601.220 Intermediate Programming

Summer 2023, Meeting 22 (July 24th)

Today's agenda

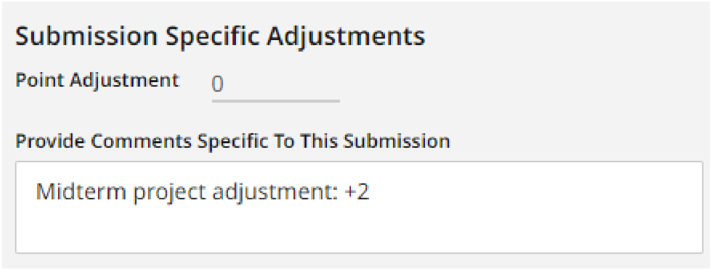
- Exercise 35 review
- Exercise 36 review
- Day 37 recap questions
- Work on final project

Reminders/Announcements

- Final project due by 11pm on **Friday, July 28th**
- Final exam in class on **Friday, July 28th**
    - Review materials are posted on the course website
    - Final exam review session: next week, we will announce on Piazza when details are finalized
- Final project contributions are due by 11pm on **Saturday, July 29th**. This survey corresponds to 5 percent of your final project.
- No extensions will be given to either the final project or the contributions survey.

Grading clarifications

- Contributions surveys are 5% of you project grades.
- Depending on the amount of work done by each member, individual grades were modified.
- See comments left on Q1 of the contributions survey to see your modifications.
- Not everybody got a modification.



Submission Specific Adjustments

Point Adjustment 0

Provide Comments Specific To This Submission

Midterm project adjustment: +2

Figure 1: midterm adjustments

Exercise 35 review

Checking whether file was opened successfully in `readFile` function:

```
std::ifstream fin(filename);
if (!fin.is_open()) {
  throw std::ios_base::failure("could not open file");
}
```

Exercise 35 review

Handling the exception in `main`:

```
std::vector<int> numbers;
try {
  numbers = readFile(argv[1]);
  // ...code to print out the values in the vector...
  return 0;
} catch (std::ios_base::failure &ex) {
  std::cerr << "Error: " << ex.what() << "\n";
  return 1;
}
```

Exercise 35 review

$\text{specific} \longrightarrow \text{general}$

Seeing what happens when `readFile` reads non-integer data:

```
$ ./exceptionExercise nonIntData.txt
Error: File contains non-integer data!
```

Handling `std::invalid_argument` in `main`:

```
try {
  // code calling readFile
} catch (std::ios_base::failure &ex) {
  std::cerr << "Error: " << ex.what() << "\n";
  return 1;
} catch (std::invalid_argument &ex) {// <-- another catch block
  std::cerr << "Error: " << ex.what() << "\n";
  return 1;
}
```

Exercise 36 review

Implementing `MyList::iterator`:

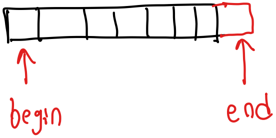
```
class iterator {
  const MyNode<T>* ptr;

public:
  iterator(const MyNode<T>* initial) : ptr(initial) { }
  // use compiler-generated copy ctor and assignment op

  iterator& operator++() { ptr = ptr->next;
                           return *this; }
  bool operator!=(const iterator& o) const
                          { return ptr != o.ptr; }
  T& operator*() const    { return ptr->data; }

};
```

Exercise 36 review



Implementing `MyList::begin()` and `MyList::end()`:

```
iterator begin() { return iterator(head); }

iterator end()   { return iterator(nullptr); }
```

Exercise 36 review

Implementing `MyList::const_iterator`:

```
class const_iterator {
private:
  MyNode<T> *ptr;

public:
  const_iterator(MyNode<T> *initial) : ptr(initial) { }
  // use compiler-generated copy ctor and assignment op

  const_iterator& operator++() { ptr = ptr->next;
                                 return *this; }
  bool operator!=(const const_iterator &other) const
                               { return ptr != other.ptr; }
  const T& operator*() const   { return ptr->data; }
};
```

Exercise 36 review

Implementing `MyList::cbegin()` and `MyList::cend()`:

```
const_iterator cbegin() const
  { return const_iterator(head); }

const_iterator cend() const
  { return const_iterator(nullptr); }
```

Exercise 36 review

Implementing `MyList` constructor from begin/end iterators:

```
template<typename Itr>
MyList<T>(Itr i_begin, Itr i_end) : head(nullptr) {
  for (Itr i = i_begin; i != i_end; ++i) {
    insertAtTail(*i);
  }
}
```

Note that the `Itr` type parameter is the iterator type for the collection we are copying data from. Because it is a type parameter, we can copy from any type of collection that supports iterators.

