Module 40: Programming in C++

Functors: Function Objects

Intructors: Abir Das and Sourangshu Bhattacharya

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

{*abir, sourangshu*}*@cse.iitkgp.ac.in*

Slides taken from NPTEL course on Programming in Modern C++

by **Prof. Partha Pratim Das**

Module Objectives

- Understand the Function Objects or Functor
- Study the utility of functor in design, especially in STL

Module Outline

1. Function Pointers
   - Callback
     - `qsort`
   - Issues
2. Functors in C++
   - Basic Functor
   - Simple Example
   - Examples from STL
     - Function Pointer

Function Pointers

- *Points to the address of a function*
  - Ordinary C functions
  - Static C++ member functions
  - Non-static C++ member functions
- *Points to a function with a specific signature*
  - List of Calling Parameter Types
  - Return-Type
  - Calling Convention

Function Pointers in C

- *Define a Function Pointer*

```
int (*pt2Function) (int, char, char);
```

- *Calling Convention*

```
int DoIt (int a, char b, char c);
int DoIt (int a, char b, char c) {
    printf ("DoIt\n");
    return a+b+c;
}
```

- *Assign Address to a Function Pointer*

```
pt2Function = &DoIt; // OR
pt2Function = DoIt;
```

- *Call the Function pointed by the Function Pointer*

```
int result = (*pt2Function) (12, 'a', 'b');
```

- *Compare Function Pointers*

```
if (pt2Function == &DoIt) {
    printf ("pointer points to DoIt\n");
}
```

Function Pointers in C

Direct Function Pointer

```
#include <stdio.h>

int (*pt2Function) (int, char, char);
int DoIt (int a, char b, char c);

int main() {
    pt2Function = DoIt; // &DoIt

    int result = (*pt2Function)(12, 'a', 'b');

    printf("%d", result);

    return 0;
}
int DoIt (int a, char b, char c) {
    printf ("DoIt\n");

    return a + b + c;
}
```

```
DoIt
207
```

Using `typedef`

```
#include <stdio.h>

typedef int (*pt2Function) (int, char, char);
int DoIt (int a, char b, char c);

int main() {
    pt2Function f = &DoIt; // DoIt

    int result = f(12, 'a', 'b');

    printf("%d", result);

    return 0;
}
int DoIt (int a, char b, char c) {
    printf ("DoIt\n");

    return a + b + c;
}
```

```
DoIt
207
```

Function Reference In C++

- *Define a Function Pointer*

```
int (A::*pt2Member)(float, char, char);
```

- *Calling Convention*

```
class A {
int DoIt (float a, char b, char c) {
    cout << "A::DoIt" << endl; return a+b+c; }
};
```

- *Assign Address to a Function Pointer*

```
pt2Member = &A::DoIt;
```

- *Call the Function pointed by the Function Pointer*

```
A instance1;
int result = (instance1.*pt2Member)(12, 'a', 'b');
```

- *Compare Function Pointers*

```
if (pt2Member == &A::DoIt) {
    cout <<"pointer points to A::DoIt" << endl;
}
```

Function Pointer: Operations and Programming Techniques

- **Operations**
  - *Assign* an Address to a Function Pointer
  - *Compare* two Function Pointers
  - *Call* a Function using a Function Pointer
  - *Pass* a Function Pointer as an Argument
  - *Return* a Function Pointer
  - *Arrays* of Function Pointers
- **Programming Techniques**
  - *Replacing switch/if-statements*
  - *Realizing user-defined late-binding*, or
    - Functions in Dynamically Loaded Libraries
    - Virtual Functions
  - *Implementing callbacks*

Function Pointers: Replace Switch/ IF Statements


```
#include <iostream>
using namespace std;
// The four arithmetic operations
float Plus(float a, float b){ return a+b; }
float Minus(float a, float b){ return a-b; }
float Multiply(float a, float b){ return a*b; }
float Divide(float a, float b){ return a/b; }
int main(){
  int ch, a, b;
  cout << "Enter 0 for add, 1 for sub, 2 for mult and 3 for div: ";
  cin >> ch;
  cout << "Enter 2 numbers: ";
  cin >> a >> b;
  switch(ch){
    case 0: cout << Plus(a, b) << endl; break;
    case 1: cout << Minus(a, b) << endl; break;
    case 2: cout << Multiply(a, b) << endl; break;
    case 3: cout << Divide(a, b) << endl; break;
    case 4: cout << "Enter valid choice" << endl;
  }
  return 0;
}
```

