An Introduction to Programming through C++

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

Lecture 10: Part-2
Functions Examples

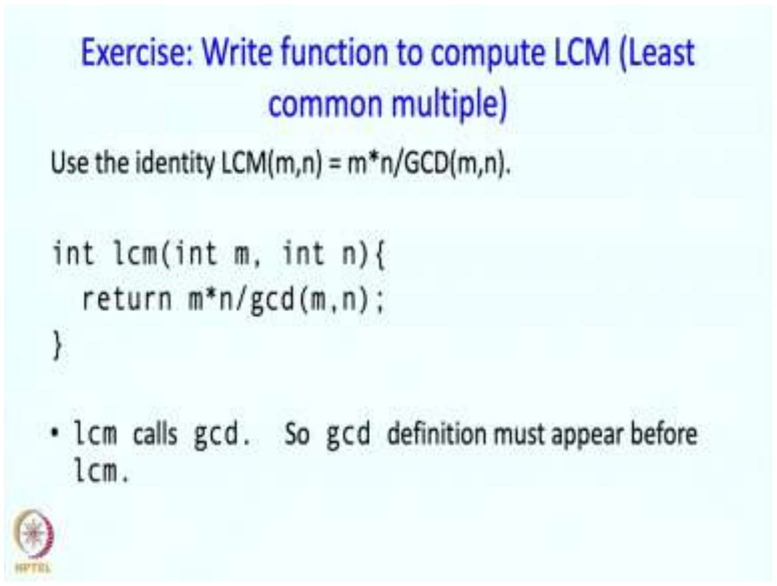
(Refer Slide Time: 00:21)

What we have discussed

- What is a function
- How to define a function
- How it executes
- Next: More examples and how to think of functions

So, in the last segment we discussed what is a function how to define it, and how it executes. Now, we are going to start by taking more examples.

(Refer Slide Time: 0:29)



Exercise: Write function to compute LCM (Least common multiple)

Use the identity LCM(m,n) = m*n/GCD(m,n).

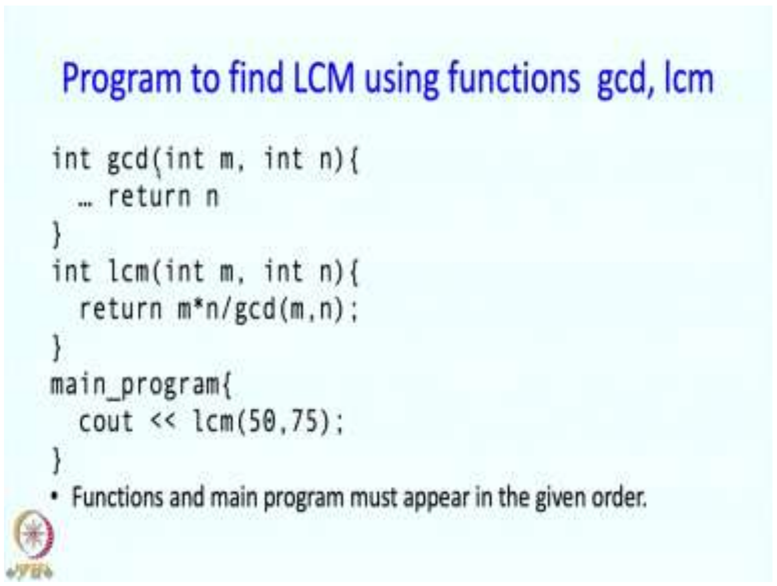
```
int lcm(int m, int n){
  return m*n/gcd(m,n);
}
```

- `lcm` calls `gcd`. So `gcd` definition must appear before `lcm`.

So, another exercise, so, let us write a function to compute the LCM or the least common multiple. So, here we are going to write this code but we are going to use the identity that the LCM of 2 numbers and m and n is their product divided by their greatest common divisor. So, now, we already know how to calculate the greatest common divisor, so we might say that why do not we use this and calculate LCM as well.

