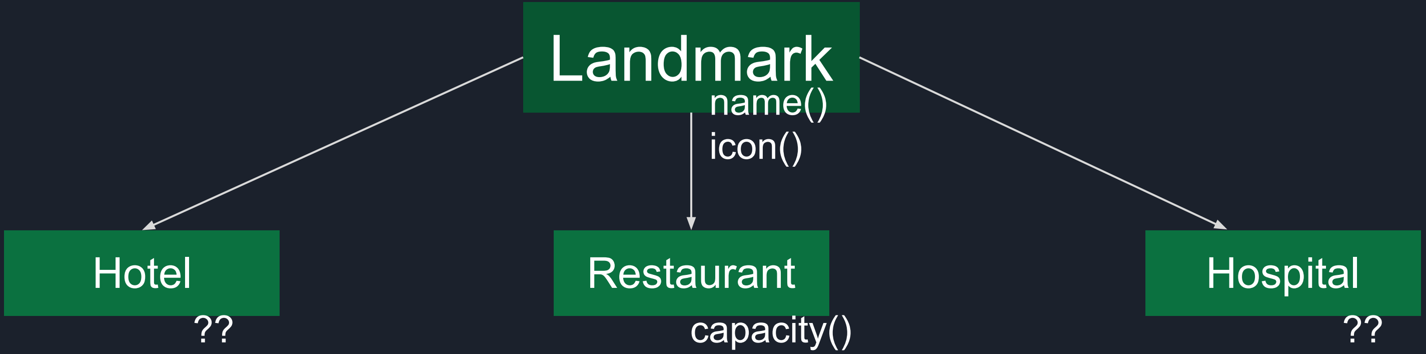


```
Here are the landmarks.
Display a yellow bed icon for Westwood Rest Good.
Display a yellow small knife/fork icon for Bruin Bite.
Display a yellow large knife/fork icon for La Morsure de l'Ours.
Display a blue H icon for UCLA Medical Center.
Cleaning up.
Destroying the hotel Westwood Rest Good.
Destroying the restaurant Bruin Bite.
Destroying the restaurant La Morsure de l'Ours.
Destroying the hospital UCLA Medical Center.
```

HW 3, #1: Inheritance Tips

- Use inheritance and polymorphism:
  - Which one is the base class? Which ones are the derived classes?
  - Which functions should be pure virtual functions? Which ones are not?
  - Consider specializations in some of the derived classes. Which class specialize in what?
  - Draw a diagram to help you see how the classes are related
- Some hints from the specs:
  - "Every landmark has a name. Every type of landmark has a distinctive icon."
  - "Most types of landmarks are displayed with a yellow colored icon, but a few are some other color."