Function Pointers: Replace Switch/ IF Statements


```
#include <iostream>
using namespace std;
// The four arithmetic operations
float Plus(float a, float b){ return a+b ; }
float Minus(float a, float b){ return a-b ; }
float Multiply(float a, float b){ return a*b; }
float Divide(float a, float b){ return a/b; }
int main(){
  float (*OpPtr[4])(float, float) = {Plus, Minus, Multiply, Divide};
  int ch, a, b;
  cout << "Enter 0 for add, 1 for sub, 2 for mult and 3 for div: ";
  cin >> ch;
  cout << "Enter 2 numbers: ";
  cin >> a >> b;
  cout << (*OpPtr[ch])(a, b) << endl;
  return 0;
}
```

Example: Callback, Function Pointers

- It is a Common C Feature

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

void A(){
  cout << "Hello" << endl;
}
// Function pointer as argument
void B(void (*fptr)()){
  // Calling back function that fptr points to
  fptr();
}
int main(){
  void (*fp)() = A;
  B(fp); // Or simply B(A)
  return 0;
}
```

Function Pointers: Callback: qsort to Quick Sort


```
void qsort(void *base,    // Pointer to the first element of the array to be sorted
           size_t nitems, // Number of elements in the array pointed by base
           size_t size,   // Size in bytes of each element in the array
           int (*compar)(const void *, const void*)); // Function that compares two elements

int CmpFunc(const void* a, const void* b) { // Compare function for int
    int ret = (*(const int*)a > *(const int*) b)? 1:
                  (*(const int*)a == *(const int*) b)? 0: -1;
    return ret;
}

int main() {
    int field[10];

    for(int c = 10; c>0; c--)
        field[10-c] = c;

    qsort((void*) field, 10, sizeof(field[0]), CmpFunc);
}
```

Function Pointers: Issues

- No value semantics
- Weak type checking
- Two function pointers having identical signature are necessarily indistinguishable
- No encapsulation for parameters

Functors or Function Objects

- Smart Functions
  - Functors are *functions with a state*
  - Functors *encapsulate C / C++ function pointers*
    - Uses templates and
    - Engages polymorphism
- Has its own *Type*
  - A class with zero or more private members to store the state and an overloaded `operator()` to execute the function
- Usually *faster* than ordinary Functions
- Can be used to implement *callbacks*
- Provides the basis for *Command Design Pattern*

Basic Functor

- Any class that overloads the function call operator:
  - `void operator()();`
  - `int operator()(int, int);`
  - `double operator()(int, double);`
  - `...`

Functors: Simple Example

- Consider the code below

```
int AdderFunction(int a, int b) { // A function
    return a + b;
}

class AdderFunctor {
public:
    int operator()(int a, int b) { // A functor
        return a + b;
    }
};

int main() {
    int x = 5;
    int y = 7;
    int z = AdderFunction(x, y);  // Function invocation

    AdderFunctor aF;
    int w = aF(x, y);             // aF.operator()(x, y); -- Functor invocation
}
```

Functors: Examples from STL: Function Pointer for Functor

- **Fill a vector with random numbers**
  - **generate** algorithm

```
#include <algorithm>
template <class ForwardIterator, class Generator>
    void generate(ForwardIterator first, ForwardIterator last, Generator gen) {
        while (first != last) {
            *first = gen();
            ++first;
        }
    }
```

    - ▷ **first, last**: Iterators are defined for a range in the sequence. "[" or "]" means include the element and "(" or ")" means exclude the element. `ForwardIterator` has a range `[first,last)` spanning from `first` element to the element before the `last`
    - ▷ **gen**: Generator function that is called with no arguments and returns some value of a type convertible to those pointed by the iterators
    - ▷ This can either be a function pointer or a function object
  - Function Pointer **rand** as Function Object

```
#include <cstdlib>

// int rand (void);

vector<int> V(100);
generate(V.begin(), V.end(), rand);
```

Functors: Examples from STL

- **Sort a vector of UDTs**

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;
class Student{
public:
    float CGPA;
    string RollNo;
  Student(string RN, float CG):RollNo(RN), CGPA(CG){};
};

int main(){
    vector<Student> students{Student("18CS10065",8.5),
                             Student("19CS30008",8.3),
                             Student("17CS10024",8.9)};
    sort(students.begin(),students.end());
    return 0;
}
```

