HO CHI MINH UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

ADVANCED PROGRAMMING

REPORT
FUNCTIONAL & OBJECT ORIENTED PROGRAMMING

LECTURER:	Truong Tuan Anh
CLASS:	CC01
STUDENT:	Pham Tan Phuoc - 1952406

HO CHI MINH CITY, 8/2021

Contents

Chapter 1

Key advantages of Polymorphism in C++

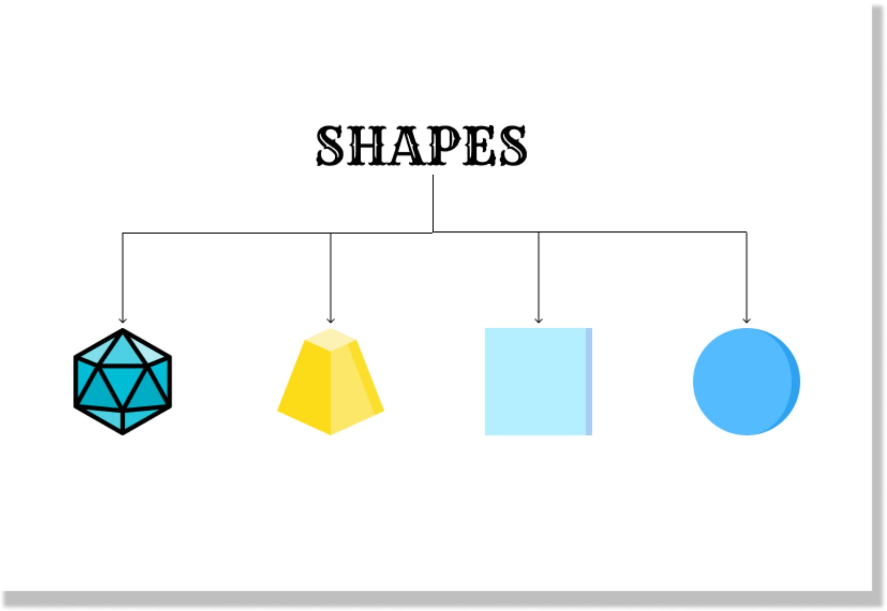
1.1 What is Polymorphism in C++?

Polymorphism in C++ means, the same entity (function or object) behaves differently in different scenarios.

Let's imagine that we'll need to write a program that'll calculate the area and perimeter of shapes. To do this, we'll need to define methods to calculate the area and perimeter of our shapes; area() and perimeter(). However, basic knowledge of geometry tells us that we can't solve for the area and perimeter of different shapes in the same way. For instance, the perimeter of a circle is 2 x Pi x radius and the perimeter of a square is 4 x Length.

So, we'll need to define the different shapes as sub-classes or derived classes of the base class shapes. Therefore, we'll have a subclass circle, square, trapezium, polygon which will all have their methods and different parameters. What we have done above is polymorphism. Our class shapes now has different forms and characteristics like circle, square, trapezium, and polygon as sub-classes.

Polymorphism is derived from two words; poly and morphism which means many and variance of form. In programming, polymorphism is defined as the ability of an object to take on different forms.



In C++ we have two types of polymorphism:

1. Compile time Polymorphism – This is also known as static (or early) binding.
2. Runtime Polymorphism – This is also known as dynamic (or late) binding.

1.2 Compile Time Polymorphism

In compile-time polymorphism, a function is called at the time of program compilation. We call this type of polymorphism as early binding or Static binding.

Function overloading and operator overloading is the type of Compile time polymorphism.

1. Function Overloading

   Function overloading means one function can perform many tasks. In C++, a single function is used to perform many tasks with the same name and different types of arguments. In the function overloading function will call at the time of program compilation. It is an example of compile-time polymorphism.

   In the below example, A function ADD() is used to perform two tasks. The two asks would be to add two integer values and add two strings (concatenate).

