# CLASS DESIGN IN C++

## STRUCTS REVIEW

- A **struct** is an object without any member functions
- structs are used for diverse data
  - a collection of values of different types
  - the collection is treated as a single item
  - **Member names** – (fields) identifiers inside the struct.

```
struct CDAccount {
    double balance;
    double interestRate;
    int term; // number of months
};
```

# STRUCTS

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

struct CDAccount {
	double balance;
	double interestRate;
	int term; // number of months
};

void getData(CDAccount& theAccount);
//postcondition: the user has entered the values

int main()
{
	//declaring struct objects
	CDAccount myAccount, yourAccount;
	cout << "\nmy Account input: ";
	getData(myAccount);
	return 0;
}

//get the data from the user by reference
void getData(CDAccount& theAccount)
{
	cout << "\nEnter the account initial balance: ";
	cin >> theAccount.balance;

	cout << "\nEnter the interest rate: ";
	cin >> theAccount.interestRate;

	cout << "\nEnter the term (must be 12 or fewer months): ";
	cin >> theAccount.term;
}
```

## DEFINTIONS

- A **class** is a data type whose variables are objects
- A **struct** is an object without any member functions
- **Object** – a special kind of variable that has its own special-purpose functions
- **Stream** – A flow of data.

## CLASSES

## CLASSES

A class is a data type whose variables are objects.

**Object** – a variable that has member functions as well as the ability to hold data values.

The object is actually the value of the variable.

- In a struct by default all member variables are Public
- In a class by default all member variables  are Private

•**Private members** are only accessible inside the class.

•**Public members** are accessible outside the class

## FEATURES OF CLASSES

**Encapsulation** is the practice of bundling data into a single unit with methods that operate on that data. Data hiding using private and public access specifiers.

**Abstraction:** Exposing only essential details.

**Inheritance:** Reusing properties and behaviors of another class. a new class (child) can inherit properties and behaviors of an existing (parent) class

**Polymorphism** is the ability of a single variable or function to handle different data types or objects. Allowing different implementations through function overloading and overriding.

## CLASSES

- Member function - is defined the same way as any other function except that the class name the scope resolution operator `(::)` are given in the function header
- Member functions are implemented outside the class declaration using `::` (scope resolution operator)
- Helps separate interface (declaration) from implementation.

```
void DayOfYear::Output()

returnType ClassName::FunctionName(formal parameter list)
{
    Function body statements
}
```

## CLASS DECLARATION

```
//class example 1 DayOfYear
#include <iostream>
#include <string>
using namespace std;

class DayOfYear
{
public:
    DayOfYear(); //default constructor
    DayOfYear(int, int); //explicit-value constructor
    void input();
    void output()const;

private:
    int month;
    int day;
};
```

# CLASS CONSTRUCTORS

```
//class example 1 DayOfYear
#include <iostream>
#include <string>
using namespace std;

class DayOfYear
{
public:
	DayOfYear(); //default constructor
	DayOfYear(int, int); //explicit-value constructor
	void input();
	void output()const;

private:
	int month;
	int day;
};
```

```
//member function definitions
//constructor initializes member variables to 0
DayOfYear::DayOfYear() : month(0), day(0)
{
    //body is intentionally left blank
}
//overloaded constructor initializes the member variables with input
//also called an explicit-value constructor
DayOfYear::DayOfYear(int newMonth, int newDay)
{
    month = newMonth;
    day = newDay;
}
```

# CLASS MEMBER FUNCTION

```
//class example 1 DayOfYear
#include <iostream>
#include <string>
using namespace std;

class DayOfYear
{
public:
	DayOfYear(); //default constructor
	DayOfYear(int, int); //explicit-value constructor
	void input();
	void output()const;

private:
	int month;
	int day;
};
```

```
//gets the date from the user
void DayOfYear::input()
{
	//dot operator is not needed because it is a member function
	cout << "\nEnter the month as a number: ";
	cin >> month;

	cout << "\nEnter the day of the month: ";
	cin >> day;
}
```

## CLASS MEMBER FUNCTION

```
//class example 1 DayOfYear
#include <iostream>
#include <string>
using namespace std;

class DayOfYear
{
public:
	DayOfYear(); //default constructor
	DayOfYear(int, int); //explicit-value constructor
	void input();
	void output()const;

private:
	int month;
	int day;
};
```

```
//prints the month and day
void DayOfYear::output()const
{
    //dot operator is not needed because it is a member function
    cout << "\nmonth is " << month
        << ", day is " << day << endl;
}
```

# CLASSES

```
//class example 1 DayOfYear
#include <iostream>
#include <string>
using namespace std;

class DayOfYear
{
public:
	DayOfYear(); //default constructor
	DayOfYear(int, int); //explicit-value constructor
	void input();
	void output()const;

private:
	int month;
	int day;
};
```

```
//main function driver
int main()
{
	DayOfYear today, birthday;
	cout << "\nEnter todays date: ";
	today.input();
	cout << "\nEnter your birthday: ";
	birthday.input();
	cout << "\nTodays date is: ";
	today.output();
	cout << "\nYour birthday is: ";
	birthday.output();

	return 0;
}
```

## CLASSES

Private and public Members

With an ideal class definition, you should be able to change the details of how the class is implemented and the only things that you should need to change in any program that uses the class are the definitions of the member functions.

Member variables should only be accessed through the member functions.
Private members cannot be directly accessed in the program except within the member functions.

The variables that follow the label private will be **private member variables** and the functions will be **private member functions.**

Programmers find that it usually makes their code easier to understand and easier to update if they make all member variables private.

Member functions set (mutator or setter)  and get (accessor or getter) are used to access the member variables

# CLASSES

```
class DayOfYear
{
public:

      //default constructor
      DayOfYear();
      //Initializes the date to January first.

