CSCI 200: Foundational Programming Concepts & Design Lecture 07

FUNctions

Complete Set1 Feedback in Canvas

Access Code: roshambo

Previously in CSCI 200

- Debugging
  - Enable compiler warnings

    ```
    g++ -Wall -Wextra -Werror
    ```

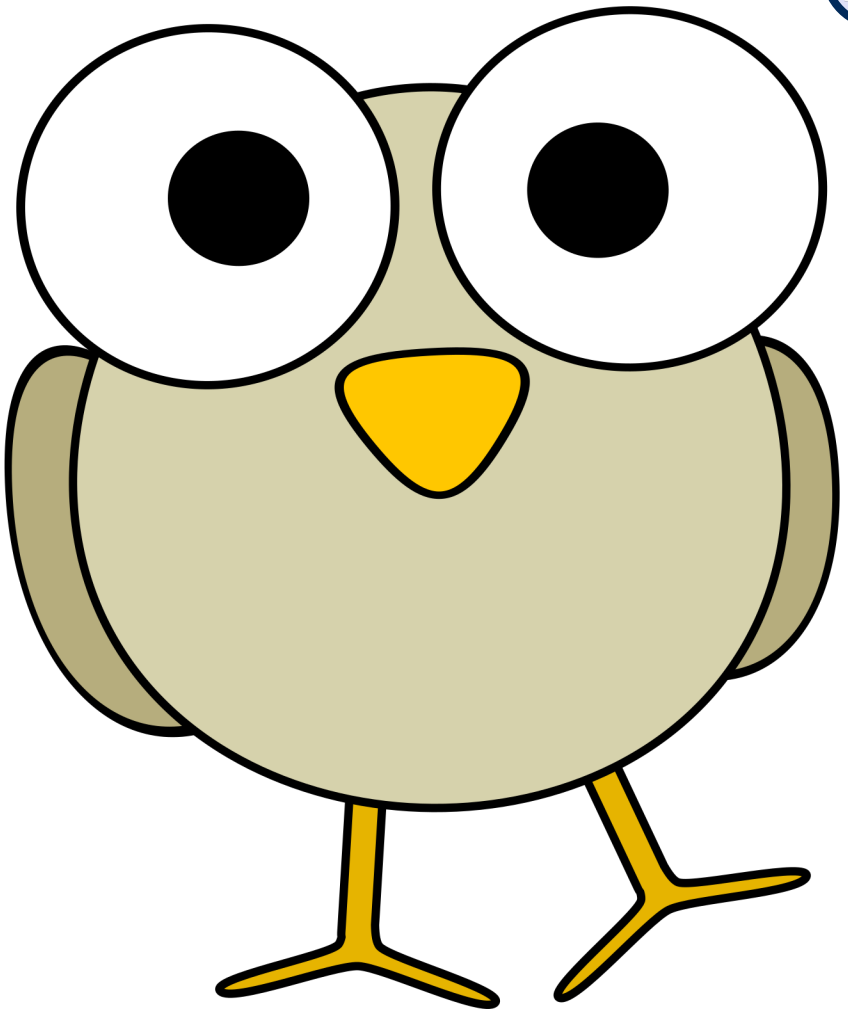
  - Print Lines

    ```
    cout << "here" << endl;
    ```

  - gdb / lldb – Debugger
    - Enable debugging

      ```
      g++ -g
      ```

Questions?



Learning Outcomes For Today

- Identify the parts of a function.
- Explain the difference between a parameter and an argument for a function. Discuss what can be returned from a function and what a void function is.
- Explain the meaning of the DRY principle and appropriate uses for functions.
- **S**OLID: Discuss how functions contribute towards the Principle of Single Responsibility.

On Tap For Today

- Functions
- Parameters & Arguments
- Pass-By-Value
- Practice

On Tap For Today

- Functions
- Parameters & Arguments
- Pass-By-Value
- Practice

Functions in Math

- $f(x) = 2x + 1$
- $\cos(90) = 0$
- What is the name of each?
- What is the input of each?
- What is the output of each?

Functions are Abstractions

- Complex operations are in a “black box”
- Are reusable
  - Don’t Repeat Yourself (DRY): put complex operations in ONE place and reference multiple times
    - `WORM Principle` : Write Once Read Many
      (aka Write Once Use Many)
  - Single Responsibility Principle: Function handles one major task instead of many minor tasks
    - **S**`OLID Principles`
  - Modular:
    - Can be moved from project to project
    - And/Or Can be replaced by a different implementation with matching interface
- Simplifies our code

Functions we've already used

- pow(), sqrt(), abs(), rand(), srand(), time()
- Library Functions
  - You didn't write them
  - Provided for you!