   Readability of the program increases by function overloading. It is achieved by using the same name for the same action.

```cpp
#include <iostream>
using namespace std;
class Addition {
    public:
        int ADD(int X,int Y) // Function with parameter
        {
            return X+Y;    // this function is performing addition of
                two Integer value
        }
        int ADD() { // Function with same name but without parameter
        string a= "HELLO";
        string b="SAM"; // in this function concatenation is
            performed
        string c= a+b;
        cout<<c<<endl;
        }
};
int main(void) {
     Addition obj; // Object is created
     cout<<obj.ADD(128, 15)<<endl; //first method is called
     obj.ADD(); // second method is called
     return 0;
}
```

```
Output:
143
HELLOSAM
```

   In the above example, we use function ADD() to perform many tasks which is a property of polymorphism in C++.

   Advantages:

   - The use of function overloading is to save the memory space, consistency and readability.
   - We can develop more than one function with the same name.
   - It speeds up the execution of programme.

- Function overloading is done for code re-usability, to save efforts, and also to save memory.
- It helps app to load the class method based on the type of parameter.
- Code maintenance is easy.

2. Operator Overloading

   Operator overloading means defining additional tasks to operators without changing its actual meaning. We do this by using operator function.

   The purpose of operator overloading is to provide a special meaning to the user-defined data types.

   For example: We will declare the addition operator that can be used to add two Box objects and returns final Box object

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

class Box {
   public:
      double getVolume(void) {
         return length * breadth * height;
      }
      void setLength( double len ) {
         length = len;
      }
      void setBreadth( double bre ) {
         breadth = bre;
      }
      void setHeight( double hei ) {
         height = hei;
      }

      // Overload + operator to add two Box objects.
      Box operator+(const Box& b) {
         Box box;
         box.length = this->length + b.length;
         box.breadth = this->breadth + b.breadth;
         box.height = this->height + b.height;
         return box;
      }

   private:
      double length;    // Length of a box
      double breadth;   // Breadth of a box
      double height;    // Height of a box
};

// Main function for the program
int main() {
   Box Box1;              // Declare Box1 of type Box
```

```
        Box Box2;               // Declare Box2 of type Box
        Box Box3;               // Declare Box3 of type Box
        double volume = 0.0;    // Store the volume of a box here

        // box 1 specification
        Box1.setLength(6.0);
        Box1.setBreadth(7.0);
        Box1.setHeight(5.0);

        // box 2 specification
        Box2.setLength(12.0);
        Box2.setBreadth(13.0);
        Box2.setHeight(10.0);

        // volume of box 1
        volume = Box1.getVolume();
        cout << "Volume of Box1 : " << volume <<endl;

        // volume of box 2
        volume = Box2.getVolume();
        cout << "Volume of Box2 : " << volume <<endl;

        // Add two object as follows:
        Box3 = Box1 + Box2;

        // volume of box 3
        volume = Box3.getVolume();
        cout << "Volume of Box3 : " << volume <<endl;

        return 0;
    }
```

```
    Output:
    Volume of Box1: 210
    Volume of Box2: 1560
    Volume of Box3: 5400
```

Note: Following is the list of operators which can not be overloaded: "::", ".*", ".", "?:".

Advantages:

- Operator overloading in C++ enables programmers to use notation closer to the target domain.
- They provide similar support to built-in types of user-defined types.
- Operator overloading in C++ makes the program easier to understand.
- Perform different operations on the same operand.

1.3 Runtime Polymorphism

In a Runtime polymorphism, functions are called at the time the program execution. Hence, it is known as late binding or dynamic binding.

Function overriding is a part of runtime polymorphism. In function overriding, more than one method has the same name with different types of the parameter list.

It is achieved by using virtual functions and pointers. It provides slow execution as it is known at the run time. Thus, It is more flexible as all the things executed at the run time.

1. Function Overriding

   In function overriding, we give the new definition to base class function in the derived class. At that time, we can say the base function has been overridden. It can be only possible in the 'derived class'. In function overriding, we have two definitions of the same function, one in the superclass and one in the derived class. The decision about which function definition requires calling happens at runtime. That is the reason we call it 'Runtime polymorphism'.

   For example:

