CS 161
Introduction to CS I
Lecture 13

- Function overloading
- How can functions make changes to their arguments?
  - Passing function arguments by value and reference
- Introduction to pointers

Oregon State University
College of Engineering

Useful tips

- Assignment 3 peer reviews – **write in the comment boxes, not the rubric (ignore if Canvas says it is not complete)**
- Practice proficiency demo status
- Revision plans – submit within 48 hours of demo + email TA
- Study session (Thursday, 6-7 p.m., **LINC 268**) – print and complete the worksheet in advance
- More practice?
  - Edabit: example problems for loops, etc.
- Assignment 3 implementation questions?

Oregon State University
College of Engineering

C++ function overloading

- Overloading: Two or more functions with the same name

```
int sum(int a, int b) {
        return a + b;
}
```

```
string sum(string a, string b) {
        return a + b;
}
```

- For the compiler to know which one to call, the functions must have:
  - Different data types for the parameters
  - Or different number of parameters

```
int sum(int a, int b, int c) {
    return a + b + c;
}
```

  - Different return types alone are NOT sufficient

C++ function overloading

- Compiler decides which function to call based on the following:
  - **Exact match**: if the number and types of arguments exactly match a definition (without any automatic type conversion), then that is the definition used
  - Match using **implicit type conversion**: if there is no exact match but there is using implicit type conversion, then the match is used

Implicit conversion from int to float

```
float sum(float a, float b) {
    return a + b;
}
...
int z = sum(3, 5);
```

Warning! Implicit conversion from float to int

```
int sum(int a, int b) {
    return a + b;
}
...
float z = sum(3.2, 5.3);
```

C++ function overloading

- If two valid options are present, it is ambiguous (cannot decide)

Ambiguous – which function to call?

```
float sum(float a, float b) {
    return a + b;
}
int sum(int a, int b) {
    return a + b;
}
...
float z = sum(3.2, 5.3);
```

Unambiguous – must use first sum()

```
float sum(float a, float b) {
    return a + b;
}
string sum(string a, string b) {
    return a + b;
}
...
float z = sum(3.2, 5.3);
```

Course map

Divide and conquer part 2
(recursion)

Structured data
(arrays and objects)

Basics
Storing data, calculations, interacting with users

Decision making (adaptation) and **repetition** (write once, repeat forever!)

Divide and conquer
(modularization and code re-use in functions)

Dynamic growth
(memory allocation and management)

Oregon State University
College of Engineering

Variable scope: practice (1)

What does this code print out?

```
int z = 14;
for (int z = 0; z < 3; z++) {
  cout << z << endl;
}
cout << z << endl;
```

```
0
1
2
14
```

Variable scope: practice (2)

Oregon State University
College of Engineering

What does this code print out?

```
for (int z = 0; z < 3; z++) {
  cout << z << endl;
}
cout << z << endl;
```

Error!
Will not compile
(z not in scope
on final line)

Oregon State University
College of Engineering

Variable scope: practice (3)

What does this code print out?

```
int z = 14;
for (z = 0; z < 3; z++) {
  cout << z << endl;
}
cout << z << endl;
```

0
1
2
3

Oregon State University
College of Engineering

Variable scope: practice (4)

What does this code print out?

```
int z = 14;
for (z = 0; z < 3; z++)
  cout << z << endl;

cout << z << endl;
```

0
1
2
3

Variable scope in functions

```
1. void compute_sum() {
2.    int sum = x + y;
3. }

4. int main() {
5.    int x = 2, y = 3;
6.    compute_sum();
7.    cout << sum << endl;
8.    return 0;
9. }
```

Where are the error(s)?

Oregon State University
College of Engineering

Variable scope in functions: errors

```
1. void compute_sum() {
2.    int sum = x + y;    /* error: x and y outside scope */
3. }

4. int main() {
5.    int x = 2, y = 3;
6.    compute_sum();
7.    cout << sum << endl;/* error: sum not declared */
8.    return 0;
9. }
```

Variable scope: errors fixed

```
1. int compute_sum(int x, int y) {
2.    int sum = x + y;
3.    return sum;
4. }

5. int main() {
6.    int x = 2, y = 3;
7.    int sum = compute_sum(x, y);
8.    cout << sum << endl;
9.    return 0;
10.}
```

