STL Function Objects (Functors)

Function is actually an object of a template class that has a single member function: the overloaded ( ) operator.

Function objects (also called functors) are an STL feature that you may not employ immediately when you start using the STL. They are, however, very useful in many situations and an STL facility with which you should become acquainted. They give the STL a flexibility that it would not otherwise have, and also contribute to STL efficiency. The most common uses for function objects are for generating data, for testing data, and for applying operations to data.

A function object (or functor) is simply any object of a class that provides at least one definition for operator() What this means is that if you then declare an object f of the class in which this operator() is defined you can subsequently use that object f just like you would use an "ordinary" function. For example, you could have an assignment statement like

someValue = f(arg1, arg2);

which is the same as

someValue = f.operator()(arg1, arg2);

so long as operator() for the given class had been defined to do the following two things:

It must take two parameters (par1 and par2, say, of whatever type(s)), with arg1 and arg2 in the above assignment statement being actual parameters of the corresponding type(s). It must return a value of the type of the variable someValue

If you were only going to use a functor as illustrated in the assignment statement example or the sample program given above, you might as well use an ordinary function. In the STL, algorithms often take a parameter which is a function (or functor) telling the algorithm how to perform some part of its task, and functors are generally more versatile and hence more useful for this purpose. In particular, the built-in functors are objects of template classes, and so the same computations performed by a given functor can be applied to different types.

Arithmetic binary functors

`plus<T> f;`

`f(arg1, arg2)` returns the value `arg1 + arg2`.

`minus<T> f;`

`f(arg1, arg2)` returns the value `arg1 - arg2`.

`multiplies<T> f;`

`f(arg1, arg2)` returns the value `arg1 * arg2`.

`divides<T> f;`

`f(arg1, arg2)` returns the value `arg1 / arg2`.

`modulus<T> f;`

`f(arg1, arg2)` returns the value `arg1 % arg2`.

Relational binary functors

`equal_to<T> f;`

`f(arg1, arg2)` returns the value `arg1 == arg2`.

`not_equal_to<T> f;`

`f(arg1, arg2)` returns the value `arg1 != arg2`.

`greater<T> f;`

`f(arg1, arg2)` returns the value `arg1 > arg2`.

`greater_equal<T> f;`

`f(arg1, arg2)` returns the value `arg1 >= arg2`.

`less<T> f;`

`f(arg1, arg2)` returns the value `arg1 < arg2`.

`less_equal<T> f;`

`f(arg1, arg2)` returns the value `arg1 <= arg2`.

Logical binary functors

`logical_and<T> f;`

`f(arg1, arg2)` returns the value `arg1 && arg2`.

`logical_or<T> f;`

`f(arg1, arg2)` returns the value `arg1 || arg2`.

```
// sortemp.cpp
// sorts array of doubles in backwards order,
// uses greater<>() function object
#include <iostream>
#include <algorithm>                  //for sort()
#include <functional>                 //for greater<>
using namespace std;
```

```
// array of doubles
double fdata[] = { 19.2, 87.4, 33.6, 55.0, 11.5, 42.2 };

int main()
{                                  // sort the doubles
       sort(fdata, fdata + 6, greater<double>());

       for (int j = 0; j<6; j++)             // display sorted doubles
              cout << fdata[j] << ' ';
       cout << endl;
       return 0;
}
```

