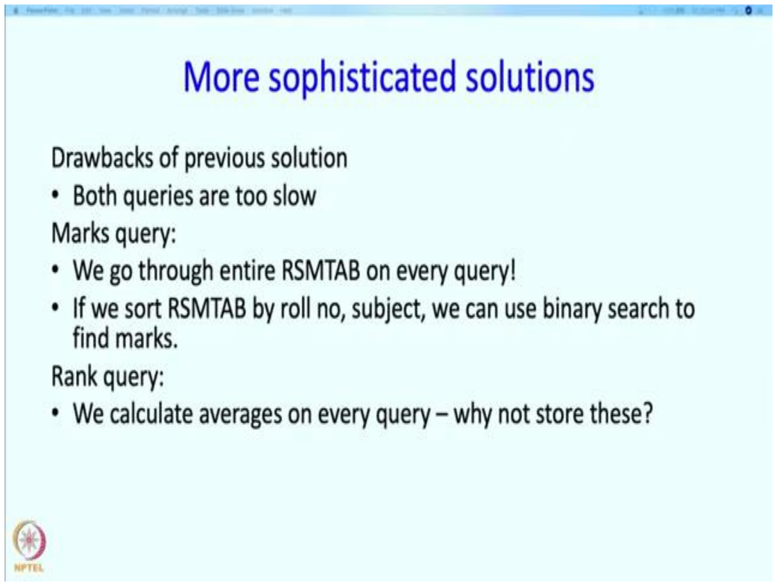
An Introduction to Programming through C++

Professor Abhiram G. Ranade
Department of Computer Science and Engineering,
Indian Institute of Technology Bombay India

Lecture 25, Part 4
Medium size programs
Sophisticated solutions to marks display

Welcome back, in the previous segment we discussed a simple solution based on manual algorithms for our marks display problem, next we are going to turn to somewhat more sophisticated solutions.

(Refer Slide Time: 00:32)



More sophisticated solutions

Drawbacks of previous solution
- Both queries are too slow

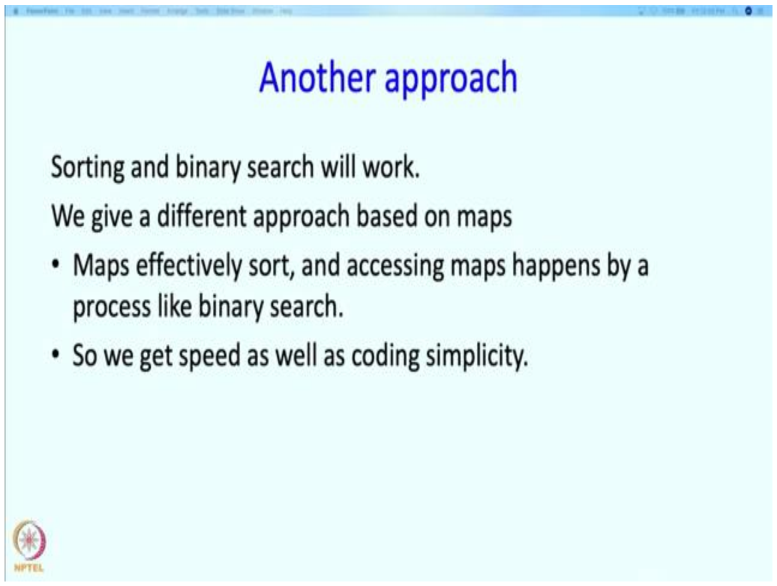
Marks query:
- We go through entire RSMTAB on every query!
- If we sort RSMTAB by roll no, subject, we can use binary search to find marks.

Rank query:
- We calculate averages on every query – why not store these?

So to motivate those let us consider the drawbacks of the previous solution, so both queries are going to be too slow if the number of students, number of subjects is very huge, say if you are going to do this for your entire university, then you will notice that the previous solutions will be quite slow. The marks queries slow because we go through the entire RSMTAB on every query so we want to look for the marks of a given student, but we go through the all the marks of all the students.

Now, if we sort RSMTAB by roll number and subject we can use binary search to find the marks. We know this idea and if we implement these ideas then we will be able to speed this up. The rank query, we calculate averages on every query. So, if I want the rank for a certain roll number, we calculate the average marks obtained by all ranks. If we then need to find the rank of some other roll number we again go through the entire process. So the natural question is why do not we do this once at the very beginning and then just store those things. In fact, once we have calculated the averages calculating the ranks itself is just one additional step, and again why should we repeat why do not we do that at the very beginning, it will save quite a bit of time while processing the queries themselves so that is another thing we should really be doing.

(Refer Slide Time: 02:22)



Another approach

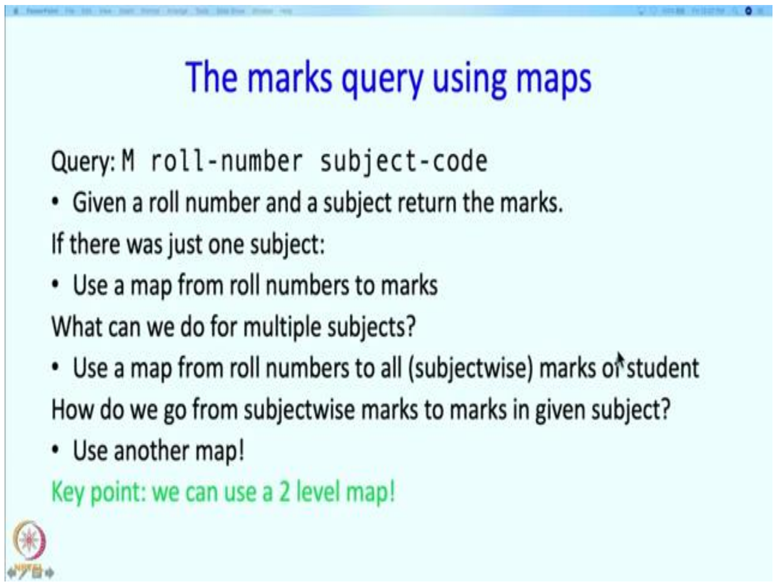
Sorting and binary search will work.