So, here is the code, we are going to write int LCM of m and n, it is going to take two arguments so it will have 2 parameters m and n and we are going to just return that expression, so return just that, that is it, and, of course, this assumes that we have a GCD already defined and where should that be defined, it should be defined before it is used. So it should, so the definition should come above LCM.

(Refer Slide Time: 01:34)



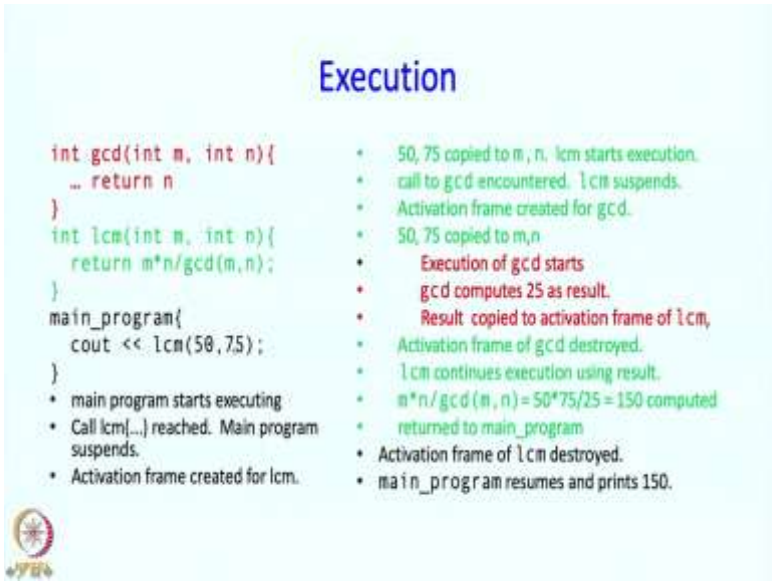
Program to find LCM using functions gcd, lcm

```
int gcd(int m, int n){
  … return n
}
int lcm(int m, int n){
  return m*n/gcd(m,n);
}
main_program{
  cout << lcm(50,75);
}
```

- Functions and main program must appear in the given order.

So, your code should look something like this, so first in your file, you should have a GCD program, then you should have the LCM program and I have not given the full GCD program, it is a little bit long but, it will return n eventually and I am. So n, of course, with the value of n will have changed inside this, and here I am going to, it is going to return the LCM and then I am going to have the main program. So, the main program is going to make call on lcm, lcm will make a call on gcd which is given over here.

(Refer Slide Time: 02:13)



Execution

```
int gcd(int m, int n){
  … return n
}
int lcm(int m, int n){
  return m*n/gcd(m,n);
}
main_program{
  cout << lcm(50,75);
}
```

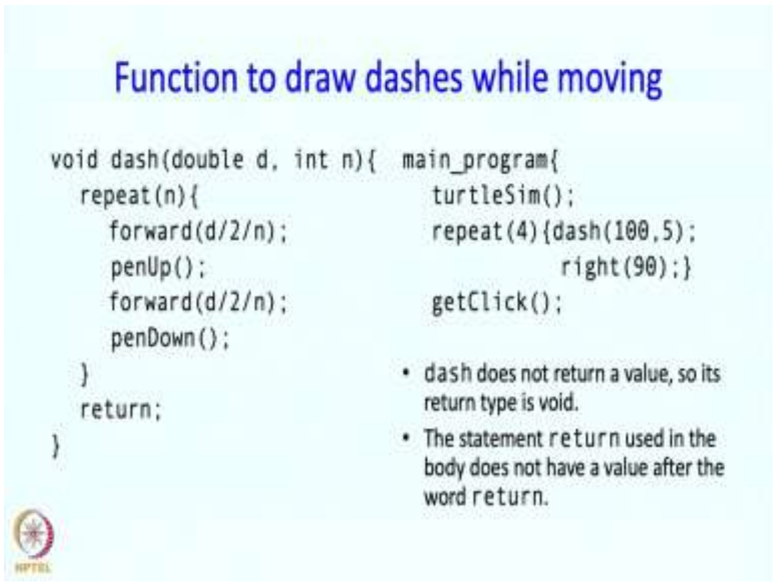
- main program starts executing
- Call lcm(...) reached. Main program suspends.
- Activation frame created for lcm.
- 50, 75 copied to m, n. lcm starts execution.
- call to gcd encountered. lcm suspends.
- Activation frame created for gcd.
- 50, 75 copied to m,n
  - Execution of gcd starts
  - gcd computes 25 as result.
  - Result copied to activation frame of lcm,
- Activation frame of gcd destroyed.
- lcm continues execution using result.
- m*n/gcd(m,n) = 50*75/25 = 150 computed
- returned to main_program
- Activation frame of lcm destroyed.
- main_program resumes and prints 150.

