

# C++

## Function pointers, Functors

1

# Introduction

- Algorithms in the STL like: **find()**, **count()**, etc execute some action based on the value of a passed parameter
- For example: **find(v.begin(),v.end(),5)** will return the first iterator **it** in the vector **v** such that **\*it == 5**
- These functions can be extended to work based on the result of a function passed as argument rather than by using an exact value
- Passing a function to another function can be done using:
  - **Function pointers**
  - **Function objects (or functors)**

2

## Function pointers

- Function pointers are inherited from C
- A function pointer is an address in memory to an executable code
- Example from the C library: **qsort** (implementation of quick sort)
  - **qsort(void\* base, size\_t nmemb, size\_t size, int (\*compare) (const void\*, const void\*))**;
  - 4<sup>th</sup> argument is of type **int (\*) (const void\*, const void\*)** and is a function pointer

3

## Example

```
#include <iostream>
using namespace std;

int min (int a, int b) {
    return a < b ? a : b;
}
int max (int a, int b) {
    return a > b ? a : b;
}

int main() {
    bool maxQ = true;
    int (*f) (int a, int b);
    if (maxQ) f = min;
    else f = max;
    cout << f(1, 3) << endl;
}
```

f is the name of a var of type function pointer. It points to a fun that take two ints and return an int

Assign the address of either **min** or **max** to f. 4

## STL example: find\_if

- **find\_if()** is an extension of **find()**:
  - **find()** returns the first iterator (in a range) matching an exact value (passed as arg). I.e. **\*it == value**, where **it** is the iterator.
  - **find\_if()** returns the first iterator (in a range) matching a predicate (passed as arg). I.e. **pred(\*it)** is true, where **it** is the iterator and **pred()** the predicate

Declaration of find\_if():

```
template <class InputIterator, class Predicate>
find_if(InputIterator first, InputIterator last, Predicate pred);
```

5

## STL example: find\_if

```
bool lt_5 (int a) { return a < 5; }

int main () {
    vector<int> v;
    v.push_back(11); v.push_back(9);
    v.push_back(7); v.push_back(5);
    v.push_back(2); v.push_back(8);

    vector<int>::iterator res;
    res = find_if(v.begin(), v.end(), lt_5);

    if (res != v.end()) cout << *res << endl;
}
```

Define a function "lt\_5"

Returns the first iterator (it) in [v.begin(), v.end()) such that lt\_5(\*it) == true (i.e. \*it < 5)

Output: 2

6

## STL example: sort()

- The **sort()** function in the STL is overloaded:
  - It sorts in ascending order a container between the range [first, last) (assuming the comparison predicate is "<"). See 1 below.
  - Or we can specify our comparison predicate as an argument to the function **sort()**. See 2 below.

(1)  
template <class Iterator>  
void sort(Iterator first, Iterator last);

(2)  
template <class Iterator, class Predicate>  
void sort(Iterator first, Iterator last, Predicate comp);

7

## STL example: sort()

```
class Student {
public:
    Student(int i, string n) : id(i), name(n) {}
    int id;
    string name;
};

bool lt_id (const Student& s1, const Student& s2)
{ return s1.id < s2.id; }
bool lt_name (const Student& s1, const Student& s2)
{ return s1.name < s2.name; }

void print(const Student& s) {
    cout << s.id << " " << s.name << endl;
}
```

8

## STL example: sort()

for\_each applies the 3<sup>rd</sup> argument (print()) to each element in the range [v.begin(), v.end())

```
int main(){
    Student s1(1111, "suzuki"), s2(5211, "nakata");
    Student s3(4312, "tanaka"), s4(1, "saito");
    vector<Student> v;
    v.push_back(s1); v.push_back(s2);
    v.push_back(s3); v.push_back(s4);
    sort(v.begin(), v.end(), lt_id);
    for_each(v.begin(), v.end(), print);
    sort(v.begin(), v.end(), lt_name);
    for_each(v.begin(), v.end(), print);
}
```

9

## Function object: introduction

- **Function object** or **functor** is a generalization of the concept of function pointer
- Any object overloading **operator()** can be "called" as if it were a function
- Example:

```
class PlusA {
public:
    int _a;
    PlusA () : _a(1) {}
    PlusA (int a) : _a(a) {}
    int operator() (int b) { return _a + b; }
};
```

```
// ...
PlusA plus_a;
plus_a(5);
```

10

## Function object

- In the previous example:
  - PlusA is a class
  - PlusA plus\_a creates an object; it is constructed by the default constructor which sets the internal parameter \_a to 1
  - Operator() has been overloaded so plus\_a(value) is a function that returns value + \_a (=1)

11

## Function object

- As seen in the previous example, function objects (like any objects) have a state
- In the previous example, the state is defined by the member data **\_a** (=1 in the example)
- Having a state is an advantage over function pointers

12

## Example of function objects

```
#include <iostream>
using namespace std;

class GT {
private:
    int _a;
public:
    GT(int a) : _a(a) {}
    bool operator() (int b) {
        return b > _a;
    }
};
```

```
int main() {
    GT gt_5(5), gt_10(10);

    if (gt_5(7)) cout << "7 > 5" << endl;
    if (gt_10(7)) cout << "7 > 10" << endl;
}
```

13

## STL example: find\_if()

- Rewrite the example using find\_if() with function objects

```
class LT {
private:
    int _a;
public:
    LT(int a) : _a(a) {}
    bool operator() (int b) { return b < _a; }
};
```

```
int main () {
    vector<int> v;
    v.push_back(11); v.push_back(9);
    v.push_back(7); v.push_back(5);
    v.push_back(2); v.push_back(8);

    vector<int>::iterator res;
    LT lt_5(5);
    res = find_if(v.begin(), v.end(), lt_5);

    if (res != v.end()) cout << *res << endl;
}
```

14

## STL example

- **find\_if** examines each element of the container (by using an iterator **it**)
- For each element, it will check the predicate, by calling the functor **lt\_5** (3<sup>rd</sup> argument) with argument **\*it**:
  - `lt_5(*it); // lt_5.operator()(*it);`

15

## STL example: for\_each

```
template <class Iterator, class UnaryFunction>
UnaryFunction for_each(Iterator first, Iterator last, UnaryFunction f);
```

Apply the function object **f** to each element in the range `[first, last)`.  
**F**'s return value if any is ignored.  
`for_each()` returns the function object after it has been applied to each element.

16

## STL example: for\_each

```
class Student {
public:
    Student(int i, string n) : id(i), name(n) {}
    int id;
    string name;
};

ostream& operator<< (ostream& os,
const Student& s) {
    os << s.id << " " << s.name << endl;
    return os;
}
```

```
template <class T>
class Print {
public:
    int _count;
    Print() : _count(0) {}
    void operator() (const T& s) {
        cout << s << endl;
        _count++;
    }
};
```

17

## STL example: for\_each

```
int main(){
    Student s1(1111, "suzuki");
    Student s2(5211, "nakata");
    Student s3(4312, "tanaka");
    Student s4(1, "saito");
    vector<Student> v;
    v.push_back(s1); v.push_back(s2);
    v.push_back(s3); v.push_back(s4);
    Print<Student> P = for_each(v.begin(), v.end(), Print<Student>());
}
```

Returns the fun object after  
it has been applied to each  
element

Create a temp object  
Print<Student> by calling the  
default constructor

18

## Summary

- An object that can be called like a function is a function object (or functor)
- Function objects are a generalization of function pointers
- They can have data members and internal state like any objects
- A function object must have the method **operator()** overloaded so that it can be called

19