We give a different approach based on maps

- Maps effectively sort, and accessing maps happens by a process like binary search.
- So we get speed as well as coding simplicity.

Now, there is another approach besides sorting and binary search which will work. So, sorting and binary search will work, but we want to suggest one more approach and this approach is going to be based on maps. Maps effectively produce the effect of sorting, and accessing maps happens really by a process like binary search, in fact sometimes it is even faster than binary search. But let us say it is sort of as good as binary search, so effectively what happens is that if we use maps, we are going to get speed as well as coding simplicity.

(Refer Slide Time: 03:03)



The marks query using maps

Query: `M roll-number subject-code`

- Given a roll number and a subject return the marks.

If there was just one subject:

- Use a map from roll numbers to marks

What can we do for multiple subjects?

- Use a map from roll numbers to all (subjectwise) marks of student

How do we go from subjectwise marks to marks in given subject?

- Use another map!

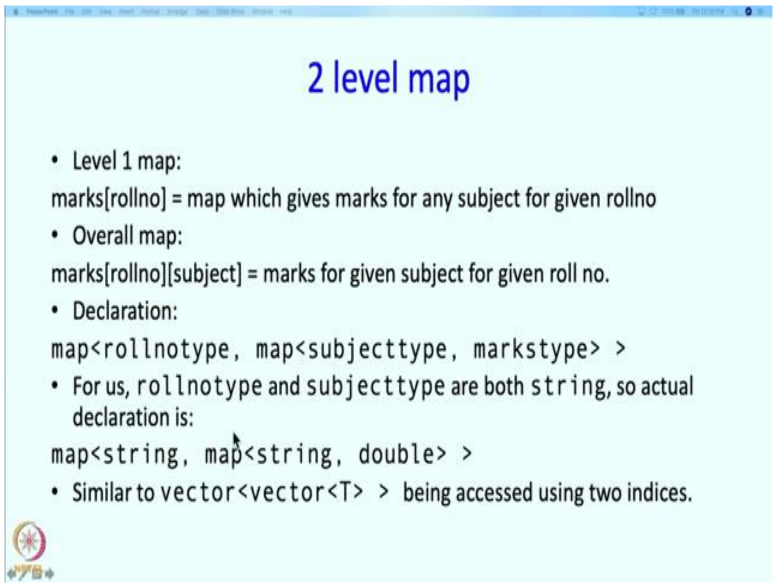
Key point: we can use a 2 level map!

So, let us first look at the marks query so let me remind you what the query was, the query was M followed by roll number followed by subject code and of course you could have given the name, but let us that is a minor matter, so let us just focus on M followed roll number followed by subject code. Given a roll number and a subject we want to return the marks. Now, if there was just one subject what would you do? Well, since we know about maps we could just use the map from roll numbers to marks.

What can we do for multiple subjects? Well we can use the map so the index will be a roll number and the value that the map produces for us could be something like all marks of the students. So, subject wise marks of the students, so the value that is produced is not one mark but all marks. So, may be it produces a vector, so what do we do with all these marks? How do we go from all these marks to the marks in a particular subject?

Well a natural idea will be to use another map because what are these subject-wise maps anyway? So they are telling us that say in mathematics the marks are so many for this given roll number, for this fixed roll number, in physics the marks are so many, in English the marks are so many. So, this is again a map, so the output of this map which where the index is roll number should naturally be another map. So, effectively what we are going to do is we are going to use a 2 level map.

(Refer Slide Time: 6:02)



2 level map

- Level 1 map:

marks[rollno] = map which gives marks for any subject for given rollno

- Overall map:

marks[rollno][subject] = marks for given subject for given roll no.

- Declaration:

```
map<rollnotype, map<subjecttype, markstype> >
```

- For us, `rollnotype` and `subjecttype` are both `string`, so actual declaration is:

```
map<string, map<string, double> >
```

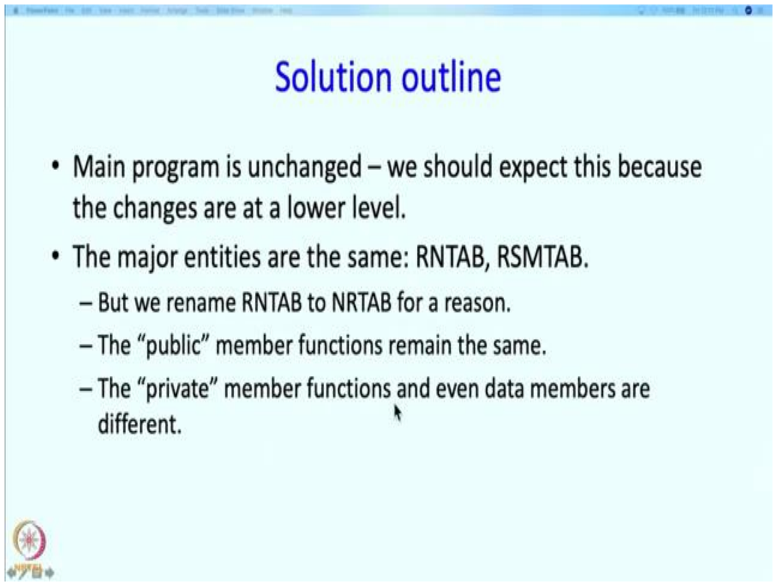
- Similar to `vector<vector<T> >` being accessed using two indices.

So, let us see how this works. So the level 1 map, let us call it marks and marks[roll number], the value produced by this map is itself a map. And what is this, it is a map which gives marks of any subject given a roll number, so given a roll number means the index for this map that is returned by this map is the roll number. So the overall map is as follows, so the 2 level map is also going to be called marks but if we just supply 1 index to it, it is going to give us a map and if we supply the 2nd index, which is the subject, then it will give us the marks for the given subject for the given roll number.

So, how will we declare something like this? So the declaration is the 1st index type, the 1st index was a roll number so whatever type we are going to have for the roll number we are going to place over here and the 1st the while the overall thing is a map which takes a roll number and produces a value but the value is itself a map. But what is this value? It is a map from a subject type to a marks type. So we are going to have 2 indices, the 2nd index is going to be a subject index and the value that that returns is going to be a marks type.