So, how does this execute. so let see that. So, the execution begins with the main program so the main program starts executing. So, it encounters this statement and it say see there is an LCM to be computed. So, since, there is an LCM to be computed, the main program suspends and an activation frame is crated for lcm, then 50 and 75 are copied to m and n, so 50 and 75 are copied to these m and n and lcm starts execution.

Now, the call to gcd as this executes, the call to gcd is encountered, this execution of lcm tries to construct this value and it encounters a call to gcd, so because of that lcm is suspended. So now, note that we have two things suspended, we have the main program suspended and we have the lcm suspended and at this point the activation frame for gcd is created because this value has to be calculated. So, 50 and 75, these values are copied to these m and n, so in the activation frame of gcd. So execution of gcd starts, gcd computes the correct result the GCD of 50 and 75 which is 25, so gcd computes this result and the result is copied to the activation frame of where this came from. So, this call came over here, so 25 gets copied over here. So, now, gcd execution is complete and so the activation frame of gcd goes away and lcm execution resumes.

So, lcm continues execution so sorry this, so m times n divided by 25 is calculated, so 50 times 75 divided by 25 which is 150 is calculated, but this is a return statement so this value gets copied in place of this expression, so the value is returned to the main program and the activation frame of lcm is now destroyed. So, remember what happened, so main program had its activation frame and activation frame was created for lcm and activation frame was created for gcd. The activation frame or gcd was destroyed, the activation frame of lcm was destroyed and the value 150 was returned from here, and it is in at place. So at this point the main program resumes execution and prints 150, and then it exits.

(Refer Slide Time: 04:57)



Function to draw dashes while moving

```
void dash(double d, int n){
  repeat(n){
    forward(d/2/n);
    penUp();
    forward(d/2/n);
    penDown();
  }
  return;
}
```

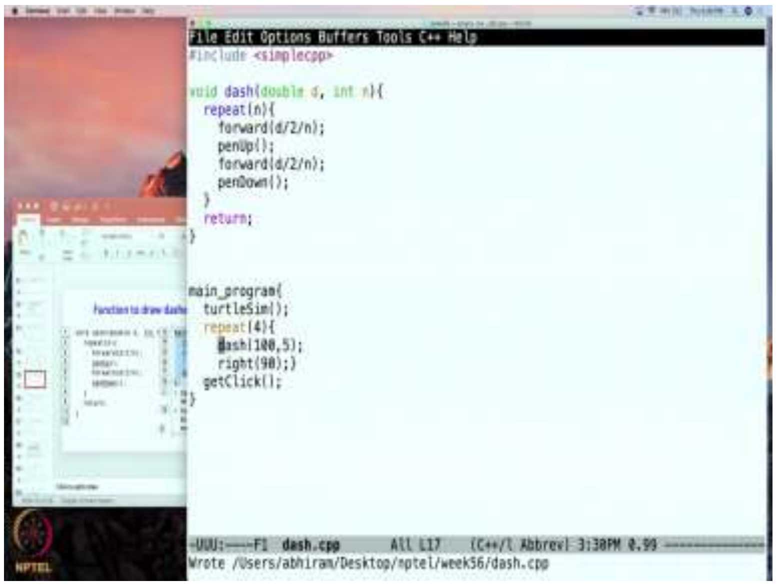
```
main_program{
  turtleSim();
  repeat(4){dash(100,5);
            right(90);}
  getClick();
```

- dash does not return a value, so its return type is void.
- The statement return used in the body does not have a value after the word return.

