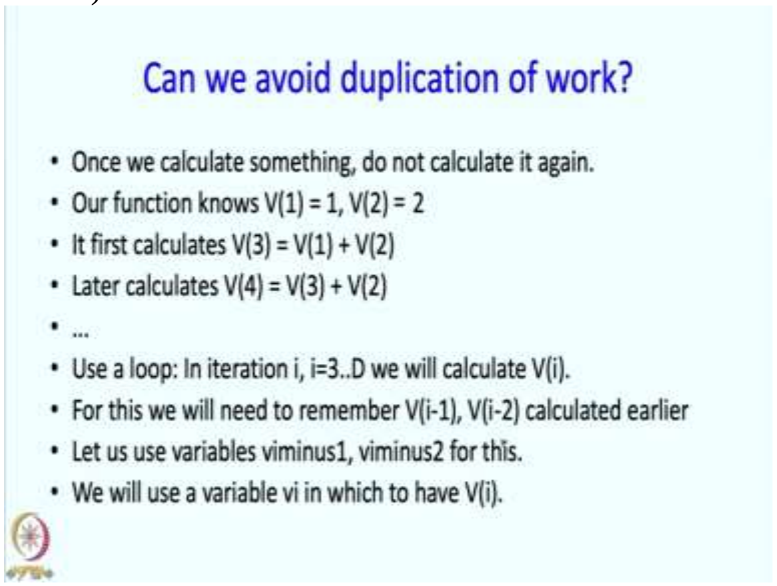
# An Introduction to Programming through C++
**Professor Abhiram G. Ranade**
**Department of Computer Science and Engineering**
**Indian Institute of Technology, Bombay**
**Lecture 12 Part 3 – Virahanka Numbers (Iterative Program and Conclusion)**

Welcome back. In the last segment, we discussed recursive function to calculate Virahanka numbers and we concluded that it was taking too much time and that was because we were seeming to repeat work. In this segment we are going to see how to avoid that. So, what exactly were we doing? And what should we do? Well, we should really not calculate something more than once. Okay?

(Refer Slide Time: 0:48)



## Can we avoid duplication of work?

- Once we calculate something, do not calculate it again.
- Our function knows V(1) = 1, V(2) = 2
- It first calculates V(3) = V(1) + V(2)
- Later calculates V(4) = V(3) + V(2)
- ...
- Use a loop: In iteration i, i=3..D we will calculate V(i).
- For this we will need to remember V(i-1), V(i-2) calculated earlier
- Let us use variables viminus1, viminus2 for this.
- We will use a variable vi in which to have V(i).

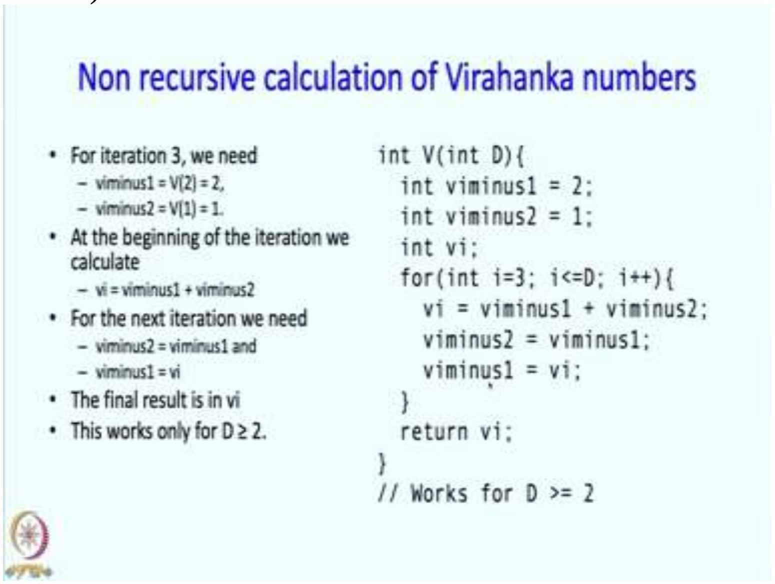
So, to begin with our function knows V(1) is 1 and V(2) is 2. So, our function was V(1) and V(2). Okay. It then calculates V(3) and V(3) is calculated using V(1)+V(2). Okay. So, after that you can calculate V(4) is V(3)+V(2).But notice that when we when we issue this call V(4), we do not really need to recalculate V(3). If we could somehow remember what we, that we calculated V(3) earlier then we could just use that value and V(2) of course we know. So, in general we could think of doing this in a loop. So, in iteration i, where i goes from 3 to D, we will calculate V(i) because for i equals 1 and 2, there is nothing to be done. Those values are already known.

