

Functors and Algorithms

Rupesh Nasre.

January 2020
OOAIA

Functors

- These are function objects.
- Called as class names.
- Implemented as parenthesis operator in C++.
 - `void operator ()(args) {...}`

Function (not functor)

```
#include <bits/stdc++.h>
using namespace std;

int increment(int x) { return (x+1); }

int main()
{
    int arr[] = {1, 2, 3, 4, 5};
    int n = sizeof(arr) / sizeof(*arr);

    transform(arr, arr + n, arr, increment);

    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
    cout << endl;

    return 0;
}
```

Transform is an algorithm. Increment is a function.

Functor

```
#include <bits/stdc++.h>
using namespace std;

class increment {
public:
    int operator () (int arr_num) {
        return arr_num + 1;
    }
};

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int n = sizeof(arr) / sizeof(arr[0]);

    transform(arr, arr+n, arr, increment());

    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
}
```

Depending upon the g++ version, parentheses are optional.

Increment is a functor.

Functor with Aggregates

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

struct vfun {
    int operator()(int n) {
        cout << n << endl;
        return n+1;
    }
};

int main() {
    vector<int> v;
    v.push_back(2);
    v.push_back(3);
    v.push_back(1);
    v.push_back(9);

    transform(v.begin(), v.end(), v.begin(), vfun());

    return 0;
}
```

Classwork: Add k to each element of v.

Parameterized Functor

```
class increment {  
public:  
    increment(int ln) { n = ln; }  
    int operator () (int arr_num) {  
        return arr_num + n;  
    }  
private:    int n;  
};  
int main() {  
    int arr[] = {1, 2, 3, 4, 5};  
    int n = sizeof(arr) / sizeof(arr[0]);  
    int k;  
  
    cin >> k;  
    transform(arr, arr+n, arr, increment(k));  
  
    for (int i=0; i<n; i++)  
        cout << arr[i] << " ";  
}
```

Can use constructor to store the argument.

```

class increment {
public:
    increment(int ln) { n = ln; }
    int operator () (int arr_num) { return arr_num + n; }
private: int n;
};

int main() {
    vector<int> arr;
    arr.push_back(1);    arr.push_back(2);    arr.push_back(3);
    arr.push_back(4);    arr.push_back(5);
    int n = arr.size();
    increment inc(20);

    transform(arr.begin(), arr.end(), arr.begin(), inc);

    // method 1: indexing
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
    cout << endl;

    // method 2: iterators
    for (vector<int>::iterator it = arr.begin(); it != arr.end(); ++it)
        cout << *it << " ";
    cout << endl;

    // method 3: algorithm
}

```

```

struct lessthanthree {
    bool operator()(int n) { return n < 3; }
};
struct tworaisedtoprint {
    void operator()(int n) { cout << (1 << n) << " "; }
};
struct tworaisedto {
    int operator()(int n) { return (1 << n); }
};
int main() {
    vector<int> v;
    v.push_back(1);    v.push_back(2);    v.push_back(3);
    v.push_back(4);    v.push_back(5);

    int small = count_if(v.begin(), v.end(), lessthanthree());
    cout << "Number of elements less than 3 = " << small << endl;

    for_each(v.begin(), v.end(), tworaisedtoprint());
    cout << endl;

    transform(v.begin(), v.end(), v.begin(), tworaisedto());
    reverse(v.begin(), v.end());
    for (vector<int>::iterator it = v.begin(); it != v.end(); ++it) {
        cout << *it << " ";
    }
    cout << endl;
    return 0;
}

```


Classwork

- Write main and functor to convert each string `w` in a set to `ww`.