Introduction to C++

Today' Class

- **Agenda**
  - Structures
  - Classes
    - Constructor
    - Destructor
    - Scope Resolution operator

C++: Struct

- **Struct**
  - A user-defined composite data type
  - Syntax:

```
struct type_name {
    type member_name1;
    type member_name2;
    type member_name3;
     . .. . .. ..
    type member_name_N;
} instances;
```

  - Structures can be nested

- Example

```
struct Product {
    int weight;
    double price;
} ;
Product apple, walnuts;
```

Refer to: Structs

C++: Struct

- **Struct**
  - Product apple, oranges;

    apple = oranges
    - Same as

    apple.weight = oranges.weight

    apple.price = oranges.price
  - Example

```
struct Product {
    int weight;
    double price;
};
```

Product apple, walnuts;

C++: Struct

- **Used for**
  - » Defining custom datatypes
  - » For grouping logical attributes together into a single entity
    - › Easier to refer and pass it as an argument in functions

C++: Class

- **Class**
  - OO concept in C++

```
class className{
        public: members & methods;
        private: members & methods;
        protected: members & methods;
}
```

- Access Specifier
  - Private - Accessible only from within other members of the same class
    - Default value if not specified
  - Protected - Accessible from members of the same class, but also by the members defined in the derived classes
  - Public - Accessible from anywhere where the object is visible

C++: Class Constructor

Refer to: SimpleClass

Constructor

- Every class has a no-argument constructor by default
  - If you define a constructor, the default is not provided anymore

```
class Person{

private:
          int SSN;
protected:
          string taxID;
public:
     string firstName = "Rishi";
     string lastName = "Saripalle";
     string email = "rishi.saripalle@ilstu.edu";
};
int main (){
/*
 * calls the default no-argument constructor
 */
          Person p;
          cout<<"Name: "<<p.lastName<<" "<<p.firstName<<endl;
}
```

C++: Class Constructor

Refer to: Class2

Constructor

- Every class has a no-argument constructor by default
  - If you define a constructor, the default is not provided anymore

```
int main(){
        Student student;
        ......
        .....
}
```

```
class Student{

        private:
                int SSN = 007;
        protected:
                string taxID = "MI5007";
        public:
                string firstName;
                string lastName;
                string email;
        Student(string fname,string lname,string email){
                this->firstName = fname;
                lastName = lname;
                this->email = email;
        }
};
int main(){
        Student p("Rishi","Saripalle","rsaripa@ilstu.edu");
        cout<<"Name: "<<p.lastName<<" "<<p.firstName<<endl;
}
```

C++: Class Copy Constructor

— Copy Constructor

» A constructor which creates a *new* object using an *existing* object of the same class

» The copy constructor must have one of the following signatures:

- MyClass(const MyClass& object);
- MyClass(MyClass& object);
- MyClass(volatile const MyClass& object);
- MyClass(volatile MyClass& object);

- MyClass(MyClass object); ⊘

C++: Class Copy Constructor

Refer to: Class3

Copy Constructor Example

```
class Teacher{

public:
     string firstName;
     string lastName;
     string email;
Teacher(string fname,string lname,string email){
     this->firstName = fname;
     lastName = lname;
     this->email = email;
}
/*
 * Copy Constructor. Only one parameter - existing object
 */
Teacher(const Teacher& obj){
     firstName = obj.firstName;
     lastName = obj.lastName;
     email = "rks-"+obj.email;
}
};
```

```
int main(){
    Teacher s("Rishi","Saripalle","rsarip@ilstu.edu");

    // Both the statements below call the copy constructor
    Teacher copy(s);
    Teacher copy2 = s;
    cout<<"Copy object email: "<<copy.email<<endl;
    cout<<"Copy2 object email: "<<copy2.email<<endl;

    Teacher newFaculty("Sashi","Saripalle","ssarip@ilstu.edu");

    newFaculty = s;
}
```

C++: Scope Resolution Operator

- **:: - Scope Operator**
  - Specifies the class to which the member/variable being declared belongs
  - Difference between *function defined within the class* vs. *its declaration and defined later outside the class*,
    - Former, the function is automatically considered an ***inline*** member function by the compiler
    - Later is a normal (not-inline) class member function.
  - **NO** differences in behavior.

C++: Scope Resolution Operator

Refer to: Class2.cpp

:: - Scope Operator Example

```
using namespace std;

class Student{

    private:
        int SSN = 007;
    protected:
        string taxID = "MI5007";
    public:
        string firstName;
        string lastName;
        string email;
    // Declare the method in the class
    Student(string,string,string);
    void sendEmail();
};
```

```
/*
 * :: scope resolution operator
 * class_name:: method_name(parameters){ function body}
 * return_type
class_name::method_name(parameters){function body}
 */
Student::Student(string fname,string lname,string
email){
        this->firstName = fname;
        lastName = lname;
        this->email = email;
}
void Student::sendEmail(){
        cout <<"Test Email is sent to "<<email<<endl;
}
int main(){
    Student p("Rishi","Saripalle","rsaripa@ilstu.edu");
    cout<<"Name: "<<p.lastName<<" "<<p.firstName<<endl;
    p.sendEmail();
}
```

C++: const Function

Refer to: Class3.cpp

const Modifier

Cannot change the ***object*** , i.e."me" that is calling sendEmail method.