- main()
  - User-defined Functions
  - Functions you create (declare & implement)

int main()

```
int main() {
    // do stuff
    return 0;
}
```

- main() is a function that takes nothing as input and returns an integer as output

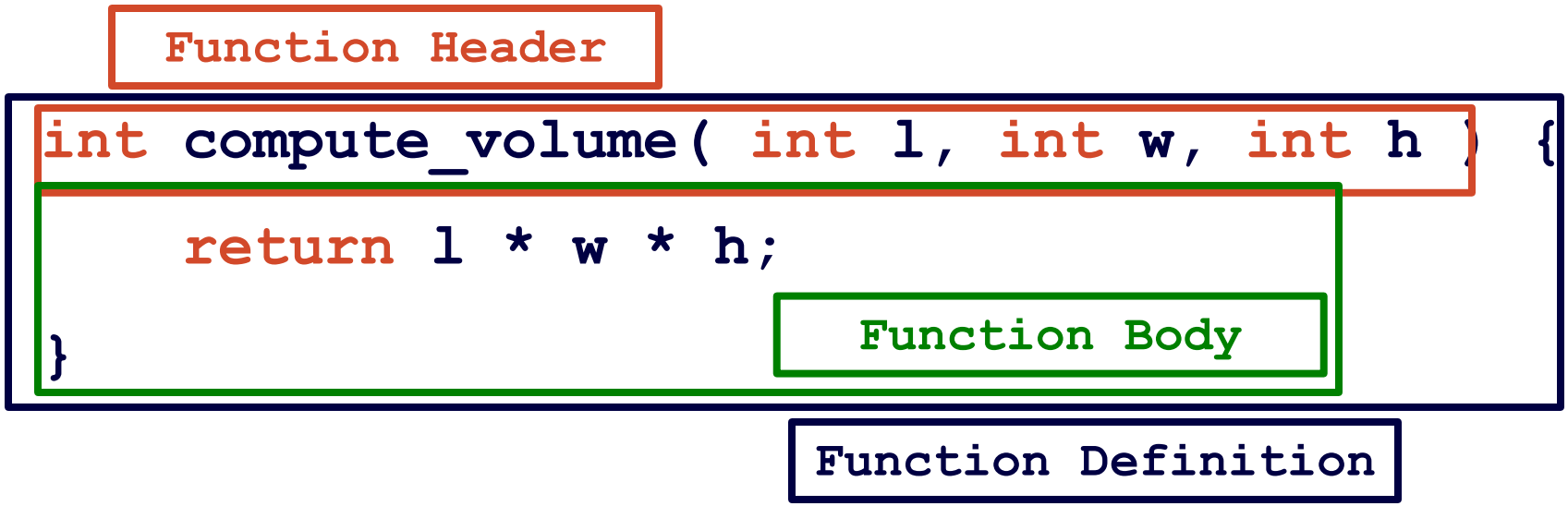
Example: volume

```
#include <iostream>
using namespace std;

int main() {
    int length(20), width(11), height(9), boxVolume;

    boxVolume = length * width * height;
    cout << "The volume is " << boxVolume << endl;

    length = length + 5;
    boxVolume = length * width * height;
    cout << "The new volume is " << boxVolume << endl;

    return 0;
}
```

Example: volume

```
#include <iostream>
using namespace std;

int compute_volume( int l, int w, int h ) {
    return l * w * h;
}
int main() {
    int length(20), width(11), height(9), boxVolume;
    boxVolume = compute_volume( length, width, height );
    cout << "The volume is " << boxVolume << endl;
    length = length + 5;
    boxVolume = compute_volume( length, width, height );
    cout << "The new volume is " << boxVolume << endl;
    return 0;
}
```

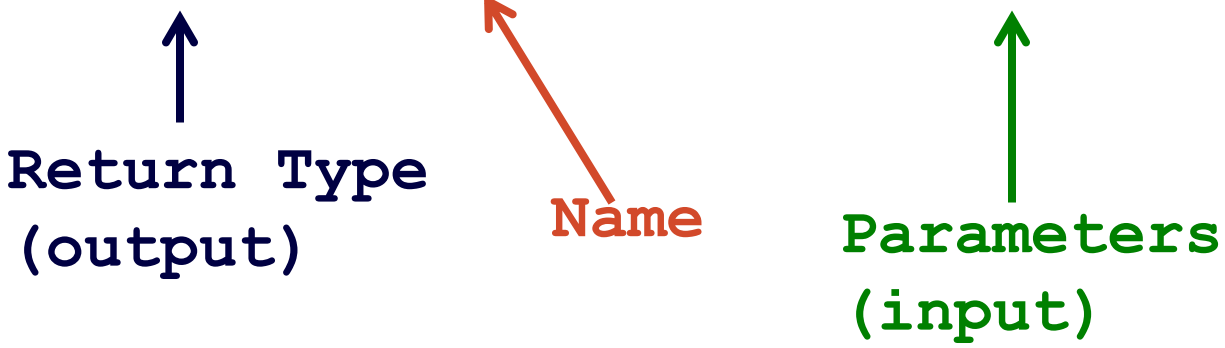
Anatomy of a Function



Function Header

- compute_volume() is a function that accepts three ints as input and returns an int as output

```
int compute_volume( int l, int w, int h )
```



