Compile time polymorphism

Overview

Compile time polymorphism is also known as static polymorphism. This type of polymorphism can be achieved through function overloading or operator overloading.

a) Function overloading: When there are multiple functions in a class with the same name but different parameters, these functions are overloaded. The main advantage of function overloading is it increases the readability of the program. Functions can be overloaded by using different numbers of arguments and by using different types of arguments.

i) Function overloading with different numbers of arguments:

In this example, we have created two functions, the first add() performs the addition of the two numbers, and the second add() performs the addition of the three numbers. Let's look at the example:

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

// Function with two parameters
int add(int num1, int num2) {
    return num1 + num2;
}

// Function with three parameters
int add(int num1, int num2, int num3) {
    return num1 + num2 + num3;
}

int main() {

    cout << add(10, 20) << endl;
    cout << add(10, 20, 30);
```

```
    return 0;
}

Output:
30
60
```

ii) Function overloading with different types of arguments:

In this example, we have created two add() functions with different data types. The first add() takes two integer arguments and the second add() takes two double arguments.

```
#include <iostream>

using namespace std;

// Function with two integer parameters
int add(int num1, int num2) {
    return num1 + num2;
}

// Function with two double parameters
double add(double num1, double num2) {
    return num1 + num2;
}

int main(void) {

    cout << add(10, 20) << endl;
    cout << add(10.4, 20.5);
    return 0;
}

Output:
30
30.9
```

Default Arguments: A default argument is a value provided in a function declaration automatically assigned by the compiler if the function's caller doesn't provide a value for

the argument with a default value. However, if arguments are passed while calling the function, the default arguments are ignored.

Example: A function with default arguments can be called with 2 or 3 or 4 arguments.

```
#include<iostream>

using namespace std;

int add(int x, int y, int z = 0, int w = 0) {
    return (x + y + z + w);
}

int main() {
    cout << add(10, 20) << endl;
    cout << add(10, 20, 30) << endl;
    cout << add(10, 20, 30, 40) << endl;
    return 0;
}

Output:
30
60
100
```

b) Operator Overloading: C++ also provides options to overload operators. For example, we can make the operator ('+') for the string class to concatenate two strings. We know that this is the addition operator whose task is to add two operands. A single operator, '+,' when placed between integer operands, adds them and concatenates them when placed between string operands.

Points to remember while overloading an operator:

- It can be used only for user-defined operators(objects, structures) but cannot be used for in-built operators(int, char, float, etc.).

- Operators = and & are already overloaded in C++, so we can avoid overloading them.
- Precedence and associativity of operators remain intact.

List of operators that can be overloaded in C++:

+	-	*	/	%	^
&	|	~	!	,	=
<	>	<=	>=	++	--
<<	>>	==	!=	&&	||
+=	-=	/=	%=	^=	&=
|=	*=	<<=	>>=	[]	()
->	->*	new	new []	delete	delete []

List of operators that cannot be overloaded in C++:

::	.*	.	?:

Example: Perform the addition of two imaginary or complex numbers.

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

class Complex {
    private:
        int real, imag;
    public:
        Complex(int r = 0, int i = 0) {
            real = r;
            imag = i;
        }
```

```
    // This is automatically called when '+' is used with
    // between two Complex objects
    Complex operator + (Complex
        const & b) {
        Complex a;
        a.real = real + b.real;
        a.imag = imag + b.imag;
        return a;
    }
    void print() {
        cout << real << " + i" << imag << endl;

    }
};

int main() {
    Complex c1(10, 5), c2(2, 4);
    Complex c3 = c1 + c2; // An example call to "operator+"
    c3.print();
}
```

Output:

```
12 + i9
```