```
#include <iostream>
using namespace std;
class Animal {
    public:
void function(){
cout<<"Eating..."<<endl;
    }
};
class Man: public Animal
{
 public:
 void function()
    {
       cout<<"Walking ..."<<endl;
    }
};
int main(void) {
 Animal A =Animal();
   A.function(); //parent class object
   Man m = Man();
   m.function(); // child class object

   return 0;
}
```

```
Output:
Eating ...
Walking ...
```

   Advantages:

- The child class having same function of parent class,can even have its independent implementation of that function.
- Helps in saving the memory space.
- Maintain consistency, and readability of our code.
- Helps in making code re-usability easy.
- The child class can access function of parent class as well.

2. Virtual Function

A virtual function is declared by keyword virtual. The return type of virtual function may be int, float, void.

A virtual function is a member function in the base class. We can redefine it in a derived class. It is part of run time polymorphism. The declaration of the virtual function must be in the base class by using the keyword virtual. A virtual function is not static.

The virtual function helps to tell the compiler to perform dynamic binding or late binding on the function.

If it is necessary to use a single pointer to refer to all the different classes' objects. This is because we will have to create a pointer to the base class that refers to all the derived objects.

But, when the base class pointer contains the derived class address, the object always executes the base class function. For resolving this problem, we use the virtual function.

When we declare a virtual function, the compiler determines which function to invoke at runtime.

Example: Sample program to illustrate use of virtual function in C++

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

class Animal{
 public:
 virtual void my_features()
 {
 cout << "I am an animal.";
 }
};
class Mammal : public Animal{
 public:
 void my_features()
 {
 cout << "\nI am a mammal.";
 }
};
class Reptile : public Animal{
 public:
 void my_features()
 {
```

```
            cout << "\nI am a reptile.";
            }
           };

           //intermediate function
           void intermediate_func(Animal *a1)
           {
            a1->my_features();
           }

           int main()
           {
            Animal *obj1 = new Animal;
            Mammal *obj2 = new Mammal;
            Reptile *obj3 = new Reptile;

            intermediate_func(obj1);
            intermediate_func(obj2);
            intermediate_func(obj3);

            return 0;
           }
```

```
           Output:
           I am an animal.
           I am a mammal.
           I am a reptile.
```

Advantages

- Polymorphism
- Reusability of code
- Small program length

1.4 Conclusion

Polymorphism allows programmers to reuse, evaluate and execute the program, modules, forms written once. In certain aspects, they can be repeated.

You may use the odd variable name to stock variables of different types of data, such as Int, Float, etc.).

Polymorphism tends to reduce the pairing of multiple functionalities.

Method overloading can be extended to builders that allow multiple ways of initializing class objects. It helps you to identify several builders for managing various forms of initializations.

Method overriding functions along with inheritance without the need for re-compilation to allow code reuse of existing groups.

Chapter 2

Object Oriented Programming in C++ and Java

Nowadays, Java and C++ programming languages are vasty using in competitive coding. Due to some awesome features, these two programming languages mostly used in Industry also. C++ is a widely popular language among coders for its efficiency, high speed, and dynamic memory utilization. Java is widely in the IT industry, It is incomparable with any other programming language in terms of software development.

Object-oriented programming (OOP) is one of the most important concepts in C++ and Java. It is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior. OOP focuses on the objects that developers want to manipulate rather than the logic required to manipulate them. This approach to programming is well-suited for programs that are large, complex and actively updated or maintained.

Let's go through the various points to compare OOP Programming between these two popular coding languages, the following table reviews some concepts and terminology as used in the text or as you might see in other places.

Term or Concept	Object-Oriented Paradigm	Java	C++
Supported Paradigm	Object-Oriented	Object-Oriented	Object-Oriented Procedural
class	class	class	class
object	object, instance	object, instance	object, instance
Create an object from a class	instantiate	instantiate	instantiate
Data not associated with a class	N/A	N/A	variable
Operation not associated with a class	N/A	N/A	function
Data associated with or stored in a class	attribute	instance variable, instance field	data member, member variable
Object associated with a class	behavior, operation	method	member function

2.1 Class vs Object

From its name, it seems like object-oriented programming is all about objects. Yet, when we write an object-oriented program, our code defines a feature called a "class" and both C++ and Java include **class** as a keyword - but neither has **object** as a keyword. We previously used a cookie-cutter and cookies metaphor to illustrate the relationship between a class and the objects instantiated from it.

In the cookie-cutter metaphor, the cutter plays the role of the class while the cookies are the objects. A cookie-cutter describes the shape and the size of a cookie, but it's inedible and does not consist of any cookie ingredients. Alternatively, we make cookies from flour, sugar, and butter. Once we make the cookie dough, we can stamp out as many cookies as wanted with the cookie cutter. Classes and objects are fundamentally connected, so at times the terms are used interchangeably. Nevertheless, the two concepts are distinct.

