Page No.
Date

Range based for loop

- for array & containers.
- returns element in each iteration.
- for each loop.

collection.

```
for (var declaration : range) {
    =
}
```

- int a[] = { 1, 2, 3 };

```
for (int i : arr)
    cout << i << endl;

for (auto i : arr)
    cout << i << endl;
```

i is copy of each element

l-value

```
for (auto &i : arr) {
    cout << i << endl;
    i++;
}
```

i is ref to element

→ modifies element.

'initializer_list<int>'

```
for (auto n : {1, 2, 3, 4}) {
    cout << n;
}
```

↑ r-value

for - loop internals.

```
int arr[] = {11, 22, 33, 44};
for (auto x : arr)
    cout << x << " ";

int *p = &arr[0], *q = &arr[4];
while (p != q) {
    cout << *p << " ";
    p++;
}
```

global fns from std namespace. <iostream>

```
int *p = begin(arr), *q = end(arr);
while (p != q) {
    cout << *p << " ";
    p++;
}
```

```
auto range = arr; // initializer_list<int>
auto p = begin(range);
auto q = end(range);
for (; p != q; p++) {
    auto x = *p;
    cout << x;
}
```

In C++ '17, begin & end pointers need not to be same.

To use range-based for loops with user-defined classes implement begin() & end() methods in the class which returns iterator / iterator like object.
std::begin() & std::end() simply calls begin() & end() on objects.

In main():

```
Bag b;
//init b
for(auto v : b)
    cout << v;
```

↓ expanded to ↓

```
for(auto itr = begin(b);
    itr != end(b);
    itr++) {
    auto v = *itr;
    cout << v;
}
```

```
std::begin(b) {
    return b.begin();
}

std::end(b) {
    return b.end();
}
```

```
class Bag {
    int arr[10];
public:
    int* begin() {
        return arr;
    }
    int* end() {
        return arr+10;
    }
};
```

Page No.
Date

Non-Static Data members Initializer

class data members can be initialized at the point of declaration (in class) from C++ 11 onwards.

```
class Queue {
private:
    int arr[5] {};   // all elems = 0
    int rear = -1;
    int front = -1;
    int count = {0};
public:
    ==
};
```

Copy init & uniform init is allowed

cannot use auto here

Compiler generates init code and add it at the start of constructor (user defined or synthesized).
Values initialized in ctor (user defined) will overwrite initializer.