87.4 55 42.2 33.6 19.2 11.5

Writing your own function objects

```
// sorts person objects stored by pointer
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
using namespace std;

class person
   {
   private:
      string lastName;
      string firstName;
      long phoneNumber;
   public:
      person() :              // default constructor
           lastName("blank"), firstName("blank"), phoneNumber(0L)
         {  }
                              // 3-arg constructor
      person(string lana, string fina, long pho) :
              lastName(lana), firstName(fina), phoneNumber(pho)
         {  }
      friend bool operator<(const person&, const person&);
      friend bool operator==(const person&, const person&);

      void display() const   // display person's data
         {
         cout << endl << lastName << ",\t" << firstName
              << "\t\tPhone: " << phoneNumber;
         }
      long get_phone() const // return phone number
         { return phoneNumber; }
   }; //end class person
//-----------------------------------------------------------
// overloaded < for person class
bool operator<(const person& p1, const person& p2)
   {
   if(p1.lastName == p2.lastName)
      return (p1.firstName < p2.firstName) ? true : false;
   return (p1.lastName < p2.lastName) ? true : false;
```

```
   }
//-----------------------------------------------------------
// overloaded == for person class
bool operator==(const person& p1, const person& p2)
   {
   return (p1.lastName == p2.lastName &&
           p1.firstName == p2.firstName ) ? true : false;
   }
//-----------------------------------------------------------
```

```
// function object to compare persons using pointers
class comparePersons
   {
   public:
   bool operator() (const person* ptrP1,
                    const person* ptrP2) const
      { return *ptrP1 < *ptrP2; }
   };
//-----------------------------------------------------------
//function object to display a person, using a pointer
class displayPerson
   {
   public:
   void operator() (const person* ptrP) const
      { ptrP->display(); }
   };
////////////////////////////////////////////////////////////////
int main()
   {                          // a vector of ptrs to persons
   vector<person*> vectPtrsPers;
                              //make persons
   person* ptrP1 = new person("KuangThu", "Bruce", 4157300);
   person* ptrP2 = new person("Deauville", "William", 8435150);
   person* ptrP3 = new person("Wellington", "John", 9207404);
   person* ptrP4 = new person("Bartoski", "Peter", 6946473);
   person* ptrP5 = new person("Fredericks", "Roger", 7049982);
   person* ptrP6 = new person("McDonald", "Stacey", 7764987);

   vectPtrsPers.push_back(ptrP1);  //put persons in set
   vectPtrsPers.push_back(ptrP2);
   vectPtrsPers.push_back(ptrP3);
   vectPtrsPers.push_back(ptrP4);
   vectPtrsPers.push_back(ptrP5);
   vectPtrsPers.push_back(ptrP6);

   for_each(vectPtrsPers.begin(),              //display vector
            vectPtrsPers.end(), displayPerson() );
                                               //sort pointers
   sort( vectPtrsPers.begin(), vectPtrsPers.end() );
```

```
   cout << "\n\nSorted pointers";
   for_each(vectPtrsPers.begin(),               //display vector
            vectPtrsPers.end(), displayPerson() );

   sort( vectPtrsPers.begin(),                  //sort persons
         vectPtrsPers.end(), comparePersons() );
   cout << "\n\nSorted persons";
   for_each(vectPtrsPers.begin(),               //display vector
            vectPtrsPers.end(), displayPerson() );
   while( !vectPtrsPers.empty() )
      {
      delete vectPtrsPers.back();               //delete person
      vectPtrsPers.pop_back();                  //pop pointer
      }
   cout << endl;
   return 0;
   }  // end main()
KuangThu,       Bruce           Phone: 4157300
Deauville,      William         Phone: 8435150
Wellington,     John            Phone: 9207404
Bartoski,       Peter           Phone: 6946473
Fredericks,     Roger           Phone: 7049982
McDonald,       Stacey          Phone: 7764987

Sorted pointers
KuangThu,       Bruce           Phone: 4157300
Deauville,      William         Phone: 8435150
Wellington,     John            Phone: 9207404
Bartoski,       Peter           Phone: 6946473
Fredericks,     Roger           Phone: 7049982
McDonald,       Stacey          Phone: 7764987

Sorted persons
Bartoski,       Peter           Phone: 6946473
Deauville,      William         Phone: 8435150
Fredericks,     Roger           Phone: 7049982
KuangThu,       Bruce           Phone: 4157300
McDonald,       Stacey          Phone: 7764987
Wellington,     John            Phone: 9207404
```

```
//plusair.cpp
//uses accumulate() algorithm and plus() function object
#include <iostream>
#include <list>
#include <numeric>        //for accumulate()
using namespace std;
////////////////////////////////////////////////////////////
class airtime
  {
  private:
    int hours;          //0 to 23
    int minutes;        //0 to 59
```

```
  public:
                    //default constructor
    airtime() : hours(0), minutes(0)
      {  }
                    //2-arg constructor
    airtime(int h, int m) : hours(h), minutes(m)
      {  }
    void display() const  //output to screen
      { cout << hours << ':' << minutes; }

    void get()          //input from user
      {
      char dummy;
      cout << "\nEnter airtime (format 12:59): ";
      cin >> hours >> dummy >> minutes;
      }
                    //overloaded + operator
    airtime operator + (const airtime right) const
      {              //add members
      int temph = hours + right.hours;
      int tempm = minutes + right.minutes;
      if(tempm >= 60)    //check for carry
        { temph++; tempm -= 60; }
      return airtime(temph, tempm); //return sum
      }
                    //overloaded == operator
    bool operator == (const airtime& at2) const
      { return (hours == at2.hours) &&
            (minutes == at2.minutes); }
                    //overloaded < operator
    bool operator < (const airtime& at2) const
      { return (hours < at2.hours) ||
            (hours == at2.hours && minutes < at2.minutes); }
                    //overloaded != operator
    bool operator != (const airtime& at2) const
      { return !(*this==at2); }
                    //overloaded > operator
    bool operator > (const airtime& at2) const
      { return !(*this<at2) && !(*this==at2); }
  };  //end class airtime
////////////////////////////////////////////////////////////////
int main()
  {
  char answer;
```

```
airtime temp, sum;
list<airtime> airlist;   //list of airtimes

do {                  //get airtimes from user
  temp.get();
  airlist.push_back(temp);
  cout << "Enter another (y/n)? ";
  cin >> answer;
  } while (answer != 'n');
                    //sum all the airtimes
sum = accumulate( airlist.begin(), airlist.end(),
              airtime(0, 0), plus<airtime>() );
cout << "\nsum = ";
sum.display();          //display sum
cout << endl;
return 0;
}
```