CSCI 200: Foundational Programming Concepts & Design Lecture 08

Functions in Separate Files

Download Lecture08 Starter Code

Previously in CSCI 200

- Function definition: header and body

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

- Function header: return type, name, parameters
- Match type for return AND params/args
- Need ( ) even if zero parameters
  - **( )** is the function call operator

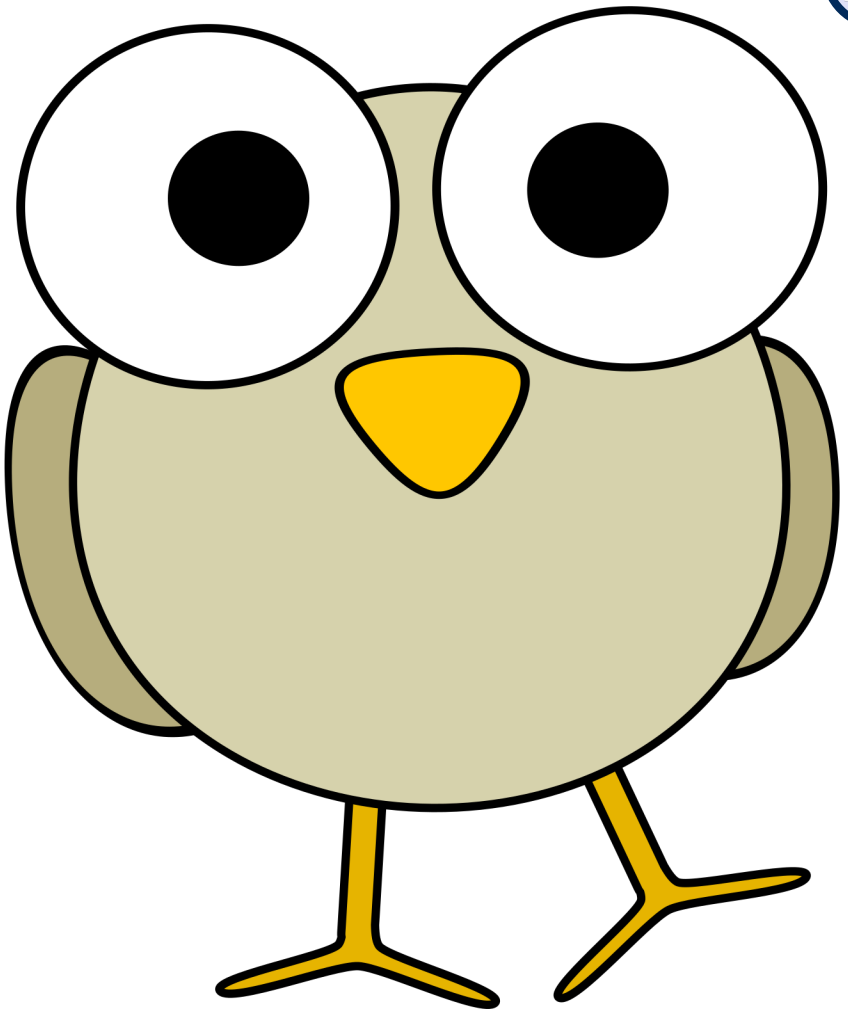
Use of ( )

```
( id )              // group order of operations

id()                // call function

(type)id            // type cast value
```

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;
}
```

Questions?



Learning Outcomes For Today

- Explain the difference between a function prototype and a function implementation. Discuss the pros/cons of separate implementations.
- Describe how a computer generates a program from code.
- Write and use a Makefile.
- Discuss the advantages of using Makefiles.

On Tap For Today

- Function Abstraction/Reuse/Modularity
- Practice

On Tap For Today

- Function Abstraction/Reuse/Modularity
- Practice

Three Ways to Declare a Function

1. Above main()
2. Declare prototype, then definition
3. Separate files

Three Ways to Declare a Function

1. Above main()
2. Declare prototype, then definition
3. Separate files

Declare Above main()

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

/**
 * @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;
}

int main() {
    cout << add( 5, 8 ) << endl;
    return 0;
}
```

What happens now?

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

int main() {
    cout << add( 5, 8 ) << endl;
    return 0;
}

/**
 * @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;
}
```

Compiler Error

add(int, int) undeclared

Three Ways to Declare a Function

1. Above main()
2. Declare prototype, then definition
3. Separate files

Function Prototype

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

/**
 * @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 );

int main() {
    cout << add( 5, 8 ) << endl;
    return 0;
}

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

Function Prototype

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

/**
 * @brief adds two values together
 * @param left hand value
 * @param right hand value
 * @return sum of left and right
 */
int add( int, int );

int main() {
    cout << add( 5, 8 ) << endl;
    return 0;
}

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

Function Prototype

- Consider

```
void drawRectangle( double, double );
```

- Which parameter is length?

What happens now?

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

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

int main() {
    cout << add( 5, 8 ) << endl;
    return 0;
}

// no function definition!
```

Linker Error

Undefined symbol add(int, int)

Three Ways to Declare a Function

1. Above main()
2. Declare prototype, then definition
3. Separate files

Separate Files

- Declare function prototypes in one file
- Define function implementations in another