So, we can say that a class is a description that describes three characteristics of the objects that are created or instantiated from it:

1. The name of the class becomes a new data type that may be used throughout the program and is the type of objects instantiated from it.
2. Each class forms an aggregate data type, which means that it can hold many, smaller variables. The class specifies the names and types of all the variables in the objects instantiated from the class.
3. Classes also contain functions or methods that can operate on the data or variables stored in the object. The class specifies the name of each function, its return type, and the number and type of each argument.

Just as cookies have ingredients, so do objects. Memory is the "raw ingredient" programmers use to make objects. One of the most striking differences between C++ and Java is where and how programmers allocate that memory.

Differences in allocating memory:

- C++: With C++, you call a standard function **new** to request a number of bytes. It returns a pointer pointing to the first of those bytes. There is no guarantee as to what values those bytes will be set to. In C++, **new** guarantees contiguous bytes meaning you can keep adding one to that pointer and reach all the bytes allocated. Once finished with that memory, you call the **delete** function, passing the pointer, to release the memory. So we have memory as a block of bytes, contiguous memory, pointer arithmetic and manual control of free.
- Java: Java uses **new** to allocate memory. We pass it the name of a Java class rather than a number of bytes. The Java runtime then measures how many bytes an instance of that class needs, allocates them and calls the object constructor. This sets the object fields to known values. **New** then returns a reference to that object. It has no meaning in terms of bytes. It allows methods to be called in that object, whose data lives in the allocated memory. There is no pointer arithmetic here. Once you're done, Java will work out that your program has no more references to that object. It will eventually release the memory automatically. There is no equivalent to **delete()**. Java has a concept of object instances not raw bytes, initialises object memory, has no pointer arithmetic and uses automatic memory management, called 'garbage collection'. Java also checks that memory references

are not accidentally zero. It will not crash if they are. It throws a NullPointerException with a stack trace showing where the program was at the time.

2.2 C++ and Java syntax

The object-oriented paradigm requires three main characteristics: encapsulation, inheritance, and polymorphism.

1. Encapsulation

   We achieve encapsulation by packaging data and the operations that use that data together into an entity called an object. Java and C++ use different terms to name class variables and operations. Java calls them instance variables or fields and methods, respectively, which denotes that they "belong" to instances or objects. C++ calls them member variables and member functions because they are members of a class. Much of the following syntax has to do with creating objects, accessing data, and calling operations.

   - Constructors

     Constructors are just functions (C++) or methods (Java), but they are special in one significant way. Each time a program instantiates a new object, it automatically calls a constructor that builds or initializes the object making it ready for use by the program. Constructors can be simple or complex. Simple constructors may simply initialize the variables in an object by transferring the values stored in the constructor arguments to the variables. Complex constructors may perform more complicated operations.

     Creating or instantiating a new object in C++ is similar to instantiating an object in Java, but there are some important differences. Suppose that we have a class named Foo and we wish to instantiate a Foo object. Assume that class Foo has a constructor that does not take any arguments. The instantiation statements look like this:

     ```
             Foo myFoo1 = new Foo(); //Java
             Foo* myFoo2 = new Foo; //C++
             Foo myFoo3; //C++
     ```

     We begin to understand why C++ is a more complex language than Java with this example: C++ has more ways of creating objects than Java. Note the use of the asterisk (*) in conjunction with the class name when the **new** keyword is used in C++. The asterisk modifies the variable myFoo2 making it a **pointer**.

     Unlike Java, C++ does not require parentheses when the constructor does not have any arguments (previously, C++ didn't allow parentheses, but they are optional now). If the constructor does require an argument, then the parentheses are needed:

     ```
             Foo myFoo1 = new Foo(5); //Java
             Foo* myFoo2 = new Foo(5); //C++
             Foo myFoo3(5); //C++
     ```

   - General method and Function calls

Now let's suppose that the **Foo** class has a method or function named **doSomething**. C++ has two ways of creating an object, which requires two ways to use the object, which in turn requires an operator not found in Java. The new operator, called arrow, is formed by the minus and greater than symbols together without any space and is used whenever the variable to the left is a pointer. Otherwise, both languages use the dot operator to make the call.

```
myFoo1.doSomething(); //Java
myFoo2->doSomething(); //C++
myFoo3.doSomething(); //C++, looks like Java
```

