Higher Order Functions

R K Joshi
IIT Bombay

Simple Functions

```
int f (int x) {
 return x*x;
}
```

A Higher Order Function
- Output parameter is a function

```
f (x,y) { return x+y};
```

```
f(x) {

        return

                g(y) {
                        return x + y;
                } ;
}
```

---

f(x) returns a function!

apply it as follows: f(10)(20);

```
#include <iostream>
#include <functional>
using namespace std;


function<int(int)> f (int x)  {
 return [x] (int y)->int {
                return x + y;
        };
};
```

```
int main()
{
   int x = f (10)(20);
   cout << x <<  endl;
}
```

Another Higher Order Function – Here, the input parameter is a function

/x./y.x y is one such function with two argument. it applies the first one to the second one as shown in the following code x is a function

```
f(x) {
   return
    g(y) {
         x(y);
      } ;
    }
```

```
#include <iostream>
#include <functional>
using namespace std;

int f (function<int(int)>g, int x)  {
        return  g(x);
};
int main()
{
   int x = f ([](int x)->int{return x*x;}, 20);
   cout << x << endl;
}
```

Map: A higher order function

- map (T->T, List<T>) → List<T>

- The function takes a function (value transformer), and a list

- It applies the transformer to each element of the list

- It collects all the results in another list

- That list is returned

```
vector<int> map (function<int(int)> f, vector<int> input)  {

        vector<int>tmp;
                for (int i=0; i<input.size(); i++)
                        tmp.push_back(f(input[i]));

        return tmp;
};

int main() {
 vector<int> v = {1,2,3,4,5,6,7};
 vector<int> w;

        w = map ( [](int x)->int{return x*x;},  v);

        for (int i=0; i<v.size(); i++) cout << v[i] << " ";
        cout << endl;
        for (int i=0; i<w.size(); i++) cout << w[i] << " ";
        cout << endl;
}
```

Fold: A higher order function

- Fold takes a function (T,T)->T, and a list<T>
- Fold applies the function to the list recursively starting from the head node of the list.
- fold (f,L) = L.head, if L.tail is NULL
- fold (f,L) = f(L.head, fold (f,L.tail))

```
int rightfold(function<int(int, int)> f, vector<int> input)  {

        int tmp;
                if (input.size()==1) return input[0];
                tmp=input[0];
                input.erase(input.begin());

                tmp = f (tmp, rightfold (f, input));
                return tmp;
};
int main() {
 vector<int> v = {1,2,3,4,5,6,7};
 int w;
        w = rightfold ( [](int x, int y){return x*y;},  v);
        cout << w << endl;

        w = rightfold ( [](int x, int y){return x+y;},  v);
        cout << w << endl;
}
```