Lecture 24: Advanced Topics

Class page: https://github.com/tsung-wei-huang/cs1410-40

Dr. Tsung-Wei Huang
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT

Announcement

- **Final Exam starts on 11/30 and ends on 23:59 PM 12/6**
  - Take-home exam, same as midterm
  - Cover all topics
    - 50% concept questions
    - 50% programming questions
  - Free to discuss with your friends and use internet resources
  - Never copy solutions
- **No more LAB!!!**

Lambda for C++

- **What's a Lambda?**
- **Lambda expression or lambda function**
  - an expression that specifies an anonymous function object
- **Imagine handing an operation or function (code) to some other operation or function**
  - For generic work
  - For a functional style
  - For concurrency
  - For readability
    - Eliminate tiny functions

Lambda for C++

```
void print_square(int i)
{
    cout << i*i << endl;
}

int main()
{
    vector<int> v;
    for_each(v.begin(), v.end(), print_square);
}
```

Lambda for C++

```
int main() {
  vector<int> v;
  for_each(v.begin(), v.end(),
  [](int i) { cout << i*i << endl; } );
}
```

Lambda for C++

[*captures* **]** *<tparams>*$^{\text{(optional) (C++20)}}$ **(** *params* **)** *specifiers exception attr* **->** *ret requires*$^{\text{(optional) (C++20)}}$ **{** *body* **}**	(1)
[*captures* **]** **(** *params* **)** **->** *ret* **{** *body* **}**	(2)
[*captures* **]** **(** *params* **)** **{** *body* **}**	(3)
[*captures* **]** **{** *body* **}**	(4)

1) Full declaration.

2) Declaration of a const lambda: the objects captured by copy are const in the lambda body.

3) Omitted trailing-return-type: the return type of the closure's `operator()` is deduced from `return` statements as if for a function whose return type is declared auto.

4) Omitted parameter list: function takes no arguments, as if the parameter list was **()**. This form can only be used if none of constexpr, mutable, exception specification, attributes, or trailing return type is used.

Lambda with Return Value

```
vector<int> v;
deque<int> d;
transform(v.begin(), v.end(),
   front_inserter(d),
   [](int n) { return n * n * n; });
transform(v.begin(), v.end(),
   front_inserter(d),
   [](int n) -> double {
       if (n % 2 == 0) {return n * n * n;}
       else {return n / 2.0;}
    });
```

Lambda with Captured Variables

```
v.erase(remove_if(v.begin(), v.end(),
        [x, y](int n) { return x < n && n < y; }),v.end());
v.erase(remove_if(v.begin(), v.end(),
        [=](int n) { return x < n && n < y; }), v.end());
for_each(v.begin(), v.end(),
        [&x, &y](int& r) {
        const int old = r;
        r *= 2;
        x = y;
        y = old;       });
```

Dynamic Object Allocation

- **We know std::malloc and std::free**
- **How can we create an object dynamically?**
  - new and delete operators

Smart Pointer

- **Smart pointer automatically manages the lifetime of an object**
- **C++11 has two commonly used smart pointers**
  - shared_ptr: shared ownership
  - unique_ptr: unique ownership

Summary

- ❑ **Lambda Expression**
- ❑ **Dynamic Memory Allocation for Objects**
- ❑ **Smart Pointers**

Thank you for attending CS1410!

Please complete the course evaluation ☺