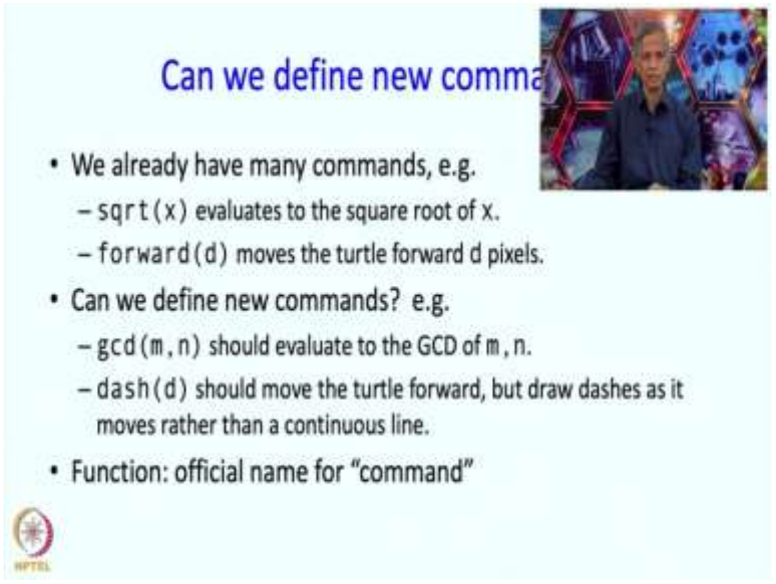
An Introduction to Programming through C++

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

Lecture 10: Part-1 Functions Basics

Hello and welcome to week five of the NPTEL course an introduction to programming through C++. Today's lecture sequence is about functions and the reading for this is from chapter 9 of the textbook. So the central question that functions address is how do we define new commands? So we really already have a whole bunch of commands for example sqrt(X).

(Refer Slide Time: 0.44)

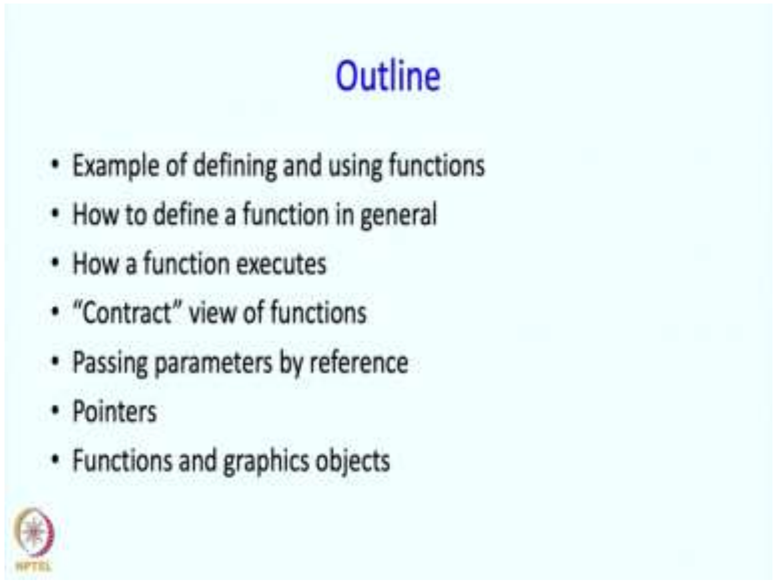


Can we define new comma

- We already have many commands, e.g.
  - `sqrt(x)` evaluates to the square root of x.
  - `forward(d)` moves the turtle forward d pixels.
- Can we define new commands? e.g.
  - `gcd(m,n)` should evaluate to the GCD of m, n.
  - `dash(d)` should move the turtle forward, but draw dashes as it moves rather than a continuous line.
- Function: official name for "command"

So 'sqrt' is a command and if you pass it the argument X then you get the square root of X. 'forward' is a command and given an argument d it causes the turtle to move d pixels. So you might ask are these the only commands that we can have or can be ourselves make new commands? For example, we would like to make a command gcd(m, n) which should evaluate the GCD the greatest common divisor of m and n. Or suppose we want the turtle to draw dashed lines as it moves rather than solid lines, continuous lines. Perhaps, you would like to have a command dash(d) which would do this. Where d is the number of pixels that you want the turtle to move. And, in fact, you can do this and the name that the technical name for what we have been calling the command so far is a function.

