ARLAB

ME/CprE/ComS 557

Computer Graphics and Geometric Modeling

Introduction to Programming

September 1st, 2015

Rafael Radkowski

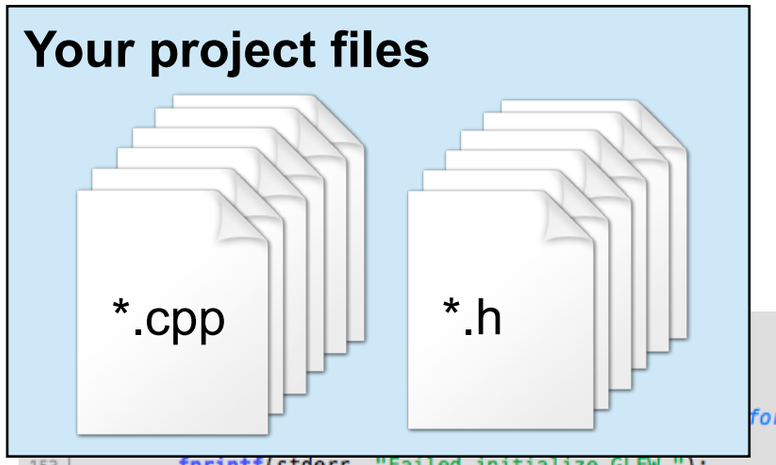
VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Content

- C++ Introduction
- Datatypes
- Operations
- Functions
- Classes
- #include / preprocessor

Software Development

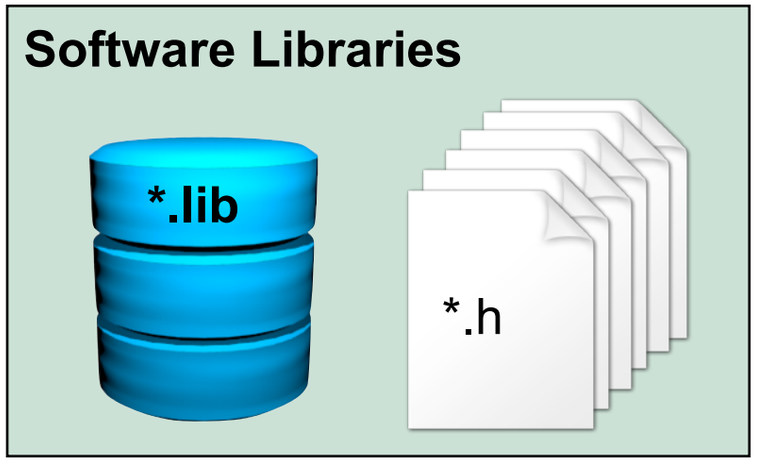


with C/C++

- .h - header files: keep all the declarations
- .cpp - implementation files: keep all the definitions