And how do we calculate V(i)? Well, V(i) requires us to take V(i-1) which we must have calculated in the previous iteration and V(i-2) which we must have calculated in the iteration previous to that and add up these two values. So, which means these values we could print out if we wish. But we do not really need to print them out, if we are only interested in the final V(D). But in any case, in i th iteration we should be remembering these values. These values should be there so that we can add them up and we can generate V(i).

So, to keep these values V(i-1) and V(i-2), we will have two variables. So, we will have a loop. We are now going to have a loop and in that loop, we are going to have 2 variables and at the beginning of the i th iteration these variables will contain V(i-1) and V(i-2). And we will also use a variable V(i) in which we will construct V(i).

So, V(i) is a variable, V(i-1), V(i-2) are going to be our variables and the important point is that the beginning of the iteration i we want V(i-1) and V(i-2) to contain V(i-1) and V(i-2) which were calculated earlier.

(Refer Slide Time: 3:41)



## Non recursive calculation of Virahanka numbers

- For iteration 3, we need
  - viminus1 = V(2) = 2,
  - viminus2 = V(1) = 1.
- At the beginning of the iteration we calculate
  - vi = viminus1 + viminus2
- For the next iteration we need
  - viminus2 = viminus1 and
  - viminus1 = vi
- The final result is in vi
- This works only for D ≥ 2.

```
int V(int D){
  int viminus1 = 2;
  int viminus2 = 1;
  int vi;
  for(int i=3; i<=D; i++){
    vi = viminus1 + viminus2;
    viminus2 = viminus1;
    viminus1 = vi;
  }
  return vi;
}
// Works for D >= 2
```

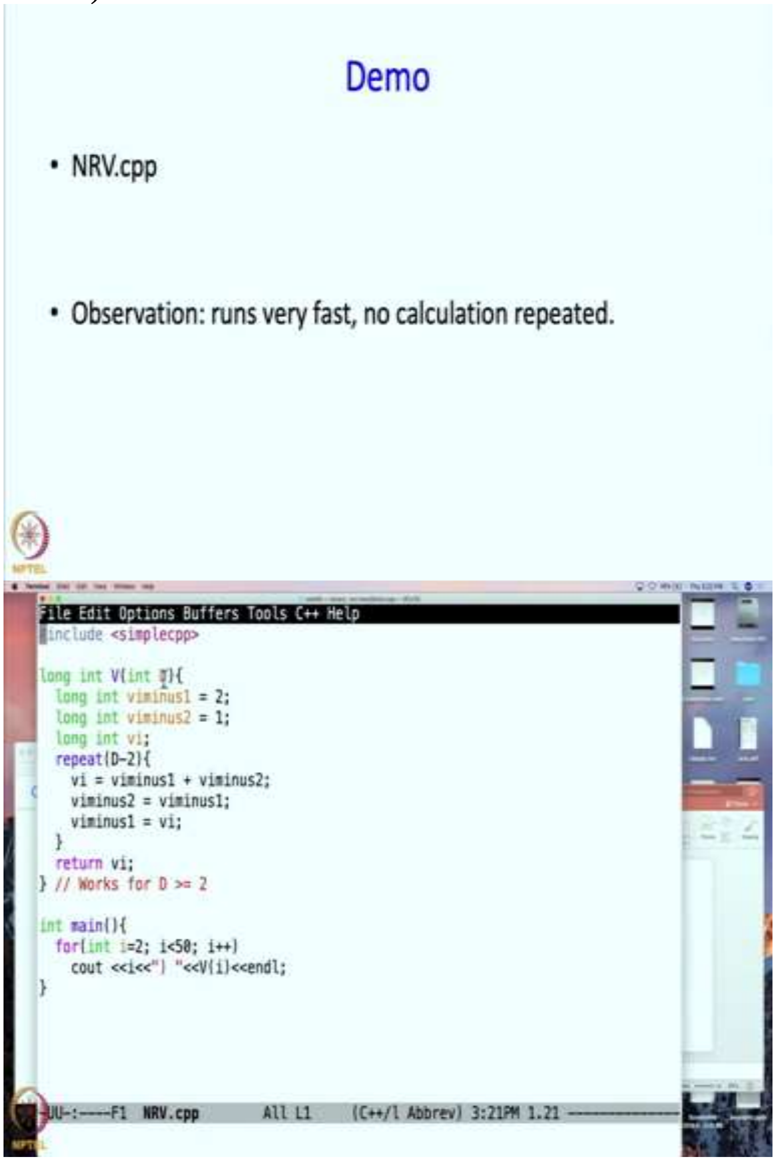
So, based on this we can create this program. So, this program is going to calculate V(D). So, as we said we are going to have iterations going from 3 to D and at the beginning V(i-1) and V(i-2) must have the values V(i-1), V bracket i minus 1 and V bracket i minus 2, so those values must be there. And then we add them up to get the value of V(i).

