**CS301 – Bootcamp: Structs and Classes**

**Case study: pet adoption**

**Structure or “struct”:** C++ construct that allows multiple variables to be grouped together

**Structures** allow us to mix \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ data types and store them within the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ group. Then, we can treat this group as a single item.

Funnel chart

Description automatically generated with medium confidence

**Example: A structure for pet adoption**

struct PetData

{

string petName; //store pet's name

string species; //store species

double ageInYears; //store age in years

char genderMorF; //store gender as 'M' or 'F'

double weightInPounds; //store weight in pounds

int cageLocation; //store location in shelter

bool isHealthy; //true/false whether the pet is healthy

bool isAvailable; //true/false whether the pet is available

double adoptionFee; //adoption fee in dollars

};

**We have already used some classes before!**

The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ class, accessed with #include <string>

* Declares C++ string objects
* Has built-in functions we have used
  + myString.length()

The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ class, accessed with #include <fstream>

* Declare the input and output filestream objects

Ifstream inFile;

ofstream outFile;

* Has built-in functions – remember these from our assignments?
  + inFile.open(“file.txt”)
  + inFile.close()

**Data hiding: Why have private members?**

* Making data members private provides data \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
* This prevents data from being accidentally \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
* If you attempt to change a private member, your compiler will \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

**Example: driving a car: Does the driver really need to tell the car when to run the fuel pump?**

**Declaring a class**

class Rectangle

{

private:

double width; //declare width and length

double length;

public:

void setWidth(double newWidth); //declaration for functions

void setLength(double newLength);

double getWidth() const;

double getLength() const; //const prevents data

double getArea() const; //from being modified

};

**Public vs Private Access Specifiers**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_: can be accessed by functions outside of the class

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_: “Members only” - can only be called by or accessed by functions that are members of the class

Note: Can list in any order. If not specified, Default setting is private.

**Using const With Member Functions**

* const appearing after the parentheses in a member function declaration specifies that the function will not \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ in the calling object.

**Other rules about classes (see example on next page)**

1. Similar to struct, include your class declaration at the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ of your program, just below the headers. Or more commonly, use separate files (demo soon)
2. Variables are generally declared to be \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.
3. Objects are declared just like \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ variables:
4. You use the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_operator (.) to call functions and access members
5. When defining a member function:
   1. Put function \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ in class declaration at the beginning of the program
   2. Define function using class name and \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (::)
   3. To call a member function, use the function’s \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_and the dot (.) operator

The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ operator (:: ) is used to tell what a member function is a member of. It is used with a class name, not with a class variable.

**Example: Rectangle program and Rectangle class**

#include <iostream>

using namespace std;

//rectangle class declaration

class Rectangle

{

private:

double width;

double length;

public:

void setWidth(double);

void setLength(double);

double getWidth() const;

double getLength() const;

double getArea() const;

};

int main()

{

Rectangle box;

double rectWidth;

double rectLength;

//get the rectangle's width and length from the user.

cout << "This program will calculate the area of a\n"

<< "rectangle. What is the width? ";

cin >> rectWidth;

cout << "What is the length? ";

cin >> rectLength;

//Store the width and length of the rectangle

//in the box object

box.setWidth(rectWidth);

box.setLength(rectLength);

//Display the rectangle's data

cout << "Here is the rectangle's data: " << endl

<< "Width: " << box.getWidth() << endl

<< "Length: " << box.getLength() << endl

<< "Area: " << box.getArea() << endl;

return 0;

}

// assigns a value to the width member

void Rectangle::setWidth(double w)

{

width = w;

}

//set a value to length

void Rectangle::setLength(double len)

{

length = len;

}

double Rectangle::getWidth() const

{

return width;

}

double Rectangle::getLength() const

{

return length;

}

double Rectangle::getArea() const

{

return width \* length;

}

**Using multiple files with classes**

1. Make a \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ file (aka interface file). Name it classname.h (where classname is the name of the class it holds). Put the declaration of the class in this file.
2. Make an \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ file. Name it classname.cpp. The implementation must have the same name as the class but end in .cpp. Put the definitions for all the member functions and overloaded operators in this implementation file. Use #include “classname.h” to reference the header/interface file in this implementation file.
3. Make an \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ file. The application file should contain your main function and any additional elements of your program. Use #include “classname.h” to reference the header/interface file in this application file.