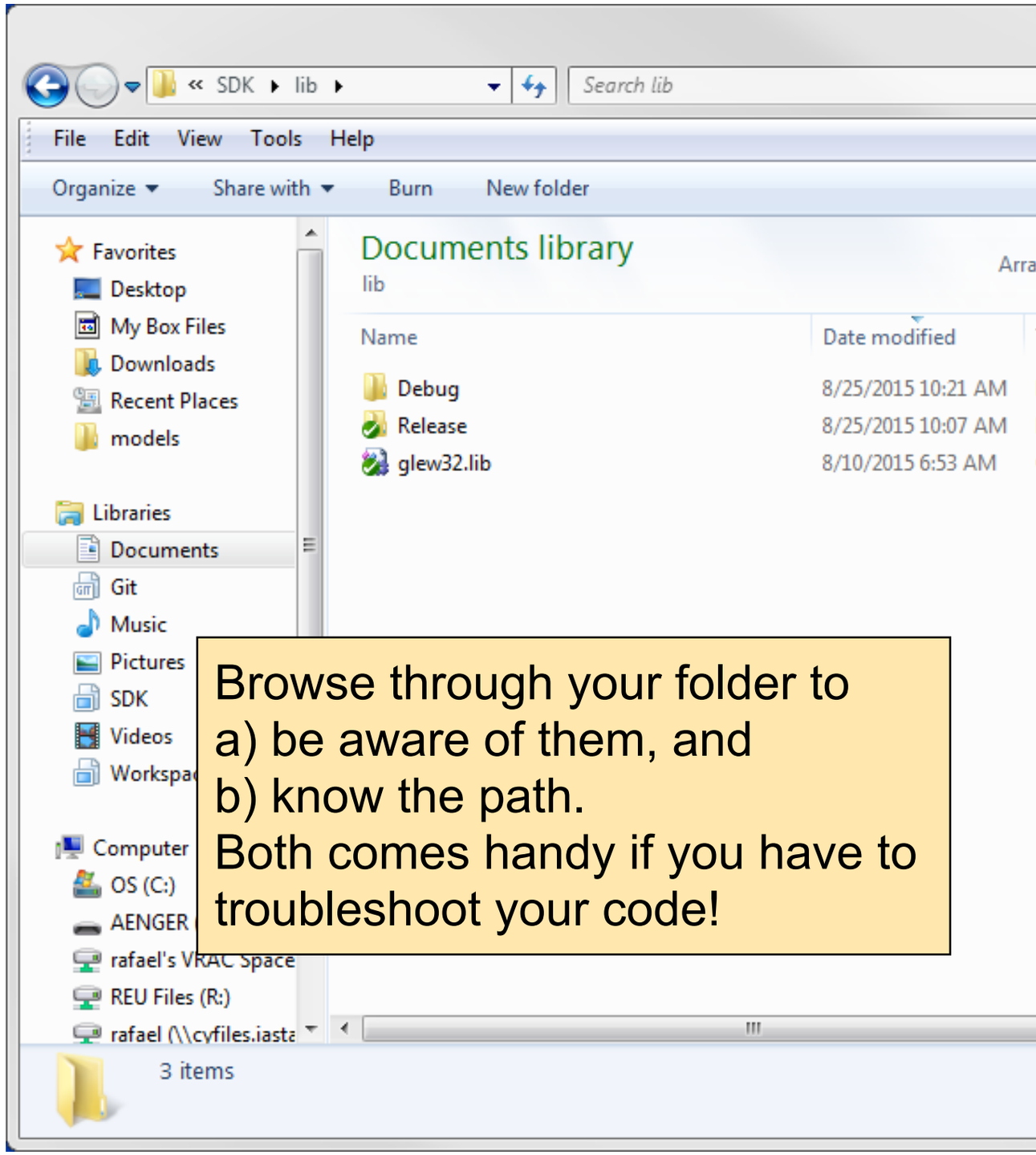
```
                                                                    for any reason, print it out to STDERR.
153         fprintf(stderr, "Failed initialize GLFW.");
154         exit(EXIT_FAILURE);
155     }
156
157     // Set the error callback, as mentioned above.
158     glfwSetErrorCallback(error_callback);
159
160     // Set up OpenGL options.
161     // Use OpenGL verion 4.1,
162     glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
163     glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 1);
164     // GLFW_OPENGL_FORWARD_COMPAT specifies whether the OpenGL context should be forward-compatible, i.e. one where all functionality
165     glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
166     // Indicate we only want the newest core profile, rather than using backwards compatible and deprecated features.
167     glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
168     // Make the window resize-able.
169     glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
170
171     // Create a window to put our stuff in.
172     GLFWwindow* window = glfwCreateWindow(800, 600, "Hello OpenGL", NULL, NULL);
173
174     // If the window fails to be created, print out the error, clean up GLFW and exit the program.
175     if(!window) {
176         fprintf(stderr, "Failed to create GLFW window.");
177         glfwTerminate();
178         exit(EXIT_FAILURE);
179     }
180
181     // Use the window as the current context (everything that's drawn will be place in this window).
182     glfwMakeContextCurrent(window);
183
184     // Set the keyboard callback so that when we press ESC, it knows what to do.
185     glfwSetKeyCallback(window, key_callback);
186
187     printf("OpenGL version supported by this platform (%s): \n", glGetString(GL_VERSION));
188
189     // Makes sure all extensions will be exposed in GLEW and initialize GLEW.
190     glewExperimental = GL_TRUE;
191     glewInit();
192
193     // Shaders is the next part of our program. Notice that we use version 410 core. This has to match our version of OpenGL we are u
194
195     // Vertex shader source code. This draws the vertices in our window. We have 3 vertices since we're drawing an triangle.
196     // Each vertex is represented by a vector of size 4 (x, y, z, w) coordinates.
```