Now I have to fill up, fill up the rest of the code over here. So, let us see, so in the iteration 3 when we first enter this loop, what do we need? So, we need, so we are going to calculate V(3), i is going to be 3 so we want to calculate V(3) over here. So, at that point we need this to be V(2) and we need this to be V 1, so these values will get set only above outside the loop. So therefore, we should initialize these variables to 2 and 1 respectively. And then when we go the next iteration of the loop, what happens? So, first of all in this iteration, what happens? In the beginning we calculate this which is as far as the plan we were talking about on the last slide and then we have to prepare for the next iteration.

So, for the next iteration what is it that we want? Well, the value that was V(i-1) in the current iteration, so is the value of calculated in the previous iteration. But now for the next iteration that will be the value calculated in the second previous iteration. So, that should really go to V(i-2) and V(i-1) for the next iteration is the value that was calculated in the current iteration and therefore that must become V(i). So, our loop must be this. And finally, what should be returned?

So, at the end of this loop V(i) will contain V(D) and therefore we should return V(i) over here, okay. Now if you look at this code you will see that this code works only for D greater than 2. Basically, if D is equal to 1, what happens? This V(i) is not even defined. So, but for for the other D it works fine and so, let us just put a comment to that effect.

(Refer Slide Time: 6:08)



## Demo

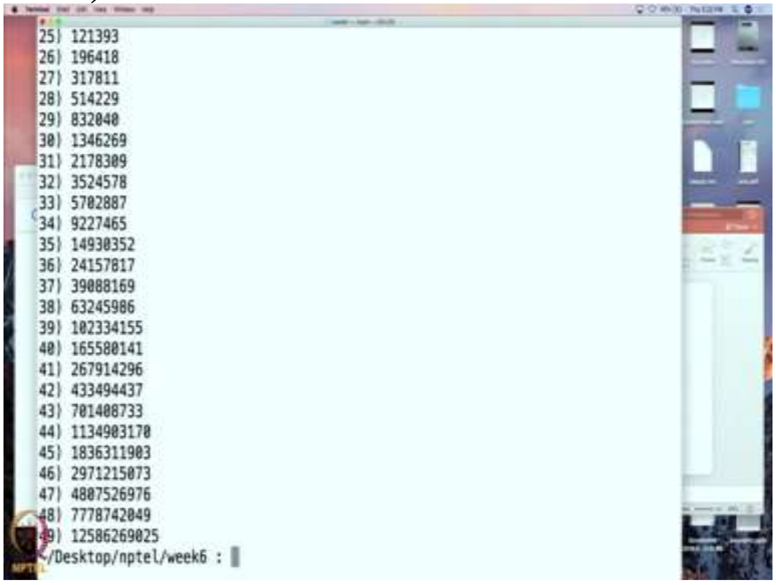
- NRV.cpp
- Observation: runs very fast, no calculation repeated.

```
#include <simplecpp>

long int V(int D){
  long int viminus1 = 2;
  long int viminus2 = 1;
  long int vi;
  repeat(D-2){
    vi = viminus1 + viminus2;
    viminus2 = viminus1;
    viminus1 = vi;
  }
  return vi;
} // Works for D >= 2

int main(){
  for(int i=2; i<50; i++)
    cout <<i<<") "<<V(i)<<endl;
}
```

```
UU-:----F1  NRV.cpp       All L1     (C++/l Abbrev) 3:21PM 1.21 ---------------
```