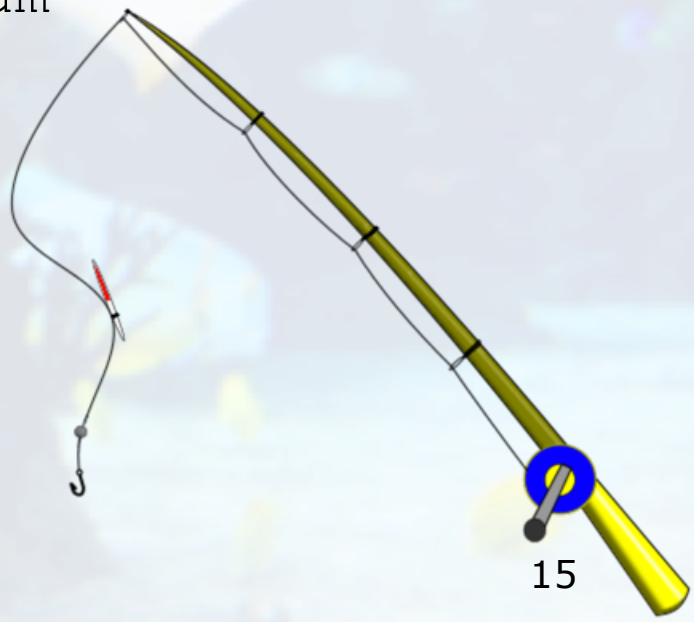
Variable scope: errors fixed (version 2)

```
1. int compute_sum(int a, int b) {
2.    int sum = a + b;
3.    return sum;
4. }

5. int main() {
6.    int x = 2, y = 3;
7.    int sum = compute_sum(x, y);
8.    cout << sum << endl;
9.    return 0;
10.}
```

Oregon State University
College of Engineering

How can we get compute_sum() to do all the work?

- Goal: `compute_sum()` updates `sum` variable in `main()`
- But `compute_sum()` can't see the `sum` variable in `main()`
- Bad solution: declare global variable `sum`
- Good solution: pass a reference to `sum`



Oregon State University
College of Engineering

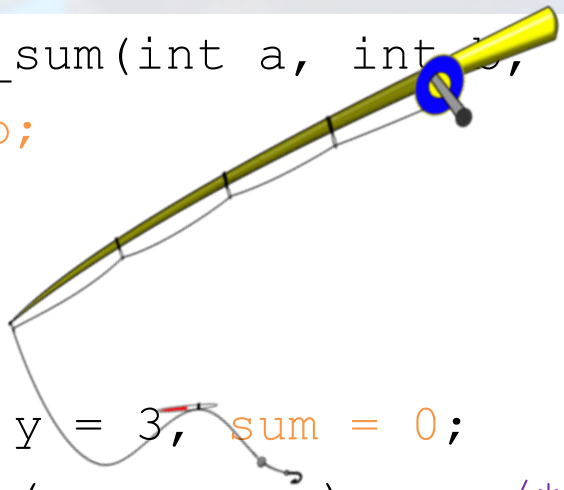
Variable values and references

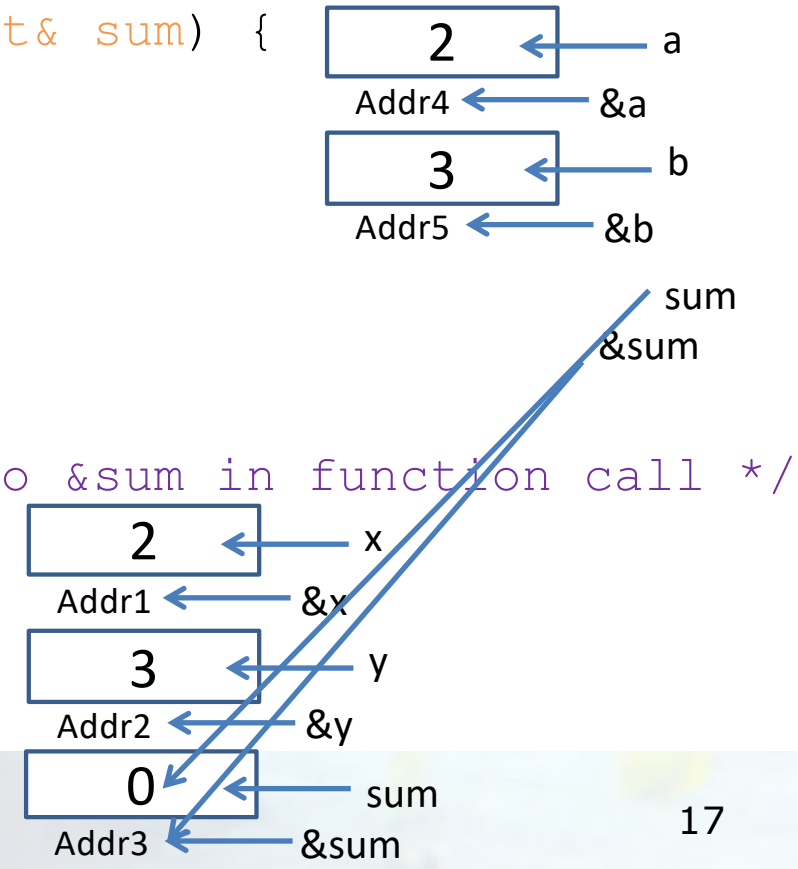
- Each variable has:
  - Value
  - Memory location (address)