Now, in our case, roll number type and subject type are both string and marks type is double and so the actual declaration is this. So, map<string, map<string, double> > and if you remember we have to put a space over here because if we do not then the C++ will interpret it as the input redirection operator, we do not want that to happen so this is the type declaration for our map. It really similar to vector of vector of T and as just as this is a matrix or this accessed using 2 indices, this is also accessed using 2 indices.

(Refer Slide Time: 07:50)

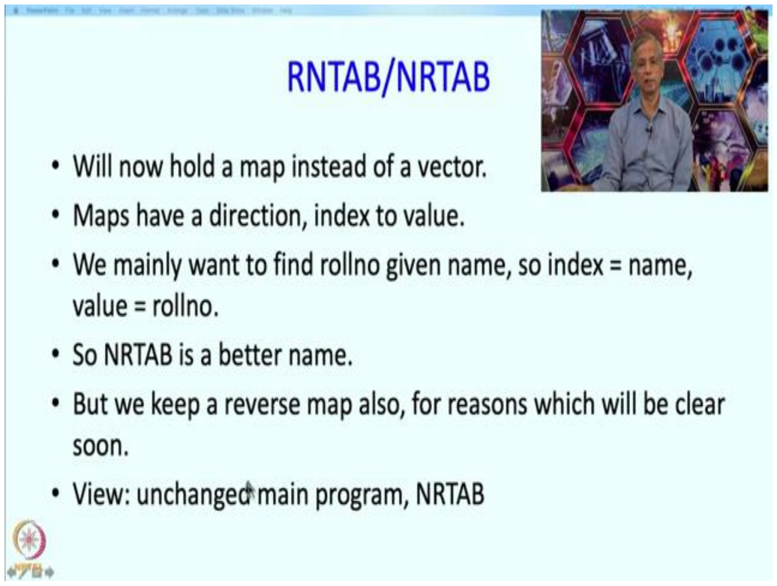


Solution outline

- Main program is unchanged – we should expect this because the changes are at a lower level.
- The major entities are the same: RNTAB, RSMTAB.
  - But we rename RNTAB to NRTAB for a reason.
  - The "public" member functions remain the same.
  - The "private" member functions and even data members are different.

So, what is the solution for this query or the entire thing actually, let us start with the entire thing first. So, the main program you will see is going to be unchanged and in the sense we should expect this, because the changes are at a low level, at a high level the queries format remains the same and our main entities our main tables are going to remain the same. But the internals of the table are going to be different. And well we are going to change the name of RNTAB to NRTAB for some reason which will become clear in minute but this is only a cosmetic name change. The program really is unchanged and there are again only these 2 tables. And while we have not designated any member functions as public or private, but if we were to and I guess once we are done we should designate some functions as public and private. So, the public member functions will remain the same. So, what are the public member functions for RNTAB? So there was lookup here and for NRTAB there was lookup and findrank, so those will remain the same. But the private member functions that we had, they will be different and even the data member were be different and the entire logic of these 2 tables is going to be different because now they are going to be using maps.

(Refer Slide Time: 9:21)



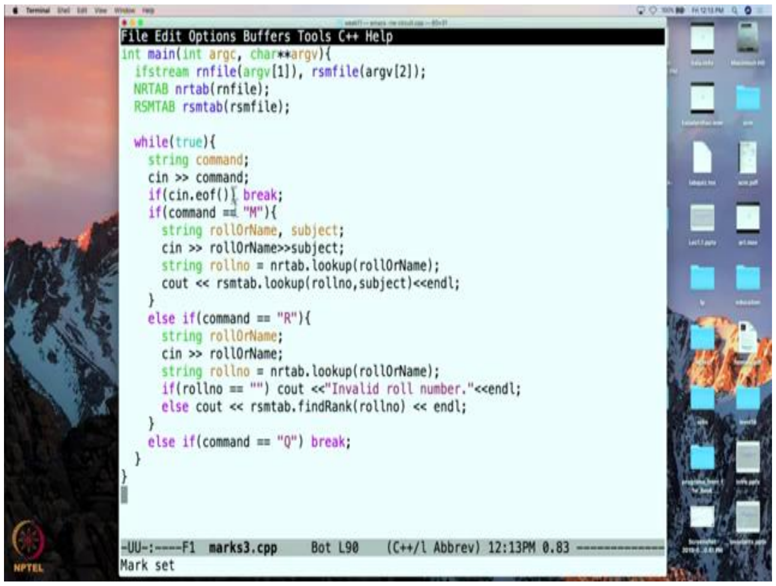
RNTAB/NRTAB

- Will now hold a map instead of a vector.
- Maps have a direction, index to value.
- We mainly want to find rollno given name, so index = name, value = rollno.
- So NRTAB is a better name.
- But we keep a reverse map also, for reasons which will be clear soon.
- View: unchanged main program, NRTAB

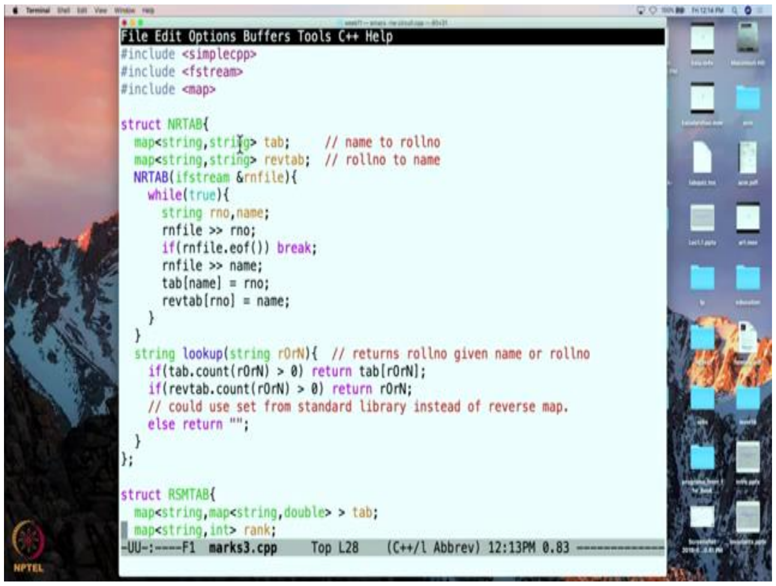
So, how does RNTAB or what we are going to call NRTAB work? So it will not hold a map instead of a vector. Now maps have a direction from index to value and we want this table to give us a roll number, given a name. So the index is a name and a value is a roll number. And therefore, we are going to call it NRTAB, to say that it goes from name to roll number, so that is the only reason why you are changing a name.