Functors: Examples from STL

- **Sort a vector of UDTs**

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;
class Student{
public:
    float CGPA;
    string RollNo;
  Student(string RN, float CG):RollNo(RN), CGPA(CG){};
};

int main(){
    vector<Student> students{Student("18CS10065",8.5),
                             Student("19CS30008",8.3),
                             Student("17CS10024",8.9)};
    sort(students.begin(),students.end());
    return 0;
}
```

- **Compilation error!**

CPPReference about sort

std::sort

Defined in header <algorithm>

```
template< class RandomIt >
void sort( RandomIt first, RandomIt last );                                   (1)  (until C++20)
template< class RandomIt >
constexpr void sort( RandomIt first, RandomIt last );                              (since C++20)

template< class ExecutionPolicy, class RandomIt >
void sort( ExecutionPolicy&& policy,                                          (2)  (since C++17)
           RandomIt first, RandomIt last );

template< class RandomIt, class Compare >
void sort( RandomIt first, RandomIt last, Compare comp );                     (3)  (until C++20)
template< class RandomIt, class Compare >
constexpr void sort( RandomIt first, RandomIt last, Compare comp );                (since C++20)

template< class ExecutionPolicy, class RandomIt, class Compare >
void sort( ExecutionPolicy&& policy,                                          (4)  (since C++17)
           RandomIt first, RandomIt last, Compare comp );
```

Sorts the elements in the range [`first` , `last`) in non-descending order. The order of equal elements is not guaranteed to be preserved.

1) Elements are compared using operator<.

3) Elements are compared using the given binary comparison function `comp` .

Parameters

first, last - the range of elements to sort

policy - the execution policy to use. See execution policy for details.

comp - comparison function object (i.e. an object that satisfies the requirements of *Compare*) which returns `true` if the first argument is *less* than (i.e. is ordered *before*) the second.

The signature of the comparison function should be equivalent to the following:

```
bool cmp(const Type1 &a, const Type2 &b);
```

Functors: Examples from STL


```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;
class Student{
public:
    float CGPA;
    string RollNo;
  Student(string RN, float CG):RollNo(RN), CGPA(CG){};
  bool operator<(const Student& rhs){
    return this->CGPA < rhs.CGPA;
  }
};
```

- **Continued in next slide**

Functors: Examples from STL


```
int main(){
    vector<Student> students{Student("18CS10065",8.5),
                             Student("19CS30008",8.3),
                             Student("17CS10024",8.9)};
    sort(students.begin(),students.end());
    for (auto st:students){
      cout << st.RollNo << ", " << st.CGPA << endl;
    }
    return 0;
}
```

```
codio@fluteregular-vivarodeo:~/workspace$ g++ -o FunctorSort_01 \
> FunctorSort_01.cpp && ./FunctorSort_01
19CS30008, 8.3
18CS10065, 8.5
17CS10024, 8.9
```

Functors: Examples from STL


```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;
class Student{
public:
    float CGPA, SGPA;
    string RollNo;
  Student(string RN, float CG, float SG):RollNo(RN), CGPA(CG), SGPA(SG){};
  bool operator<(const Student& rhs){
    return this->CGPA < rhs.CGPA;
  }
};
class SGPAComp{
public:
  bool operator()(const Student& lhs, const Student& rhs){
    return lhs.SGPA < rhs.SGPA;
  }
};
```

- **Continued in next slide**

Functors: Examples from STL


```
int main(){
  vector<Student> students{Student("18CS10065",8.5,9.1),
                           Student("19CS30008",8.3,7.8),
                           Student("17CS10024",8.9,8.5)};
  // Sort using the functor to compare SGPAs
  sort(students.begin(),students.end(), SGPAComp());

  for (auto st:students){
    cout << st.RollNo << ", " << st.CGPA << ", "
              << st.SGPA << endl;
  }
  return 0;
}
```

```
adas@mcurie:~/workspace/sw_engg_2023/codes$ ./FunctorSort_02
19CS30008, 8.3, 7.8
17CS10024, 8.9, 8.5
18CS10065, 8.5, 9.1
```