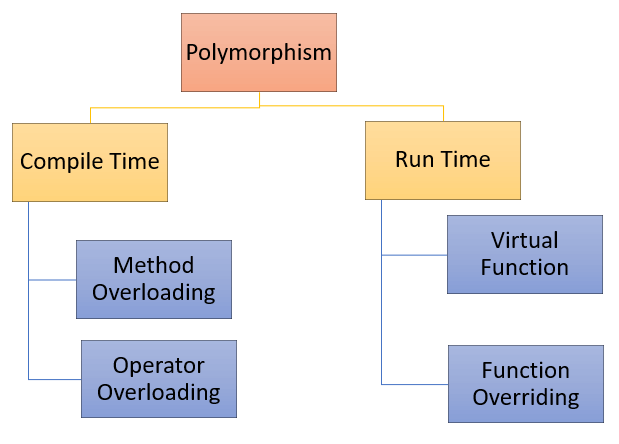
**What is Polymorphism**

**The word “poly” means many and “morphs” means forms, So, the word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.**

**Example: A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, and an employee. So, the same person possesses different behaviour in different situations. This is called polymorphism.**

****

**Types of polymorphism**

* **Compile-time Polymorphism**
* **Runtime Polymorphism**

**Compile-time Polymorphism**

Compile time polymorphism is also known as early binding or static polymorphism. It is performed during compilation time and hence the name compile-time polymorphism.  It is achieved by function overloading and  operator overloading.

**Compile time polymorphism have**

* **Function Overloading**
* **Operator Overloading**

**Function Overloading**

Function overloading is a feature of object-oriented programming where two or more functions can have the same name but different arguments or return type. Function overloading takes place when you create more than one functions of the same name and these functions serve different purposes.

If we have to perform only one operation and having same name of the functions increases the readability of the program.

**Rules of Function Overloading**

* The functions must have the same name.
* The functions must have different types of parameters.
* The functions must have a different set of parameters.
* The functions must have a different sequence of parameters.

**Syntax**

**return\_type funcion\_name (data\_type\_1 variable1, data\_type\_2 variable2, .......) {**

**//statements**

**}**

**C++ code of function overloading**

**#include<iostream>**

**using namespace std;**

**// Function to add two integers**

**int add(int a, int b) {**

**return a + b;**

**}**

**// Function to add two doubles**

**double add(double a, double b) {**

**return a + b;**

**}**

**int main() {**

**// Calls int add(int a, int b)**

**int addition\_int\_values = add(5, 3);**

**// Calls double add(double a, double b)**

**double addition\_double\_values = add(2.5, 3.7);**

**cout << "Addition of int values: " << addition\_int\_values << endl;**

**cout << "Addition of double values: " << addition\_double\_values << endl;**

**return 0;**

**}**

**Operator Overloading**

Operator overloading is a compile-time polymorphism in which the operator is overloaded to provide the special meaning to the user-defined data type. Operator overloading is used to overload or redefines most of the operators available in C++. It is used to perform the operation on the user-defined data type.

The advantage of Operators overloading is to perform different operations on the same operand.

**Operator that cannot be overloaded are as follows:**

* Scope operator (::)
* Sizeof operator(sizeof)
* member selector(.)
* member pointer selector(\*)
* ternary operator(?:)

**Syntax**

**return\_type operator symbol()**

**{**

**//statements**

**}**

**Rules for Operator Overloading**

* Existing operators can only be overloaded, but the new operators cannot be overloaded.
* The overloaded operator contains atleast one operand of the user-defined data type.
* We cannot use friend function to overload certain operators. However, the member function can be used to overload those operators.
* When unary operators are overloaded through a member function take no explicit arguments, but, if they are overloaded by a friend function, takes one argument.

When binary operators are overloaded through a member function takes one explicit argument, and if they are overloaded through a friend function takes two explicit arguments.

**Types of operator overloading**

* **Unary operator overloading**
* **Binary operator overloading**

**Unary Operator Overloading**

The unary operators operate on a single operand and following are the examples of Unary operators −

* [The increment (++) and decrement (--) operators](https://www.tutorialspoint.com/cplusplus/increment_decrement_operators_overloading.htm).
* The unary minus (-) operator.
* The logical not (!) operator.

**Syntax**

**class className {**

**...**

**public:**

**returnType operator<symbol>()**

**{**

**// custom behaviour for the operator**

**}**

**...**

**};**

**C++ code of unary operator overloading with class function**

**#include <iostream>**

**using namespace std;**

**class Negation {**

**public:**

**int x, y;**

**Negation(int a, int b)**

**{**

**this->x = a;**

**this->y = b;**

**}**

**//overload - operator**

**void operator-()**

**{**

**x=-x;**

**y=-y;**

**}**

**};**

**int main()**

**{**

**Negation d1(8, 9);**

**cout << "\nx : " << d1.x << " , y : " << d1.y;**

**-d1;**

**cout << "\nx : " << d1.x << " , y : " << d1.y;**

**return 0;**

**}**

**C++ code of unary operator overloading with friend function**

**#include <iostream>**

**using namespace std;**

**class Negation {**

**public:**

**int x, y;**

**Negation(int a, int b)**

**{**

**this->x = a;**

**this->y = b;**

**}**

**friend void operator-(Negation &N);**

**};**

**void operator-(Negation &N)**

**{**

**N.x=-N.x;**

**N.y=-N.y;**

**}**

**int main()**

**{**

**Negation d1(8, 9);**

**cout << "\nx : " << d1.x << " , y : " << d1.y;**

**-d1;**

**cout << "\nx : " << d1.x << " , y : " << d1.y;**

**return 0;**

**}**

**Difference Between Compile Time and Run Time Polymorphism**

| **Compile Time Polymorphism** | **Runtime Polymorphism** |
| --- | --- |
| * It is also known as static polymorphism, early binding, or overloading | * It is also known as dynamic polymorphism, late binding, or overriding |
| * It is executed at the compile time | * It is executed at the run time |
| * The method is called with the help of a compiler | * The method is not called with the help of a compiler |
| * The program’s execution is fast as it involves the use of a compiler | * The program’s execution is slow as it does not involve a compiler |
| * It is achieved by function overloading and operator overloading | * It is achieved by virtual functions and function overriding |
| * Compile-time polymorphism tends to be less flexible as all commands are operated at the compile time | * Run time polymorphism tends to be more flexible as all commands are executed at the run time |