CS100 Introduction to Programming

Recitation 11

<Yang Feiming>

<yangfm@shanghaitech.edu.cn>

NO PLAGIARISM!!!

- The most likely cause for failing this course.
- You WILL be caught!
- We WILL punish!
- They WILL know!
  - Parents
  - University
  - School
  - Fellows

Overview

- Lambda
  - Concepts
  - Captures
  - Usage
  - Functors
  - Exercises

Lambda expressions

Syntax

- introducer: capture list in square brackets
- declarator: parameter list in parentheses followed by return type
  using trailing return-type syntax
- compound statement in brace brackets

```
[capture-list] (parameter-list) -> return-type { body }
```

Concepts walkthrough

- What is capture?
- Omit some part?
- Difference between captured variables & parameters?
- What can be in the body?
  - return?
  - cin/cout?
  - new?

```
[capture-list] (parameter-list) -> return-type { body }
```

Examples

- `[] (double x) - >int {return floor ( x );}`
- `[] (int x , int y) {return x < y ;}`
- `[] {std::cout << "Hello, World!\n";}`

Captures

- locals only available if captured; non-locals always available
- can capture by value or by reference
- different locals can be captured differently
- can specify default capture mode
- can explicitly list objects to be captured or not
- might be wise to explicitly list all objects to be captured (when practical) to avoid capturing objects accidentally (e.g., due to typos)
- to capture class members within member function, capture `this`
- capture of `this` must be done by value

Using captures

- Trivial use case

```
void print_larger_than(std::vector<int> &v, int pivot) {
    auto predicate = [pivot](int x) { return x > pivot; };
    for (auto x : v)
        if (predicate(x))
            std::cout << x << '\n';
}
```

- We will see a better version later

Using captures

- Does this compile?

```
// old C library from 1998...
typedef void (*callback_t)(int);
void register_callback(callback_t callback);
// Using it now in C++11...
ofstream a(str);
register_callback([& a](int lhs) { a << lhs; });
```

- Now?

Variations of captures

- Capture all by value

```
[=](int x) { return x + a + b; }
```

- Capture all by reference

```
[&](int x) { return x + a + b; }
```

- Capture all by reference, but a by value

```
[=, &a](int x) { return x + a + b; }
```

- Capture all by value, but a by reference

```
[&, a](int x) { return x + a + b; }
```

Variations of captures

- Move

```
[a = std::move(myvector)](int x) { return x + a[0]; }
```

- Change name

```
[a = x](int x) { return x + a; }
```

Captures - mutate

```
int main() {
    int count = 5;
    // Must use mutable in order to be able to
    // modify count member.
    auto get_count = [&]() mutable -> int {
        return count++;
    };
    int c;
    while ((c = get_count()) < 10) {
        std::cout << c << '\n';
    }
}
```

- Not recommended, but sometimes handy

Using captures

- What could possibly go wrong?

```
auto getCounter(int initial) {
    return [&]() mutable -> int { return initial++; };
}

int main() {
    auto counter = getCounter(5);
    int c;
    while ((c = counter()) < 10) {
        std::cout << c << '\n';
    }
}
```

Expecting lambda

- Formally called higher order functions
- Basically a function that takes a function as parameter
- C counterpart: function pointers

- When to use?
  - Expect user code to fill your template
- Side note: normal functions
  - Expect user value to fill your template

Expecting lambda

- Function pointers?
  - Capture unfriendly
- Template way: as a template argument

```
template <typename Predicate, typename T>
void print_if(const std::vector<T> &vec, Predicate predicate) {
    for (auto const& t: vec)
        if (predicate(t))
            std::cout << t << std::endl;
}
```

  - Won’t work well if you want to store it

Expecting lambda

- std::function: wrapper around callables

```
class EventBus {
public:
    void setHandler(std::function<void(const Event &)>);
    std::vector<std::function<void(const Event &)>> handlers;
};

int main() {
    EventBus eventBus;
    eventBus.setHandler([](const Event &evt) { std::cout << evt; });
}
```

Lambda vs struct

- Lambda better?
- Struct better?
- What cannot be done with struct?

Lambda vs struct

- Which is better?

```
struct IsPositive {
    bool operator()(int arg) { return arg > 0; };
};

int main() {
    std::vector<int> v{1, 2, 3, 0, -1};
    print_if(v, IsPositive());                  // struct version
    print_if(v, [](auto i) { return i > 0; });  // lambda version
    return 0;
}
```

Lambda vs struct

- Which is better?

```
struct IsBiggerThan {
    int pivot;
    bool operator()(int arg) { return arg > pivot; };
};

int main() {
    std::vector<int> v{1, 2, 3, 0, -1};
    int pivot = 1;
    print_if(v, IsBiggerThan {pivot});                  // struct version
    print_if(v, [pivot](auto i) { return i > pivot; }); // lambda version
    return 0;
}
```

Lambda vs struct

- Pros:
  - less tedious to write
- Cons
  - less readable
  - less reusable

Exercises

- higher.hpp & higher.cpp
  - Reinvent a STL function (naïve form)
  - Implement a basic logger that allow user to specify how to do logging
    - The log handler may be asynchronous!
- application.cpp & main.cpp
  - Try to write no loops here
  - Use the given function

Clone the skeleton code

- `git clone https://github.com/llk89/cs100recitation11`

Exercises

```
typedef std::function<std::string()> log_message_type;
void Logger::log(Logger::log_message_type message);
```

- Why use a lambda as log messages?

QA Time

- If you have any problems with…
  - Functional programming
  - CMake
  - Recitation 11
- Ask now