So one more quick function, so we want to draw dashes while moving, so we have over turtle and normally if you just say forward it draws a line so now, we are going to draw dashes. So here is a function. We are going to have, the name is going to be dash and we are going to say how many pixels we want to move, so we want to move forward d pixels, and we want to say over here, how many dashes should there be so this distance is going to be divided in to these many dashes. So, how does this work, so this first of all this function, let me observe is not going to return a value it is going to cause some drawing to happen. So, because it is not returning a value we are going to state its return type as being void, if you wanted to return a value in addition to drawing something we could have put that type over here, but as it is we do not, we are not interested in returning a value. So we want to draw 10 dashes or whatever n dashes so we are going to have a repeat statement, so repeat n times and we are going to go forward, d by 2n so a distance d by 2n we are going to go forward, and during this we are going assume at the beginning that the pen is down so a line will, be drawn. Now, at this point we will raise the pen so at this point again we will start going forward so we will have forward d by 2n again so this time the turtle will move the same distance but, with the pen up. So, no line will be drawn. Then we will put the pen down again and that would be the end of the loop. So this whole thing would be executed n times and so the turtle will cover d over 2 distance twice in each iteration, so d distance in, d by n distance in each iteration so overall it will indeed cover d distance.

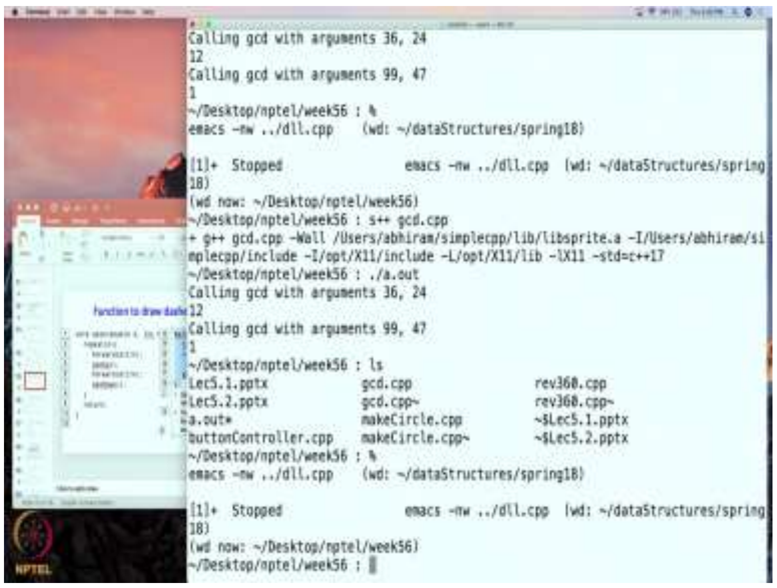
And at the end it will return. So, here note that we have not set return something, so here, we are not expecting to return anything and therefore, we just say return followed by semicolon. So that is it, so the main program could look something like this, turtleSim repeat(4) dash 100, 5 and right 90, so this should be drawing a square we have put and get click just to see that, that square has been drawn. Alright, so let me suspend the program over here and let us see this, dashes program in execution.

(Refer Slide Time: 07:56)