- Example: `int x = 3;`
  - Variable value `x`: 3
  - Variable reference `&x` (address): 0x7ffee5799b10

Pass by reference

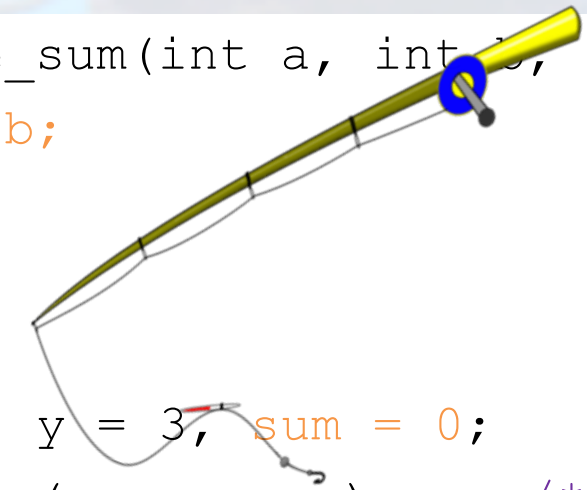
```
1. void compute_sum(int a, int b, int& sum) {
2.   sum = a + b;
3. }

4. int main() {
5.   int x = 2, y = 3, sum = 0;
6.   compute_sum(x, y, sum);      /* no &sum in function call */
7.   cout << sum << endl;
8.   return 0;
9. }
```

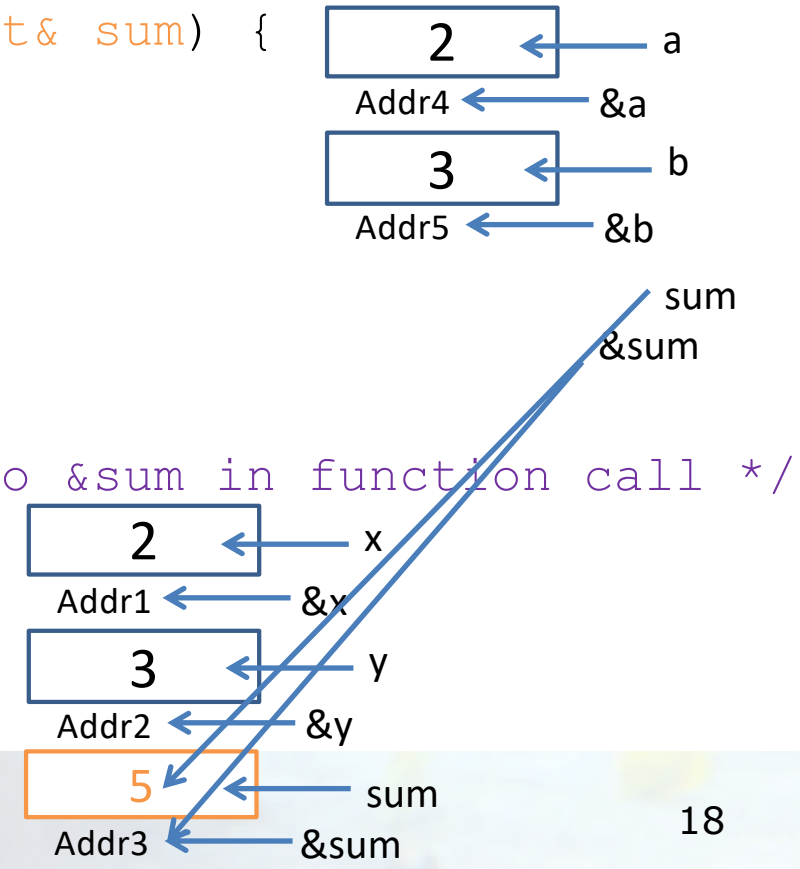




Pass by reference

```
1. void compute_sum(int a, int b, int& sum) {
2.   sum = a + b;
3. }

4. int main() {
5.   int x = 2, y = 3, sum = 0;
6.   compute_sum(x, y, sum);      /* no &sum in function call */
7.   cout << sum << endl;
8.   return 0;
9. }
```