      //explicit-value constructor
      DayOfYear(int the_month, int the_day);
      //Precondition: the_month and the_day form a possible date.
      //Initializes the date according to the arguments.

      //get the month and day from the user
      void input();

      //print the date onto the screen
      void output()const;

      //Returns the month, 1 for January, 2 for February, etc.
      int getMonth() const;

      //Returns the day of the month.
      int getDay() const;

      void set(int newMonth, int newDay);
      //sets the month and day of an object

private:
      int month;
      int day;
};
```

# CLASSES

```
class DayOfYear
{
public:

      //default constructor
      DayOfYear();
      //Initializes the date to January first.

      //explicit-value constructor
      DayOfYear(int the_month, int the_day);
      //Precondition: the_month and the_day form a possible date.
      //Initializes the date according to the arguments.

      //get the month and day from the user
      void input();

      //print the date onto the screen
      void output()const;

      //Returns the month, 1 for January, 2 for February, etc.
      int getMonth() const;

      //Returns the day of the month.
      int getDay() const;

      void set(int newMonth, int newDay);
      //sets the month and day of an object

private:
      int month;
      int day;
};
```

```
DayOfYear Christmas, today;

    cout << "\nEnter todays date: ";
    today.input();

    Christmas.set(12, 25);

    if (today.getMonth() == Christmas.getMonth())
    {
        cout << "\nChristmas is this month!!!\n\n";
    }
```

# CLASSES

Once you make a member variable private, there is no way to change its value except by using one of the member functions.

**Encapsulation** – also known as data hiding. It is the technique of making the variables in a class private and accessible only by a controlled interface of public or protected functions. This allows you to modify the internal variables and data structures without breaking the code that uses your class.

TIP: Make all member variables Private

Define **Accessor** (i.e. Getter) and **mutator** (i.e. Setter) functions.

## STRUCTS VS. CLASSES

All members of structures are public by default and all members of classes are private by default. Technically a structure can do everything a class can do but if you do that you would have two names with different syntax rules for the same concept.

Structures are generally used with only member variables, all public, and no member functions.

**Constructor** – a member function of a class that has the same name as the class.

A constructor is called automatically when an object of the class is declared.

Constructors are used to initialize objects.

A constructor must have the same name as the class of which it is a member.

## CONSTRUCTORS AND DESTRUCTORS

**Constructors** initialize objects

**Destructors** clean up resources when objects go out of scope

```
class MyClass {
private:
    int data;
public:
    MyClass(int value); // Constructor
    ~MyClass();         // Destructor
};

MyClass::MyClass(int value) { data = value; }
MyClass::~MyClass() { /* Cleanup code */ }
```

## ACCESS SPECIFIERS

•**Private:** Accessible only within the class.

•**Public:** Accessible from outside the class.

•**Protected:** Used in inheritance; accessible in derived classes.

```
class Example {
private:
    int a; // Private
protected:
    int b; // Protected
public:
    int c; // Public
};
```

# INHERITANCE OVERVIEW

**Derived class** inherits features from **Base class**.

Supports **code reuse** and **polymorphism**.

```
class Base {
public:
    void show() { std::cout << "Base class"; }
};

class Derived : public Base {
public:
    void display() { std::cout << "Derived class"; }
};
```

## ABSTRACT DATA TYPES

A **data type** consists of a collection of values together with a set of basic operations defined on those values.

A data type is called an **Abstract Data Type (ADT)** if the programmers who use the type do not have access to the details of how the operations are implemented.

## ABSTRACT DATA TYPES

**HOW TO define a class so it is an ADT:**

- Separate the specification of how the type is used by a programmer from the details of how the type is implemented
- Make all member variables private members
- Basic operations a programmer needs should be public member functions
- Fully specify how to use each public function
- Helper functions should be private members

## ABSTRACT DATA TYPES

**The ADT interface tells how to use the ADT in a program**
The interface consists of

- The public member functions
- The comments that explain how to use the functions
- The interface should be all that is needed to know how to use the ADT in a program

**ADT Benefits:**

- Changing an ADT implementation does require changing a program that uses the ADT
- ADT's make it easier to divide work among different programmers
- Writing and using ADTs breaks the larger programming task into smaller tasks

## THE CONST PARAMETER

const can be used at the end of a member function to indicate that it is a "const member function" and the function will not make any modifications to the object, for example:

```
//Returns the month, 1 for January, 2 for February, etc.
int getMonth() const;

//Returns the day of the month.
int getDay() const;
```

```
bool equal(DayOfYear date1, DayOfYear date2)
{
        return (date1.getMonth() == date2.getMonth() &&
                date1.getDay() == date2.getDay());
}
```