**Demo: Full Example: Building the Rectangle class**

Procedure we will follow in this demo:

1. Create new project in CLion
2. Create new class “Rectangle”
3. Add class header with public and private members
4. Update CMakeLists.txt so that compiler knows to combine three files when compiling
5. Use Generate Definitions to make empty function definitions
6. Fill in function definitions in Rectangle.cpp
7. Update main function and run

**Reminder: Let CLion help you!**

* From within CLion, you can make a new class. Enter the class name, and CLion will make the header and implementation files for you.
* Other code will be added to prevent your class from being declared multiple times (if not defined and #define)
* CLion will even help you add function definitions using a class header.
* If you want to compile and run all the files with a class in Clion, you need to update the CMakeLists.txt file and use add\_executable to run the header, implementation, and main function together.

**add\_executable(BankAccount BankAccount.h BankAccount.cpp mainBankAccount.cpp )**

**More buzzwords: Accessors and Mutators (“getters and setters”)**

* Mutator: Think mutate. A member function that stores a value in a private member variable, or changes its value in some way
* Accessor: Think access. A function that retrieves a value from a private member variable. Accessors do not change an object's data, so they should be marked const.

Remember, code outside the class can only access the private variables by using these functions!

**A few more things to watch out for: Common mistake I see in assignments using classes:**

* Make sure you remember to write all your function definitions in the implementation file.
* If you forget to write a function definition for a function in the implementation file, you will get compiler errors from CLion (“undefined reference”)

**How to declare and use structs**

Diagram

Description automatically generated

**How to use struct variables:**

Once you declare your struct, you can declare variables of structure type.

structName structVariableName;

Example: We can declare more variables of type PetData using:

PetData pet1;

PetData pet2;

**Using structures in programs**

Diagram

Description automatically generated with low confidence

**The Dot operator and Using data in structures**

Use the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_operator to refer to members of struct variables:

structVariableName.memberVariabletoget

Examples: pet1.name = “Fluffy”; //get name from pet1

**Example: Employee Pay**

//This program demonstrates use of structures

#include <iostream>

#include <string>

#include <iomanip>

using namespace std;

struct PayRoll

{

int empNumber;//employee ID

string name;//employee Name

double hours;//hours worked

double payRate;//hourly pay rate

double grossPay;//Gross pay

};

int main()

{

PayRoll employee; //declare PayRoll structure called employee

//get the employee ID number

cout << "Enter the employee's number: ";

cin >> employee.empNumber;

//get the employee's name

cout << "Enter the employee's name: ";

cin.ignore(); //ignore the '\n' at the end of the name

getline(cin, employee.name);

//get the hours worked

cout << "How many hours did the employee work? ";

cin >> employee.hours;

//Get the employee's pay rate

cout << "What is the employee's hourly pay rate? ";

cin >> employee.payRate;

//Calculate the employee's gross pay

employee.grossPay = employee.hours \* employee.payRate;

//Display the employee data

cout << "Here is the employee's payroll data: \n";

cout << "Name: " << employee.name << endl;

cout << "Hours worked: " << employee.hours << endl;

cout << "Pay Rate: $" << employee.payRate << " per hour." << endl;

cout << "Gross Pay: $" << employee.grossPay << endl;

return 0;

}

**The problem with structures**

**Two main limitations of structs:**

1. **structures are used to store data. They are not used to store functions.**

If we need to make changes, we have to modify BOTH structures AND the functions in our program!

Especially in massive programs with tons of data and functions, it can be a big problem to have data and functions kept separate.

\*In the C programming language, structs could only store data and could not store functions. In C++ a struct can contain both variables and functions. But conventionally, programmers generally use structs to store sets of data and classes when we need to store both functions and data.

