Pairs

Introduction:

Let's suppose that we have to store data in the form of pairs. For example, assume you are the manager of a football team and you want to record all the results of all matches and store them. A match result can be modeled as a pair of two integers where the first integer represents the number of goals your team has scored, and the second one represents the number of goals your team has received. How can we do that?

One solution is to define a structure that has 2 integers as attributes as shown below:

```
typedef struct result {
int x;
int y;
} result;
```

We can then use this structure to manipulate data as we want. We can for example declare a vector of results (or pairs):

```
vector <result> v;
```

Now suppose we want to sort this vector. We should first define what it means to compare 2 results. Let's agree upon the following definition:

Let (x1,y1) and (x2,y2) be 2 results.

- (x1,y1)=(x2,y2) if and only if x1=x2 and y1=y2
- (x1,y1)<(x2,y2) if and only if x1<x2 or (x1=x2 and y1<y2)

In other words, to compare two pairs, we first base our decision on the first element of each pair, if it happens that they are equal, we move to compare the second element of each pair.

However, the compiler won't understand our agreement unless we ourselves define it to him.

Fortunately, we don't need to. C++ provides us with a built-in structure of a pair that facilitates our coding.

1- *Declaration of a pair:*

To declare a pair, we use the following method:

pair < type1, type2 > name;

examples:

- pair<int,int> p; //represents a pair of integers, its name is p
- pair<double,int> q; //represents a pair composed of a double and an integer, its name is q

We can even declare more complex structures such as a pair of vector and set:

pair < vector <int> , set <double> > z;

in this declaration , each pair z is composed of 2 elements. The first one is a vector of integers, and the second one is a set of doubles.

2- *Accessing pair elements:*

We can simply access the elements of a pair using the keywords 'first' and 'second'.
Assume we have the following declaration of a pair:

pair<int,int>p;

To access the first element , we write:
p.first;

To access the second one, we write:
p.second;

For better understanding, observe this code below which declares a pair, reads its elements from the user and then displays them of the screen:

```
int main () {

pair<int,int>z;

cin>>z.first;
cin>>z.second;

cout<<z.first<<" "<<z.second<<endl;

return 0;
}
```

3- *The function make_pair()*

The function make_pair() accepts two arguments and returns a pair composed of the given arguments in order.

For example, suppose we have the following declaration of a pair:

pair<int,int>x;

Writing : x=make_pair(2,3); gives the pair x the value (2,3).

We often use this when filling a vector of pairs.

Method 1:

```
int main () {

// Declaring a vector of pairs
vector < pair<int,int> > x;

// filling the vector with 10 pairs
for (int i=0;i<10;i++) {
    pair <int,int>p;
    cin>>p.first>>p.second;
    x.push_back(p);
}

// printing a vector of pairs
for (int i=0;i<10;i++) {
    // x[i] is itself a pair having two components named 'first' and 'second'
    cout<<x[i].first<<" "<<x[i].second<<endl;
}
return 0;
}
```

Method 2:

```
int main () {

// Declaring a vector of pairs
vector < pair<int,int> > x;

// filling the vector with 10 pairs
for (int i=0;i<10;i++) {
    int a,b;
    cin>>a>>b;
    //make_pair(a,b) creates a pair (a,b) and returns it.
    x.push_back(make_pair(a,b));
}

// printing a vector of pairs
for (int i=0;i<10;i++) {
    // x[i] is itself a pair having two components named 'first' and 'second'
    cout<<x[i].first<<" "<<x[i].second<<endl;
}
return 0;
}
```

At the end of the day, we can avoid using this function. It's up to you to choose your style.

4- *Pair as a pointer.*

Assume we have declared a vector of pairs of integers. Let's traverse this vector and display it using iterators. Denote by itr the iterator iterating over v. As we agreed before, the value of the currently traversed element isn't itr, instead it is *itr.

In this case also, *itr represents the value of the currently traversed element , which is a pair. Thus to access the components of the ith pair of the vector, we simply use:

(*itr).first and (*itr).second;

The parenthesis are used to avoid confusion and to make it clear that the * refers strictly to itr.

Alternatively, we can treat the iterator as a pointer , and thus we can access the components of the pair it points to as follows:

itr->first; itr->second;

Printing the same vector using an iterator:

```
#include <bits/stdc++.h>
using namespace std;
int main () {
vector < pair<int,int> > v;
for (int i=0;i<10;i++) {
    int a,b;
    cin>>a>>b;
    v.push_back(make_pair(a,b));
}

vector < pair<int,int> > :: iterator itr;
//Method 1:
for ( itr=v.begin() ; itr!=v.end(); itr++) cout<<(*itr).first<<" "<<(*itr).second<<endl;

//Method 2:
for (itr=v.begin(); itr!=v.end(); itr++) cout<<itr->first<<" "<<itr->second<<endl;

return 0;
}
```

Final Note: sorting a vector of pairs using the function sort() , sorts the vector applying the definition of pair comparison given in the introduction.

Thank you.

Adel Haj Hassan