CHAPTER 2: OBJECT ORIENTED PROGRAMMING CONCEPTS

Concepts associated with OOP

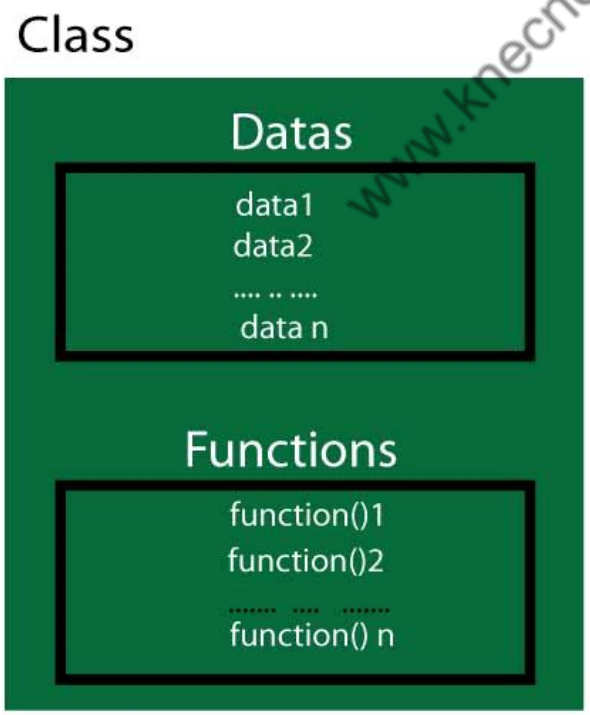
Concepts of OOP:

- ✓ Objects
- ✓ Classes
- ✓ Data Abstraction and Encapsulation
- ✓ Inheritance
- ✓ Polymorphism
- ✓ Overloading
- ✓ Reusability

Objects and Classes

Class is a collection of data member and member function. **Class** is a user **define** data type. **Object** is a **class** type variable. **Objects** are also called instance of the **class**. Each **object** contains all members (variables and functions) declared in the **class**.

A **class** is the collection of related data and function under a single name. A C++ program can have any number of classes. When related data and functions are kept under a class, it helps to visualize the complex problem efficiently and effectively.



A Class is a blueprint for objects

When a class is defined, no memory is allocated. You can imagine like a datatype.
int var;

The above code specifies *var* is a variable of type integer; int is used for specifying variable *var* is of integer type. Similarly, class are also just the specification for objects and object bears the property of that class.

Defining the Class

Class is defined in C++ programming using keyword class followed by identifier(name of class). Body of class is defined inside curly brackets an terminated by semicolon at the end in similar way as structure.

```
class class_name
   {
   // some data
   // some functions
   };
```

Example of Class in C++

```
class temp
   {
      private:
         int data1;
         float data2;
      public:
         void func1()
           {   data1=2;  }
        float func2(){
              data2=3.5;
              retrun data;
           }
   };
```


Explanation

As mentioned, definition of class starts with keyword class followed by name of class(*temp*) in this case. The body of that class is inside the curly brackets and terminated by semicolon at the end. There are two keywords: private and public mentioned inside the body of class.

Keywords: private and public

Keyword private makes data and functions private and keyword public makes data and functions public. Private data and functions are accessible inside that class only whereas, public data and functions are accessible both inside and outside the class. This feature in OOP is known as data hiding. If programmer mistakenly tries to access private data outside the class, compiler shows error which prevents the misuse of data. Generally, data are private and functions are public.

Objects

When class is defined, only specification for the object is defined. Object has same

relationship to class as variable has with the data type. Objects can be defined in similary way as structure is defined.

Syntax to Define Object in C++

class_name variable name;

For the above defined class *temp*, objects for that class can be defined as:

temp obj1,obj2;

Here, two objects(*obj1* and *obj2*) of *temp* class are defined.

Data member and Member functions

The data within the class is known as data member. The function defined within the class is known as member function. These two technical terms are frequently used in explaining OOP. In the above class *temp*, *data1* and *data2* are data members and *func1()* and *func2()* are member functions.

Accessing Data Members and Member functions

Data members and member functions can be accessed in similar way the member of structure is accessed using member operator(.). For the class and object defined above, *func1()* for object *obj2* can be called using code:
obj2.func1();

Similary, the data member can be accessed as:

object_name.data_memeber;

Note: You cannot access the data member of the above class *temp* because both data members are private so it cannot be accessed outside that class.