So, NRTAB is a better name, and for some reason which will become clear a little bit later we are going to keep the reverse map also. But that is not going to be the main map, so in fact, we will call it the reverse map so to know what is reverse and what is forward it is important to give NRTAB as the name for the entire thing. So, we will see the details soon, in fact, immediately, because now we are going to look at unchanged main program and our main first major entity NRTAB.

(Refer Slide Time: 10:36)



```
int main(int argc, char**argv){
  ifstream rnfile(argv[1]), rsmfile(argv[2]);
  NRTAB nrtab(rnfile);
  RSMTAB rsmtab(rsmfile);

  while(true){
    string command;
    cin >> command;
    if(cin.eof()) break;
    if(command == "M"){
      string rollOrName, subject;
      cin >> rollOrName>>subject;
      string rollno = nrtab.lookup(rollOrName);
      cout << rsmtab.lookup(rollno,subject)<<endl;
    }
    else if(command == "R"){
      string rollOrName;
      cin >> rollOrName;
      string rollno = nrtab.lookup(rollOrName);
      if(rollno == "") cout <<"Invalid roll number."<<endl;
      else cout << rsmtab.findRank(rollno) << endl;
    }
    else if(command == "Q") break;
  }
}

-UU-:----F1  marks3.cpp    Bot L90   (C++/l Abbrev) 12:13PM 0.83 ------------
Mark set
```



```
#include <simplecpp>
#include <fstream>
#include <map>

struct NRTAB{
  map<string,string> tab;     // name to rollno
  map<string,string> revtab;  // rollno to name
  NRTAB(ifstream &rnfile){
    while(true){
      string rno,name;
      rnfile >> rno;
      if(rnfile.eof()) break;
      rnfile >> name;
      tab[name] = rno;
      revtab[rno] = name;
    }
  }
  string lookup(string rOrN){  // returns rollno given name or rollno
    if(tab.count(rOrN) > 0) return tab[rOrN];
    if(revtab.count(rOrN) > 0) return rOrN;
    // could use set from standard library instead of reverse map.
    else return "";
  }
};

struct RSMTAB{
  map<string,map<string,double> > tab;
  map<string,int> rank;
-UU-:----F1  marks3.cpp    Top L28   (C++/l Abbrev) 12:13PM 0.83 ------------
```

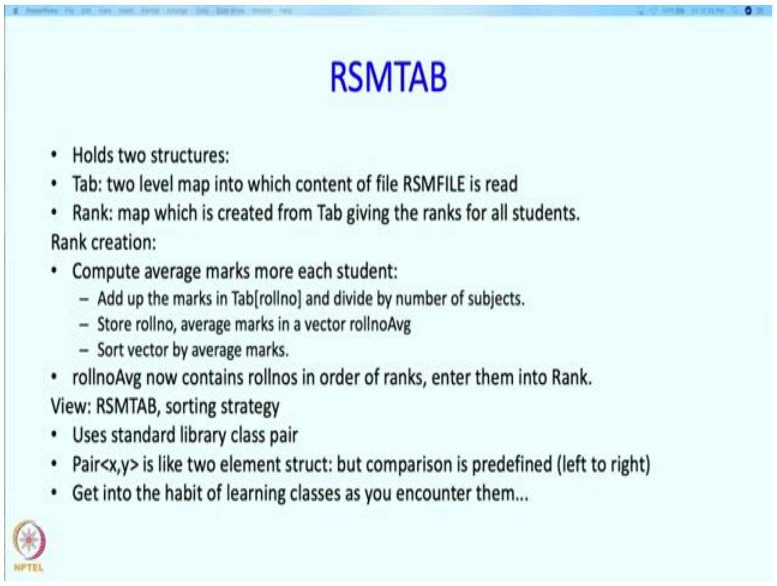
So, this is our 2nd marks display program and I have called it marks 3, sort of marks 2 would have used binary search and sorting but we have skipped that and we have gone to directly version 3. So, let me show you the main program first, the main program really has remained the same except for this name. The name is NRTAB rather RNTAB. So, now let us look at this entity NRTAB.

So here is out entity NRTAB. So as I said it is going to contain a table which is a map now, and it is a map from string to string but it is a map from name to roll number. And we will also keep a reverse map and we will call it REVTAB, and it will give the name given the roll number. So how do we construct NRTAB? so the data is going to be taken from rnfile which was a stream corresponding to the file opened on 3d in the main program? And we are going to go through this file or go through this stream, and as before we are going to read the pairs rno and names so we need to read the roll number into rno and if we find that the file has ended, then we break otherwise we read the name and we store in the table, the roll number corresponding to the name. So, now this is a map and so we can just store it in this manner. And by the way even here we are assuming that the names are single words. So, as indicated earlier, you can do a little bit more work to deal with the case where the names are longer and contain spaces. In the reverse tab, we are going to store the roll number as the index and the name as the value. And now what does lookup look like? Well, as before we are going to supply it an argument which would either be a roll number or a name. So, we are going to check - does this appear as an index in the forward table? And for that we just have to ask tab of count of this greater than 0. If it is then we know that it is appearing as an index in the forward table so that is it is a name. So, in that case we are going to return tab of this index otherwise we are going to check does it appear as the index in the reverse table. The reverse table has roll numbers as indices, so if this does appear we are still going to return that itself, but why we are doing this, well we want to know whether this is a valid roll number. So if it is a valid roll number, then we will return itself. If it is not a valid name or not a valid roll number then we are going to return the empty string. So, this return protocol is exactly like the protocol like we had in the previous implementation as well. Now, I want to make a comment over here, this reverse table we are not using in any sensible way. We are not really indexing into it we are just checking whether the given string appears as a proper index or we are checking whether is given string is a member of the indices in the set of indices in this table.

