OBJECT ORIENTED ANALYSIS DESIGN & PROGRAMMING UNIT-2

P. Balamurugan, Assistant Professor
PG & Research Department of Computer Science
Government Arts College, Coimbatore -641018
Email: spbalamurugan@rediffmail.com

Topics to be Covered….

- OOP Paradigm
- OOP Advantages
- Key concepts of OOP
- Classes & Objects
- I/O statements
- Declarations
- Data types
- Control structures
- Examples

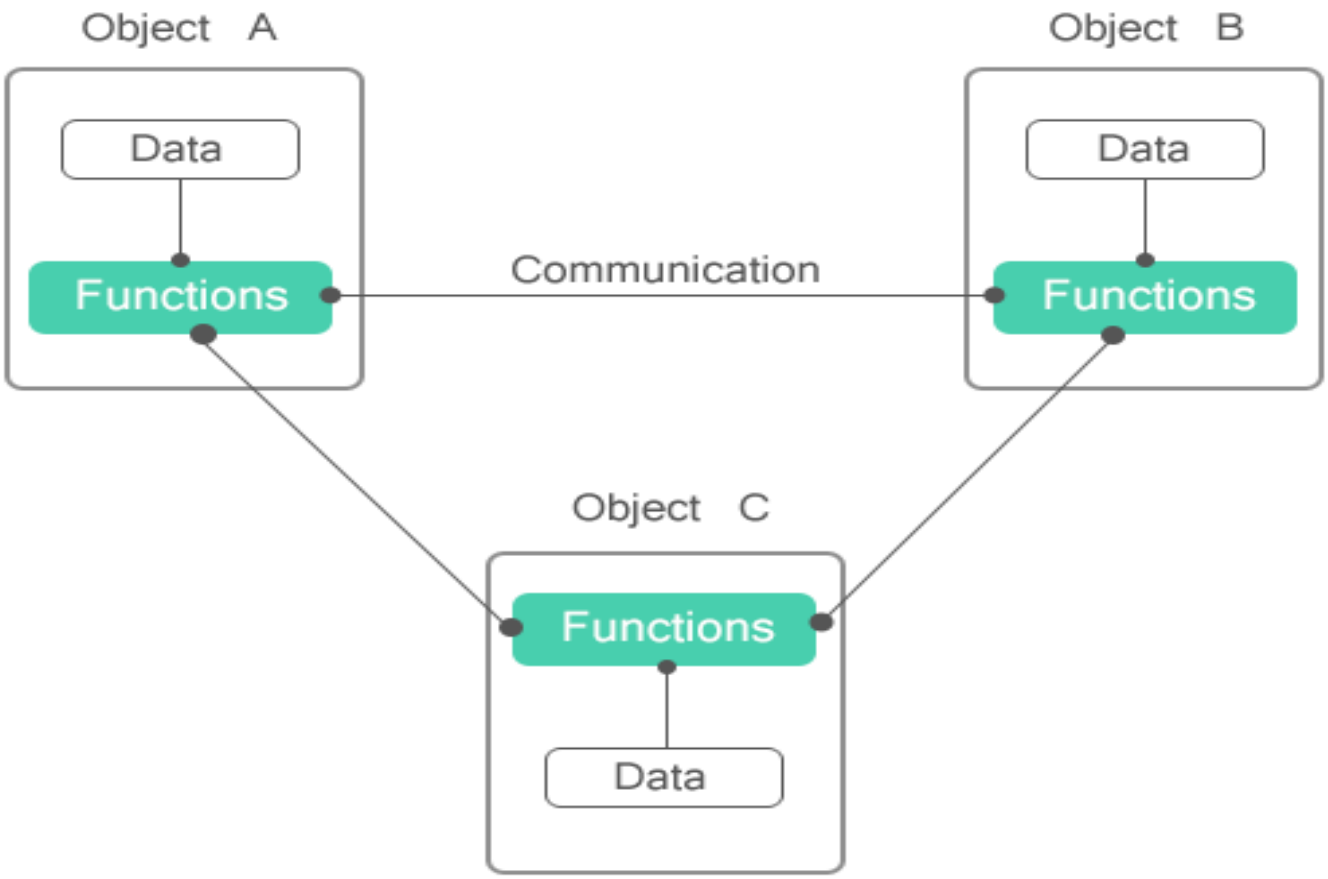
OOP - Paradigm

Object-oriented programming (**OOP**) is a **programming paradigm** based on the concept of "**objects**", which can contain **data** and **code**:

Data in the form of fields (often known as attributes or properties), and

Code, in the form of procedures (often known as methods).

….OOP – Paradigm



….OOP - Paradigm

Object-oriented programming (**OOP**) approach provides solution to some of the flaws encountered in Procedural approach.