OpenGL Files (GLEW, GLM, GLFW)

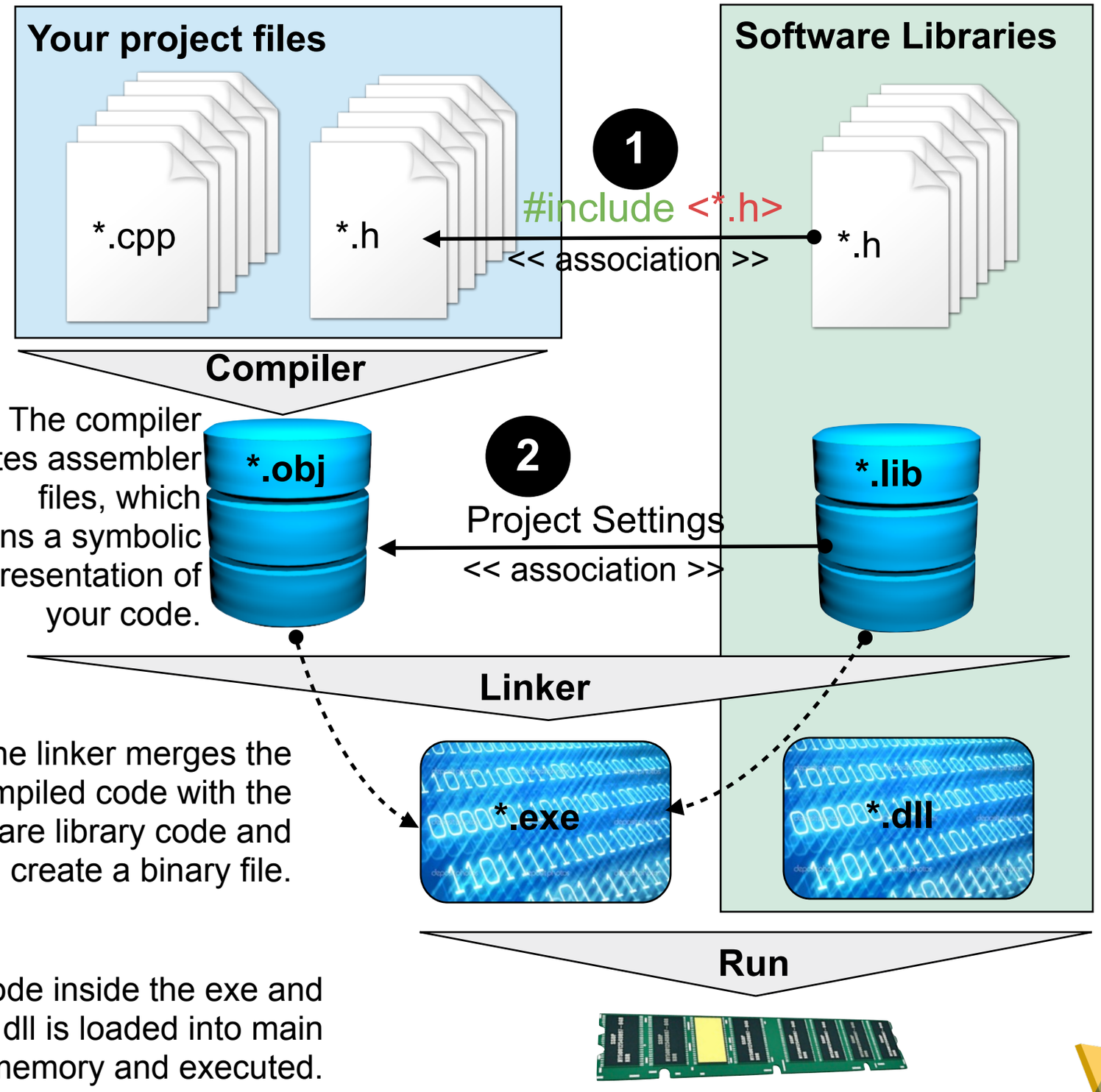


- GLEW, GLM, and GLFW provide header files with all file declaration
- GLEW library file: glew32.lib (64bit version)
  The library files provides declarations in machine code
- GLFW library file: glfw.lib
- GLM is a "header only" programming packages.