(Refer Slide Time: 01:51)

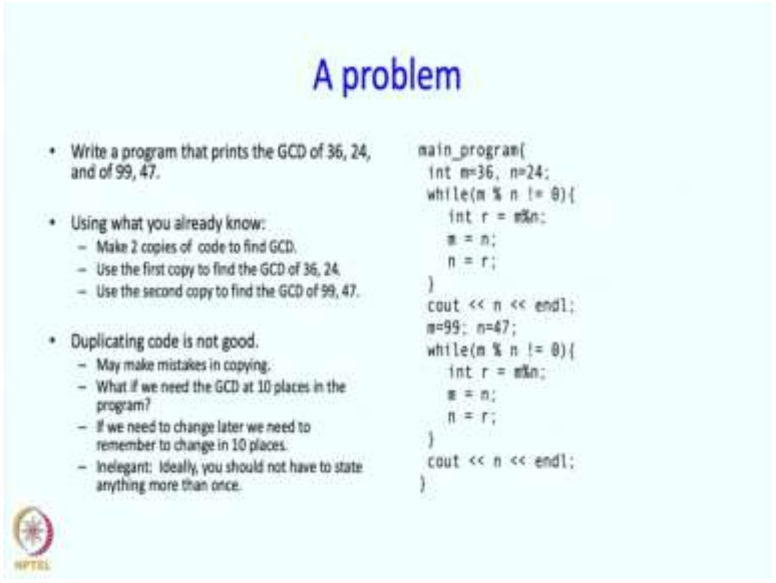


Outline

- Example of defining and using functions
- How to define a function in general
- How a function executes
- "Contract" view of functions
- Passing parameters by reference
- Pointers
- Functions and graphics objects

So, in this lecture we are going to do the following things. We are going to give first some examples of defining and using functions, then we talked about how to define a function in general. How a function executes? We will talk about what is called the contract view of functions. So, how should you think about functions informally. Then we will talk about how the arguments can be given or what is called parameter passing and there are various ways of doing this. One is the so-called parameter passing by reference which we will talk about. And we can also pass parameters using pointers. So this is also a concept that we will introduce and finally we will talk about how functions can interact with graphics objects such as rectangles circles and, of course, turtles, the turtles we will already have discussed in the earlier part of the sequence. So let us begin with a problem.

(Refer Slide Time: 03:06)



A problem

- Write a program that prints the GCD of 36, 24, and of 99, 47.
- Using what you already know:
  - Make 2 copies of code to find GCD.
  - Use the first copy to find the GCD of 36, 24.
  - Use the second copy to find the GCD of 99, 47.
- Duplicating code is not good.
  - May make mistakes in copying.
  - What if we need the GCD at 10 places in the program?
  - If we need to change later we need to remember to change in 10 places.
  - Inelegant: Ideally, you should not have to state anything more than once.

```
main_program{
 int m=36, n=24;
 while(m % n != 0){
    int r = m%n;
    m = n;
    n = r;
 }
 cout << n << endl;
 m=99; n=47;
 while(m % n != 0){
    int r = m%n;
    m = n;
    n = r;
 }
 cout << n << endl;
}
```

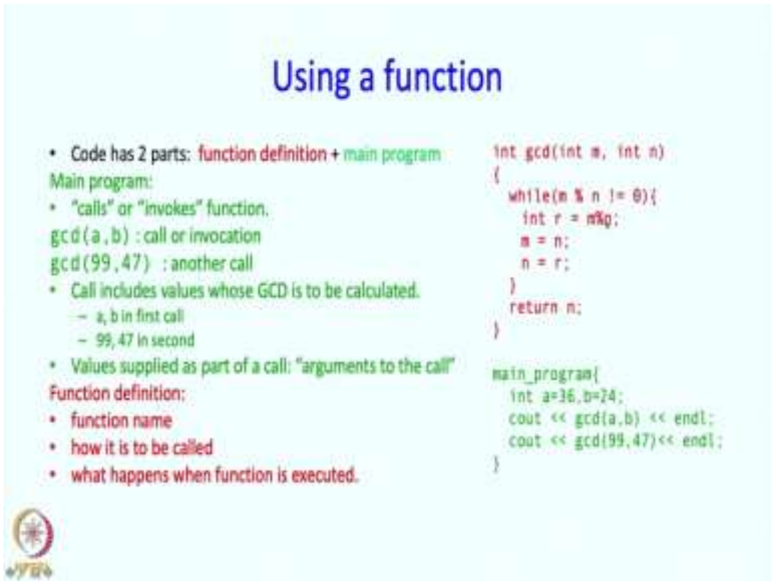
Our problem is write a program that prints the GCD of 36, 24 and of 99, 47. Here, I have explicitly stated the numbers but in general you can say that I want to write a program which first finds the GCD of one set of numbers maybe finds a GCD, does something and then finds the GCD of some other set of numbers maybe does something else and finds a GCD of third set of numbers and so on. So, this is just a representative program representing the kinds of things you might want to do.

