# Functors and Algorithms

Rupesh Nasre.

January 2018
OOAIA

```
#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.**

```
#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] << " ";
}
```

**Increment is a functor.**

```
#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;
}
```

**Applicable on aggregates too.**
**Question: What if vfun needs an argument?**

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

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]);

    transform(arr, arr+n, arr, increment(20));

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

**Can use constructor to store the argument.**

```
class increment {
public:
    increment(int In) { n = In; }
    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);

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

    for (vector<int>::iterator it = arr.begin(); it != arr.end(); ++it)
        cout << *it << " ";
    cout << endl;
}
```

**We can use iterators too.**

```
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;
}
```