Day 37 recap questions

1. How do you pass functionality as an argument to a function in C?
2. If you template the function type, what else can you pass in as an argument to a function in C++?
3. What are two advantages of using lambdas?
4. Why is the `auto` keyword essential?
5. Why else is the `auto` keyword useful?

1. How do you pass functionality as an argument to a function in C?

In C, we can use a *function pointer* to pass the identity of a function as an argument to another function.

The C library `qsort` function takes advantage of this possibility.

```c
void qsort(void *base, size_t nmemb, size_t size,
           int (*compar)(const void *, const void *));
```

func pointer

qsort example program

```c
// qsort.c:
#include <stdio.h>
#include <stdlib.h>
int compare_doubles(const void *left, const void *right)
  { double diff = *((const double *)left) - *((const double *)right);
    if (diff < 0.0) return -1;
    if (diff > 0.0) return 1;
    return 0; }

int main(void) {
  double arr[] = { 8.7, 2.8, 1.8, 9.9, 1.2, 5.1, 8.4 };
  qsort(arr, 7, sizeof(double), compare_doubles);
  for (int i = 0; i < 7; i++) { printf("%.1f ", arr[i]); }
  printf("\n");
  return 0;
}
```

```
$ gcc -g -std=c11 -Wall -Wextra -pedantic qsort.c -o qsort
$ ./qsort
1.2 1.8 2.8 5.1 8.4 8.7 9.9
```

Disadvantages of function pointers

Function pointers have concrete parameter types and return type. So, there's no direct way to allow them to be generic. (Hence, the use of `const void *` as the pointer type in the comparison function.)

Also, a function pointer is just that, a pointer to a function. It is not an object, and can't have member variables to hold extra data that the function might need.

(In Linux, run the command $\boxed{\text{man 3 qsort}}$, and notice how there is a `qsort_r` function. Its purpose is to allow the comparison function to have access to additional information, beyond just pointers to the elements being compared.)

2. If you template the function type, what else can you pass in as an argument to a function in C++?

You can pass in a *functor*, which is simply an object which has an overloaded function call operator.

Because an object is an instance of a class, it can have member variables to store extra information. The class can also be generic.

inst();

A template function supporting a functor

Consider how `std::sort` is declared:

```
template< class RandomIt, class Compare >
void sort( RandomIt first, RandomIt last, Compare comp );
```

`Compare` is a type parameter, so `comp` could be an object. A function pointer would work too! It just has to be something that can be called on the values being compared, and return true if the left value should be ordered before the right one.

Case-insensitive string sorting in C++

```cpp
// cisort.cpp:
#include <iostream>
#include <string>
#include <algorithm>
#include <vector>
#include <cctype>

std::string make_lower(const std::string &s) {
  std::string result;
  for (unsigned i = 0; i < s.size(); i++) { result += ::tolower(s[i]); }
  return result;
}

struct CaseInsensitiveCompare {  // a functor type!
  bool operator()(const std::string &left, const std::string &right) const
    { return make_lower(left) < make_lower(right); }
};
```

Functor

```cpp
int main() {
  std::vector<std::string> v { "cat", "Fish", "Zebra", "whale" };
  std::sort(v.begin(), v.end(), CaseInsensitiveCompare());
  for (const std::string &s : v) { std::cout << s << " "; }
  std::cout << "\n";
}
```

```
$ g++ -g -std=c++11 -Wall -Wextra -pedantic cisort.cpp -o cisort
$ ./cisort
cat Fish whale Zebra
```

3. What are two advantages of using lambdas?

C++ lambdas are a mechanism for creating functor objects.

They are defined "on the fly", right at the point in the program where they are needed. This means you don't have to define an explicit `struct` or `class` type that your functor is an instance of.

There is a mechanism for capturing values or references to variables that are in scope at the point where the lambda is created. This means they can naturally take advantage of surrounding values and variables in the program.