OOP treats data as critical element. It is not allowed to flow freely around the program.

It ties closely with functions and protects it from accidental modification.

….OOP - Paradigm

- **Object-oriented programming** (**OOP**) allows to decompose the problem into number of entities called objects.
- Emphasis data rather than procedure.
- Data is hidden not accessed by external functions
- Object communicates with each other through functions.
- Bottom-up approach used in program design

OOP - Advantages

- Faster and Easier to Execute
- Clear structure for the program
- C++ code helps DRY(Don’t Repeat Yourself)
- DRY makes code easier to maintain, modify and debug
- OOP used to create complete reusable code with less code and short time.

OOP - Key Concepts

- Objects
- Class
- Data Abstraction
- Data Encapsulation
- Inheritance
- Polymorphism
- Dynamic binding
- Message Passing

….OOP - Key Concepts

- Class – Template for objects. User-defined data type. Example - CAR
- Objects – Instances of Class. Example – BMW, Audi, Maruthi
- Data Abstraction - Providing only essential information to the outside world and hiding their background details.
- Data Encapsulation - Encapsulation is placing the data and the functions that work on that data in the same place.
- Inheritance - Process of forming a new class from an existing class that is from the existing class called as base class, new class is formed called as derived class.

….OOP - Key Concepts

- Polymorphism - The ability to use an operator or function in different ways. Otherwise giving different meaning or functions to the operators or functions is called polymorphism.
- Dynamic binding – Binding associates the procedure call to the code to be executed. Dynamic means the code to be executed for particular procedure call is not known until the runtime.
- Message Passing – Helps to communicate within object and between objects. Establishing communication among objects.

Classes & Objects

- Class contains attributes and methods.
- Attributes are called member variable.
- Methods are called member functions.
- Attributes and methods are commonly called class members.
- Example :

```
class myClass {       // The class
        public:             // Access specifier
        int myNum;        // Attribute (int variable)
        string myString;  // Attribute (string variable)
};
```

Classes & Objects

- How to create an object 'myObj' of class "MyClass"?
- Example :

```
class myClass {       // The class
    public:           // Access specifier
    int myNum;        // Attribute (int variable)
    string myString;  // Attribute (string variable)
};
int main() {
  MyClass myObj;  // Create an object of MyClass

  // Access attributes and set values
  myObj.myNum = 15;
  myObj.myString = "Some text";
  return 0;
};
```

- May create multiple objects 'myObj1' , 'myObj2' ,etc.,

Classes & Objects

- Class Methods are **functions** that belongs to the class.
- There are two ways to define functions that belongs to a class:
  - Inside class definition
  - Outside class definition
- How to create a method called myMethod in the class myClass?
- Example :

```
class MyClass {              // The class
  public:                    // Access specifier
    void myMethod() {  // Method/function defined inside the class  }
};
int main() {
  MyClass myObj;     // Create an object of MyClass
  myObj.myMethod();  // Call the method
  return 0;
   }
```

I/O Statements

- **'cin'** is a input statement.
- **'cin'** is a keyword that reads data from the keyboard with the extraction operator (>>)
- It is used to get user input.
- Syntax :

  cin >> *n*; // Get user input from the keyboard

- **'cout'** object, together with the insertion operator(<<), is used to display values/print text in output device.
- Syntax :

  cout << "Hello World!";

- Example code segment:

```
int x, y, sum;
cout << "Type a number: ";
cin >> x;
cout << "Type another number: ";
cin >> y;
sum = x + y;
cout << "Sum is: " << sum;
```

Declarations and Data types

- All the variables must be declared before they are used in executable statements.
- Declaration statements introduce the identifiers/variables with some attributes such as size, data type, initial value.
- Syntax :

  datatype v1[,v2,v3,…vn];
- Data types are used to specify the type of value hold by the variable.
- Data types may be a Built-in or User-defined.
- Built-in data types are grouped into two i) Basic data types and ii) Derived data types.
- Basic data types are : int, float, double, boolean, char
- Derived data types are the combination of basic data type and qualifiers.
- Qualifiers are signed, unsigned, short, long
- Example for Derived data types : short int, long int, unsigned short int etc.,

Control Structures

- Decision making statements
  - Simple if - to specify a block of code to be executed, if a specified condition is true
  - if …else - to specify a block of code to be executed if condition is true, otherwise another block of code is executed.
  - else if - to specify a new condition to test, if the first condition is false
  - switch - to specify many alternative blocks of code to be executed
  - Ternary operator(?:) – Short hand if...else statement.