Now for this we could have used the set data structure from the standard library, and that is the similarity restriction but I use the reverse map since you have not really learnt about it, you could keep this implementation or we could learn the set data structure and in fact, now that you are coming to the end of the course you could get into the habit of looking up new things when you hear about them.

So, this is something you could try out. In any case, I have told you how NRTAB is organized, and you can see that it is actually simpler because the whole searching business is now gone. And the map data structure and the map standard library structure takes care of all of that. So, let us go back to the presentation.

(Refer Slide Time: 15:56)



RSMTAB

- Holds two structures:
- Tab: two level map into which content of file RSMFILE is read
- Rank: map which is created from Tab giving the ranks for all students.

Rank creation:

- Compute average marks more each student:
  - Add up the marks in Tab[rollno] and divide by number of subjects.
  - Store rollno, average marks in a vector rollnoAvg
  - Sort vector by average marks.
- rollnoAvg now contains rollnos in order of ranks, enter them into Rank.

View: RSMTAB, sorting strategy

- Uses standard library class pair
- Pair<x,y> is like two element struct: but comparison is predefined (left to right)
- Get into the habit of learning classes as you encounter them...

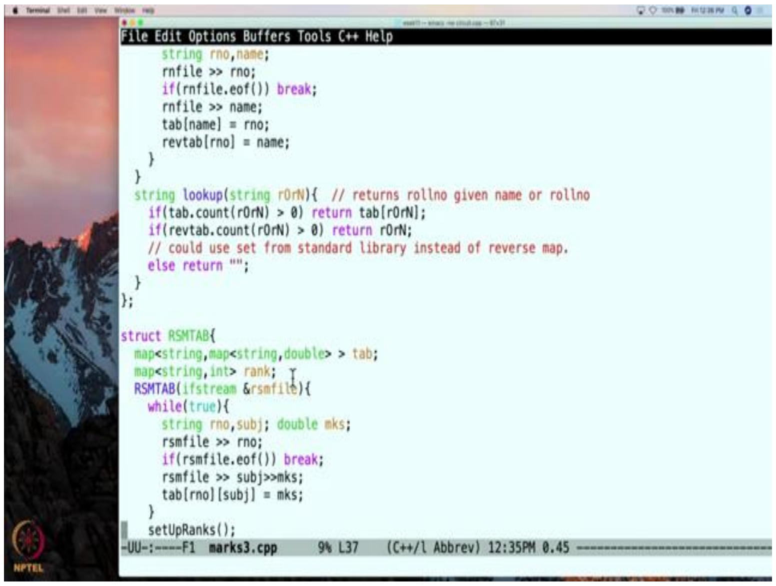
So, now let us look at RSMTAB, so RSMTAB is going to hold 2 structures. So there is going to be a TAB, which is going to be the 2 level map which we talked about earlier into which the content of RSMFILE is read. But then as we discussed, we are going to precompute all the ranks to begin with. So, this is going to be stored again in a map, so its index is going to be the roll number and we are going to create this when we create the entire RSMTAB.

So, we are calling it the RSMTAB, but internally it also contains a table for the ranks. So, how RSMFILE is read is probably quite obvious by now. It is pretty much like how the RNFILE was read to create RNTAB or NRTAB, we will similarly read RSMFILE and create this first tab. How do we create the rank? So, that is a little bit more interesting. So first we have to calculate the average marks for each student. So we have our tab, and we want to add up the marks in tab[roll number], tab[roll number] if you remember, is going to give you another map it is going to give you a sub map. So what is that sub map, the sub map is going to be a map from subjects to marks. So the sub map gives us all the subjects at this particular roll number has taken and we are just going to add up those marks and divided by the number of subjects. So this is how the sub map structure is actually coming in quite useful. And then we are going to store the roll number and the average marks which that we got over here into a vector called roll number average, so this is going to be a 2nd an additional data structure that we will need, but this is going to be a very temporary data structure as we will see. So, we will not put it up as a major thing over here, so we are going to store the roll number and average pairs into this vector. Then we are going to sort the vector by average marks and when this happens we will have the entries arranged in decreasing order of average marks and so then we can go over these entries pick up the roll numbers as they appear and those are the rank holders. So the earliest roll number is rank 1, then the next is rank 2 and so on. So, at the end of this step, roll number average will now contain roll numbers in order of ranks and so we can just enter them into the map called rank. So that is how all this is going to work. So, let us now take a look at the code of RSMTAB. And I want to tell you beforehand that it is going to contain the library class pair. So this pair is going to be used, this is what is going to be stored in this vector roll number average. So let me just introduce this pair class a little bit. So what is the pair class, so the pair class contains 2 elements after all it's a pair and it is really like a 2 element struct. So, it is a 2 element struct that you do not have to, you can sort of define as you go along. But there are some important things that are already defined for pair, for example you can compare 2 pairs without actually yourself defining a comparison operator, if you have defined a struct, then you would have to define the comparison operator but if it is a pair a comparison operator is implicitly defined.

So, how does it work? If you are comparing 2 pairs, then you first compare their x values then you compare their y values. And if you find that the x value of one is smaller than the other, then you declare that to be smaller. Only if they are equal, only if the x values are equal you move on to y. So here is another class and as I said this is a new class, I am going to show you the class and I am going to explain enough about the class but you should not get scared of seeing new things at this point.

So, as I said you are pretty close to the end of the course and at this point you should have the confidence to be able to lookup documentation. So, see the online documentation on pairs to know more about them. So let us know go and look at the code for RSMTAB. So let me just make this a little bit bigger. So, as I said this RSMTAB.