Example to Explain Working of Object and Class in C++ Programming

```
/* Program to illustrate working of Objects and Class in C++
Programming */
#include <iostream>
using namespace std;
class temp
{
    private:
        int data1;
        float data2;
    public:
       void int_data(int d){
          data1=d;
          cout<<"Number: "<<data1;
         }
       float float_data(){
           cout<<"\nEnter data: ";
           cin>>data2;
```

```
            return data2;
          }
};
 int main(){
      temp obj1, obj2;
      obj1.int_data(12);
      cout<<"You entered "<<obj2.float_data();
      return 0;
 }
```

Output:

```
Number: 12
Enter data: 12.43
You entered: 12.43
```

Explanation of Program

In this program, two data members *data1* and *data2* and two member function int_data() and float_data() are defined under *temp* class. Two objects *obj1* and *obj2* of that class are declared. Function int_data() for the obj1 is executed using code obj1.int_data(12);, which sets 12 to the *data1* of object *obj1*. Then, function float_data() for the object *obj2* is executed which takes data from user; stores it in *data2* of *obj2* and returns it to the calling function.

Note: In this program, *data2* for object *obj1* and *data1* for object *obj2* is not used and contains garbage value.

obj1		obj2	
data1	data2	data1	data2
12	Random value	Random value	12.43

Defining Member Function Outside the Class

A large program may contain many member functions. For the clarity of the code, member functions can be defined outside the class. To do so, member function should be declared inside the class(function prototype should be inside the class). Then, the function definition can be defined using scope resolution operator **::**. Learn more about defining member function outside the class.

Abstraction and Encapsulation

Abstraction allows us to represent complex real world in simplest manner. It is process of identifying the relevant qualities and behaviors an object should possess, in other word represent the necessary feature without representing the back ground details.

Encapsulation It is a process of hiding all the internal details of an object from the outside real world. The word Encapsulation, like Enclosing into the capsule. It restrict

client from seeing its internal view where behavior of the abstraction is implemented

Inheritance and polymorphism

Inheritance is when an object or class is based on another object or class, using the same implementation (inheriting from a class) specifying implementation to maintain the same behavior (realizing an interface; inheriting behavior).

Polymorphism is a *concept* wherein a name may denote instances of many different classes as long as they are related by some common superclass.

Comparison between structured and OOP

Keywords and identifiers

Keywords:

Keywords are the reserved words used in programming. Each keywords has fixed meaning and that cannot be changed by user. For example:

int *money*;

Here, int is a keyword that indicates, *'money'* is of type integer.

As, C++ programming is case sensitive, all keywords must be written in lowercase. Here is the list of all keywords predefined by C++.

Keywords in C++ Language			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
continue	for	signed	void
do	if	static	while
default	goto	sizeof	volatile
const	float	short	unsigned

Identifiers

In C++ programming, identifiers are names given to C++ entities, such as variables, functions, structures etc. Identifier are created to give unique name to C++ entities to identify it during the execution of program. For example:

```
int money;
```

```
int mango_tree;
```

Here, *money* is a identifier which denotes a variable of type integer. Similarly, *mango_tree* is another identifier, which denotes another variable of type integer.

Rules for writing identifier

1) An identifier can be composed of letters (both uppercase and lowercase letters), digits and underscore '_' only.

2) The first letter of identifier should be either a letter or an underscore. But, it is discouraged to start an identifier name with an underscore though it is legal. It is because, identifier that starts with underscore can conflict with system names. In such cases, compiler will complain about it. Some system names that start with underscore are *_fileno*, *_iob*, *_wfopen* etc.

3) There is no rule for the length of an identifier. However, the first 31 characters of an identifier are discriminated by the compiler. So, the first 31 letters of two identifiers in a program should be different.

Literals and constants

A literal is some data that's presented directly in the code, rather than indirectly through a variable or function call.

Here are some examples, one per line:

```
42
128
3.1415
'a'
"hello world"
```

A value written exactly as it's meant to be interpreted. In contrast, a variable is a name that can represent different values during the execution of the **program**. And a constant is a name that represents the same value throughout a **program**. But a **literal** is not a name -- it is the value itself

A constant is an identifier with an associated value which cannot be altered by the program during normal execution – the value is **constant**. This is contrasted with a variable, which is an identifier with a value that can be changed during normal execution – the value is variable.

Comments and Punctuators

Comments are portions of the code ignored by the compiler which allow the user to make simple notes in the relevant areas of the source code. Comments come either in block form or as single lines.

- **Single-line comments** (informally, *C++ style*), start with // and continue until the end of the line. If the last character in a comment line is a \ the comment will continue in the next line.

- **Multi-line comments** (informally, *C style*), start with /* and end with */.

Punctuators: Punctuators are used to group or separate the part of code. These helps to demarcate the program structure
In the above code, {, } and ; are the punctuators.

- Braces are used to group the multiple statements into a separate block.
- Semicolon is used to terminate the statement.

Operators: It combines the expressions or transforms them.
Ex: a + b.
'+' is called as an operator which combines a and b and performs addition.
In the above example code, ., (), + and = are the operators.
There are several kinds of operators are there in **C++** which does different operations based on the operands(literals).

Reasons for embracing OOP

- Code Reuse and Recycling: Objects created for Object Oriented Programs can easily be reused in other programs.
- Encapsulation (part 1): Once an Object is created, knowledge of its implementation is not necessary for its use.