2. Inheritance

- In Java, all classes inherit from the Object class directly or indirectly. Therefore, there is always a single inheritance tree of classes in Java, and Object class is root of the tree. In Java, if we create a class that doesn't inherit from any class then it automatically inherits from Object class . In C++, there is forest of classes; when we create a class that doesn't inherit from anything, we create a new tree in forest. Following Java example shows that Test class automatically inherits from Object class.

```
class Test{
    //members of test
}
class Main{
    public static void main(String[] args){
        Test t = new Test();
        System.out.println("t is instanceof Object: " + (t
            instanceof Object));
    }
}
```

- In Java, members of the grandparent class are not directly accessible.

- Java uses extends keyword for inheritence. Unlike C++, Java doesn't provide an inheritance specifier like public, protected or private. Therefore, we cannot change the protection level of members of base class in Java, if some data member is public or protected in base class then it remains public or protected in derived class. Like C++, private members of base class are not accessible in derived class. Unlike C++, in Java, we don't have to remember those rules of inheritance which are combination of base class access specifier and inheritance specifier.

- The meaning of protected member access specifier is somewhat different in Java. In Java, protected members of a class "A" are accessible in other class "B" of same package, even if B doesn't inherit from A (they both have to be in the same package). For example, in the following program, protected members of A are accessible in B.

```
// filename B.java
class A{
    protected int x = 10, y = 20;
}
```

```
class B{
    public static void main (String args[]){
        A a = new A();
        System.out.println(a.x + " " + a.y);
    }
}
```

- Unlike C++, Java doesn't support multiple inheritance. A class cannot inherit from more than one class. A class can implement multiple interfaces though.
- In C++, default constructor of parent class is automatically called, but if we want to call parametrized constructor of a parent class, we must use Initializer list. Like C++, default constructor of the parent class is automatically called in Java, but if we want to call parametrized constructor then we must use super to call the parent constructor. See following Java example.

```
package main;

class Base{
    private int b;
    Base(int x){
        b = x;
        System.out.println("Base constructor called");
    }
}

class Derived extends Base{
    private int d;
    Derived(int x, int y){
        //Calling parent class parameterized constructor
        //Call to parent constructor must be the first line in
            a Derived class
        super (x);
        d = y;
        System.out.println("Derived constructor called");
    }
}

class Main{
    public static void main(String[] args){
        Derived obj = new Derived(1,2);
    }
}
```

3. Polymorphism

Polymorhism can be turned on or off in C++ by a keyword virtual. In Java, polymorphism is always on by default. If one does not want it, the keyword "final" can be used.

Polymorphism is achieved by method overloading and method overriding. In C++ we also have both. But C++ has operator overloading which Java has not.

2.3 Arrays

Strictly speaking, arrays are not an object-oriented construct. However, Java implements them as objects, while C++ treats them as a primitive or fundamental data type.

Conceptually and functionally, an array is a collection of variables managed by a common name. Individual variables, called elements, within the array are selected by an index or subscript. Arrays have a size, which is the number of elements or variables that they hold. Programs determine the size of each array when it creates them. C++ and Java implement zero-indexed arrays, which means that valid array index values range from 0 to the size of the array - 1. All of the variables in the array must be the same data type (e.g., int or double). The following figure demonstrates how programmers create arrays in C++ and Java. The figure also shows an abstract representation of an array in computer memory.

```
\\Java
double[] scores = new double[8];
```

```
\\C++
double scores[8];
double* scores = new double[8];
```

- In Java, an array is an instance of an unnamed class that is always created with the new operator. (The java.util.Arrays class contains a collection of static methods that are useful when working with arrays but is not involved with creating arrays).
- Arrays may be created in two different ways in a C++ program. The second way is very similar (both in appearance and in the underlying concepts) to what is done in Java.
- Arrays created in both languages have a similar appearance. The illustrated array has eight double variables or elements that are are accessed as scores[0], scores[1], scores[2], ..., scores[7].