- In addition: GLEW provides a glew32.dll which keeps the code definition in machine code.



C/C++ Compiler



Every software project consists of two set of code: your own code and code from software libraries.

Your project code incorporates a set of cpp-files and header files.

The software library incorporates a set of header files, a library (multiple library files), and a binary file (dll), which contains the executables.

C/C++ code is generated in two steps.

First, a compiler compiles your project files and generates object files (obj). The contain assembler code. During this step, your code needs to know all the libraries and the provided function. This association is established using the #include command in your header files. The obj files contain a symbolic link to each library function.

Secondly, the Linker merges the generated obj files to one binary file. During this process, the Linker searches the lib files for the binary code, related to the symbolic links.

The result is an executable file containing machine code.

During program start, the machine code from the exe and the dll are loaded into computer's main memory. Thus, the program runs.

C++

C++ is a high-level, general-purpose programming language.

C++ is standardized by the International Organization for Standardization (ISO). The current standard version (December 2014) is ISO/IEC 14882:2014 (also known as C++14).

History

Bjarne Stroustrup, a Danish computer scientist, began his work on C++'s predecessor "C with Classes" in 1979.

The first edition was was ready and released in 1985.

Philosophy

- Programmers should be able to program in their own style
- No implicit violations of the type system but allows explicit violations;
- User-created types need to have the same support and performance as built-in types.
- .....
- The programmer has a lot of freedom BUT must know what he or she does !!!

C++ Code

C++ code example

```
147
148
149 int main(int argc, const char * argv[])
150 {
151     // Initialize GLFW, and if it fails to initialize for any reason, print it out to STDERR.
152     if (!glfwInit()) {
153         fprintf(stderr, "Failed initialize GLFW.");
154         exit(EXIT_FAILURE);
155     }
156
157     // Set the error callback, as mentioned above.
158     glfwSetErrorCallback(error_callback);
159
160     // Set up OpenGL options.
161     // Use OpenGL verion 4.1,
162     glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
163     glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 1);
164     // GLFW_OPENGL_FORWARD_COMPAT specifies whether the OpenGL context should be forward-compatible, i.e. one where all functionality
165     glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
166     // Indicate we only want the newest core profile, rather than using backwards compatible and deprecated features.
167     glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
168     // Make the window resize-able.
169     glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
170
171     // Create a window to put our stuff in.
172     GLFWwindow* window = glfwCreateWindow(800, 600, "Hello OpenGL", NULL, NULL);
173
174     // If the window fails to be created, print out the error, clean up GLFW and exit the program.
175     if(!window) {
176         fprintf(stderr, "Failed to create GLFW window.");
177         glfwTerminate();
178         exit(EXIT_FAILURE);
179     }
180
181     // Use the window as the current context (everything that's drawn will be place in this window).
182     glfwMakeContextCurrent(window);
183
184     // Set the keyboard callback so that when we press ESC, it knows what to do.
185     glfwSetKeyCallback(window, key_callback);
186
187     printf("OpenGL version supported by this platform (%s): \n", glGetString(GL_VERSION));
188
189     // Makes sure all extensions will be exposed in GLEW and initialize GLEW.
190     glewExperimental = GL_TRUE;
191     glewInit();
192
193     // Shaders is the next part of our program. Notice that we use version 410 core. This has to match our version of OpenGL we are u
```

Structure of a Program

```
// my first program in C++
#include <iostream>

int main()
{
  std::cout << "Hello World!";
}
```

Every program starts at the entry point "main".