**2. Everything is public within a struct (there is no data protection)**

Sometimes we don’t want all parts of a program to be able to access certain data all the time. Examples:

* Programmer in industry working on a program with a big team.
  + Do you want your teammates’ code to be able to modify your data without restriction?
* Programs storing important data
  + we want to have protections in place so we don’t accidentally modify data
  + ensure updates are made properly without corrupting other data or calculations.

**How to improve a struct**

A class is basically an improved struct with more capabilities:

* **Encapsulation:** we can put both data and functions together to make self-contained, modular programs.
* **Data hiding:** we can declare some items to be public and other items to be private to help protect data.

An object is an instance of a class, in the same way that a variable can be an instance of a struct

**We have already used some classes before!**

The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ class, accessed with #include <string>

* Declares C++ string objects
* Has built-in functions we have used
  + myString.length()

The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ class, accessed with #include <fstream>

* Declare the input and output filestream objects

Ifstream inFile;

ofstream outFile;

* Has built-in functions – remember these from our assignments?
  + inFile.open(“file.txt”)
  + inFile.close()

**Data hiding: Why have private members?**

* Making data members private provides data \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
* This prevents data from being accidentally \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
* If you attempt to change a private member, your compiler will \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

**Example: driving a car: Does the driver really need to tell the car when to run the fuel pump?**

**Declaring a class**

class Rectangle

{

private:

double width; //declare width and length

double length;

public:

void setWidth(double newWidth); //declaration for functions

void setLength(double newLength);

double getWidth() const;

double getLength() const; //const prevents data

double getArea() const; //from being modified

};

**Public vs Private Access Specifiers**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_: can be accessed by functions outside of the class

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_: “Members only” - can only be called by or accessed by functions that are members of the class

Note: Can list in any order. If not specified, Default setting is private.

**Using const With Member Functions**

* const appearing after the parentheses in a member function declaration specifies that the function will not \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ in the calling object.

**Example: Rectangle program and Rectangle class**

#include <iostream>

using namespace std;

//rectangle class declaration

class Rectangle

{

private:

double width;

double length;

public:

void setWidth(double);

void setLength(double);

double getWidth() const;

double getLength() const;

double getArea() const;

};

int main()

{

Rectangle box;

double rectWidth;

double rectLength;

//get the rectangle's width and length from the user.

cout << "This program will calculate the area of a\n"

<< "rectangle. What is the width? ";

cin >> rectWidth;

cout << "What is the length? ";

cin >> rectLength;

//Store the width and length of the rectangle

//in the box object

box.setWidth(rectWidth);

box.setLength(rectLength);

//Display the rectangle's data

cout << "Here is the rectangle's data: " << endl

<< "Width: " << box.getWidth() << endl

<< "Length: " << box.getLength() << endl

<< "Area: " << box.getArea() << endl;

return 0;

}

// assigns a value to the width member

void Rectangle::setWidth(double w)

{

width = w;

}

//set a value to length

void Rectangle::setLength(double len)

{

length = len;

}

double Rectangle::getWidth() const

{

return width;

}

double Rectangle::getLength() const

{

return length;

}

double Rectangle::getArea() const

{

return width \* length;

}

**Using multiple files with classes**

1. Make a \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ file (aka interface file). Name it classname.h (where classname is the name of the class it holds). Put the declaration of the class in this file.
2. Make an \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ file. Name it classname.cpp. The implementation must have the same name as the class but end in .cpp. Put the definitions for all the member functions and overloaded operators in this implementation file. Use #include “classname.h” to reference the header/interface file in this implementation file.
3. Make an \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ file. The application file should contain your main function and any additional elements of your program. Use #include “classname.h” to reference the header/interface file in this application file.

**Demo: Full Example: Building the Rectangle class**

Procedure we will follow in this demo:

1. Create new project in CLion
2. Create new class “Rectangle”
3. Add class header with public and private members
4. Update CMakeLists.txt so that compiler knows to combine three files when compiling
5. Use Generate Definitions to make empty function definitions
6. Fill in function definitions in Rectangle.cpp
7. Update main function and run

**Reminder: Let CLion help you!**

* From within CLion, you can make a new class. Enter the class name, and CLion will make the header and implementation files for you.
* Other code will be added to prevent your class from being declared multiple times (if not defined and #define)
* CLion will even help you add function definitions using a class header.
* If you want to compile and run all the files with a class in Clion, you need to update the CMakeLists.txt file and use add\_executable to run the header, implementation, and main function together.

