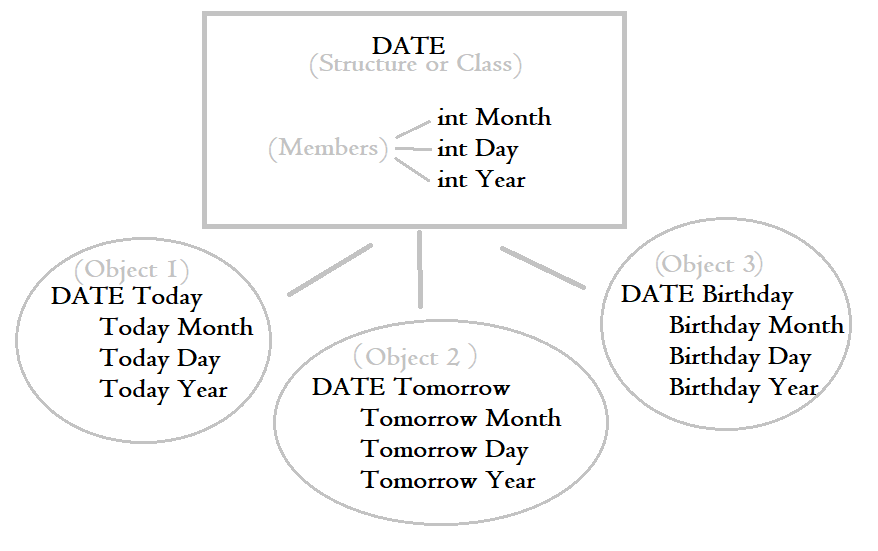
**Structures**

 A structure is a collection of variables (called members), grouped under a single, global name. The global name is general and relates to its members, such as the **Date** *(Structure Name)* would include a **Month**, **Day**, and **Year** *(Various Members)*. The members are given values when defined within a speific structure variable(s)/Object(s), such as **Today** *(Structure Variable)* having a specific **Date** *(Structure Name).*

Structures are used primarily for small, un-privated allocations of complex data storage.

**Structure Declaration Example 1 & 2:**

*Structure Tag* struct StudentRecord {

*Member Variable*  int studentNumber;

*Member Variable* char grade;

}; *// <- Don’t forget end ;*

int main (){

*Structure Variable* StudentRecord yourRecord;  
 *//^ Creation of StudentRecord Object*

*Member Value*  yourRecord.studentNumber = 2001;

*Member Value*  yourRecord.grade = ‘A’;

*//^ Value Initialization for Object*

StudentRecord yourRecord2(2002,’B’);

*//^ Object in-line value initialization*

cout << yourRecord.studentNumber;

cout << yourRecord.grade; }

**Structure Declaration Example 3:**

*You can also declare Structure Variables at the end inbetween brace and semicolon of the Structure Tag*

struct StudentRecord {

int studentNumber;

char grade;

} yourRecord, myRecord(2001,’A’),herRecord(2002,’B’);

*//^Default Object ^Defined Object ^Defined Object*

**Classes**

Classes are nearly exactly the same to structures, however the general rule of thumb is to use structs for simple data structures, and classes for bigger more complex ones when you want to use encapsulation (although both technically have the same features and functions).

**Class Declaration Example (External Function):**

Class Tag class DayOfYear {

Access Specifier public:

Member Function void output(); // Function Declaration

Member Variables int month;

int day;

};

int main (){

Object DayOfYear today;

Member Values today.month = 9;

today.day = 6;

today.output(); // Calling member function

}

**// ~ The :: operator connects a function to a class ~ \\**

Member Function void DayOfYear::output(){ // Function Definition

switch (month) {

case 1:

cout << “January”; break;

//… rinse repeat

case 12:

cout << “December”; break;

default:

cout << “No such Month exists.”;

}

cout << “, “ << day;

}

Note: You can define the function before or after the main. Location doesn’t matter.

**Class Declaration Example 2 (Internal Function):**

*You can also declare/define class functions within the class*

class DayOfYear {

public:

-----------🡪 void output(){

- switch (month) {

- case 1:

- cout << “January”; break;

- //… rinse repeat

- case 12:

- cout << “December”; break;

- default:

- cout << “No such Month exists.”;

- }

-----------🡪 cout << “, “ << day; }

int month;

int day; };

**Basic Member Functions:**

**Mutators -** Made to specifically change the member’s value of an object

class Person {

public:

void setName(string n) { n = name;}

void setAge(int a) {a = age;}

void setAll(int a, string n) { n = name; a = age; }

string Name;

int Age;

**Acessors** - Made to specifically access a member (Usually when in the private sector of a class)

class Person {

public:

string getName() { return Name; }

int getAge() { return Age; }

string Name;

int Age;

**Encapsulation**

The hiding of information. Hiding information is not only to keep code secret to oneself, but is also could be used to prevent accidental code from accessing it, to hide irrelevant code that other coders wouldn’t need to see, or to help make public code easier and faster to read.

**Public Vs Private**

In example of before, public means the main function can call any of the variable’s values from a simple cout. If put in a private class membership, only a function from the public section of the class could access those private variables and values.