Case-insensitive string sorting using a lambda

$$[\,\underbrace{\,}_{1}\,]\,(\underbrace{\qquad\qquad}_{2})\,\{\underbrace{\qquad\qquad\qquad}_{3}\}\underbrace{;}_{4}$$

```cpp
// cisort2.cpp:
#include <iostream>
#include <string>
#include <algorithm>
#include <vector>
#include <cctype>

std::string make_lower(const std::string &s) {
  std::string result;
  for (unsigned i = 0; i < s.size(); i++) { result += ::tolower(s[i]); }
  return result;
}

int main() {
  std::vector<std::string> v { "cat", "Fish", "Zebra", "whale" };
  auto compare = [](const std::string &left, const std::string &right) {
    return make_lower(left) < make_lower(right);
  };
  std::sort(v.begin(), v.end(), compare);
  for (const std::string &s : v) { std::cout << s << " "; }
  std::cout << "\n";
}
```

$\}$ lambda

```
$ g++ -g -std=c++11 -Wall -Wextra -pedantic cisort2.cpp -o cisor
$ ./cisort2
cat Fish whale Zebra
```

std::find_if

The `std::find_if` function searches a range of elements to find the first one (if any) matched by a *predicate*. The predicate is a functor taking an element as a parameter, and returning true or false.

It is declared as follows:

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

`std::find_if` is a generalized *sequential search* function.

Find first even element in a range of integers

```cpp
// findeven.cpp:
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
  std::vector<int> v { 17, 23, 27, 44, 97, 93, 14 };

  auto is_even = [](int val) { return (val % 2) == 0; };
                  ①     ②            ③                 ④

  auto i = std::find_if(v.begin(), v.end(), is_even);

  if (i != v.end()) { std::cout << *i << "\n"; }
               else { std::cout << "not found\n"; }
}
```

```
$ g++ -g -std=c++11 -Wall -Wextra -pedantic findeven.cpp -o find
$ ./findeven
44
```

Find first multiple of N

```
// findmultn.cpp:
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
  int n;
  std::cin >> n;

  std::vector<int> v { 17, 23, 27, 44, 97, 93, 14 };

  auto pred = [n](int val) { return (val % n) == 0; };

  auto i = std::find_if(v.begin(), v.end(), pred);

  if (i != v.end()) { std::cout << *i << "\n"; }
                else { std::cout << "not found\n"; }
}
```

```
$ g++ -g -std=c++11 -Wall -Wextra -pedantic findmultn.cpp -o fin
$ echo "11" | ./findmultn
44
$ echo "6" | ./findmultn
not found
```

Captures

The `findmultn.cpp` program created a lambda which uses the value of an `int` variable `n`:

```
int n;
std::cin >> n;
// ...
auto pred = [n](int val) { return (val % n) == 0; };
```

The `[n]` at the beginning of the lambda is its *capture list*. In this case, the variable `n` is captured by value. If the capture list were specified as `[&n]`, then `n` would be captured by reference, meaning that the lambda would be able to modify `n`.

4. Why is the auto keyword essential?

A lambda has a type. You can think of this type as being equivalent to the struct or class type you would have had to write. E.g.:

```
auto pred = [n](int val) { return (val % n) == 0; };
```

vs.

```
struct MatchMultipleOfN {
  int n;
  MatchMultipleOfN(int n) : n(n) { }
  bool operator()(int val) const { return (val % n) == 0; }
};

// ...

auto pred = MatchMultipleOfN(n);
```

Lambda types are generated by the compiler

Because the type of a lambda is generated by the compiler, as a programmer you have no way of knowing what the name of the type is!

The `auto` keyword allows you to declare a variable whose type is based on the value you use to initialize the variable.

This is perfect for lambdas, since only the compiler knows the type of the lambda.

5. Why else is the auto keyword useful?

The auto keyword allows you to declare a variable without explicitly having to name its type. For example:

```
for (auto i = coll.begin(); i != coll.end(); ++i) {
  // ...do something with *i...
}
```

This loop will work as long as coll is a type with begin() and end() member functions returning iterators. We don't need to explicitly name the iterator type.

Work on final project

There is no exercise on lambdas. So, please work on the final project, and use Slack to let us know if you have any questions!

Notes

Notes

Notes

Notes

Notes