Alright. Let us see. Let us run that program and let us see what happens with it. So, this is the program, okay. So again, the function is what I just showed you, okay. And in the main program I am going to have the same thing again. I am going to go from 2 to 50 as before. I did not go from go with 1 because we know that this program does not work for 1, okay. And as you can see again to accommodate the large numbers, I have had long ints, okay. D does not have to be long because D does not contain long numbers, okay. So, the arguments to D does not contain long numbers. So, that does not have to be long.

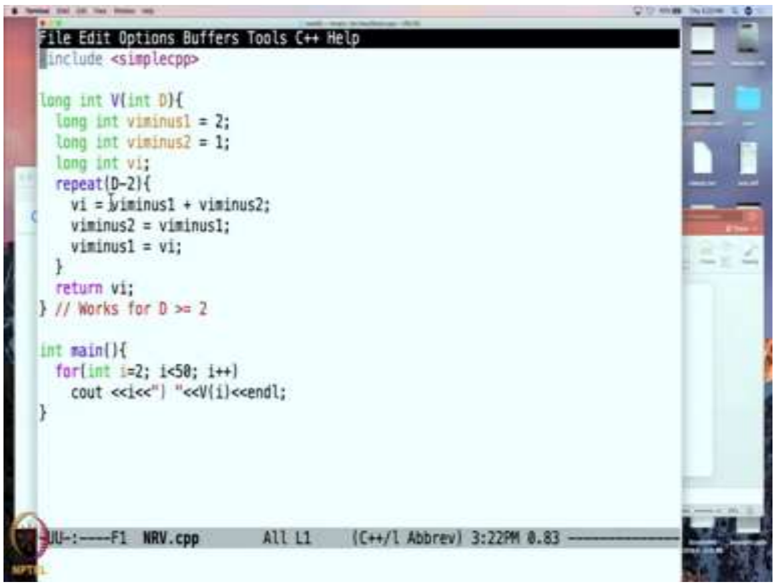
(Refer Slide Time: 6:49)



```
25) 121393
26) 196418
27) 317811
28) 514229
29) 832040
30) 1346269
31) 2178309
32) 3524578
33) 5702887
34) 9227465
35) 14930352
36) 24157817
37) 39088169
38) 63245986
39) 102334155
40) 165580141
41) 267914296
42) 433494437
43) 701408733
44) 1134903170
45) 1836311903
46) 2971215073
47) 4807526976
48) 7778742049
49) 12586269025
~/Desktop/nptel/week6 :
```

So, let us compile this. So, this time it produced an answer instantaneously essentially, so as soon as I hit return the answer was there. There was no waiting at all. And that is to be expected because in each iteration of the loop we are doing really very little work.