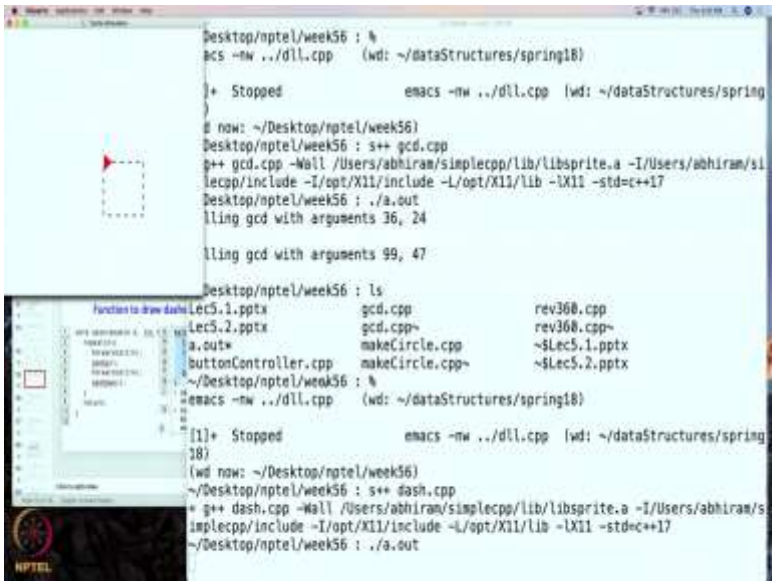


```
#include <simplecpp>

void dash(double d, int n){
  repeat(n){
    forward(d/2/n);
    penUp();
    forward(d/2/n);
    penDown();
  }
  return;
}


main_program{
  turtleSim();
  repeat(4){
    dash(100,5);
    right(90);}
  getClick();
}
```

```
-UUU:----F1  dash.cpp       All L17    (C++/l Abbrev) 3:30PM 0.99 ---------------
Wrote /Users/abhiram/Desktop/nptel/week56/dash.cpp
```



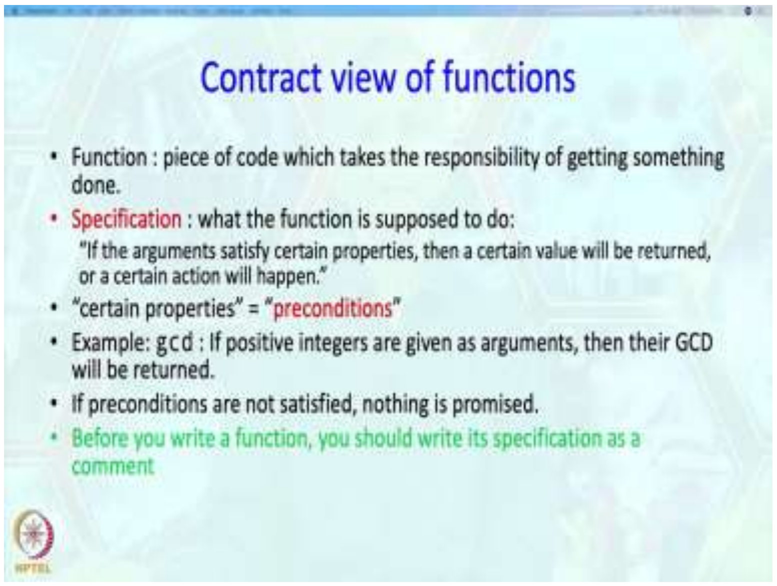
```
Calling gcd with arguments 36, 24
12
Calling gcd with arguments 99, 47
1
~/Desktop/nptel/week56 : %
emacs -nw ../dll.cpp    (wd: ~/dataStructures/spring18)

[1]+  Stopped                 emacs -nw ../dll.cpp  (wd: ~/dataStructures/spring
18)
(wd now: ~/Desktop/nptel/week56)
~/Desktop/nptel/week56 : s++ gcd.cpp
+ g++ gcd.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/si
mplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week56 : ./a.out
Calling gcd with arguments 36, 24
12
Calling gcd with arguments 99, 47
1
~/Desktop/nptel/week56 : ls
Lec5.1.pptx             gcd.cpp                 rev360.cpp
Lec5.2.pptx             gcd.cpp~                rev360.cpp~
a.out*                  makeCircle.cpp          ~$Lec5.1.pptx
buttonController.cpp    makeCircle.cpp~         ~$Lec5.2.pptx
~/Desktop/nptel/week56 : %
emacs -nw ../dll.cpp    (wd: ~/dataStructures/spring18)

[1]+  Stopped                 emacs -nw ../dll.cpp  (wd: ~/dataStructures/spring
18)
(wd now: ~/Desktop/nptel/week56)
~/Desktop/nptel/week56 : 
```



```
Desktop/nptel/week56 : %
acs -nw ../dll.cpp    (wd: ~/dataStructures/spring18)

]+  Stopped                 emacs -nw ../dll.cpp  (wd: ~/dataStructures/spring
)
d now: ~/Desktop/nptel/week56)
Desktop/nptel/week56 : s++ gcd.cpp
g++ gcd.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/si
lecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
Desktop/nptel/week56 : ./a.out
lling gcd with arguments 36, 24

lling gcd with arguments 99, 47

Desktop/nptel/week56 : ls
Lec5.1.pptx             gcd.cpp               rev360.cpp
Lec5.2.pptx             gcd.cpp~              rev360.cpp~
a.out*                  makeCircle.cpp        ~$Lec5.1.pptx
buttonController.cpp    makeCircle.cpp~       ~$Lec5.2.pptx
~/Desktop/nptel/week56 : %
emacs -nw ../dll.cpp    (wd: ~/dataStructures/spring18)

[1]+  Stopped                 emacs -nw ../dll.cpp  (wd: ~/dataStructures/spring
18)
(wd now: ~/Desktop/nptel/week56)
~/Desktop/nptel/week56 : s++ dash.cpp
+ g++ dash.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/s
implecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week56 : ./a.out
```