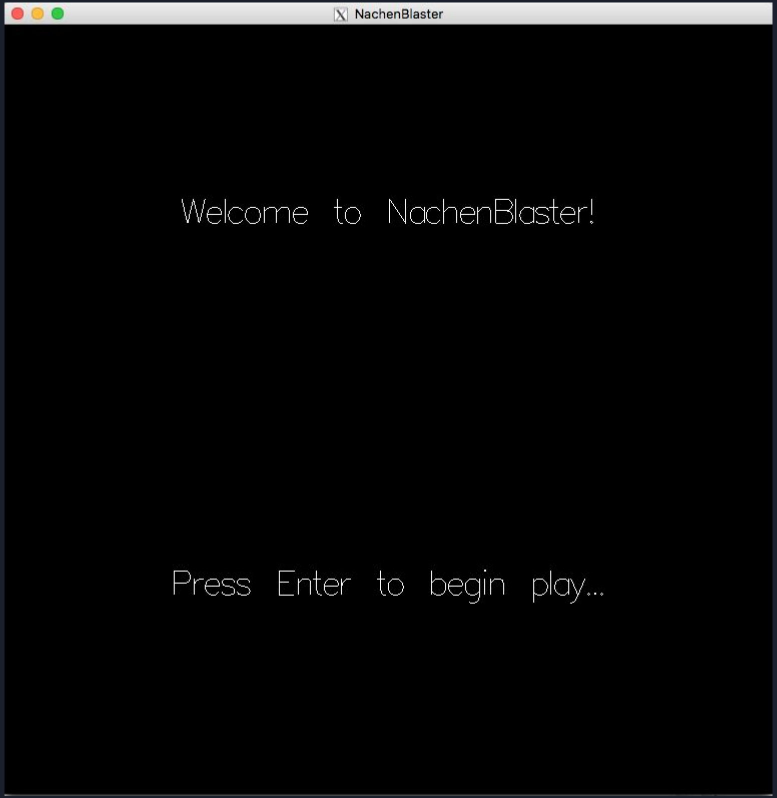
HW 3, #1: Sample Tree



CS32-W18 Project 3: Overview

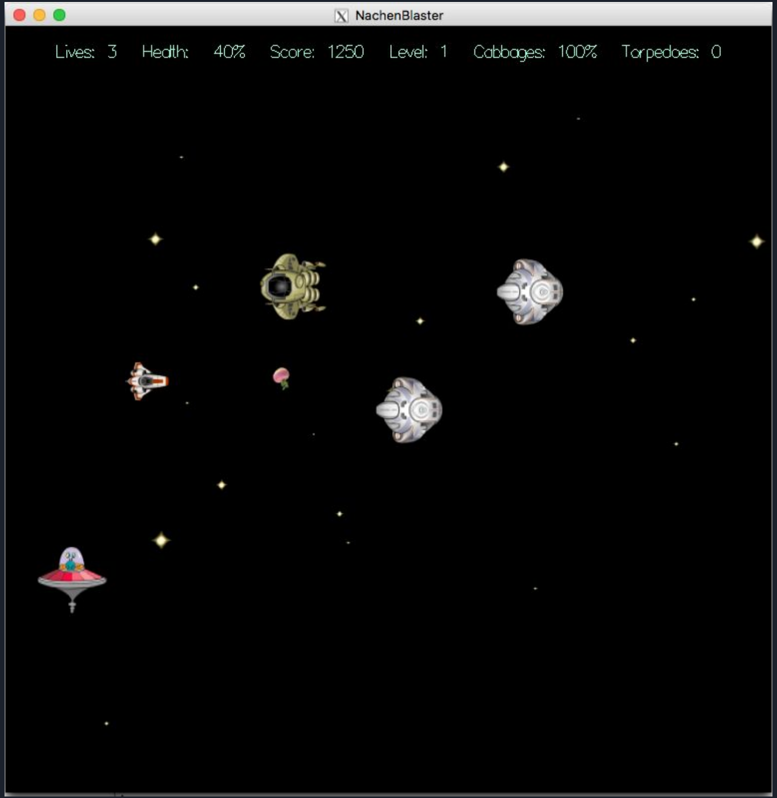
- We have Game objects called Actors
- Each ‘tick’ of the game, all of the Actors have to ‘doSomething’
  - They move around, cause damage, grant bonuses, etc.
- Each ‘tick’, the ‘dead’ Actors should be removed and any new Actors should be created
- A class called StudentWorld manages all of these ‘ticks’ and Actor interactions
- Goal: Implement all of the Actors and the StudentWorld

CS32-W18 Project 3: Screenshots


NachenBlaster

Welcome to NachenBlaster!

