CSCI 200: Foundational Programming Concepts & Design Lecture 07

FUNctions

Complete Set1 Feedback in Canvas

Access Code: `reno`

Previously in CSCI 200

- Debugging
  - Enable compiler warnings (Required)

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

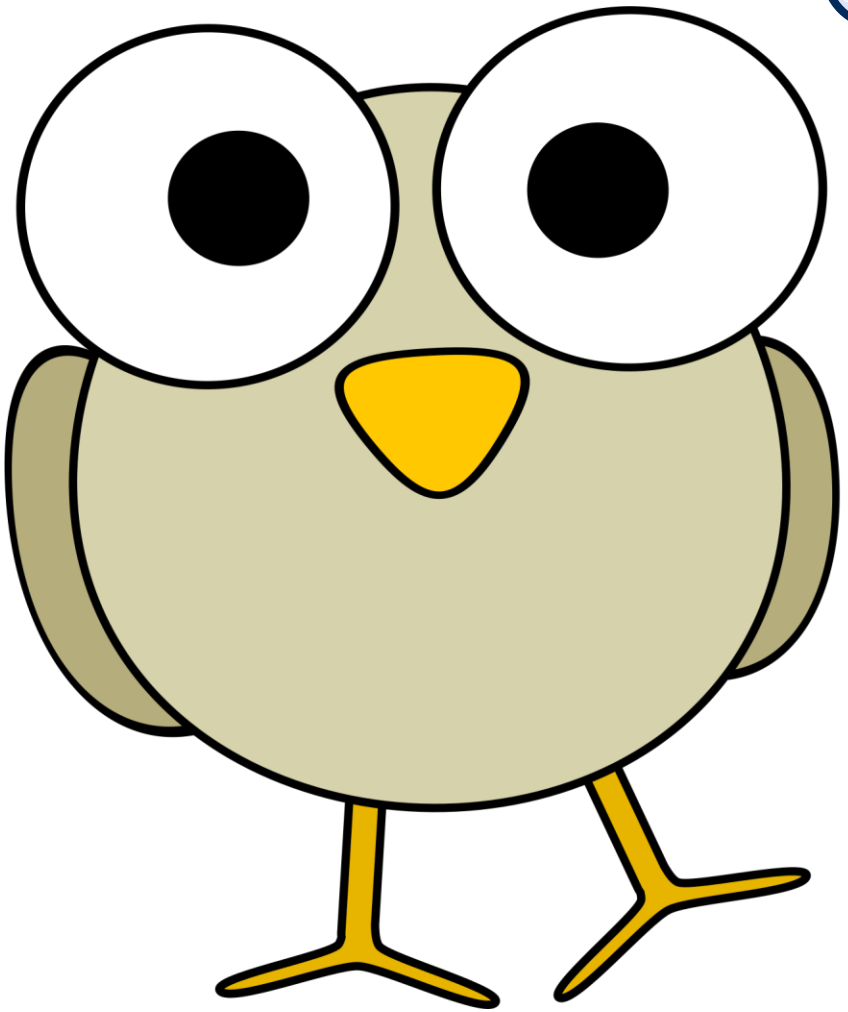
  - Print Lines

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

  - **`gdb`** / **`lldb`** - Debugger (Preferred)
    - 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
- Pass-By-Value
- Practice

On Tap For Today

- Functions
- Pass-By-Value
- Practice

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
    - Can be replaced by a different implementation with matching interface
- Simplifies our code

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), 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;
}
```

Function Header

- **`compute_volume()`** is a function that accepts three integers as input and returns an integer as output

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

Return Type (output)

Name

Parameters (input)

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;
}
```

- Tip: Install the Doxygen Documentation Generator extension in VS Code

Function Documentation

```
bool draw(
  GLuint shaderProgramHandle,
  GLint matDiffLocation,
  GLint matSpecLocation,
  GLint matShinLocation,
  GLint matAmbLocation,
  GLenum diffuseTexture
);
```

Function Documentation

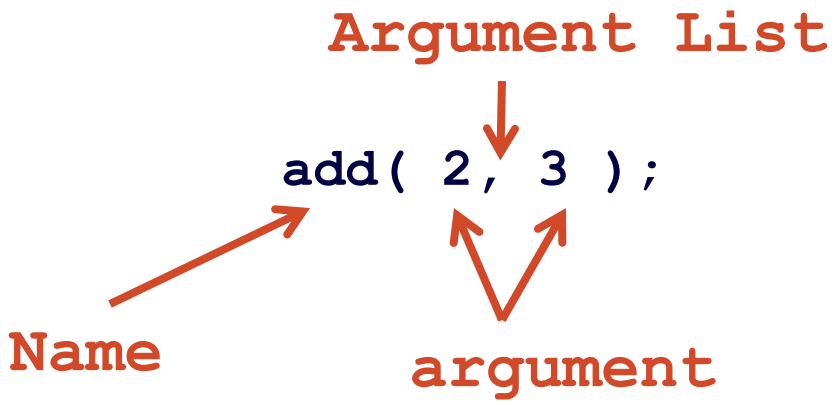
```
/**
 * @brief Renders a model against the provided shader program
 * @param shaderProgramHandle shader program handle to use
 * @param matDiffLocation uniform location of material diffuse component
 * @param matSpecLocation uniform location of material specular component
 * @param matShinLocation uniform location of material shininess component
 * @param matAmbLocation uniform location of material ambient component
 * @param diffuseTexture texture number to bind diffuse texture map to
 * @return true if draw succeeded, false otherwise
 */
bool draw(
  GLuint shaderProgramHandle,
  GLint matDiffLocation,
  GLint matSpecLocation,
  GLint matShinLocation,
  GLint matAmbLocation,
  GLenum diffuseTexture
);
```

On Tap For Today

- Functions
- 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 << "a+b is " << add(a, b) << endl;
  return 0;
}
```

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

Pass By Value

- Literal values are passed by value

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

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

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	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

Will this compile?

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

The function body is valid. Returning from a void function with just return; is allowed.

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
- 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