2.4 Conclusion

1. Type of Programming Language
   - C++ is both a procedural and object-oriented programing language. Hence, C++ has features specific to procedural languages as well as features of object-oriented programming language.
   - Java is a completely object-oriented programming language.
2. Memory Management - Used to make objects
   - C++: Managed by developers using pointers. Supports structures and union.
   - Java: Controlled by system, does not use pointers. Supports Threads and Interfaces.
3. Inheritance
   - C++: Provide single and multiple inheritance both.
   - Java: Does not support multiple inheritance. Uses the concept if Interface to achieve.

4. Overloading
   - In C++, methods and operators can be overloaded. This is static polymorphism.
   - In Java, only method overloading is allowed. It does not allow operator overloading.
5. Virtual Keyword
   - As a part of dynamic polymorphism, in C++, the virtual keyword is used with a function to indicate the function that can be overridden in the derived class. This way we can achieve polymorphism.
   - In Java, the virtual keyword is absent. However, in Java, all non-static methods by default can be overridden. Or in simple terms, all non-static methods in Java are virtual by default.

Chapter 3

OOP & Functional programming

3.1 A brief overview of OOP and Functional Programming

1. What is OOP?

   OOP, which is Object-Oriented Programming, as mentioned in chapter 2, is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior. OOP focuses on the objects that developers want to manipulate rather than the logic required to manipulate them.

   This approach to programming is well-suited for programs that are large, complex and actively updated or maintained.

   Some concepts of OOP:

   - Class: A class in C++ is the building block, that leads to Object-Oriented programming. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A C++ class is like a blueprint for an object.
   - Object: Objects are created from classes. Objects can represent anything the program has to handle, such as a shape, a place, time, and many others. Class objects are declared in a similar way as variables are declared. The class name must start, followed by the object name.
   - Encapsulation: When all the data members and member functions are combined in a single unit called class, this process is called Encapsulation. In other words, wrapping the data together and the functions that manipulate them.
   - Inheritance: Inheritance is a way to reuse once written code again and again. The class which is inherited is called the Base class & the class which inherits is called the Derived class. They are also called parent and child class. So when, a derived class inherits a base class, the derived class can use all the functions which are defined in base class, hence making code reusable.

     Example code:

     ```
     #include <iostream>
     using namespace std;
     class ParentClass {
       //data member
       public:
         int var1 =100;
     };
     class ChildClass: public ParentClass {
       public:
       int var2 = 500;
     };
     int main(void) {
       ChildClass obj;
     }
     ```

     Now this object obj can use the properties (such as variable var1) of ParentClass.

- Polymorphism: Explained in chapter 1, basically, it is a feature, which lets us create functions with same name but different arguments, which will perform different actions. That means, functions with same name, but functioning in different ways. Or, it also allows us to redefine a function to provide it with a completely new definition.

  Example code:

```cpp
#include <iostream>
using namespace std;
class Sum {
  public:
    int add(int num1,int num2){
      return num1 + num2;
    }
    int add(int num1, int num2, int num3){
      return num1 + num2 + num3;
    }
};
int main(void) {
   //Object of class Sum
   Sum obj;

   //This will call the second add function
   cout<<obj.add(10, 20, 30)<<endl;

   //This will call the first add function
   cout<<obj.add(11, 22);
   return 0;
}
```

```
Output:
60
33
```

- Abstract: Abstraction refers to showing only the essential features of the application and hiding the details. In C++, classes can provide methods to the outside world to access & use the data variables, keeping the variables hidden from direct access, or classes can even declare everything accessible to everyone, or maybe just to the classes inheriting it. This can be done using access specifiers – public, private, protected.

2. What is Functional Programming?

   In computer science, functional programming is a programming paradigm where programs are constructed by applying and composing functions. It is a declarative programming paradigm in which function definitions are trees of expressions that map values to other values, rather than a sequence of imperative statements which update the running state of the program.

   Generally, functional programming means using functions to the best effect for solving problems.