(Refer Slide Time: 21:44)

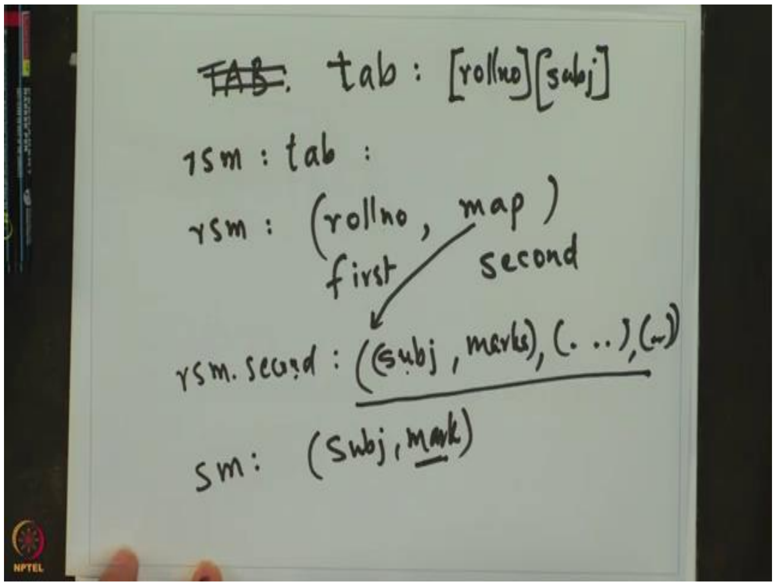


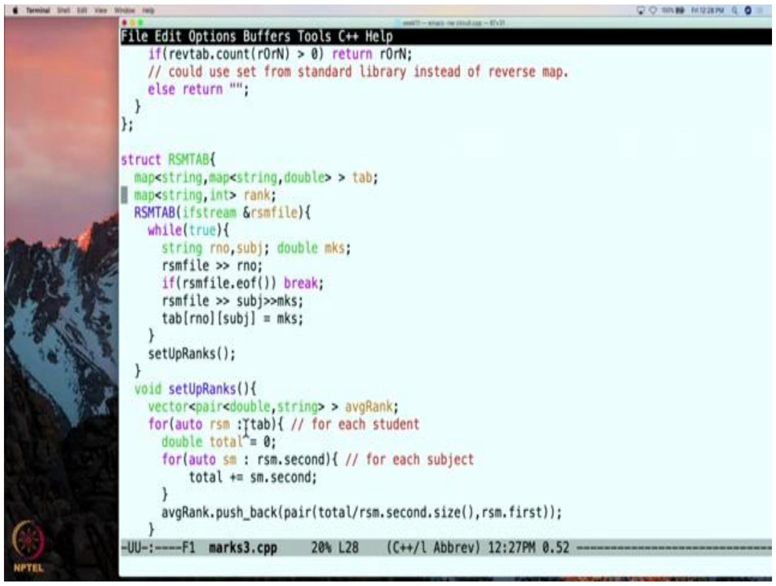
```
      string rno,name;
      rnfile >> rno;
      if(rnfile.eof()) break;
      rnfile >> name;
      tab[name] = rno;
      revtab[rno] = name;
    }
  }
  string lookup(string rOrN){  // returns rollno given name or rollno
    if(tab.count(rOrN) > 0) return tab[rOrN];
    if(revtab.count(rOrN) > 0) return rOrN;
    // could use set from standard library instead of reverse map.
    else return "";
  }
};

struct RSMTAB{
  map<string,map<string,double> > tab;
  map<string,int> rank;
  RSMTAB(ifstream &rsmfile){
    while(true){
      string rno,subj; double mks;
      rsmfile >> rno;
      if(rsmfile.eof()) break;
      rsmfile >> subj>>mks;
      tab[rno][subj] = mks;
    }
    setUpRanks();
-UU-:----F1  marks3.cpp      9% L37    (C++/l Abbrev) 12:35PM 0.45 ----------------------------
```

This is the structure RSMTAB, our 2nd major entity and we said that it contains this 2 level maps called TAB which will be index by, the 1st index being be the roll number, the 2nd being the subject and then this our rank map. So this given roll number will return the rank. So, the construction of RSMTAB is fairly straight forward. I am not going to go through it, but let me just look at how the ranks are going to be created.

So for that we have a member function called setup ranks, so this member function will use a local variable which is a vector and this will contain the pairs, the elements of this vector will be pairs, and they will consist of a double and a string. The 1st element of the pair is supposed to hold the average marks when they are calculated, and the second element of the pair is going to hold the roll number. So, how do we setup the ranks? First, we are going to over the entire RSMTAB, so RSM is going to be an entry in this table. So, essentially we are going over each student, so remember that the first index of this is the roll number. So, what we are picking out over here is an element of it so which means for a fixed roll number, we are going to get the entry. So, we are going to set total to be 0 and we are going to calculate what the total marks are, corresponding to every entry in this. So, the roll number is going to be obtained by taking the first element member of this pair. RSM itself is a pair just like these other pairs. So, the 1st element is this RSM.first so that is the role number and we are going to push on to this average rank, this part is going to be the average, and this part is going to be the rank and this loop is going to be about calculating the average. So, how do we calculate the average? well we are going to set total to 0 and then we are going to go over sm, which is the 2nd part of this RSM.

