std::vector
std::list
std::map
==> simple examples

---

l-value
r-value
//R-read, L-left
int a,b,c;
a=10; //10:R, a:L
b=20; //20:R, b:L
c=a+b; //a:R,b:R, a+b:R, c:L

a, b ==> both l-value & r-value
==> address can be obtained
10, 20, a+b, 3.5 ==> r-value only

int *ptr=&a; ==> *ptr : both l & r
ptr, *ptr ==> both L & R
&a ==> r-value only
sum(a,b) ==> r-value only

int sum(int , int);

sum(a,b)=20; //error, no l-value

l-value
r-value

iterator

custom container,iterator

int &ref=a; ==> ref : both L & R
const int &rc=a; ==> r-val is fine
=> restricted l-value
++rc; //error
const int *pc=&a;
==> *pc : restricted L-V
*pc = 200; //error

R-value only expressions
Pure R-Values (prvalues)

Function returning reference
can have l-value also

```
int& maxval(int& r1,int& r2) {
	if(r1 > r2)
		return r1;
	else
		return r2;
}
```

```
max(a,b)++;
max(a,b)=25;
```

Normal References : l-vaue references
C++11 introduced : r-value references

Move Constructor
Move Assignment Operator

```
std::vector<int> v1{10,20,30,40,50}; //C++11
v1.at(3)=42;
v1[2]=35;	//v1[2]==>v1.operator[](2)
```

```
employees[103]="Hello";
```

```
std::list<MyString> names;

names.push_back(MyString("Scott"));
```

```
MyString getName(char *ps) {
	return MyString(ps);
}
```

In both cases the object is temporary, instead of duplicating resources, we may transfer the resources

duplicate + destroy old ==> transfer

Expressions wrapped in std::move (casting), Anonymoys objects will match r-value references

- copy constructor -- clone/duplicate
- move constructor -- transfer/detach+attach

sum(10,20)
--> sum.operator()(10,20)

int x=10; Move Constructor
int y=x; Move Assignment Operator
int z; R-Value References, std::move
z=x;

```
std::vector<int> v1{20, 50, 10, 30, 40 };

std::sort(v1.begin(), v1.end());                    //operator < , M1
```

```
bool rcompare(int x,int y) { return x > y; }    //or any custom logic

std::sort(v1.begin(), v1.end(), rcompare)       //M2
```

```
std::sort<v1.begin(), v1.end(), MyCompare<int>());
MyCompare<int> mygreater;
std::sort<v1.begin(), v1.end(), mygreater);
std::sort<v1.begin(), v1.end(), std::greater<int>());          //M3
```

```
[](int x,int y) { return x * y; }
[](int x,int y) { return x > y; }

std::sort<v1.begin(), v1.end(), [](int x,int y) {
   return x > y;
});                                                 //M4
```

```
std::list<int> mylist { 12, 35, 23, 45, 64, 75, 82 };

bool myfilter(int x) { return x%5==0; }
std::count_if(mylist.begin(), mylist.end(), myfilter);

std::count_if(mylist.begin(), mylist.end(), [](int x) {
    return x%5==0;
});

auto iter = std::find(mylist.begin(), mylist.end(), 64);
```