   Some concepts of Functional Programming:

- Pure functions: is one whose results are dependent only upon the input parameters, and whose operation initiates no side effect, that is, makes no external impact besides the return value. Example:

```
multi :: Int -> Int -> Int
multi x y = x * y
```

- Recursion: Recursion is important to Haskell because unlike imperative languages, you do computations in Haskell by declaring what something is instead of declaring how you get it. That's why there are no while loops or for loops in Haskell and instead we many times have to use recursion to declare what something is. Example:

```
replicatex :: (Num i, Ord i) => i -> a -> [a]
replicatex n x
    | n <= 0   = []
    | otherwise = x:replicatex (n-1) x
```

- Referential Transparency: Referential transparency, a term commonly used in functional programming, means that given a function and an input value, you will always receive the same output. That is to say there is no external state used in the function. Example:

```
multi :: Int -> Int -> Int
multi x y = x * y
```

  If we input x = 2 and y = 3, the ouput always gets 6 at anytime you run since there's no external variable which interferes in our defined function.

```
*Main> multi 2 3
6
```

- Functions are First-Class and can be Higher-Order:

  First class functions are functions that are treated like an object (or are assignable to a variable).

  Higher order functions are functions that take at least one first class function as a parameter, or return at least one first class function.

  Example: The map function takes another function (head) and a list ([“dog”, “cat”, “mouse”]) as arguments and applies that function to each element in the list:

```
GHCi> map head ["dog","cat", "mouse"]
"dcm"
```

- Variables are Immutable: When a variable is immutable, once a value is bound to a name, you can't change that value. Example:

```
Prelude> let a = 2
Prelude> a
2
Prelude> :i a
a :: Num p => p       -- Defined at <interactive>:1:5
Prelude> let f x = a + x
Prelude> let a = 3
Prelude> a
3
Prelude> f (-2)
0
```

From the output, we can conclude, f never needs to care that you've defined a new variable that's also called a; from its perspective a was one immutable variable that always stays as it is.

3.2 Compare the two programming paradigms

1. Main differences between Functional Programming and OOP

    - Functional programming emphasizes on evaluation of functions while object oriented programming is based on the concept of objects.
    - Functional programming uses immutable data while object oriented programming uses the mutable data.
    - Functional programming follows the declarative programming model while object oriented programming follows the imperative programming model.
    - Functional programming supports parallel programming while object oriented programming does not.
    - In functional programming, statements can be executed in any order. In OOP, statements are executed in a particular order.
    - In functional programming, recursion is used for iterative data while in OOP, loops are used for iterative data.
    - Variables and functions are the basic elements of functional programming. Objects and models are the basic elements of object oriented programming.
    - Functional programming is used only when there are few things with more operations. Object-oriented programming is used when there are many things with few operations.
    - In functional programming, a state does not exist. In object-oriented programming, the state exists.
    - In functional programming, a function is the primary manipulation unit. In object-oriented, an object is the primary manipulation unit.
    - Functional programming provides high performance in processing large data for applications. Object-oriented programming is not good for big data processing.

- Functional programming does not support conditional statements. In Object-oriented programming, conditional statements can be used like if-else statements and switch statement.

2. Illustration of the comparison between Functional Programming and OOP

Functional Programming	**OOP**
#1. Definition	
FP emphasizes on evaluation of mathematical functions	OOP based on concept of objects
#2. Data	
FP uses immutable data	OOP uses the mutable data
#3. Model	
FP does follow declarative programming model	OOP does follow imperative programming model
#4. Support	
Parallel programming supported by FP	OOP does not support parallel programming
#5. Execution	
In FP, the statements can be executed in any order	In OOPs, the statements should be executed in particular order
#6. Iteration	
In FP, recursion is used for iterative data	In OOPs, loops are used for iterative data
#7. Element	
The basic elements of FP are Variables and Functions	The basic elements of OOP are objects and methods
#8. Use	
The FP is used only when there are few things with more operations	OOP is used when there are many things with few operations
#9. Languages	
Haskell, Lisp, Racket are some languages that support FP	C++, Java, Python are some languages that support OOP

3.3 Advantages and Disadvatages of Functional Programming and OOP

1. Functional Programming

   (a) Advantages:

   - The order of execution doesn't matter since it is handled by the system to compute the value we have given rather than the one defined by programmer. In other words, the declarative of the expressions will become unique. Because functional programs have an approach toward mathematical concepts, the system will be designed with the notation as close as possible to the mathematical way of concept.
   - Variables can be replaced by their value since the evaluation of expression can be done any time. The functional code is then more mathematically traceable because the program is allowed to be manipulated or transformed by substituting equals with equals. This feature is called referential transparency.
   - Immutability makes the functional code free of side effects. A shared variable, which is an example of a side effect, is a serious obstacle for creating parallel code and result in non-deterministic execution. By removing the side effect, we can have a good coding approach.
   - The power of lazy evaluation will make the program run faster because it only provides what we really required for the queries result. Suppose we have a large amount of data and want to filter it by a specific condition, such as showing only the data that contains the word Name. In imperative programming, we will have to evaluate each operation of all the data. The problem is that when the operation takes a long time, the program will need more time to run as well. Fortunately, the functional programming that applies LINQ will perform the filtering operation

only when it is needed. That's why functional programming will save much of our time using lazy evaluation.

