Copyright Warning

ViSLAB
High-Performance Visualization, Communication, Computing

Object Oriented Design

Week 12-2

Function Pointer

- Week 6, tutorial
- replaced return address with a function pointer

```
void foo(int a, int b) {
    int c = a + b;
    cout << "Sum is :" << c << endl;
    dumpStack32(&c, 10);
    unsigned long *ulong_ptr = (unsigned long *)&a;
    printf(" %08x", *(ulong_ptr-1));
    org_ptr = (unsigned long *)*(ulong_ptr-1);
    printf(" %08x", (unsigned int)(&main));
    *(ulong_ptr-1) = (long unsigned int)&boo;
}

void boo(){
    int c = 5;
    cout << endl << "Boo!" << endl;

    unsigned long *ulong_ptr = (unsigned long *)&c;
    printf(" %08x", *(ulong_ptr+2));
    printf(" org = %08x", org_ptr);
    *(ulong_ptr+2) = (long unsigned int)org_ptr;
    dumpStack32(&c, 10);
}
```

Function Object

```
class Func {
public:
   int operator() (int a, int b) const {
      return a + b;
   }
}
```

- works like a function, but it's not a function
- has a function "operator()"

```
Func f;
...
cout << f(5, 53) << endl;
```

Function Object

```
f(5, 53)

f.operator()(5, 53)
```

`operator()` : is the name of the function.

you can override `operator()` with different args.

```
    bool strSizeComp(const string& a, const string& b) {
    return a.size() < b.size();
}

int main() {
    string s;
    vector<string> data;
    while (getline(cin, s))
        data.push_back(s);

    sort(data.begin(), data.end(), strSizeComp);
    vector<string>::iterator it;
    for ( it = data.begin(); it != data.end(); ++it)
        std::cout << *it << std::end;
     return 0;
}
```

```
    class StrSizeComp {
public:
    bool operator() (const string& a, const string& b) const {
        return a.size() < b.size();
    }
}

int main() {
    string s;
    vector<string> data;
    while (getline(cin, s))
        data.push_back(s);
    StrSizeComp f;
    sort(data.begin(), data.end(), f);
    vector<string>::iterator it;
    for ( it = data.begin(); it != data.end(); ++it)
        std::cout << *it << std::end;
    return 0;
}

    StrSizeComp f;
    sort(data.begin(), data.end(), f);
```

or

```
    sort(data.begin(), data.end(), StrSizeComp());
```

function object ...

- more work to do?
- function pointer is simpler?

- F.O.
  - instantiate
  - destruct
  - combine

find_if

- find : search elements .... a value specified
  - 3rd arg: value
- find_if : <3, >=5.... condition(s) can be used
  - 3rd arg: evaluation function object.

find_if

```
#include <iostream>
#include <list>
#include <algorithm>
#include <functional>

using namespace std;

int main() {
    list<int> l;

    l.push_back(5);
    l.push_back(3);
    l.push_back(2);
    l.push_back(1);
    l.push_back(4);

    list<int>::iterator it;
    for (it = l.begin(); it != l.end();) {
        it = find_if(it, l.end(), bind2nd(greater<int>(), 3));
        if (it != l.end()) {
            cout << "found! " << *it << endl;
            ++it;
        }
    }
```

- 1st and 2nd args : specify range of search
- `bind2nd(greater<int>(), 3)`
  - `std::greater<int>()` : function object (>)
  - `<int>` : template
  - `()` : calling a constructor
  - `bind2nd` : combined F.O(>) and 3 ....3>

```
    #include <iostream>
#include <functional>

using namespace std;

int main() {
    greater<int> fobj;
    cout << "result = " << fobj(2, 3) << endl;
    return 0;
}
```

- `greater<int>` .... local instance “fobj”,
- fobj : use “>” like a function
- “>” function is implemented as `operator()`

```
#include <iostream>
#include <functional>

using namespace std;

int main() {
    less<double> fobj;
    cout << "result = " << fobj(2.0, 3.0) << endl;
    return 0;
}
```

- plus (+), minus(-), multiplies(*), divides(/), modulus(%)
- negate (-), equal_to(==), not_equal_to(!=)
- greater(>), less(<), greater_equal(>=), less_equal(<=)
- logical_and(&&), logical_or(||), logical_not(!)

equation -> object

- combine function objects
  - represent an equation
- 1 + 4 * 5
  - plus(1, multiplies(4, 5))

- `bind2nd(greater<int>(), 3)`
  - `std::greater<int>()` : function object (>)
- `bind2nd` : combines binary_function and const
  - creates unary_function

create unary_function

- subclass “std::unary_function”
- define “operator()”

```
class myFuncObj : public std::unary_function<int, bool> {
public:
    bool operator() (int right) {
        return (right > 3) ? true : false;
    }
}
```

  - template <int, bool>:
    - arg data type
    - return data type

```
    class myFuncObj : public std::unary_function<int, bool> {
public:
    bool operator() (int right) {
        return (right > 3) ? true : false;
    }
}

int main() {
    std::list<int> l;
    l.push_back(5);
    l.push_back(3);
    l.push_back(2);
    l.push_back(1);
    l.push_back(4);

    myFuncObj fo;
    std::list<int>::iterator it;
    for (it = l.begin(); it != l.end();) {
        it = std::find_if(it, l.end(), fo);
        if (it != l.end()) {
            cout << "found! " << *it << endl;
            ++it;
        }
    }
    return 0;
}
```

inside of find_if ?

```
iterator find_if(iterator begin, iterator end, Predicate fobj) {
for(iterator it = begin; it != end; ++it)
    if (fobj(*it))
        return it;
    return end;
}
```

bind2nd

- binary_function ➛ unary_function
- `bind2nd(`FunctionObject, `Constant)` generates
  - FunctionObject `bind2nd`
    - operator():

```
bool operator()(int right) {
   if (functionObject(right, const))
      return ture;
   return false;
}
```

bind2nd...1st?

binder	result of bind	eq.
std::bind1st(std::greater<int>(), 3)	greater(3, var)	3 > var
std::bind2nd(std::greater<int>(), 3)	greater(var, 3)	var > 3