So here, I have the dash program so let us compile it and run it, so see the turtle draw a dash rectangle we can finish we can stop the execution by clicking and that is it. So, at this point we have seen several examples of functions, and now, I want to say something about how you should think about functions.

(Refer Slide Time: 08:32)



Contract view of functions

- Function : piece of code which takes the responsibility of getting something done.
- Specification : what the function is supposed to do:
  "If the arguments satisfy certain properties, then a certain value will be returned, or a certain action will happen."
- "certain properties" = "preconditions"
- Example: gcd : If positive integers are given as arguments, then their GCD will be returned.
- If preconditions are not satisfied, nothing is promised.
- Before you write a function, you should write its specification as a comment

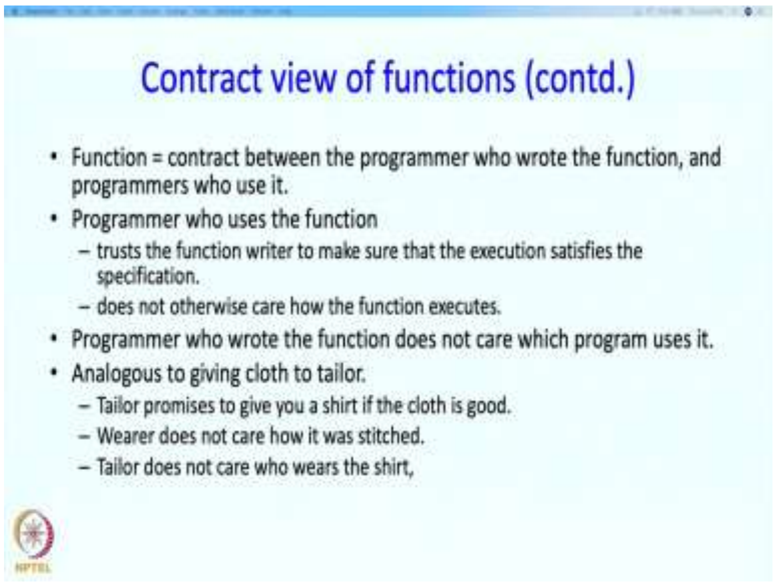
So a function if you think of as a peace of code which takes the responsibility of getting something done. The specification is a description of what the function is supposed to do, so typically the specification look something like this, "if the argument satisfies certain properties then a certain value will be returned, or a certain action will happen," that is what the specification is going to look like.

The certain properties are sometimes called preconditions, so, for example, for gcd, we might say that if positive integers are given as arguments, then their GCD will be returned, so this is a specification for gcd. If precondition is not satisfied so if you, for example, give negative values, then what happens well, nothing is promised. The program may run forever, the program may return some non-sense, of course, but nothing is guaranteed to you.

