Function & Operator Overloading

Function Overloading

- Defining multiple functions with same name but different arguments is called as **Function Overloading**
- Function Overloading allow programmer to use same name for different functions, to perform same or different functionalities in same scope.
- **Return type is ignored in Function Overloading, Why?**
- Function Overloading is used to enhance readability of program.

Ways to Overload function

By changing Number of arguments

```
#include<iostream>
using namespace std;
int sum(int a, int b)
{return a+b;}

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

```
int main(){
//call sum with 2 args
cout<<"\n"<<sum(10,20);
//call sum with 3 args
out<<"\n"<<sum(10,20,30);
return 0;
}
```

Ways to Overload function

Different data type of arguments

```
#include<iostream>
using namespace std;
int sum(int a, int b)
{return a+b;
}
double sum(double a, double b)
{return a+b;
}
```

```
int main(){
//call sum with 2 args
out<<"\n"<<sum(10,20);
//call sum for double args
cout<<"\n"<<sum(10.5,20.7);
return 0;
}
```

Name Mangling/ Name Decoration

- Even if we have same names for multiple functions in function overloading compiler sets unique identifier to overloaded function.
- **The process of generating unique identifiers by compiler is called as Name Mangling.**
- Name mangling is complier dependant process, there is no standard for identifier generation.

Mangled Names : sum@2@i@i, sum@3@i@i@i

Constructor Overloading

As constructor is also a function it can be overloaded

```
 #include<iostream>
using namespace std;
class Complex{
 private:
 int i,j;
public:
Complex():i(0),j(0){
     cout<<"\n 0 args Ctor"; }
Complex(int p):i(p),j(p){
     cout<<"\n 1 args Ctor";}
 Complex(int p,int k){
 i=p; j=k;
 cout<<"\n 2 args
Ctor";
}
};
int main(){
Complex c1;
Complex c2(5);
Complex c3(10,20);
 return 0;}
```

Default Arguments

- The argument for which default value is provided is called as default argument.
- Once we provide default value to argument that arguments become optional.
- The argument can be made default argument only if it is right most argument or all the arguments on RHS of that argument are also default arguments

Default Arguments Example

```
 #include<iostream>
using namespace std;
int sum( int a, int b, int c = 0) {
 return a+b+c;
}
int sum( int a, int b =0, int c )
{
 return a+b+c;
} // This type of default arg
is not permitted, WHY?
```

```
int main(){
 cout<<"\n"<<sum(10,20,30);
 cout<<"\n"<<sum(10,20);
// Default arg becomes optional
 return 0;
}
```

Constructor with Default Arguments

```
 #include<iostream>
using namespace std;
class Complex{
 private:
 int i,j;
public:
Complex(int p =0, int
k=0):i(p),j(k)
     {
     cout<<"\n Ctor called";
     }
};
```

```
int main(){
Complex c1;
Complex c2(5);
Complex c3(10,20);
 return 0;
}
//Only one ctor can be
used for 0,1 and 2 args
using default args
```

Placeholder Arguments

- The arguments in a function are declared without identifier or name are placeholders

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

- Placeholder arguments are some times used in Operator Overloading

Need of Operator Overloading

- **All built in operator works with built in data types like int, float and double etc. but we can use these operators on User Defined Data types.**

```
int main()
{ int a(10),
 int b(10);
cout<<a+b;
return 0;
}// As a and b are of int type + operator know how to add them
```

Need of Operator Overloading

Operators doesn’t understand User Defined Data types

```
class Complex{
 private:
 int i,j;
public:
Complex(int p =0, int k=0):i(p),j(k)
{}
};
```

```
int main(){
Complex c1(10,10),c2(5,5);
Complex c3;
c3 = c1+ c2;  //error
 return 0;
}
```

//+ operator does not know how to add two complex numbers hence we need to tell it through overloading.

Rules for Operator Overloading

- Only existing operators can be overloaded
- We can not change Precedence and Associativity of operator
- We can not invent new operators using operator overloading
- We can change meaning of operator for our data type only
- We can overload operator as member function and global function.
- **Operators are internally functions, hence they can be overloaded.**
- **sizeof(), :: , .* ,?:** etc operators can not be overloaded.

Operator Overloading Example

1. Complex class
2. Array class

Assignment: Create class Time with hour, minutes and seconds as data members and overload +, - operators for it

Thank You

Overload me if you can!!!!!

……….Operators