Three Ways To Declare Functions

1. Above main()
   - (Today)
2. Declare prototype, then definition
3. Use an external file

Example: volume

```
#include <iostream>
using namespace std;

int compute_volume( int l, int w, int h ) {
    return l * w * h;
}

int main() {
    int length(20), width(11), height(9);

    cout << "The volume is "
         << compute_volume( length, width, height ) << endl;

    return 0;
}
```

Function Rules

```
#include <iostream>
using namespace std;

int compute_volume( int l, int w, int h ) {
    int v;
    v = l * w * h;
    return v;
}

int main() {
    int length(20), width(11), height(9);
    int boxVolume = compute_volume( length, width, height );
    cout << "The volume is " << boxVolume << endl;
    return 0;
}
```

Return Type matches return value

Function input (arguments) match parameters

void Functions

- If no output is needed from the function, use special data type void
- return statement in function is optional

```
void print_smiley() {
    cout << ":)" << endl;
    return;
}
```

Function Documentation

```
/**
 * @brief adds two values together
 * @param x left hand value
 * @param y right hand value
 * @return sum of x and y
 */
int add( int x, int y ) {
    return x + y;
}
```

On Tap For Today

- Functions
- Parameters & Arguments
- Pass-By-Value
- Practice

Parameters & Arguments

- Function add() takes two int parameters x and y

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

Parameters & Arguments

- "I call the add() function passing it two int arguments a and b"

```
int add( int x, int y ) {
    return x + y;
}
int main() {
  int a(2), b(3);
  cout << "2+3 is " << add(a, b) << endl;
  return 0;
}
```

Parameters & Arguments

- A function specifies parameters for input
- A function call passes arguments as input

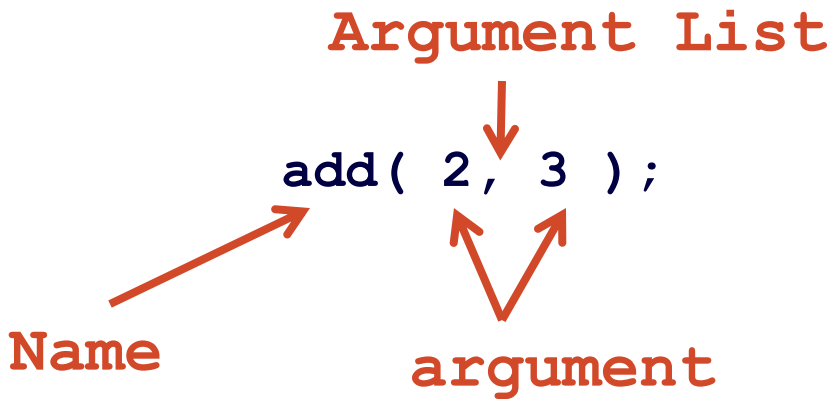
```
int add( int x, int y ) {
    return x + y;
}
int main() {
  int a(2), b(3);
  cout << "2+3 is " << add(a, b) << endl;
  return 0;
}
```

On Tap For Today

- Functions
- Parameters & Arguments
- Pass-By-Value
- Practice

Calling a function

- Call / Use / Invoke a function



- "We call add, passing it two integer arguments"
  - We know because of documentation or the function header

Pass By Value

- Primitive arguments are passed by value

```
int add( int x, int y ) {
    return x + y;
}
int main() {
  int a(2), b(3);
  cout << "2+3 is " << add(a, b) << endl;
  return 0;
}
```

- When evaluating the function, x & y have a value of 2 & 3 respectively

Precedence Table

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

Will this compile?

```
void fake_function( int x, y ) {
    // it does something
    return;
}
int main() {
  int a, b;
  fake_function( a, b );
  return 0;
}
```

Will this compile?

```
void fake_function( int x, int y ) {
    // it does something
    return;
}
int main() {
  int a(4), b(7);
  fake_function( a, b );
  return 0;
}
```

Will this compile?

```
int fake_function( int x, int y ) {
    // it does something
    return x, y;
}
int main() {
  int a(4), b(7);
  fake_function( a, b );
  return 0;
}
```

Will this compile?

```
int fake_function( int x, int y ) {
    // it does something
    return x;, y;
}
int main() {
  int a(4), b(7);
  fake_function( a, b );
  return 0;
}
```

On Tap For Today

- Functions
- Parameters & Arguments
- Pass-By-Value
- Practice

To Do for Next Time

- Work on L2A

Structured Programming Quiz

- Make Canvas Full Screen
  - Put everything else away
- Access Code:
- 12 Minutes