(Refer Slide Time: 7:20)

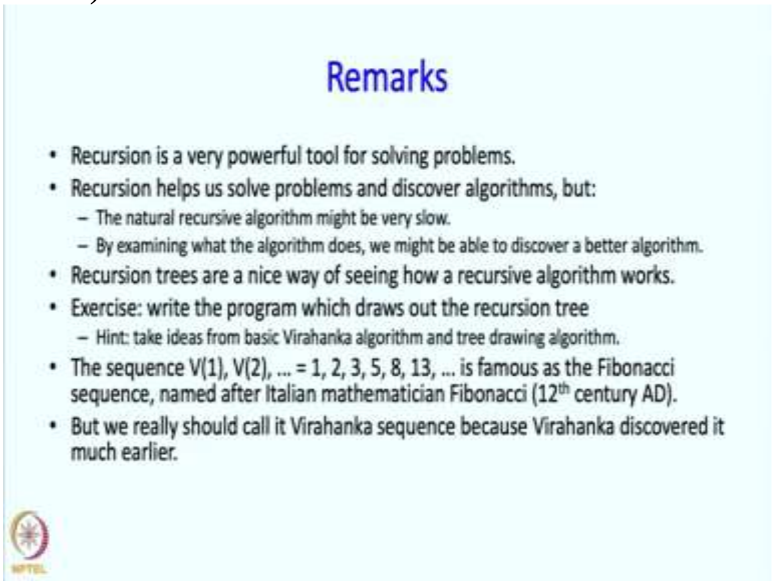


```
#include <simplecpp>

long int V(int D){
  long int viminus1 = 2;
  long int viminus2 = 1;
  long int vi;
  repeat(D-2){
    vi = viminus1 + viminus2;
    viminus2 = viminus1;
    viminus1 = vi;
  }
  return vi;
} // Works for D >= 2

int main(){
  for(int i=2; i<50; i++)
    cout <<i<<") "<<V(i)<<endl;
}
```

```
UU-:----F1  NRV.cpp       All L1     (C++/l Abbrev) 3:22PM 0.83
```

So, so there is this repeat and in this, oh, yeah so there is a slight change over here, we really did not need those i's and therefore we can also use repeat over here. So, you can compare these two codes and see that they are really the same. But I could have put for here as well, does not really matter. But the point is that we are having iterations of this loop and the time required for each value, for each V(i) is exactly the same and therefore the loop runs really fast, okay. So, we have done this and we were able to get V(D) to be calculated very fast.

(Refer Slide Time: 8:09)



## Remarks

- Recursion is a very powerful tool for solving problems.
- Recursion helps us solve problems and discover algorithms, but:
  - The natural recursive algorithm might be very slow.
  - By examining what the algorithm does, we might be able to discover a better algorithm.
- Recursion trees are a nice way of seeing how a recursive algorithm works.
- Exercise: write the program which draws out the recursion tree
  - Hint: take ideas from basic Virahanka algorithm and tree drawing algorithm.
- The sequence V(1), V(2), ... = 1, 2, 3, 5, 8, 13, ... is famous as the Fibonacci sequence, named after Italian mathematician Fibonacci (12th century AD).
- But we really should call it Virahanka sequence because Virahanka discovered it much earlier.

So, what do we learn from this? So, we can see that recursion is a very powerful tool for solving problems. Recursion helps us solve problems and discover algorithms, but the natural recursive algorithm might be very slow. By examining the algorithm, by seeing what it does we might be able to discover a better algorithm. Then we also saw that the recursion trees are a nice way of seeing how a recursive algorithm works.

And I will leave you with an exercise: Write the program which draws out the recursion tree. So, for this you need to take the ideas from the basic recursive Virahanka algorithm and also the tree drawing algorithm. And you sort of have to fuse these two programs together, okay.

Now this sequence, these Virahanka numbers V(1), V(2), V(3) which are 1, 2, 3, 5, 8, 13 is actually a famous sequence and it is well known as the Fibonacci sequence named after the Italian mathematician, Fibonacci who lived, Fibonacci or Fibonacci, who lived in 12th century AD. But Virahanka discovered it much earlier and we probably should be calling it the Virahanka sequence or call these numbers Virahanka numbers, isn’t it?

So, with that we will conclude this lecture sequence noting some rather interesting facts about recursion. Thank you.