The **const** modifier at the end of the sendEmail function wants to make sure that the calling object is not modified.

```
class Teacher{

        ……….
        public:
                string firstName;
                string lastName;
                string email;
        Teacher(string fname,string lname,string email){
                this->firstName = fname;
                lastName = lname;
                this->email = email;
        }
        ……
        }
        void sendEmail() const{
                cout <<"sending Email to "<<email<<endl;
                this->firstName = "Rishi Kanth";
        }
};
int main(){
                Teacher me("Rishi","Saripalle","rsarip@ilstu.edu");
                me.sendEmail();
}
```

C++: const Function

Refer to: Class3.cpp

— const Modifier

In this example, sendEmail is const function. However, it is calling **verifyEmail**().

The **verifyEmail** must also be **const**.

```
class Teacher{

      ……….
      public:
            string firstName;
            string lastName;
            string email;
      Teacher(string fname,string lname,string email){
            this->firstName = fname;
            lastName = lname;
            this->email = email;
      }
      ……
      }
      void sendEmail() const{
            verifyEmail();
            cout <<"sending Email to "<<email<<endl;
      }
      string verifyEmail() const{
            return email;
      }
};
int main(){
            Teacher me("Rishi","Saripalle","rsarip@ilstu.edu");
            me.sendEmail();
}
```

C++: Class Destructor

- **Destructor**
  - » Special member function of a class
  - » Executed whenever an object goes out of scope
    - › Delete expression is applied to a pointer to the object of that class (discussed later)
  - » Same name as the class prefixed with a tilde (~)
  - » **CANNOT** return a value nor take any parameters.
  - » Very useful for releasing resources

C++: Class Destructor

- **Destructor**
  - » Reverse order of construction → First constructed, last destructed
  - » **CANNOT** overload them → ONLY ONE ~Rectangle()
  - » **DON’T CALL** destructor directly
    - › Will get called once out of scope.
    - › Happens automagically
    - › If required, use DELETE only when using NEW

C++: Class Destructor

- **Destructor**
  - » *me* an object of Faculty has only the scope in Main method

```
#include <iostream>

using namespace std;

class Faculty{

	public:
		string firstName;
		string lastName;
		string email;
	Faculty(string fname,string lname,string email){
		this->firstName = fname;
		lastName = lname;
		this->email = email;
	}
	~Faculty(){
		cout<<"The faculty object is destroyed"<<endl;
	}
};
int main(){
		Faculty me("Rishi","Saripalle","rsarip@ilstu.edu");
}
```

C++: Static Modifier

- **Static**
  - Member Variables
    - Every object will only have ONE static variable
    - Its not tied to the instance, but to the class
  - Member Methods
    - Method is independent of the class
    - CANNOT access *this* pointer
    - ONLY access static member variables

C++: Operator Overloading

- **Operator Overloading**
  - Overload most of the built-in operators in C++ to work with you user-define datatypes
    - You CANNOT overload
      - :: (scope resolution), . (member access), .* (through pointer), and ?:(ternary conditional)
    - CANNOT change the precedence, grouping, or number of operands of operators

C++: Operator Overloading

- **Operator Overloading**
  - » Syntax

    Return_type Operator*operator*(parameters)

    › Operator***operator*** → operator+, operator*, operator-, etc.

    › Most of the operator overloading function need NOT be member functions of the class

C++: Operator Overloading

— Example

Refer to: OperatorOverloading

```
class Money{
      double savings;
      double checking;
      public:
            Money(){
                        savings=checking=1000;
            }
            Money(double checking,double savings){
                  this->checking=checking;
                  this->savings=savings;
            }
      …….
      ……
      Money operator +(const Money& a) const;
      Money operator -(const Money& a) const;
      bool operator ==(const Money& a) const;
      // For postfix, you pass a int as parameter, to differentiate
      with prefix. The argument has not value or use.
      void operator++(int);
      void operator++();//prefix
};
```

```
Money Money::operator +(const Money& a) const{
      Money temp;
      temp.setChecking(this->getChecking()+a.getChecking());
      temp.setSavings(this->getSavings()+a.getSavings());
      return temp;
}

bool Money::operator ==(const Money& a) const{
      if(this->checking == a.getChecking() && this-
      >getSavings()==a.getSavings())
            return true;
      else
            return false;
}


int main(){
            Money me(5000,3000);
            Money partner(2000,4000);

            Money total = me+partner;

            return 0;
}
```

C++: Operator Overloading

Refer to: OperatorOverloading

— Details

```
int main(){
        Money me(5000,3000);
        Money partner(2000,4000);

        Money total = me + partner;

        return 0;
}
```

Implicitly converted into

me

partner

```
Money::operator +(const Money& this, const Money& a)
```

```
Money Money::operator +(const Money& this, const Money& a) {
    Money temp;
    temp.setChecking(this->getChecking()+a.getChecking());
    temp.setSavings(this->getSavings()+a.getSavings());
    return temp;
}
```

Thank You

Question, Comments & Feedback