CS203 (May 2)

Substitute: KWANG HYUN KIM

https://cs100.qccmathcs.com/LEC/CS203.html
https://cs100.qccmathcs.com/LEC/CS203.pdf

REVIEW I

```
template <typename T> void sequentialWriteToEnd(vector<T> & v)
{
    auto start = chrono::steady_clock::now();
    for (long i =0; i < 100000; i++) {
        v.push_back(i);
    }
    auto end = chrono::steady_clock::now();
    cout << "VectorSWE:Elapsed time in milliseconds : "
    << chrono::duration_cast<chrono::milliseconds>(end - start).count()
    << " ms" << endl;
}
```

https://tinyurl.com/yc4c76n4

REVIEW II

```
template <typename T> void sequentialWriteToFront(vector<T> & v)
{
    auto start = chrono::steady_clock::now();
    for (long i =0; i < 100000; i++) {
        v.insert(v.begin(), i);
    }
    auto end = chrono::steady_clock::now();
    cout << "VectorSWF:Elapsed time in milliseconds : "
    << chrono::duration_cast<chrono::milliseconds>(end - start).count()
    << " ms" << endl;

}
```

https://tinyurl.com/mr2wf6hu

Question

`vector<long>`

```
sequentialWriteToEnd: 100ms
sequentialWriteToFront:  229611ms
```

“ Why? ”

Possible explanation

`sequentialWriteToEnd` for `vector` requires `copy` operation.

```
| 1 |
| 1 | 1 | copy (with a possible allocation)
| 2 | 1 |
```

Review III

```
template <typename T> void sequentialWriteToEnd(list<T> & v) // T is long here.
{
    auto start = chrono::steady_clock::now();
    for (long i =0; i < 1000000; i++) {
        v.push_back(i);
    }
    auto end = chrono::steady_clock::now();
    cout << "VectorSWE:Elapsed time in milliseconds : "
    << chrono::duration_cast<chrono::milliseconds>(end - start).count()
    << " ms" << endl;
}
```

https://tinyurl.com/59nc9v4r

Review IV

```
template <typename T> void sequentialWriteToFront(list<T> & l)
{
    auto start = chrono::steady_clock::now();
    for (long i =0; i < 100000; i++) {
        l.push_front(i);
    }
    auto end = chrono::steady_clock::now();
    cout << "ListSWF:Elapsed time in milliseconds : "
    << chrono::duration_cast<chrono::milliseconds>(end - start).count()
    << " ms" << endl;
}
```

https://tinyurl.com/yvvwcbhw

Review V

```
template <typename T> void sequentialWriteToEnd(deque<T> & d)
{
    auto start = chrono::steady_clock::now();
    for (long i =0; i < N; i++) {
        d.push_back(i);
    }
    auto end = chrono::steady_clock::now();
    cout << "DequeSWE:Elapsed time in milliseconds : "
    << chrono::duration_cast<chrono::milliseconds>(end - start).count()
    << " ms" << endl;
}
```

https://tinyurl.com/muf8vkkr

Review VI

```
template <typename T> void sequentialWriteToFront(deque<T> & l)
{
    auto start = chrono::steady_clock::now();
    for (long i =0; i < 1000000; i++) {
        l.push_front(i);
    }
    auto end = chrono::steady_clock::now();
    cout << "DequeSWF:Elapsed time in milliseconds : "
    << chrono::duration_cast<chrono::milliseconds>(end - start).count()
    << " ms" << endl;
}
```

https://tinyurl.com/bdddp6rw

Note

`list<long>`

```
sequentialWriteToEnd: 3ms
sequentialWriteToFront: 4ms
```

`deque<long>`

```
sequentialWriteToEnd: 3ms
sequentialWriteToFront: 7ms
```

`sequentialWriteToFront` for `list` and `deque` does not require `copy` operation.

Practice I

```
template <typename T> void sequentialRead(vector<T> &);
template <typename T> void sequentialRead(deque<T> &);
template <typename T> void sequentialRead(list<T> &);
```

Practice I: Note

- Use the previously filled containers and add one to every element's value.
- While the vector and deque containers support array subscript notation (objectname[index]), the list container does not.
- Use an `iterator` and `dereferencing` (`*`) to traverse the list elements. For a list `l`, use

```
auto it = l.begin();
```

Demo main function for `vector`

```
template <typename T> void sequentialRead(vector<T> &);
int main(int argc, const char * argv[]) {
    vector<long> myVector;
    sequentialWriteToEnd(myVector);
    sequentialRead(myVector);
    return 0;
}
```

