C++ Function Overloading

In C++ programming, two functions can have same identifier(name) if either number of arguments or type of arguments passed to functions are different. These types of functions having similar name are called overloaded functions.

```
/* Example of function overloading */

int test() { }
int test(int a){ }
int test(double a){ }
int test(int a, double b){ }
```

All 4 functions mentioned above are overloaded function. It should be noticed that, the return type of all 4 functions is same,i.e, `int`. **Overloaded function may or may not have different return type but it should have different argument**(either type of argument or numbers of argument passed). Two functions shown below are not overloaded functions because they only have same number of arguments and arguments in both functions are of type `int`.

```
/* Both functions has same number of argument and same type of argument*/
/* Hence, functions mentioned below are not overloaded functions. */
/* Compiler shows error in this case. */

int test(int a){ }
double test(int b){ }
```

Example 1: Function Overloading

```
/*Calling overloaded function test() with different argument/s.*/

#include <iostream>
using namespace std;
void test(int);
void test(float);
void test(int, float);
int main() {
    int a = 5;
    float b = 5.5;

    test(a);
    test(b);
    test(a, b);

    return 0;
}

void test(int var) {
    cout<<"Integer number: "<<var<<endl;
}

void test(float var){
    cout<<"Float number: "<<var<<endl;
```

```
}

void test(int var1, float var2) {
    cout<<"Integer number: "<<var1;
    cout<<" And float number:"<<var2;
}
```

Output

```
Integer number: 5
Float number: 5.5
Integer number: 5 And float number: 5.5
```

In above example, function `test()` is called with integer argument at first. Then, function `test()` is called with floating point argument and finally it is called using two arguments of type `int` and `float`. Although the return type of all these functions is same, that is, `void`, it's not mandatory to have same return type for all overloaded functions. This can be demonstrated by example below.

Example 2: Function Overloading

```
/*  C++ Program to return absolute value of variable types
    integer and float using function overloading       */

#include <iostream>
using namespace std;

int absolute(int);
float absolute(float);
int main() {
    int a = -5;
    float b = 5.5;

    cout<<"Absolute value of "<<a<<" = "<<absolute(a)<<endl;
    cout<<"Absolute value of "<<b<<" = "<<absolute(b);
    return 0;
}

int absolute(int var) {
     if (var < 0)
         var = -var;
    return var;
}

float absolute(float var){
    if (var < 0.0)
        var = -var;
    return var;
}
```

Output

```
Absolute value of -5 = 5
```

```
Absolute value of 5.5 = 5.5
```

In above example, two functions `absolute()` are overloaded. Both take single argument but one takes integer type argument and other takes floating point type argument. Function `absolute()` calculates the absolute value of argument passed and returns it.

Example 3: Function Overloading

```
1:   // Listing 5.8 - demonstrira
2:   // funkcijski polimorfizam
3:
4:   #include <iostream.h>
5:
6:   int Double(int);
7:   long Double(long);
8:   float Double(float);
9:   double Double(double);
10:
11:  int main()
12:  {
13:      int      myInt = 6500;
14:      long     myLong = 65000;
15:      float    myFloat = 6.5F;
16:      double   myDouble = 6.5e20;
17:
18:      int      doubledInt;
19:      long     doubledLong;
20:      float    doubledFloat;
21:      double   doubledDouble;
22:
23:      cout << "myInt: " << myInt << "\n";
24:      cout << "myLong: " << myLong << "\n";
25:      cout << "myFloat: " << myFloat << "\n";
26:      cout << "myDouble: " << myDouble << "\n";
27:
28:      doubledInt = Double(myInt);
29:      doubledLong = Double(myLong);
30:      doubledFloat = Double(myFloat);
31:      doubledDouble = Double(myDouble);
32:
33:      cout << "doubledInt: " << doubledInt << "\n";
34:      cout << "doubledLong: " << doubledLong << "\n";
35:      cout << "doubledFloat: " << doubledFloat << "\n";
36:      cout << "doubledDouble: " << doubledDouble << "\n";
37:
38:       return 0;
39:  }
40:
41:  int Double(int original)
42:  {
43:    cout << "U Double(int)\n";
44:    return 2 * original;
45:  }
46:
47:  long Double(long original)
48:  {
49:    cout << "U Double(long)\n";
50:    return 2 * original;
51:  }
52:
53:  float Double(float original)
54:  {
55:    cout << "U Double(float)\n";
56:    return 2 * original;
57:  }
58:
59:  double Double(double original)
60:  {
61:    cout << "U Double(double)\n";
62:    return 2 * original;
63:  }
```

IZLAZ

```
myInt: 6500
myLong: 65000
myFloat: 6.5
myDouble: 6.5e+20
U Double(int)
U Double(long)
U Double(float)
U Double(double)
DoubledInt: 13000
DoubledLong: 130000
DoubledFloat: 13
DoubledDouble: 1.3e+21
```

```
#include <iostream>
using namespace std;
void RadianNeShkalle(double alfa, double &shkalle)
{
   shkalle=alfa/3.1415*180.;
}
void RadianNeShkalle(double alfa, int &shkalle, int &minuta, int &sekonda)
{
   double ndihmese;
   ndihmese=alfa/3.1415*180.;  //mundet edhe: RadianNeShkalle(alfa,ndihmese);
   shkalle=int(ndihmese);
   ndihmese=ndihmese-shkalle;
   ndihmese=ndihmese*60;
   minuta=int(ndihmese);
   ndihmese=ndihmese-minuta;
   sekonda=int(ndihmese*60);
}

int main()
{
// x - ndryshorja per shkalle; y - ndryshorja per minuta; z - ndryshorja per sekonda
      int x,y,z;
// alfa - ndryshorja per kendin ne RADIAN, s- ndryshorja per kendin ne SHKALLE (nr. Real);
      double alfa,s;
      cout << "Lexo kendin ne RADIAN alfa= "; cin >> alfa;
// Funksioni “kthen” vleren e kendit ne shkalle si numer real
       cout<<"Kendin e futur ne RADIAN, ne SHKALLE si numer real ka vlere:"<<endl;
        RadianNeShkalle(alfa,s);
// Funksioni "kthen" shkalle, minuta dhe sekonda si numra te plote
      RadianNeShkalle(alfa,x,y,z);
      cout<<"Shkalle: "<<s<<endl;
      cout<<" gjegjesisht:"<<endl;
      cout<<"Shkalle: "<<x<< ", Minuta: "<<y<< ", Sekonda: "<<z<<endl;
      return 0;
}
```