**Example:**

class DayOfYear {

public:

void OutputMonth();

void InputMonth{);

void OutputDay();

void InputDay{);

private:

int month;

int day;

};

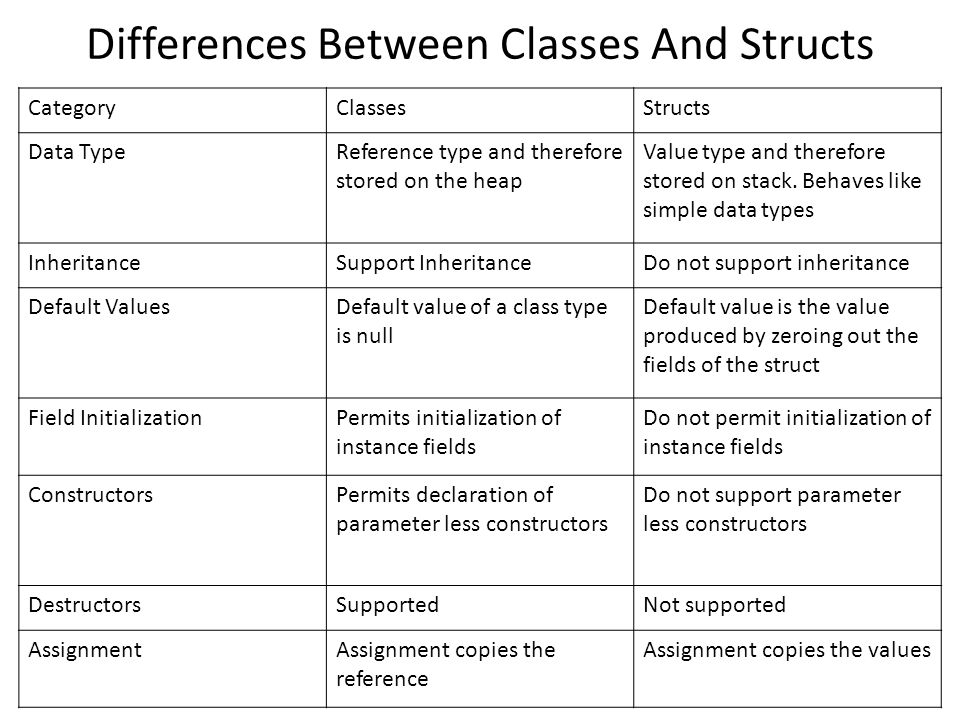
ONLY PUBLIC MEMBERS (input & output) CAN ACCESS PRIVATE MEMBERS (month & day).

today.month = 1 // would not work

today.InputMonth(1) // Would work (with declared function)

Cout << today.month // would not work.

today.OutputMonth // would work (with declared function)



\*\* Structures are PUBLIC by default, and Classes are PRIVATE by default\*\*

\*\* Classes have default constructors that are automatically created (if not done so already) by the compiler when run\*\*

**Constructors**

A constructor is a member function that is automatically called when an object of that class is declared. Its purpose is to initialize **some or all** values of the member variables of the class’s object.

To bypass using constructors we’ve been using an input function to input the variables into their object members. With a constructor this is no longer as necessary.

**Automatic Compiler Constructor Creation (with default member values) Example:**

(Note: No constructor is called, thus compiler makes one for you, however the default constructor prevents you from updating the member values unless an input function is created.)

class Computer {

public:

string getType();

string getBrand();

string getOs();

private:

// c++11 class member initialization

string type = "unknown";

string brand = "unknown";

string operatingSystem = "unknown";

};

**Default Constructor Declaration Example:**

(Note: ‘Default’ referring to what the default variables of object members are without any definition)

class DayOfYear {

public:

DEFAULT Constructor

DayOfYear();

int getMonthNumber);

int getDay();

private:

int month;

int year;

};

DEFAULT Constructor values

DayOfYear::DayOfYear() : month("unknown"),year("unknown"){}

**User-Defined Constructor Declaration Example:**

(Note: This constructor will initialize the object members values when inputted)

class DayOfYear {

public:

Constructor

DayOfYear(int monthValue, int dayValue);

int getMonthNumber();

int getDay();

private:

int month;

int year;

};

Constructor values (All One Line)

DayOfYear::DayOfYear  
(int monthValue,int dayValue) : month(monthValue),year(dayValue){}

**Using Multiple Constructors Example:**

(Note: This example has both the default constructor, along with the user-defined one, and one that allows a mix of filled values and a default one. Each constructor prior to the filled user-defined one will call the constructor ‘before’ (after) it. Default -> Partially filled, Partially Filled -> Complete)

class Computer {

public:

Computer();

Computer(string t, string b);

Computer(string t, string b, string os);

string getType(){return type;}

string getBrand(){return brand;}

string getOs(){return operatingSystem;}

private:

string type;

string brand;

string operatingSystem;

};

// default constructor (c++11 constructor delegation)

Computer::Computer() : Computer("unknown", "unknown", "unknown"){}

// user defined constructor (c++11 constructor delegation)

Computer::Computer(string t, string b): Computer(t, b, "unknown"){}

// user defined constructor (c++11 constructor delegation)

Computer::Computer(string t, string b, string os): type(t), brand(b), operatingSystem(os){}

**Operator Overloading**

Overloading an operator simply means that class members can use certain operations used in int main( ) without the hassle of using an accessor function.

Operator Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operator Type | Operators | Member | Non-Member | Friend |
| Arithmetic | + - \* / | Yes | Yes | Yes |
| Conditional | == != < > <= >= | Yes | Yes | Yes |
| Stream | << >> |  |  | Yes |
| Array | [ ] | Yes |  |  |