Every program can only have ONE main - entry point

Output on screen:

Hello World!

http://www.cplusplus.com/doc/tutorial/program_structure/

C++ Program

ARLAB

```
/ operating with variables

#include <iostream>
using namespace std;

int main ()
{
  // declaring variables:
  int a, b;
  int result;

  // process:
  a = 5;
  b = 2;
  a = a + 1;
  result = a - b;

  // print out the result:
  cout << result;

  // terminate the program:
  return 0;
}
```

Datatypes

VRAC | HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

C++ Program

```
/ operating with variables

#include <iostream>
using namespace std;

int main ()
{
  // declaring variables:
  int a, b;
  int result;

  // process:
  a = 5;
  b = 2;
  a = a + 1;
  result = a - b;

  // print out the result:
  cout << result;

  // terminate the program:
  return 0;
}
```

Datatypes (Declaration)

Datatypes:

The values of variables are stored somewhere in the memory of your computer. The program does not need to know where.

The program needs to know what kind of data will be stored and how much storage is required, and how the programmer wants to refer to this storage:

type *[name]*

- type: what data should be stored
- name: the name to link the storage with your program

You **must** define every variable before you use it.

Datatypes

ARLAB

Fundamental datatypes:

Basic datatype which are "implemented" as part of your compiler

```
int a, b;

double dA, dB;

bool bA, bB;

char cA, cB;

string strA, strB;
```

:is a STL (Standard Template Library) datatype

API datatypes

Datatypes, provided by a programming API such as OpenGL

```
GLuint a, b;

glm::mat4 a_matrix, b_matrix;
```

VRAC|HCI IOWA STATE UNIVERSITY OF SCIENCE AND TECHNOLOGY

Datatypes

Here is the complete list of fundamental types in C++:

Group	Type names*	Notes on size / precision
Character types	**char**	Exactly one byte in size. At least 8 bits.
	char16_t	Not smaller than char. At least 16 bits.
	char32_t	Not smaller than char16_t. At least 32 bits.
	wchar_t	Can represent the largest supported character set.
Integer types (signed)	**signed char**	Same size as char. At least 8 bits.
	signed **short** *int*	Not smaller than char. At least 16 bits.
	signed **int**	Not smaller than short. At least 16 bits.
	signed **long** *int*	Not smaller than int. At least 32 bits.
	signed **long long** *int*	Not smaller than long. At least 64 bits.
Integer types (unsigned)	**unsigned char**	(same size as their signed counterparts)
	unsigned short *int*	
	unsigned *int*	
	unsigned long *int*	
	unsigned long long *int*	
Floating-point types	**float**	
	double	Precision not less than float
	long double	Precision not less than double
Boolean type	**bool**	
Void type	**void**	no storage
Null pointer	**decltype(nullptr)**	

http://www.cplusplus.com/doc/tutorial/variables/

C++ Program

```
/ operating with variables

#include <iostream>
using namespace std;

int main ()
{
  // declaring variables:
  int a, b;
  int result;

  // process:
  a = 5;
  b = 2;
  a = a + 1;
  result = a - b;

  // print out the result:
  cout << result;

  // terminate the program:
  return 0;
}
```

Basic Operations (Definition)

- Every line is terminated with a semicolon ;
- Declared variables can be combined with basic operators

Operators

Assignment operator (=)

```
x = 5;
x = y;
x = y = z = 5
```

Arithmetic operators

operator	description
+	addition
-	subtraction
*	multiplication
/	division
%	modulo

```
y = 2 + x;
```

```
x = 11 % 3;
```

Increment and decrement (++, --)

```
x = 3;
y = ++x;
```

// x contains 4, y contains 4

```
x = 3;
y = x++;
```

// x contains 4, y contains 3

What is the difference?

http://www.cplusplus.com/doc/tutorial/operators/

Basic Input / Output

```
// my first program in C++
#include <iostream>

int main()
{
  int age = 9;
  int zipcode = 50011;
   std::cout << "Hello World!"
   std::cout << "I am " << age << "
             years old and my zipcode is " << zipcode << std::endl;
}
```

stream	description
cin	standard input stream
cout	standard output stream
cerr	standard error (output) stream
clog	standard logging (output) stream

