Overloading (1A)

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice.

Function Overloading (1)

C <math.h>	C++ <cmath>
int **abs** (int n);	int **abs** (int n);
long int **labs** (long int n);	long int **abs** (long int n);
double **f**abs (double x);	double **f**abs (double x);

the same
function name

different function
prototypes

Function Overloading (2)

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

int sum (int x, int y, int z) {
  return x+y+z;
}

int sum (int x, int y, int z, int w) {
  return x+y+z+w;
}
```

```
s1 = sum(10, 20);

s2 = sum(10, 20, 30);

s3 = sum(10, 20, 30, 40);
```

the same function name

different function prototypes

the compiler determines which function is called

Constructor Functions

```
class Ccircle {
public:
  int r;

  Ccircle ()       { r = 1; }
  Ccircle (int x) { r = x; }

  void    setR (int x) { r = x; }
  int     getR ()       { return r; }
  double area () ;
}
```

the constructor function name:
the same as the **class name**

no return type; not even void

automatically called whenever a new object of this class is created

used for initialization purpose

```
void main(void)  {

  Ccircle C1;
  Ccircle C2(10);


}
```

The **default constructor** is without any parameter.

the **default constructor** must be declared in addition to any other constructors defined

Overloaded Constructor Functions

```
class Ccircle {
public:
  int r;

  Ccircle ()       { r = 1; }
  Ccircle (int x)  { r = x; }

  void    setR (int x) { r = x; }
  int     getR ()      { return r; }
  double area () ;
}
```

the same function name

different function prototypes

Operator Functions

```
class Ccircle {
public:
  int r;

  Ccircle ()       { r = 1; }
  Ccircle (int x) { r = x; }

  void    setR (int x) { r = x; }
  int     getR ()       { return r; }
  double area () ;
}
```

```
void main(void)  {

  Ccircle C1(10), C2(30), C3;

  C3 = C1 + C2;

}
```

```
C1 + C2;
```

implicit call of the function **operator+**

```
C1.operator+(C2);
```

explici call of the function **operator+**

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

Overloaded Operator Functions (1)

```
class Ccircle {
public:
  int r;

  Ccircle ()       { r = 1; }
  Ccircle (int x) { r = x; }

  void    setR (int x) { r = x; }
  int     getR ()      { return r; }
  double area () ;

  Ccircle operator+(Ccircle);

}
```

```
void main(void)  {

  Ccircle C1(10), C2(30), C3;

  C3 = C1 + C2;

}
```

int 10 + 30; int

overloaded

Ccircle C1 + C2; Ccircle

```
Ccircle Ccircle::operator+ (Ccircle y) {
      Ccircle z;
      z.r = r + y.r;
       return z;
}
```

Overloaded Operator Functions (2)

```
Ccircle Ccircle::operator+ (Ccircle y) {
      Ccircle z;
      z.r = r + y.r;
      return z;
}
```

```
const Ccircle Ccircle::operator+ (const Ccircle& y)
{
     Ccircle z;
     z.r = r + y.r;
     return z;
}
```

```
Ccircle& Ccircle::operator= (const Ccircle& y) {
     r =  y.r;
     return *this;
}
```

Return l-values

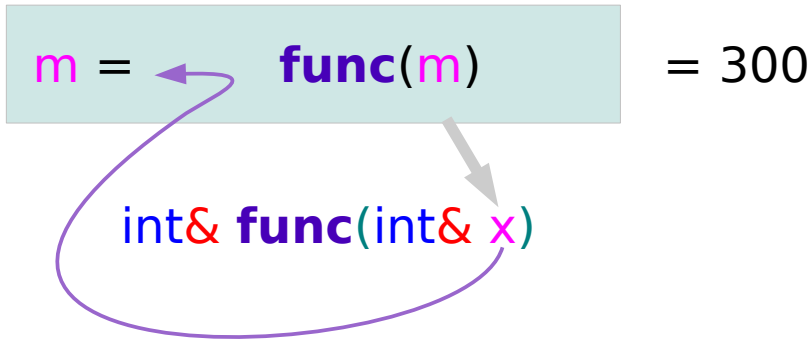
```
int& func(int& x)
{
   return x;
}
```

m*2 : invalid initialization

```
int m =10;
int n =20;

n = func(m);

func(m) = 300;
```



Function's Return Value Types (1)

```
T          func(    );
const T    func(    );
T&         func(    );
const T&   func(    );
```

```
T          func(    );
const T    func(    );  : often problematic
```

copy constructor is called
when returning objects

```
CC CO(10);
CC CO2(CO);
```

the same class' object :
the *only* parameter
as a *reference* variable
with the ***const*** specifier

```
const T    func(    ); : more preferred
const T&   func(    ); : often problematic
```

used to return private members
used not **const** for returning a string

const returning is meaningless when
simple data type such as int or char
(3 or 'A' : already constant)

```
func().mutator()
```

```
T&         func(    );
const T&   func(    );  : often problematic
```

returning as a *l-value*
do not return local variables

```
cout << a << b << endl;
(cout << a) << b << endl;
(cout << b) << endl;
cout << endl;

func(m) =300;
```

Function's Return Value Types (2)

		l-value	f().mutator()	copy constructor
T	func( );	X	OK	OK
const T	func( );	X	X	OK
T&	func( );	OK	OK	X
const T&	func( );	OK	X	X

References

[1] W Savitch, “Absolute C++”
[2] P.S. Wang, “Standard C++ with objected-oriented programming”
[3] http://www.cplusplus.com