- Include declaration file in any programs requiring functions

Preprocessor Directives

- Can include previously written files
  - Library Files

```
#include <iostream>
```

  - Files we've written

```
#include "MyFunctions.h"
```

Header & Implementation Files

- Header File (*.h) declares function prototypes
- Implementation File (*.cpp) defines corresponding function implementations

Add Source Files to Makefile

- SRC_FILES contains list of all cpp files – all function implementations
- OBJECTS contains list of all object files to compile
- Link all OBJECTS into executable

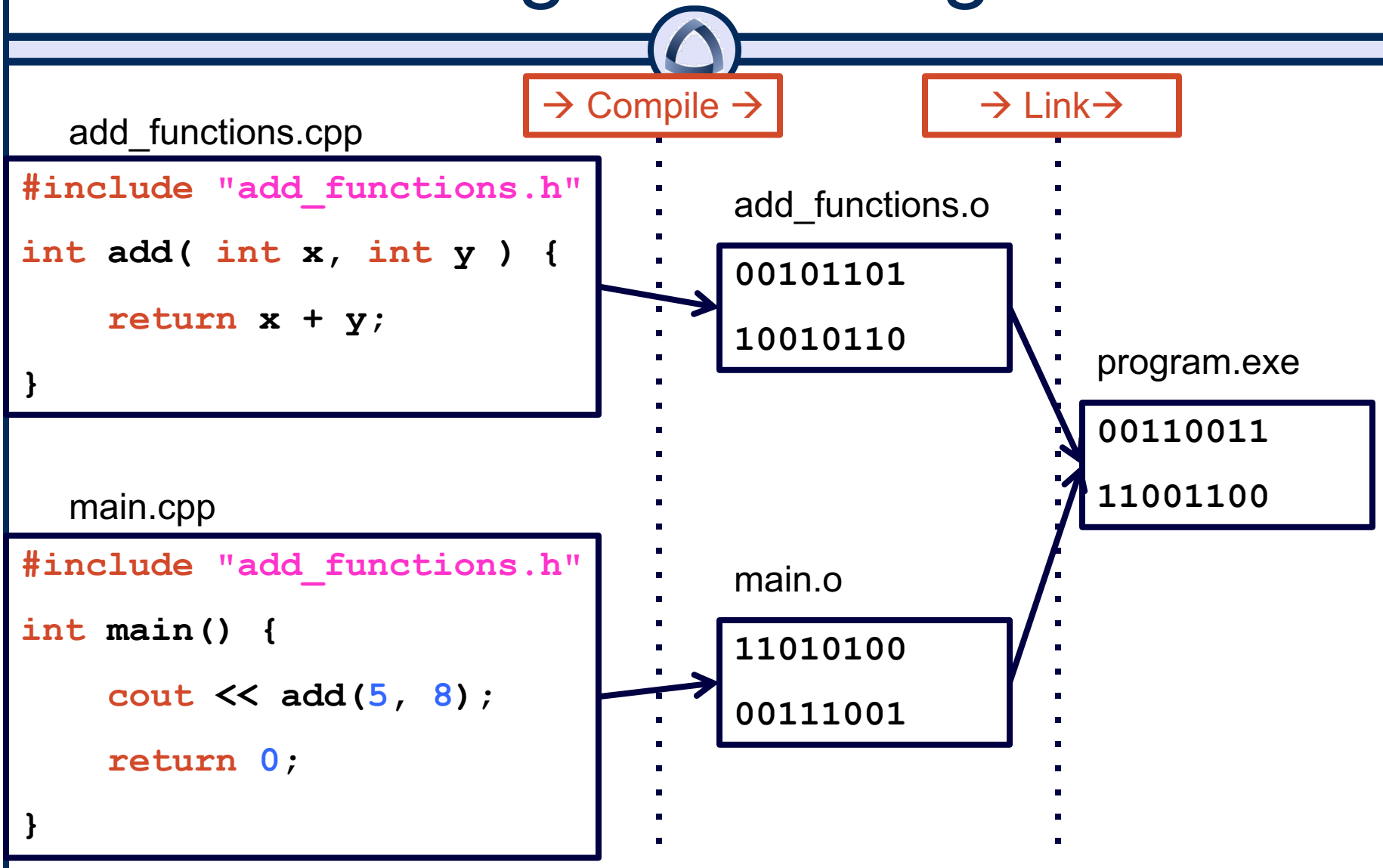
Multiple Files

```cpp
// main.cpp
#include <iostream>
using namespace std;

#include "add_functions.h"

int main() {
    cout << add(5, 8) << endl;
    return 0;
}
```

```cpp
// add_functions.h
#ifndef ADD_FUNCTIONS_H
#define ADD_FUNCTIONS_H
int add( int x, int y );
#endif //ADD_FUNCTIONS_H
```

```cpp
// add_functions.cpp
#include "add_functions.h"

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

Building Your Program



What Happens During

- Compiling?
- Linking?

Development Cycle

- Whenever a source file changes, it only needs to be recompiled

- Let's take a look!
  - And see what errors can occur

On Tap For Today

- Function Abstraction/Reuse/Modularity
- Practice

To Do For Next Time

- Watch video on Pointers before next class
- Continue with zyBooks
  - Read up on pointers!