The basic input / output APIs allow us to show values on display and to request input values from a keyboard.

http://www.cplusplus.com/doc/tutorial/basic_io/

Basic Input / Output

```
// my first program in C++
#include <iostream>

int main()
{
  int age = 9;
  int zipcode = 50011;
  std::cout << "Hello World!"
  std::cout << "I am " << age << "
            years old and my zipcode is " << zipcode << std::endl;
}
```

Include the file with the required definitions (you need to know it)

Use the namespace do call a function.

stream	description
cin	standard input stream
cout	standard output stream
cerr	standard error (output) stream
clog	standard logging (output) stream

http://www.cplusplus.com/doc/tutorial/basic_io/

What is different?

```
#include <iostream>

int main()
{
  int age = 9;
  int zipcode = 50011;
   std::cout << "Hello World!"
   std::cout << "I am " << age << "
          years old and my zipcode is " << zipcode << std::endl;
}
```

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

int main()
{
  int age = 9;
  int zipcode = 50011;
   cout << "Hello World!"
   cout << "I am " << age << "
          years old and my zipcode is " << zipcode << endl;
}
```

What is different?

ARLAB

```
#include <iostream>

int main()
{
   int age = 9;
   int zipcode = 50011;
    std::cout << "Hello World!"
    std::cout << "I am " << age << "
            years old and my zipcode is " << zipcode << std::endl;
}
```

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

int main()
{
   int age = 9;
   int zipcode = 50011;
    cout << "Hello World!"
    cout << "I am " << age << "
            years old and my zipcode is " << zipcode << endl;
}
```

namespace

VRAC|HCI IOWA STATE UNIVERSITY OF SCIENCE AND TECHNOLOGY

Statements and Control Flow

Please review them on http://www.cplusplus.com/doc/tutorial/control/

Selection statements: if and else

```
if (x == 100){
  cout << "x is 100";
}
else{
  cout << "x is not 100";
  }
```

Selection statements: switch

```
switch (x) {
  case 1:
    cout << "x is 1";
    break;
  case 2:
    cout << "x is 2";
    break;
  default:
    cout << "value of x unknown";
  }
```

Statements and Control Flow

Please review them on http://www.cplusplus.com/doc/tutorial/control/

The for loop

```
for (int n=10; n>0; n--) {
    cout << n << ", ";
}
```

The while loop

```
int n = 10;

while (n>0) {
    cout << n << ", ";
    --n;
}
```

C++ Functions

A function is a group of statements that is executed when it is called from some point of the program. It allows to structure programs in segments of code to perform individual tasks.

The following is its format:

type name (parameter1, parameter2, ...) { statements }

where:

- type is the data type specifier of the data returned by the function.
- name is the identifier by which it will be possible to call the function.
- parameters (as many as needed): Each parameter consists of a data type specifier followed by an identifier, like any regular variable declaration (for example: int x) and which acts within the function as a regular local variable. They allow to pass arguments to the function when it is called. The different parameters are separated by commas.
- statements is the function's body. It is a block of statements surrounded by braces { }.

C++ Function Example (1/2)

cpp file:

```
// function example
#include <iostream>
using namespace std;

int addition (int a, int b)
{
  int r;
  r=a+b;
  return (r);
}

int main ()
{
  int z;
  z = addition (5,3);
  cout << "The result is " << z;
  return 0;
}
```

Includes "copy & paste" the content of a header file

Function

All variables that are defined inside a function are only valid inside this function.

Returns the value of r.

Function call: the function must be defined before it is called in a cpp file.

C++ Function Example (2/2)

cpp file:

```
// function example
#include <iostream>
using namespace std;

void addition (int a, int b, int* r);

int main ()
{
  int z;
  addition (5,3, &z);
  cout << "The result is " << z;
  return 0;
}

int addition (int a, int b, int* r)
{
  (*r)=a+b;
}
```

Function prototype; can also be specified in a header file. The header file must be included.

A third variable is added for the return value: **call-by-reference**.

Function implementation. Using a prototype, the implementation can be located after the function call in the main function. If a header file for the prototype is used, it is recommended to move the implementation into a related cpp-file.