Sample code: https://tinyurl.com/5n8nd4wd

Demo main function for `list`

```
template <typename T> void sequentialRead(list<T> &);
int main(int argc, const char * argv[]) {
    list<long> myList;
    sequentialWriteToEnd(myList);
    sequentialRead(myList);
    return 0;
}
```

Sample: https://tinyurl.com/2p8nveba

Demo main function for `deque`

```
template <typename T> void sequentialRead(deque<T> &);
int main(int argc, const char * argv[]) {
    deque<long> myDeque;
    sequentialWriteToEnd(myDeque);
    sequentialRead(myDeque);
    return 0;
}
```

One `sequentialRead` for `vector`, `list` and `deque`

```
template <typename T> void sequentialRead(const T & C){

}
```

c random

REF: `random` is related to OS security.
Old random function **rand()** in `cstdlib` generates a pseudo-random integral number between 0 and RAND_MAX.

```
#include<cstdlib>
#include<ctime>
#include<iostream>
using namespace std;
int main(){
    srand(static_cast<int> (time(NULL)));// Reset random sequence.
    for(int i=0;i<10;++i)
        cout<<rand()<<endl; //Generate random number from 0 to RAND_MAX.
}
```

random between `a` and `b`

Assume $a < b < RAND\_MAX$.

```
number%10l // between 0 and 9
```

```
number%(b-a+1)     // between 0 and (b-a)
(number%(b-a+1))+a // between a and b
```

Practice III: `rand_atob`

Recall

```
(number%(b-a+1))+a // between a and b
```

Make `int rand_atob(int a, int b)` which returns a random number from `a` to `b` using `rand` function.

Note: It is not secure for a commercial program.
Video, Doc

C++11 random

```cpp
#include <random>
#include <iostream>
using namespace std;
int main()
{
    std::random_device rd;//Make non-deterministic seed.
    std::mt19937 gen(rd());// Mersenne Twister 19937bit generator.
    std::uniform_int_distribution<int> dis(1, 10);// 1<= x <= 10
    //std::uniform_real_distribution<double> dis(1.0, 10.0);
    for (int n=0; n<10; ++n)
        std::cout << dis(gen) << ' ';
    std::cout << std::endl;
}
```

Practice IV: Count 100 random integers from 10 and 14.

Example of screenshot.

```
10:21
11:23
12:25
13:17
14:14
sum:100
```

Sample:https://tinyurl.com/mr24hc8p https://tinyurl.com/2a9sc8x2

`print` function

Make a `print` which prints all elements of a given container of `vector`, `list` or `deque`.

```
template <typename T> void print(const T & C);
```

Hint1: Use an iterator with `begin()` and `end()` member functions.
Hint2: Use `++` operator to move your iterator.

Sample: https://tinyurl.com/3whnkyc2

Practice V

```
template <typename T> void randomWrite(vector<T> &);
template <typename T> void randomWrite(list<T> &);
template <typename T> void randomWrite(deque<T> &);
```

- Generate a random index `i` where $0 \leq i < 100000$.

Note: For a random access, you need to **reserve enough space** first to avoid any out-of-range issues.

```
vector<long> myVector(100000);
list<long> myList(100000);
```

How to move an `iterator`

For a given iterator `it`, `++it` ot `it++` will move `it` into the next.

For `vector` and `deque`, you can use `it+n` to access the `n` th successor of iterator `it`.

```
vector<long> v= {0,1,2,3,4,5};
auto it=v.begin();
cout<<*(it+2);
```

How about `list`?

`next` and `advance`

- `std::next(it, n)` returns the `n`th successor of iterator `it`.

https://en.cppreference.com/w/cpp/iterator/next

```
list<long> L={5,6,7,8,9};
auto it=L.begin(); // *it=5
auto it2=next(it,3); // *it2=8
```

- `std::advance(it, n)` increments `it` by `n` elements.

https://en.cppreference.com/w/cpp/iterator/advance

```
advance(it, 2); // *it=7
```

Demo main function for `vector`

```
#define N 100000
template <typename T> void randomWrite(vector<T> &);
int main(int argc, const char * argv[]) {
    srand(static_cast<int> (time(NULL)));
    vector<long> myVector(N);
    randomWrite(myVector);
    return 0;
}
```

Sample: https://tinyurl.com/2p8t958n

Demo main function for `list`

```
#define N 100000
template <typename T> void randomWrite(vector<T> &);
int main(int argc, const char * argv[]) {
    srand(static_cast<int> (time(NULL)));
    list<long> myList(N);
    randomWrite(myList);
    return 0;
}
```

Sample: https://tinyurl.com/w6hmhnu8 https://tinyurl.com/mydsf45k

Extra: `Concepts`

https://www.youtube.com/watch?v=HddFGPTAmtU

https://en.cppreference.com/w/cpp/language/constraints