(Refer Slide Time: 24:47)


~~TAB~~: tab : [rollno][subj]
rsm : tab :
rsm : (rollno , map)
first second
rsm.second : ((subj , marks), (. . .), (..))
sm: (subj, mark)

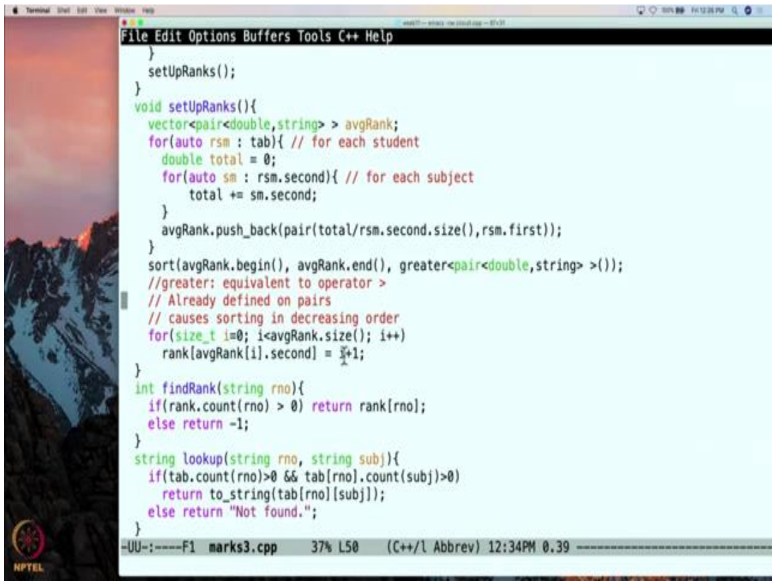


```
File Edit Options Buffers Tools C++ Help
    if(revtab.count(rOrN) > 0) return rOrN;
    // could use set from standard library instead of reverse map.
    else return "";
  }
};

struct RSMTAB{
  map<string,map<string,double> > tab;
  map<string,int> rank;
  RSMTAB(ifstream &rsmfile){
    while(true){
      string rno,subj; double mks;
      rsmfile >> rno;
      if(rsmfile.eof()) break;
      rsmfile >> subj>>mks;
      tab[rno][subj] = mks;
    }
    setUpRanks();
  }
  void setUpRanks(){
    vector<pair<double,string> > avgRank;
    for(auto rsm : tab){ // for each student
      double total = 0;
      for(auto sm : rsm.second){ // for each subject
          total += sm.second;
      }
      avgRank.push_back(pair(total/rsm.second.size(),rsm.first));
    }
-UU-:----F1  marks3.cpp     20% L28    (C++/l Abbrev) 12:27PM 0.52 ------------------------
```

So, I think I should draw picture over here. So, TAB was consisting of pairs, so the pairs we have called RSM so there are several such pairs. And this itself consists of pairs, because it is a 2 level map. So these are sm’s. So RSM consist of pairs, so what does our TAB look like? So TAB or rather I should say our tab look like, so tab you remember takes 1 index which is the roll number and the other index which is the subject.

And therefore, if I look at RSM which is an entry of TAB what does it look like, well this tab is a bunch of pairs, so what are the pairs in it? so each RSM looks like a roll number and then it looks like a map itself. So, this is the 1st and this is the 2nd entry in this pair. So over here, when we fetch an element of this tab, what we are getting is a pair like this. So we have got the roll number, so what we are doing in this loop is we will be calculating the average for a given roll number. And then we are going to determine the average for it and we are going to push it against the roll number. So our average rank vector will get the average over here and the rank over here, so as you can see that is the last step in this 1st procedure. So, in the 1st procedure, at this point, we have only calculated the averages and they are pushed on this vector. So how are they pushed well this part is the roll number and this part is the average and this part is being determined over here. So, let us see how it is determined, so RSM is all this. So, RSM.second is going to be this map, so what is this map, so this is a map from subject to marks. So it is pairs of this kind. So what is happening in this loop is that we are picking up each such pairs, so sm is going to be a pair subject and mark, so sm is this so total, we are going to create a total for this given roll number, and we are going to add sm.second to it. So rsm.second is the mark, so this we are going to add to total, so at this point after we have executed this entire loop, what do we have. Well we have the total marks obtained by a given student in this variable total. So if we want the average, we are going to divide total by RSM.second.size. So what is RSM.second? RSM.second is this entire thing so it is subject marks subject marks subject marks for a given student. So, how many such entries are there, so the number of such entries are the number of subjects. So in fact this quantity is simply is precisely the number of subjects so we will indeed get the average marks over here. And this is the roll number, so at this point this vector average rank will contain the average marks and the roll number.

(Refer Slide Time: 29:44)

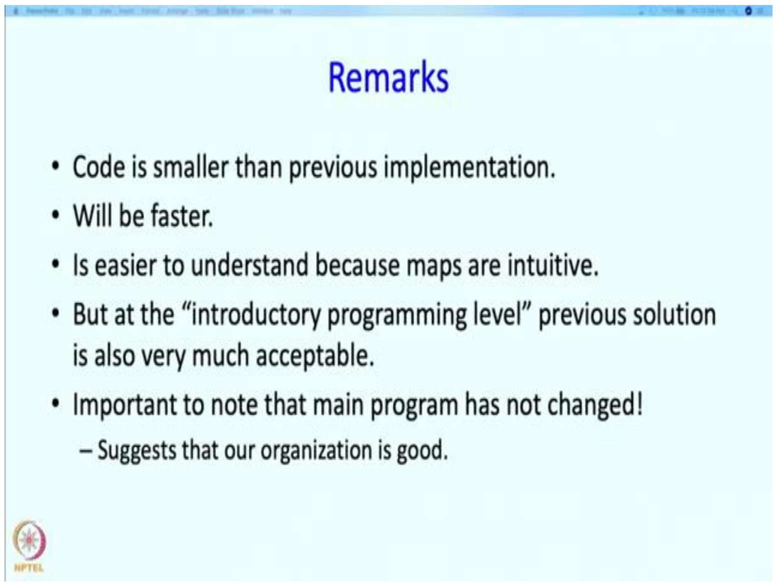


```
File Edit Options Buffers Tools C++ Help
    }
    setUpRanks();
  }
  void setUpRanks(){
    vector<pair<double,string> > avgRank;
    for(auto rsm : tab){ // for each student
      double total = 0;
      for(auto sm : rsm.second){ // for each subject
          total += sm.second;
      }
      avgRank.push_back(pair(total/rsm.second.size(),rsm.first));
    }
    sort(avgRank.begin(), avgRank.end(), greater<pair<double,string> >());
    //greater: equivalent to operator >
    // Already defined on pairs
    // causes sorting in decreasing order
    for(size_t i=0; i<avgRank.size(); i++)
      rank[avgRank[i].second] = i+1;
  }
  int findRank(string rno){
    if(rank.count(rno) > 0) return rank[rno];
    else return -1;
  }
  string lookup(string rno, string subj){
    if(tab.count(rno)>0 && tab[rno].count(subj)>0)
      return to_string(tab[rno][subj]);
    else return "Not found.";
  }
-UU-:----F1  marks3.cpp     37% L50    (C++/l Abbrev) 12:34PM 0.39 ---------------------------
```