The writer of the function gcd is not making any claim about what will happen, you could say that, if the values supplied are incorrect, then the function should say so, but that is not really always required, so the function writer has to say that these are my preconditions, so it is your responsibility to check if the arguments satisfy the preconditions, if not, I do not know what is going to happen. But, you could have other function definitions in which there are no preconditions because if the wrong arguments are given, the function may take some action and return some value saying that, "Oh, you gave me wrong arguments." Before you write a function, you should write its specification as a comment in the code, so this is just so that you are sure yourself that you know what it is that you want to do.

So, there is a certain kind of humility that you have to have when you write programs and I think most programmer have that humility because they might write programs and might claim that, “Oh this program is going to do this,” but actually when they execute they find that “No, the program does something else,” and then they have to spend lots of time figuring out why the program made a mistake. So, because of this humility it is a good idea that you start up the whole process saying, “Let me be clear to myself what is it that, my program is supposed to accomplish, and therefore, let me first write down the specification in as clear words as possible.” Now all this is really taking you towards some kind of a contract view, what does it mean, the writer of the program is promising something.

(Refer Slide Time: 11:43)



Contract view of functions (contd.)

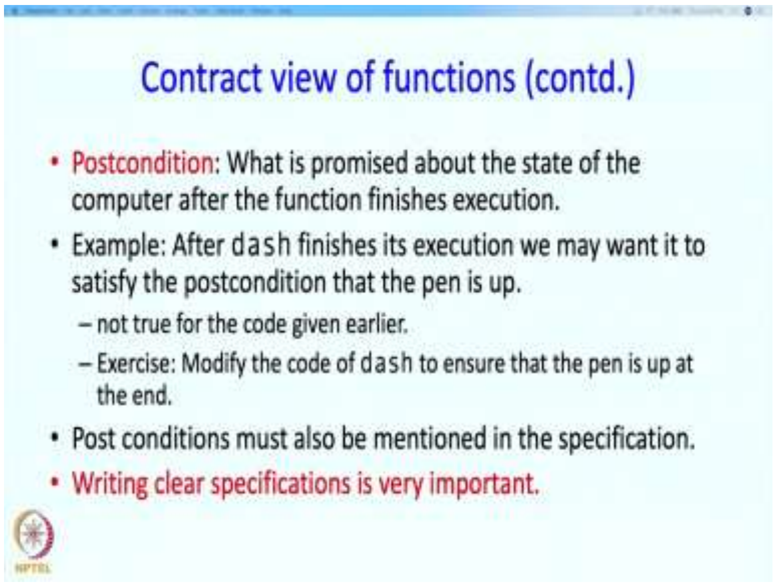
- Function = contract between the programmer who wrote the function, and programmers who use it.
- Programmer who uses the function
  - trusts the function writer to make sure that the execution satisfies the specification.
  - does not otherwise care how the function executes.
- Programmer who wrote the function does not care which program uses it.
- Analogous to giving cloth to tailor.
  - Tailor promises to give you a shirt if the cloth is good.
  - Wearer does not care how it was stitched.
  - Tailor does not care who wears the shirt,

So a function is a contract between the programmer who wrote the function, and the programmers who use the function, sometimes both of these parties could be the one and the same party but, they need not, so if they are not, then there is the contract view sort of make more sense, the programmer who uses the function says that “Look, I trust the function writer to make sure that the execution satisfies the specification.” So when I make the call I am not going to worry about how the function is actually going to execute, that is not my job, so the whole programming process is sort of to do things like this. So, once I write a function subsequently when I use it myself I am not going to worry about how that function executes. Because when I wrote the function, when I tested it, I have satisfied myself that it runs correctly and from that point on wards, I am not going to worry about how it works internally.

And if it is some other party's function, that some other party wrote that function, then, of course, I am not going to care how that other party wrote the function, but, in fact, I am not going to care even if I wrote it myself because when I wrote it I would have tested it and have sort of made sure that it works correctly so now I do not want to be bothered with it.