Pass by reference: note variable names

```
1. void compute_sum(int a, int b, int& s) {
2.   s = a + b;
3. }

4. int main() {
5.   int x = 2, y = 3, sum = 0;
6.   compute_sum(x, y, sum);     /* no &sum in function call */
7.   cout << sum << endl;
8.   return 0;
9. }
```

Names do not have to match; they are in different scopes

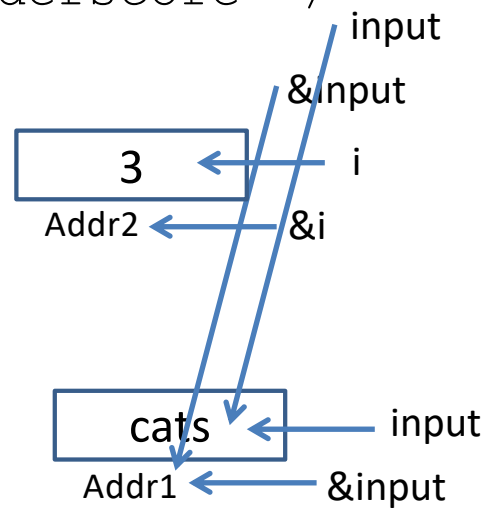
Passing arguments to functions

- `int v = 3;`
- `void fn(int w);`
- `void fn2(int& w);`

- Pass by value: make a copy
  - `fn(v);`
- Pass by reference: pass the address of the variable
  - `fn2(&v);`

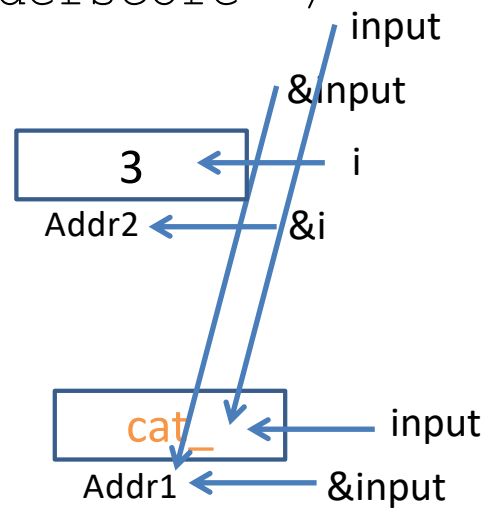
Pass strings by reference

```
1. void destroy_character(string& s, int i) {
2.   /* Change character i to an underscore */
3.   s[i] = '_';
4. }

5. int main() {
6.   string input;  cin >> input;
7.   destroy_character(input, 3);
8.   cout << input<< endl;
9.   return 0;
10.}
```



Pass strings by reference

```
1. void destroy_character(string& s, int i) {
2.   /* Change character i to an underscore */
3.   s[i] = '_';
4. }

5. int main() {
6.   string input;  cin >> input;
7.   destroy_character(input, 3);
8.   cout << input<< endl;
9.   return 0;
10.}
```



What vocabulary did we learn today?

- Function overloading
- Pass by value
- Pass by reference

Oregon State University
College of Engineering

What ideas and skills did we learn today?

- When function overloading is ambiguous
- How to make persistent changes to variables that are not in **function** scope (pass function arguments **by reference**)

Oregon State University
College of Engineering

Week 5 continues

- ❑ Attend lab (laptop required)
- ❑ Read **Rao Lesson 7** (pp. 166-167) – functions
  Read **Rao Lesson 8** (pp. 205-210) – references
- ❑ Continue working on **Assignment 3** (due **Sunday, Feb. 9**)
- ❑ Study session – **Thursday, Feb. 6, 6-7 p.m.**
  - ❑ Bring printed worksheet and writing utensil

See you Friday!