Next we are going to sort this vector; we are going to sort the average mark vector. So if we had just said the usual things sort average rank begin to end then we would have got the sorting in increasing order, we want it decreasing, so we pass it the function greater, the greater is equivalent to the operator greater than. So, greater than if we pass the greater than operator over here then we are going to get sorting in decreasing order. Otherwise by default, if we do not say anything, the less than operator get passed over here and therefore we get the usual sorting but we want the other sorting and since we want decreasing order, we pass the greater than operator and I said earlier that all such operators are already defined on pairs. And they are defined in sort of lexicographic order that is the 1st sort on this field and then they sort on this field. So if this field is different then they sort on this field. But what is this field, this is the average and so at the end of it this vector will be sorted on the average. So it is sorted on the average, so all I have to do, is to go through this vector, so we are going to examine every element of this vector and if I come to an entry of this vector, will the Ith entry of this vector has 2 parts. 1st part is going to be the average marks, and the 2nd part is going to be the rank associated with that average marks. But I do not care about the average marks I care about the roll number and since this roll number averageRank[i].second appears at position I in this vector I know that its rank must be I, if I count ranks starting at 0 but I want to count starting at I plus at1 and therefore its rank I am going to set to be equal to us I plus 1. So, what now I get is that I have built up my rank table as well over here in my RSMTAB I had this rank map, and I have built it up as well.

So, that is what we need to know about this RSMTAB and our find rank function is now extremely simple. The calculation was already done, and so if the roll number appears in this as an index in this array we just return there and we just already calculated. Otherwise we return minus 1. I guess would have even here given that use of protocol of returning the string and we could have said not found.

(Refer Slide Time: 33:11)



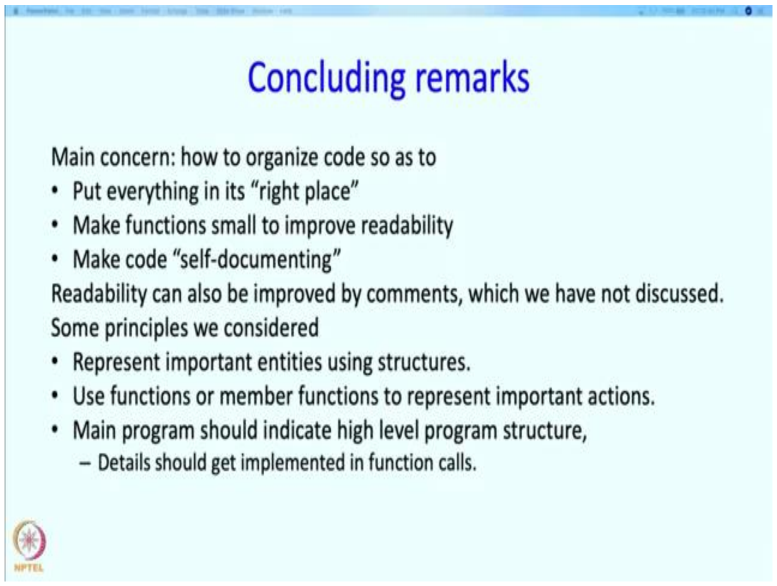
Remarks

- Code is smaller than previous implementation.
- Will be faster.
- Is easier to understand because maps are intuitive.
- But at the "introductory programming level" previous solution is also very much acceptable.
- Important to note that main program has not changed!
  - Suggests that our organization is good.

But that is an improvement that you certainly should make to this code. So let us return to our presentation. So as you can see the code is actually smaller than the previous implementations and that is because the library codes are not only faster but it are also a lot more compact. The member functions for maps do a lot of interesting work, they effectively do something like sorting and it is easier to understand actually because in some sense maps are intuitive I mean the notion of mapping an index to value this is quite easy to understand.

So, if you are having a difficulty with maps that is ok, in some sense, maps are a bit of an optional part of this whole course. But I think many people will understand maps, and if you do I think you should see how powerful they are and you should start using them when you code. And by the way, in this entire thing I have used maps but I could have used unordered maps as well. So, unordered maps will also be perfectly fine for all of this. And I should again point out that the main program has not changed, so in some sense this means that our decomposition of our logic into high level and low level seems like a good decomposition.

(Refer Slide Time: 34:47)



Concluding remarks

Main concern: how to organize code so as to
- Put everything in its "right place"
- Make functions small to improve readability
- Make code "self-documenting"

Readability can also be improved by comments, which we have not discussed.

Some principles we considered
- Represent important entities using structures.
- Use functions or member functions to represent important actions.
- Main program should indicate high level program structure,
  - Details should get implemented in function calls.

So let me make some concluding remarks, so the main concern in this entire lecture was how to organize code so as to put everything in its right place. Why should everything be in its right place, well it is like why everything in the room should be in its right place because if it is in its right place, we know where we should be searching, we know where we kept it, we know where we will find it that kind of thing.

Then we should make functions small to improve readability, we should make code self documenting, well of course we can write comments, but we have not discussed that but better than that is to write the code in a very obvious manner. So give the names of the functions, give the names of variables nicely so that the logic is very obvious. In addition to that, we considered some additional principles. So we set something like represent important entities using structures. When we said that if there is an important action, then you should have a named function doing it, or a named member function doing it because then it is giving importance to that piece of code and giving it a name which helps in readability and we also said that the main program should indicate high level program structure, and the detail should get implemented in function calls.

So, that concludes this lecture and you are invited to go over the code and you are invited to make the code even more readable. And certainly you really should develop a taste and you do not have to take my taste, but you should develop a taste and you should have a notion of what is the right way to write this piece of logic, and you should try to develop a consistent style so that at least you yourself will be able to find your code and hopefully it will be easier for other people also to understand your code and find things in it very easily. So we will conclude with that and Thank you.