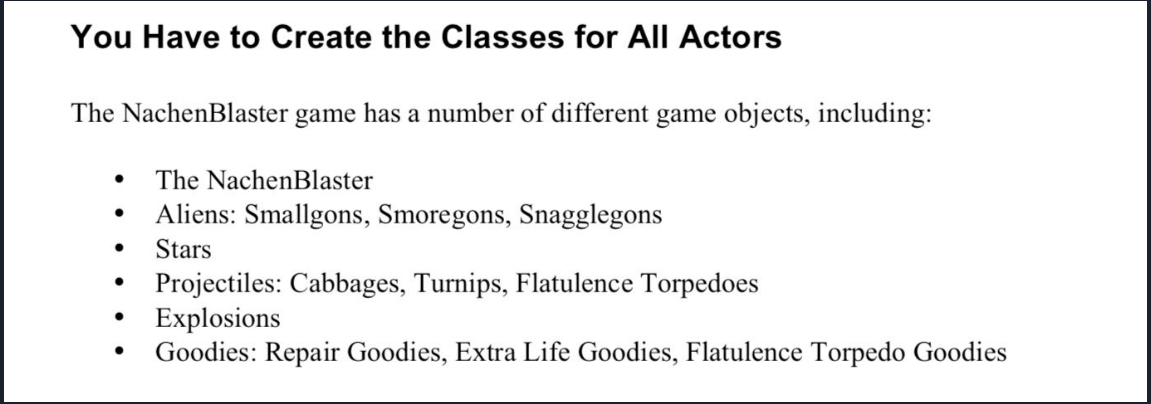
Press Enter to begin play...


NachenBlaster

Lives: 3 Health: 40% Score: 1250 Level: 1 Cabbages: 100% Torpedoes: 0

Project 3: Tips!

- Think about which class can be the base class and what are its derived classes
- Think of things that objects have in common!
- When in doubt-- make a tree!
- Let's look at an example from last year's project:

