Functions (1A)

Young Won Lim
8/22/13

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 **fabs** (double x);	double **fabs** (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

Friend Functions

```
class Ccircle {
  int r;   // private member

public:

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

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

}
```

```
double peri(Ccircle c)
{
      // r: private member
      return 3.14*r*r ;
}
```

```
friend double peri(Ccircle c);
```

anywhere (public or private members)

Static Functions – internal linkage

```
// a.cpp

static void func( );


// func is visible only in this file (a.cpp)
// internal linkage
```

Use namespace ...

```
namespace {

      void func ( );

}
```

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 Member 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

```
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;
}
```

Virtual Member Functions

```
#include <stdio.h>

class Poly {
public:
   virtual void func()
  { printf("Poly::func() is called... \n"); }
};

class Rect : public Poly {
public:
   void func()
   { printf("Rect::func() is called... \n"); }
};

class Circle : public Poly {
public:
   void func()
  { printf("Circle::func() is called... \n"); }
};
```

```
int main(void) {

  Poly PolyObj, *PolyPointer;
  Rect RectObj, *RectPointer;
  Circle CircleObj, *CirclePointer;

  PolyPointer = &PolyObj;
  PolyPointer->func();              Poly::func()

  PolyPointer = &RectObj;
  PolyPointer->func();              Rect::func()

  PolyPointer = &CircleObj;
  PolyPointer->func();              Circle::func()

}
```

*Without the **virtual** keyword,*
Poly::func is called 3 times.

Pure Virtual Member Functions

```
#include <stdio.h>

class Poly {
public:
   virtual void func() = 0;

};

class Rect : public Poly {
public:
   void func()
   { printf("Rect::func() is called... \n"); }
};

class Circle : public Poly {
public:
   void func()
  { printf("Circle::func() is called... \n"); }
};
```

```
int main(void) {

  Poly ~~PolyObj~~, *PolyPointer;
  Rect RectObj, *RectPointer;
  Circle CircleObj, *CirclePointer;

  ~~PolyPointer = &PolyObj;~~
  ~~PolyPointer->func();~~

  PolyPointer = &RectObj;
  PolyPointer->func();                Rect::func()

  PolyPointer = &CircleObj;
  PolyPointer->func();                Circle::func()

}
```

Classes containing pure virtual functions are termed "**abstract**";
they cannot be instantiated directly.

Static Member Functions

```
class Ccircle {
  int r;   // private member

public:

  static void func ( );
}
```

static method

can call a static member
function within the class

```
void main(void)  {

  Ccircle C1;
  Ccircle C2(10);

  C1.func();                // OK
  C2.func();                // OK

  Ccircle::func();          // OK

}
```

no **this** pointer :
a static function cannot
have non-static members

Static Member Functions – Example

```
#include <stdio.h>

class CRect {
public:
  int r;
  static int s;

  // CRect () { s = 0; }

  static void func() {
        printf("static s=%d\n", s++);
  }
};

int CRect::s = 0;

int main(void) {
  CRect Cobj;
  // int CRect::s = 0;

  CRect::func();
  CRect::func();

  CRect.func();

  return 0;
}
```

constructor cannot initialize static members

Initialization int is needed

not working

References

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