- Loop statements
  - for loop
  - while loop
  - do…while loop

Topics to be covered today(19-11-2020)

- Decision making statement
- Looping statement
- break & continue statement
- Arrays
- Pointers
- Functions
- Function Overloading

Decision Making Statements

- Simple if statement -

Syntax:

```
if (condition) {
  // Code block will be executed if the condition is true
}
```

Example:

```
int x = 20;
int y = 18;
if (x > y) {
  cout << "x is greater than y";
}
```

Decision Making Statements

- if ...else statement

Syntax:

```
if (condition) {
  // True code block
} else {
  // False code block
}
```

Example:

```
int x = 20;
int y = 18;
if (x > y) {
  cout << "x is greater than y";
}  else {
  cout << "y is greater than y ";
}
```

Decision Making Statements

- else if statement

Syntax:

```
if (condition1) {
  // True code block of condition1
} else if (condition2) {
  // True code block of condition2
} else {
  // code block will be executed  if both conditions are false
}
```

Example:

```
int number = 20;
if (number  < 0) {
  cout << "Negative";
} else if (number > 0) {
  cout << "Positive";
}else {
  cout << "Zero";
}
```

Decision Making Statements

- Short hand if...else (Ternary Operator)

  Syntax:

  variable = (*condition*) ? *True expression* : *False expression*;

  Example:

```
int  x = 20, y = 18;
string result = (x>y) ? “x is greater." : “y is greater.";
cout << result;
```

```
int  x = 20, y = 18;
int max = (x>y) ? x : y;
cout << max;
```

Decision Making Statements

- switch statement – Multiple conditions are tested. It is an alternate statement to else if statement.

Syntax:

```
switch(expression) {
  case x:
    // code block-x
    break;
  case y:
    // code block-y
    break;
  .....
  .....
 case n:
    // code block-n
    break;
default:
    // code block
}
```

Decision Making Statements

- switch statement

Example:

```
int day = 4;
switch (day) {
  case 1:
    cout << "Monday";
    break;
  case 2:
    cout << "Tuesday";
    break;
  case 3:
    cout << "Wednesday";
    break;
 case 4:
    cout << "Thursday";
    break;
  case 5:
    cout << "Friday";
    break;
  case 6:
    cout << "Saturday";
    break;
  case 7:
    cout << "Sunday";
    break;
  default :
    cout << "Invalid";
    break;
}
```

Looping Statements

- Repeated execution of one or more statement(s) without retyping
- Loops are handy – save time, reduce errors, more readable.
- Different types of loop statement:
  - for loop
  - while loop
  - Do…while loop
- for loop – use When you know exactly how many times you want to loop through a block of code.

  Syntax :

  for (*statement 1*; *statement 2*; *statement 3*) {
  // code block to be executed
  }

  where, **Statement 1** is executed (one time) before the execution of the code block called initialization.

  Statement 2 is condition statement which defines the condition for executing the code block.

  Statement 3 is executed (every time) after the code block has been executed called controlling statement.

Looping Statements

- **for loop statement**

  Example:

```
for (int i = 0; i < 5; i++) {
 cout << i << "\n";
}
```

- **while loop statement -** loops through a block of code as long as a specified condition is true.

  Syntax :

```
while (condition) {
// code block to be executed
}
```

  Example:

```
int i = 0;
while (i < 5) {
cout << i << "\n";
i++;
}
```

Looping Statements

- **Do..while loop statement :**
  - The do/while loop is a variant of the while loop.
  - This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax :

```
do {
// code block to be executed
}
while (condition);
```

Example:

```
int i = 0;
do {
cout << i << "\n";
i++;
}
while (i < 5);
```

break and continue statements

- The ***break*** statement is used to jump out of a **loop**.
- Terminate the loop.

Example :

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    break;
  }
   cout << i << "\n";
 }
```

- The ***continue*** statement breaks current iteration (in the loop), if a specified condition true, and continues with the next iteration in the loop.

Example :

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
   continue;
  }
   cout << i << "\n";
 }
```

Arrays

- Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
- To declare an array, define the variable type, specify the name of the array followed by square brackets and specify the size.
- Array elements are stored in contiguous memory locations.

Example :

without initial value

```
int  myNum[5]
string myCar[4]
```

with initial value

```
int  myNum[5] = {10,20,30,40,50};
string myCar[4] = {"Volvo", "BMW", "Ford", "Mazda"};
```

- Access the array elements using index number. Starting index is 0.