The and symbol returns the address of the variable:

```
int z;
&z;
(&r); // gives access to the data
```

C++ Class

A **class** is an expanded concept of a data structure: instead of holding only data, it can hold both data and functions.

Classes are generally declared using the keyword class, with the following format:

```
class class_name {
  access_specifier_1:
    member1;
  access_specifier_2:
    member2;
   function1();
   function2();
};
```

access_specifier:

- private members of a class are accessible only from within other members of the same class or from their friends.
- protected members are accessible from members of their same class and from their friends, but also from members of their derived classes.
- Finally, public members are accessible from anywhere where the object is visible.

http://www.cplusplus.com/doc/tutorial/classes/

C++ Class

```
// classes example
#include <iostream>
using namespace std;

class Rectangle
{
  private:
    int x, y;
  public:
   Rectangle();
   ~Rectangle();
    void set_values (int,int);
    int area () {return (x*y);}
};
```

Class definition

Specification of data

Constructor and destructor

Specification of functions

```
void Rectangle::set_values (int a, int b)
{
  x = a;
  y = b;
}
```

Class implementation

The class name must be added.

```
int main ()
{
  Rectangle* rect = new Rectangle;
  rect->set_values (3,4);
  cout << "area: " << rect->area();
  return 0;
}
```

main function

C++ Class

```
// classes example
#include <iostream>
using namespace std;

class Rectangle
{
  private:
    int x, y;
  public:
    void set_values (int,int);
    int area () {return (x*y);}
};
```

Class definition

Header file .h

```
void Rectangle::set_values (int a, int b)
{
  x = a;
  y = b;
}
```

Class implementation

.cpp file

```
int main ()
{
  Rectangle rect;
  rect.set_values (3,4);
  cout << "area: " << rect.area();
  return 0;
}
```

main cpp file

Objects and Pointers

An **object** is an instantiation of a class. In terms of variables, a class would be the type, and an object would be the variable.

Classes can be initialized as objects or as pointers that refer to objects.

```
int main ()
{
  Rectangle rect;
  rect.set_values (3,4);
  cout << "area: " << rect.area();
  return 0;
}
```

Object of a class

Access via dot operator

```
int main ()
{
  Rectangle* rect = new Rectangle();
  rect->set_values (3,4);
  cout << "area: " << rect->area();
  delete rect;
  return 0;
}
```

Pointer of a class; requires a *new operator* for initialization

Access via pointer operator

- Use the . (dot) operator to access functions and variables, if you work with objects.
- Use the -> (pointer) operator to access functions and variables, if you work with pointers.
- Every new requires a delete. Otherwise, you create memory leaks!

Objects and Pointers

An **object** is an instantiation of a class. In terms of variables, a class would be the type, and an object would be the variable.

Classes can be initialized as objects or as pointers that refer to objects.

```
int main ()
{
  Rectangle rect;
  rect.set_values (3,4);
  cout << "area: " << rect.area();
  return 0;
}
```

Object of a class

Access via dot operator

```
int main ()
{
  Rectangle* rect = new Rectangle();
  rect->set_values (3,4);
  cout << "area: " << rect->area();
  delete rect;
  return 0;
}
```

Pointer of a class; requires a *new operator* for initialization

Access via pointer operator

Pointers must be deleted after usage / end of program

- Use the . (dot) operator to access functions and variables, if you work with objects.
- Use the -> (pointer) operator to access functions and variables, if you work with pointers.
- Every new requires a delete. Otherwise, you create memory leaks!

Including API's

The preprocessor command #include is required to add the APIs, datatypes, function, and classes to our program

```
// stl include
#include <iostream>
#include <string>


// GLEW include
#include <GL/glew.h>

// GLM include files
#define GLM_FORCE_INLINE
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>

// glfw includes
#include <GLFW/glfw3.h>
```

Thank you!

Questions

Rafael Radkowski, Ph.D.
Iowa State University
Virtual Reality Applications Center
1620 Howe Hall
Ames, Iowa 50011, USA
+1 515.294.5580

rafael@iastate.edu
http://arlab.me.iastate.edu

www.linkedin.com/in/rradkowski

ARLAB

VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY