Function Overloading

If any class have multiple functions with same names but different parameters then they are said to be overloaded.

Function overloading allows you to use the same name for different functions, to perform, either same or different functions in the same class.

Ways to overload a function:

1. By changing number of Arguments.
2. By having different types of argument.

By Changing Number of Arguments:

```
int sum(int a,int b)
{
return a + b;
}

int sum(int a, int b, int c)
{
return a + b + c;
}
```

By having different types of argument:

```
int sum(int a,int b)
{
   return a+b;
}

float sum(double x, int y)
{
   return x+y;
}
```

Operator Overloading

Operator overloading is a compile-time polymorphism.
As the name indicated the operator is overloaded to provide the special meaning to the user-defined data type.

For example: Overloading the Unary operator ++

```
#include <iostream>
using namespace std;
class Een
{
private:
int n;
public:
Een(): n(4){}
```

```
void operator ++() {
n = n+2;
}
void Print() {
cout<<n;
}
};
int main()
{
Een E;
++E;  // calling of a function “void operator ++()”
E.Print();
return 0;
}
```

Related posts:

1. Introduction to Compiler
2. Analysis and synthesis model of compilation
3. Bootstrapping and Porting
4. Lexical Analyzer: Input Buffering
5. Storage Allocation Strategies
6. Type Checking
7. Specification & Recognition of Tokens
8. Front end and back end of the compiler
9. LEX

10. Analysis synthesis model of compilation
11. Data structure in CD
12. Register allocation and assignment
13. Loops in flow graphs
14. Dead code elimination
15. Syntax analysis CFGs
16. L-attribute definition
17. Operator precedence parsing
18. Analysis of syntax directed definition
19. Recursive descent parser
20. Storage allocation strategies
21. Equivalence of expression in type checking
22. Storage organization
23. Parameter passing
24. Run time environment
25. Type checking
26. Code generation issue in design of code generator
27. Boolean expression
28. Declaration and assignment in intermediate code generation
29. Code optimization
30. Sources of optimization of basic blocks
31. Loop optimization
32. Global data flow analysis
33. Data flow analysis of structure flow graph (SFG)