Programmer who wrote the function, does not care which program uses it, so I write a function, when I am writing the function I am making, trying to make sure, that it is going to run correctly no matter who uses it, no matter what arguments are supplied to it, so it sort of like giving cloth to a tailor, so when I give the clock the tailor promises to give me a good shirt, if the cloth is good, and the wearer does not care how the shirt was stitched, and the tailor does not care who wears it, so it is a contract between them, that I give you cloth you give me shirt. Similarly, where a function, it is a contract between the function writer and the function user. So, the contract is made so that the responsibilities are clearly delineated and each party does not reduce, it is meant to reduce the worry for each party so, the user does not worry about how exactly the function was written, how it executes, and the party that wrote the function, does not worry about where is it going to get used. Because while writing the has made sure that it is going to pick correct in all cases. When the function finishes something is promised and in particular something might be promised about the state of the computer.

(Refer Slide Time: 14:51)



Contract view of functions (contd.)

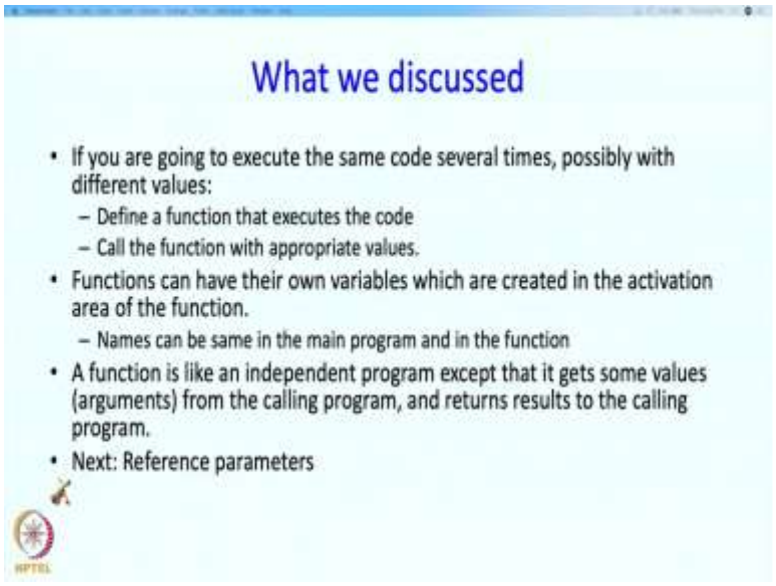
- Postcondition: What is promised about the state of the computer after the function finishes execution.
- Example: After `dash` finishes its execution we may want it to satisfy the postcondition that the pen is up.
  - not true for the code given earlier.
  - Exercise: Modify the code of `dash` to ensure that the pen is up at the end.
- Post conditions must also be mentioned in the specification.
- Writing clear specifications is very important.

So, for example, after the dash finishes its execution, if you go back and look at it, it was going to leave the pen down and it is also expecting the pen to be down but you may change this you may say look, I would like you do assume that the pen is up so maybe you should yourself make sure that the first command is pen down and I want you to leave the pen up, so maybe you should do that.

So, whatever it is, whatever the expectations might be it is a good idea to spell them out clearly, so that the user of the function is not surprised and the user of the function does not expect more than, what here she is going to get.

So, in this case here is a small exercise, modify that code so that, these post conditions will be satisfied and, of course, the post conditions must also be mentioned in the specification, and this is, this cannot be stressed too much writing clear specifications is very important for the writer of the function as well as for the user of the function alright. So what did we discuss in this segment?

(Refer Slide Time: 16:10)



What we discussed

- If you are going to execute the same code several times, possibly with different values:
  - Define a function that executes the code
  - Call the function with appropriate values.
- Functions can have their own variables which are created in the activation area of the function.
  - Names can be same in the main program and in the function
- A function is like an independent program except that it gets some values (arguments) from the calling program, and returns results to the calling program.
- Next: Reference parameters

So, we started off at the very beginning by saying that if you are going to execute the same codes several times, possibly with different values, then you define a function that executes the code. You call the function with the appropriate values. Functions can have their own variables which are created in the activation frame or activation area of the function. And the names can be the same in the main program and in the functions.

We also said that the function is like an independent program, except that it gets it, get some values the arguments from the calling program and returns results to the calling program. Next we are going to talk about reference parameters and we will take a short break.