601.220 Intermediate Programming

Spring 2023, Day 23 (March 15th)

Today's agenda

- Day 23 recap questions
- Exercise 23

Announcements/reminders

- Midterm project due Friday at 11 pm
  - No late submissions

Day 23 recap questions

1. What is a template in C++?
2. What is the standard template library (STL)?
3. How do you iterate over a `std::vector` and print out its elements?
4. What is an iterator in C++?
5. How do you add an element to an existing vector.
6. (Bonus) What is the output of the program below?

1. What is a template in C++?

A template allows a `struct` type, a `class` type, or a function to be instantiated with a variety of data types or combinations of data types.

In C, a linked list node type must hard-code the payload data type, e.g.:

```
// this node type is only useful for linked lists of
// char values
struct Node {
  char data;
  struct Node *next;
};
```

C++ template linked list node type

In C++, the payload data type can be specified with a “type parameter”:

```
template<typename E>
struct Node {
  E data;
  struct Node<E> *next;
};
```

Now we can have `Node<char>`, `Node<int>`, `Node<std::string>`, etc.

2. What is the standard template library (STL)?

The STL is a collection of useful template functions and classes provided by the standard C++ library.

Examples: `std::vector`, `std::list`, `std::map`, `std::sort`, many others.

Observation: to a large degree, effective programming means finding efficient and elegant ways to store, access, and do computations on data.

It is challenging to do these things in C because the only "built in" feature for aggregating data is the array, and the "built in" support for doing computations is very limited (e.g., `qsort`.)

In C++, the STL provides

1. very powerful ways to organize and access data, and
2. powerful tools for doing computations on data

3. How do you iterate over a `std::vector` and print out its elements?, 4. What is an iterator in C++?

Traversing a collection of values in an STL container (such as a `vector`) is done using an *iterator*. An iterator is a generalization of a pointer, and is used in a way that is very similar to the way pointers are used.

In fact, a pointer to an array element *is* an iterator, because it supports all of the operations required of iterators.

Iterator example

```
std::vector<int> myvec;

// assume some values are added to myvec

for (std::vector<int>::const_iterator i = myvec.cbegin();
     i != myvec.cend();
     ++i) {
  int value = *i;
  std::cout << value << " ";
}
```

5. How do you add an element to an existing vector?

Use the push_back member function.

```
std::vector<int> myvec;
assert(myvec.size() == 0); // myvec is initially empty

myvec.push_back(1);
myvec.push_back(2);
myvec.push_back(3);

assert(myvec.size() == 3); // 3 elements were added
assert(myvec[0] == 1);
assert(myvec[1] == 2);
assert(myvec[2] == 3);
```

6. (Bonus) What is the output of the program below?

```
#include <iostream>
#include <vector>

using std::cin; using std::cout; using std::endl;
using std::vector;

int main() {
  vector<double> numbers;
  for (int i = 1; i <= 10; i++) {
    if (i % 2 == 1)
      numbers.insert(numbers.begin(), i / 2.0);
    else
      numbers.push_back(i * 2.0);
  }

  vector<double>::iterator it = numbers.begin();
  cout << "first == " << *it << endl;
  cout << "middle1 == " << *(it + 4) << endl;
  cout << "middle2 == " << *(it + 5) << endl;
  cout << "last == " << *(it + 9) << endl;
}
```

Exercise 23

- Practice basic input and output using iostreams
- Practice using `std::vector`
- Recursion (merge sort)
- Talk to us if you have questions!

Notes

Notes

Notes

Notes

Notes