Maps

Introduction:

Let f be an array of integers whose size is n.

Let J be the set {0,1,2,3, 4, … ,n-1}.

We can thus say that f represents a mapping from J to K, where K is a subset of Z, the set of all integers.

We are already familiar with integer arrays and we know their different uses. Suppose in some problem we needed to create a mapping that is somehow different than the one just defined. Suppose that in a class, each student loves some other student. We want to create a mapping that maps the name of each student to the name of the student he loves. In other words, let this mapping be f, let x and y be the names of two students in the class.

f[x]=y if and only if x loves y.

Obviously, this can't be achieved using simple arrays since in this case the index is not an integer, instead, it's a string (a student's name). In this case, we need a data structure called map.

In simple words, a map is a mapping that associates to every "key" one and only one "value". This is the exact formal definition of a mapping in Mathematics. For example, if f[x]=y, x is said to be the "key", and y is the "value" that is associated to x.

Declaration:

A map can be declared in the following way:

map < type 1, type 2 > m;

This declaration defines a map whose name is m that is a mapping from type1 to type2.

In the problem proposed in the introduction, we would need the following declaration:

map<string,string>m;

Formally, type1 represents the type of key (or the index).

Type 2 represents the type of the value.

Example:

Let's now write a program that takes from the user 4 pairs of names, where the pair (s,t) means that student s loves student t. (s and t are names of type string). The program will then repeat the following process 10 times:

- Take a student's name as an input from the user (let it be x)
- Print the name of the student that x loves.

```
#include <bits/stdc++.h>
using namespace std;
int main() {
//Declaration of the map
map <string,string> m;

//filling the map
for (int i=0;i<4;i++) {
    string s,t;
    cin>>s>>t;
    //mapping s to t
    m[s]=t;
}

for (int i=0;i<10;i++) {
    string x;
    cin>>x;
    // m[x] represents the name of the student that x loves
    cout<<m[x]<<endl;
}
return 0;
}
```

Iterating through a map:

A map can be iterated through using an iterator just as a set or vector of pairs. If the iterator is itr, then itr->first represents the key and itr->second represents the value to which this key is mapped.

```
#include <bits/stdc++.h>
using namespace std;
int main() {
//Declaration of the map
map <int,int> m;

//filling the map
for (int i=0;i<10;i++) {
    int a,b;
    cin>>a>>b;
    //mapping s to t
    m[a]=b;
}

map <int, int> :: iterator itr;
// itr->first represents the key , itr->second represents the value.
for (itr=m.begin(); itr!=m.end(); itr++)
    cout<<itr->first<<" "<<itr->second<<endl;
return 0;
}
```

Important notes:

If x is an array of integers, we can write cin>>x[i]; but if x is a map , we don't usually write cin>>x[i]; instead , we take the key and the value from the user and then map them to each other (just as shown in the previous code).

- If x is an array of integers that is not initialized, then the value of x[i] has an undefined unpredictable value. However, if x is a map from strings to integers, it's by default initialized to 0. In other words, any string is mapped, by default, to 0 unless we change this map.
- To clear a map x, i.e. resetting all mappings to 0, we just write x.clear();
- Let m be a map from integers to integers, let x be an integer, suppose we want to check whether we have mapped x to some value or not, we do the following:

  ```
  if (m.find(x)==m.end()) cout<<"x isn't mapped yet"<<endl;
  else cout<<"x is already mapped"<<endl;
  ```

  This idea is very similar to checking whether an element belongs to a set or not.

Final note: this is one of the simplest data structures to use. In order to have a better understanding of maps, go through the exercises and feel free to ask any questions when needed.

Thank You.