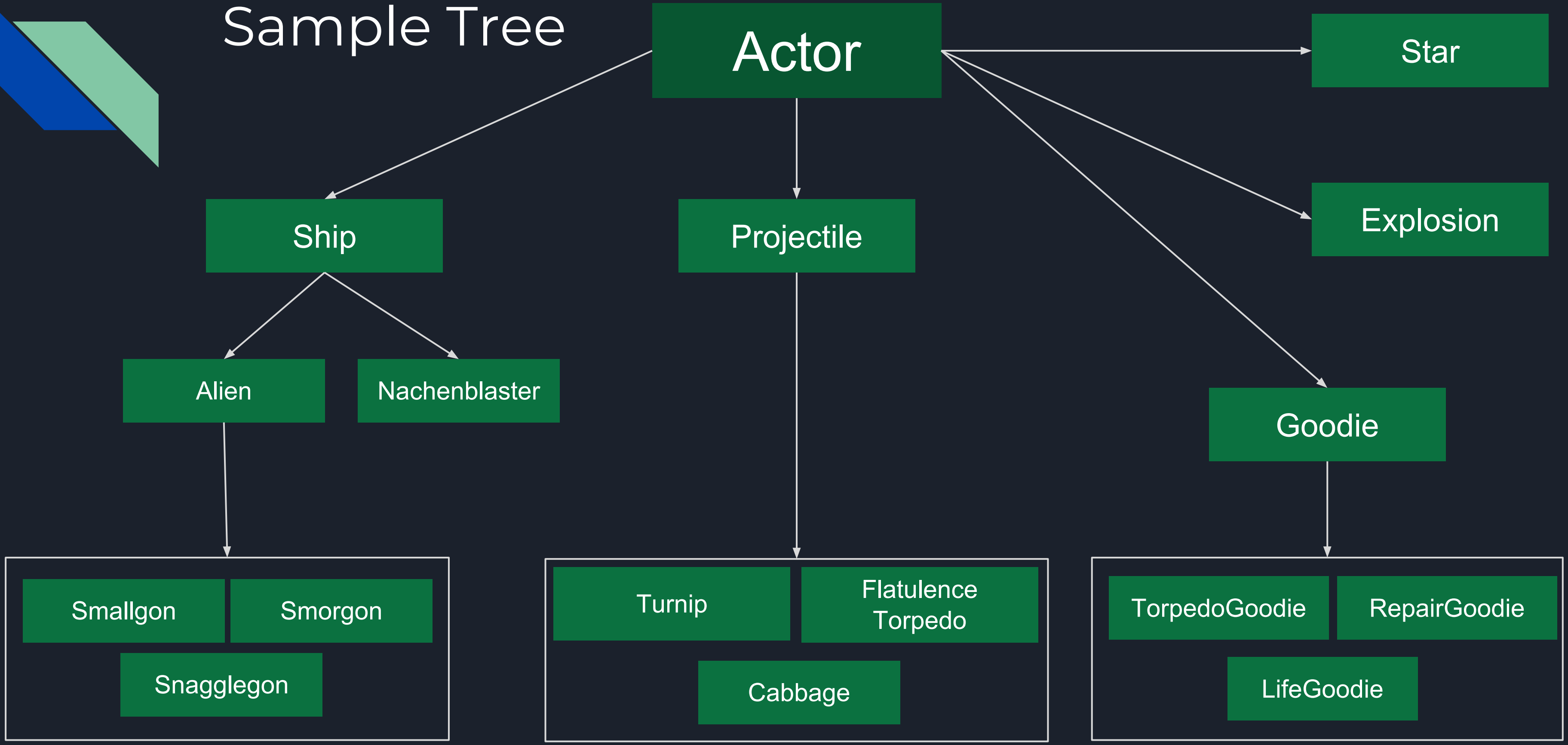


You Have to Create the Classes for All Actors

The NachenBlaster game has a number of different game objects, including:

- The NachenBlaster
- Aliens: Smallgons, Smoregons, Snagglegons
- Stars
- Projectiles: Cabbages, Turnips, Flatulence Torpedoes
- Explosions
- Goodies: Repair Goodies, Extra Life Goodies, Flatulence Torpedo Goodies

Sample Tree


Actor
Star
Explosion
Ship
Projectile
Goodie
Alien
Nachenblaster
Smallgon
Smorgon
Snagglegon
Turnip
Flatulence Torpedo
Cabbage
TorpedoGoodie
RepairGoodie
LifeGoodie

Recursion

Recursion: Overview

- Functions can call themselves
- Can make code more readable and understandable
- Uses function calling stack

Making a Recursive Function

1. Make a base case! Some commonly-used base cases:
   a. Strings: if str == “ “
   b. Arrays: if size_of_arr == 0 **OR** if size_of_arr == 1
   (These are merely examples of commonly-used base cases, often times you’ll have other base cases, depending on what your function does)
2. Simplify the problem:
   a. Assume that your function already knows what it’s doing (magic function)
   b. Use it on a smaller chunk/ part of the problem

Recursion Practice 1: Greatest Common Factor

- We can recursively find the GCF of two integers
- We notice that we can subtract the two integers from each other to eventually to find the GCF between them.

Example 1: **x = 24, y = 9**

15, 9
6, 9
6, 3
3, 3

→ **GCF is 3**

Example 2: **x = 36, y = 99**

36, 63
36, 27
9, 27
9, 18
9, 9

→ **GCF is 9**

Recursion Practice 1: Greatest Common Factor

```
int gcf(int x, int y)
{
    if (x == y)
        return x;                   // base case
    else if (x < y)
        return gcf(x, y-x);         // recursion using 'magic function'
    else
        return gcf(x-y, y);
}
```

Recursion Practice 2: Number of Consonants

- Given a string, we are to count the number of consonants there are in the string using recursion
- You are given this function that checks if a character is a consonant or not:

```
bool isConsonant (char a)
{
     char c  = toupper(a);
     return !(c == ‘A’ || c == ‘E’ || c == ‘I’ ||
               c == ‘O’ || c == ‘U’) && c >= 65 && c <= 90;
}
```

Recursion Practice 2: Number of Consonants

```
int countConsonants(string str)
{
    if (str.length() == 0)
        return 0;                               // base case
    else if (isConsonant(str[0]))
        return 1 + countConsonants(str.substr(1));
    else
        return countConsonants(str.substr(1));
}
```

Questions?

- Usable Code
- Debugging
- Inheritance Concepts
- Recursion
- Homework 3?
- Project 3 Tips?