**add\_executable(BankAccount BankAccount.h BankAccount.cpp mainBankAccount.cpp )**

**More buzzwords: Accessors and Mutators (“getters and setters”)**

* Mutator: Think mutate. A member function that stores a value in a private member variable, or changes its value in some way
* Accessor: Think access. A function that retrieves a value from a private member variable. Accessors do not change an object's data, so they should be marked const.

Remember, code outside the class can only access the private variables by using these functions!

**What is a constructor?**

* A constructor is a member function that can be used to make more objects of that class (it “constructs” more objects of your class)
* Special rules for constructors:
  + Constructors are automatically called when an object of that class is declared
  + The constructor function’s name is the same name as the class name
  + Has no return type
  + Usually public
  + We can use overloading to declare different versions of a constructor to initialize objects using various input arguments.

**How do we use constructors?**

They are just member functions which initialize our member variables. We declare and define the constructor in a similar way to other member functions in our class.

1. Declare the constructor in our class
2. Define the constructor and specify how we want things to be initialized.
3. Call the constructor when we want to make and initialize more objects using our class.

**Example: constructors for Rectangle**

**1. Declaring a Constructor: Rectangle**

class Rectangle

{

private:

double width;

double length;

public:

Rectangle(double initialWidth, double initialLength);

void setWidth(double newWidth);

void setLength(double newLength);

double getWidth() const;

double getLength() const;

double getArea() const;

};

A constructor must have the same \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_as the class

A constructor definition cannot return a \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ or have any specified return type.

Be sure to include preconditions and postconditions for all function definitions in your assignments!

**2. Defining constructors**

Rectangle::Rectangle(double initialWidth, double initialLength)   
{  
 width = initialWidth;

length = initialLength;

}

Constructor definitions go in the same place as other member functions (in the implementation file, if multiple cpp files used)

Here, the rectangle constructor will initialize a rectangle object with the initial width and length passed in the argument.

**3. Calling a constructor**

Constructors are NOT called like a normal \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

box1.Rectangle(2, 3); ***//will not work***

The correct way to call a constructor is when declaring the object

Rectangle box1(2, 3);

//creates Rectangle object named box1 and

//initializes width to 2, and length to 3

**Overloading Constructors**

* Sometimes you need to create objects based on different sets of parameter values
* Constructors can be overloaded by defining constructors with different parameter lists

Rectangle(double initialWidth, double initialLength);

Rectangle();

* Remember, the compiler determines which constructor to use based on the number and type of arguments

**The default constructor**

* A default constructor uses no \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
* It used to create an object when no other input arguments are given. (It is the “default” version of a constructor)
* A default constructor for the Rectangle class could be declared in this way

class Rectangle

{

private:

double width;

double length;

public:

Rectangle();

//default constructor

//Precondition: none

//Postcondition: default constructor. Initializes length and width to zero.

//then add the rest of class declaration

};

**An undefined default constructor**

* If we do not define the default constructor, it will be a function with no parameters and an empty definition:

Rectangle::Rectangle()   
{  
 //empty

}

* **You should ALWAYS declare and define your default constructor** (at least ALWAYS in this class. You may find very rare exceptions later).

**Defining the default constructor**

* The default constructor for the Rectangle class could be defined as

Rectangle::Rectangle()   
{  
 width = 0;

length = 0;

}

* **You should ALWAYS declare and define your default constructor** (at least ALWAYS in this class. You may find very rare exceptions later).

**What should the default constructor do? What variables should it change?**

General rule of thumb: Most of the time, your default constructor should initialize *all private variables* to some initial value (often zero).

**Calling the default constructor**

* The default constructor is called during declaration of an object

Rectangle box1;   
 // uses the default constructor

//since no arguments given

* You can only have one default constructor for a class.