Arrays

- Loops are used to handle the array easily.
- Array size is optional. It may be omitted in declaration.
- Example :

```
int  myNum[] = {10,20,30,40,50};
string myCar[] = {"Volvo", "BMW", "Ford", "Mazda"};
```

- Address operator(&)
- Example :

string myCar **= “Volvo”**

cout<<myCar<<&myCar

Pointers-Declaration

- References operators (&) and Pointers are important in C++.
- Offer you the ability to manipulate the data in the computer's memory - which can reduce the code and improve the performance.
- A pointer is a variable that stores the memory address as its value
- A pointer variable points to a data type (like int or string) of the same type, and is created with the *(dereference) operator.
- Example :

```
string  myCar = "BMW";  // Variable declaration
string *ptr = &myCar;    // Pointer declaration

// Reference: Output the memory address of myCar with the pointer (0x6dfed4)
cout << ptr << "\n";

// Dereference: Output the value of myCar with the pointer (BMW)
cout << *ptr << "\n";
```

Pointers-Modification

- Changing pointer's value.
- Example :

```
string  myCar = "BMW";  // Variable declaration
string *ptr = &myCar;    // Pointer declaration

// Reference: Output the memory address of myCar with the pointer (0x6dfed4)
cout << ptr << "\n";

// Dereference: Output the value of myCar with the pointer (BMW)
cout << *ptr << "\n";
 // Modify: Output the value of myCar with the pointer Audi)
 *ptr = "Audi";
 cout << *ptr << "\n";
```

Functions

- A function is a block of code which only runs when it is called.
- Function is a self-contained code segment.
- Pass data into function, known as parameters.
- Functions are used to perform certain actions, and they are important for reusing code.
- Define the code once, and use it many times.
- C++ provides some pre-defined functions, such as main(), which is used to execute code.
- Create own functions to perform certain actions.
- Create a function often referred to as *declare*, specify the name of the function, followed by parentheses **()**:

Functions

Syntax:

Return-data type *myFunction*(argument list) {

// code to be executed

}

- Declared functions are not executed automatically/ immediately.
- It is call by called function statement i.e. write the function's name followed by two parentheses () and a semicolon (;).

Example:

```
void myFunction() {
  cout << “Executed Function body ";
}
int main() {
  myFunction(); // call the function
  return 0; }
```

Function with Arguments

- Information can be passed to functions as a parameter.
- Parameters act as variables inside the function.
- Parameters are specified after the function name, inside the parentheses.
- Pass as many parameters as you want, just separate them with a comma:

Syntax:

Return-data type *myFunction*(arg-1, arg-2,...) {
    // code to be executed
}

Example:

```
void myName(string fname) {
  cout<< “My name:  “<< fname<<“\n"; }
int main() {
  myName(“Krishna");
   return 0; }
```

Function with Default Parameters

- Use a default parameter value, by using the equals sign (=).
- **Syntax:**

Return-data type *myFunction*(data-type arg-1= value1, arg-2,...) {
// code to be executed
}

Example:

```
void myName(string fname =“Kannan”) {
 cout<< “My name:  “<< fname<<“\n"; }
int main() {
 myName(“Krishna");
 myName();
 return 0; }
```

Function with Return Value

- Function may return value of valid data type such as int, string, etc.,

Example:

```
int myFunction(int x) {
  return x + 5;
}

int main() {
  cout << myFunction(3);
  return 0;
}
```

Function with Reference

- Also pass a reference to the function.
- Share the common memory by both actual and formal parameters.

Example:

```
void swapNums(int &x, int &y) {
 int z;
 z = x; x = y;   y = z;
   }
int main() {
     int firstNum = 10, secondNum = 20;
     cout << "Before swap: " << "\n";
     cout << firstNum << secondNum << "\n";
     swapNums(firstNum, secondNum);
     cout << "After swap: " << "\n";
     cout << firstNum << secondNum << "\n";
     return 0;
   }
```

Function Overloading

- Multiple functions can have the same name with different parameters.

Example:

```
int myFunction(int x)
float myFunction(float x)
double myFunction(double x, double y)
```

```
int addFunc(int x, int y) {
  return x + y; }
double addFunc(double x, double y) {
  return x + y; }
int main() {
  int myResult1 addFunc(8, 5);
  double myResult2 = addFunc(4.3, 6.26);
  cout << "Int: " << myResult1 << "\n";
  cout << "Double: " << myResult2;
  return 0;}
```

References

- https://www.tutorialspoint.com/object_oriented_analysis_design/ooad_object_oriented_model.htm
- https://www.geeksforgeeks.org/object-oriented-programming-in-cpp
- https://www.javatpoint.com/cpp-oops-concepts

TEXT BOOKS

1) Grady Booch, "Object Oriented Analysis and Design with Applications", Second Edition, Pearson Education. 2. Ashok N. Kamthane, "Object Oriented Programming with ANSI &Turbo C++" , First Indian Print, Pearson Education, 2003.

REFERENCE BOOKS

1) Balagurusamy "Object Oriented Programming with C++", TMCH, Second Edition, 2003.