- We have a solution for complex problems using composability. It is a rule principle that manages a problem by dividing it, and it gives pieces of the problem to several functions. The concept is similar to a situation when we organize an event and ask different people to take up a particular responsibility. By doing this, we can ensure that everything will done properly by each person.

(b) Disadvantages:

- Since there's no state and no update of variables is allowed, loss of performance will take place. The problem occurs when we deal with a large data structure and it needs to perform a duplication of any data even though it only changes a small part of the data.
- Compared to imperative programming, much garbage will be generated in functional programming due to the concept of immutability, which needs more variables to handle specific assignments. Because we cannot control the garbage collection, the performance will decrease as well.

2. Object Oriented Programming

(a) Advantages:

- A real-world idea can be demonstrated, as everything in OOP is treated as an object.
- As we use the concept of encapsulation, programs are easier to test and maintain.
- Faster development of code is done as we develop classes parallel instead of sequentially.
- OOP provides greater security due to data abstraction. The outside world cannot access the hidden data.
- Reusability can be achieved by using classes that have been already written.

(b) Disadvantages:

- Designing a program with an OOP concept can be tricky.
- A programmer needs to plan beforehand for developing a program in OOP.
- The size of programs developed with OOP is bigger than those developed with a procedural approach.
- Since OOP programs are larger in size, the execution time for these programs is also more.

3.4 Conclusion

Both OOP and FP have the shared goal of creating understandable, flexible programs that are free of bugs. But they have two different approaches for how to best create those programs.

In all programs, there are two primary components: the data (the stuff a program knows) and the behaviors (the stuff a program can do to/with that data). OOP says that bringing together data and its associated behavior in a single location (called an "object") makes it easier to understand how a program works. Functional programming says that data and behavior are distinctively different things and should be kept separate for clarity.

In functional programming, data cannot be stored in objects, and it can only be transformed by creating functions. In object-oriented programming, data is stored in objects. Object-oriented programming is widely used by programmers and successful also.

In Object-oriented programming, it is quite hard to maintain objects while increasing the levels of inheritance. In functional programming, it requires a new object to execute functions, and it takes a lot of memory for executing the applications.

Each has their own advantages and disadvantages, it is up to the programmers or developers to choose the programming language concept that makes their development productive and easy.

Bibliography

[1] Wisnu Anggoro. Packt Publishing. *Functional C#*. 2016

[2] C. Thomas Wu. McGraw-Hill Education. *An Introduction to Object-Oriented Programming with Java 5th Edition*. 2009

[3] Bjarne Stroustrup. Addison Wesley. *The C++ Programming Language, 4th Edition*. 2013

[4] Priya Pedamkar. Educba website. *What is OOP?*. Available at: https://www.educba.com/what-is-oop/

[5] hackr.io website. *Functional Programming: Concepts, Advantages, Disadvantages, and Applications*. Available at: https://hackr.io/blog/functional-programming

[6] studytonight website. *Object Oriented Programming in C++*. Available at: https://www.studytonight.com/cpp/cpp-and-oops-concepts.php

[7] Godday Success. Dev.to website. *Functional programming vs object oriented programming*. Available at: https://dev.to/thecodess/functional-programming-vs-object-oriented-programming-2he8

[8] Teach Computer Science website. *Polymorphism*. Available at: https://teachcomputerscience.com/polymorphism/#Advantage_and_Disadvantage_of_polymorphism

[9] Weber State University Website. *A review of OOP*. Available at: https://icarus.cs.weber.edu/~dab/cs1410/textbook/1.Basics/review.html

[10] Great Learning Team. *Polymorphism In C++ and Types of it?*. Available at: https://www.mygreatlearning.com/blog/polymorphism-in-cpp/