So how would you do this? Well using what you already know. You might make two copies of the code that we wrote earlier for finding the greatest common divisor. The first copy would be used to find the GCD of 36 and 24 and the second copy to find the GCD of 99 and 47. So, here is what it might look like. So, we have a main program in which we first define m to be 36 and n to be 24 and then while m mod n is not equal to 0, we set r equal to m mod n, m equal to n, n equal to r and this is just the code that you saw earlier. So, let me not explain that code but the result, the GCD will be contained in the variable n at the end and so we can just print out that GCD. So that finishes the first part of what we were required to do. The second part is doing the same thing for m equals 99 and n equals to 47. So, here we might start off by saying m equals 99, n equals 47. Note that we are not really clearing the variables. The variables have been declared earlier and we can just use those variables.

Then we have the same code again and at the end again we print out whatever is contained in m. So this is, this is a program which does what we wanted. Which is printed GCD of 36 and 24 first and then the GCD of 99 and 47. Now, this contains code duplication. Code duplication is not really good okay. So, if I am copying code maybe I will make mistakes in copying. So, why increase the chance of making mistakes?

And suppose I need to have GCD - The Greatest Common Divisor computed at 10 places in my program then it does not seem a good idea to make ten copies of this code and this is, this becomes worse if we need to make a change to the code that we wrote. So, now I have to remember to make the change in ten places. So all this is rather inelegant. Ideally, you should not have to state anything more than once and functions allow you to do exactly that.

(Refer Slide Time: 06:17)



Using a function

- Code has 2 parts: function definition + main program

Main program:
- "calls" or "invokes" function.

gcd(a,b) : call or invocation

gcd(99,47) : another call
- Call includes values whose GCD is to be calculated.
  - a, b in first call
  - 99, 47 in second
- Values supplied as part of a call: "arguments to the call"

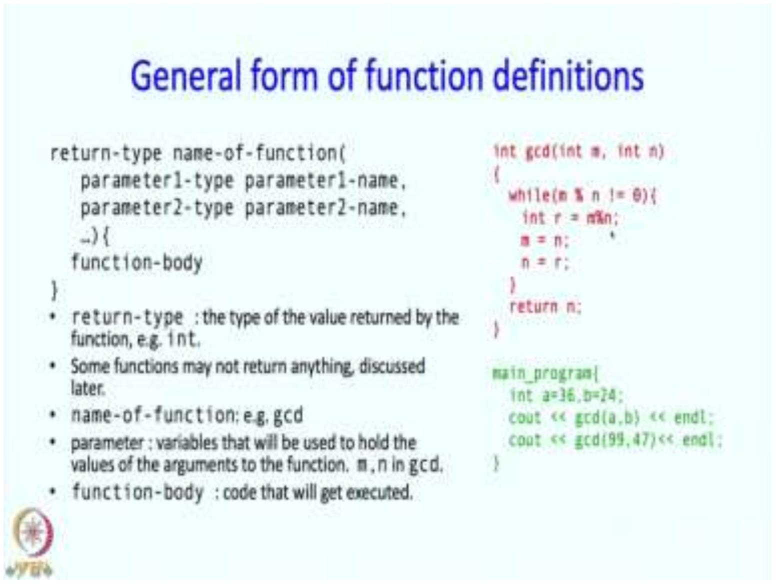
Function definition:
- function name
- how it is to be called
- what happens when function is executed.

```
int gcd(int m, int n)
{
  while(m % n != 0){
   int r = m%n;
   m = n;
   n = r;
  }
  return n;
}

main_program{
  int a=36,b=24;
  cout << gcd(a,b) << endl;
  cout << gcd(99,47)<< endl;
}
```

So, I am going to show you the code which uses functions and it has two parts. The first part which you will see in the red is the function definition and then there is the main program. So here it is. The red part is the function definition and the green part is the main program. The main program calls or invokes the function. So, the first call is GCD of a, b. Now, I could have put 36 in place of a that would have been perfectly fine but just to show you that I can also have a variable I have put in int a equal to 36, b equal to 24 and then called or invoked the function GCD with names of the variables rather than the values directly. And notice that I am invoking the function, and I am expecting that the value that comes will sort of sit in the place where that call is, where GCD of a, is and then I can directly print.

In the next line I am invoking or calling GCD again and this time the arguments 99 and 47 I am passing directly. I am sending, I am writing them down directly. So, in general the call is going to include the values whose GCD is to be calculated. The values at a, b in the first call and 99 and 47 in the second call. So, the values supplied as a part of the call are called arguments to the call. The red part is the function definition. So, first it contains a function name. Well gcd is the name. Before that there is this int, so that is not the name the name is the gcd and after that it contains int m, n, and that says so gcd together with int m and int n says how the function is going to be called. So GCD needs to be supplied an integer argument which could be referred to as m, which will be referred to as m, inside the GCD code, inside this red code and it needs to be supplied another integer value. So, that is what the top line is saying and the first int, the int at the beginning of that code is saying that the value returned is going to be an integer value. So, this int is saying that this gcd, the call to gcd is going to return an integer value and (this part) this part is the so-called body of the function and this says what you need to execute in order to compute the GCD.

(Refer Slide Time: 09:23)



General form of function definitions

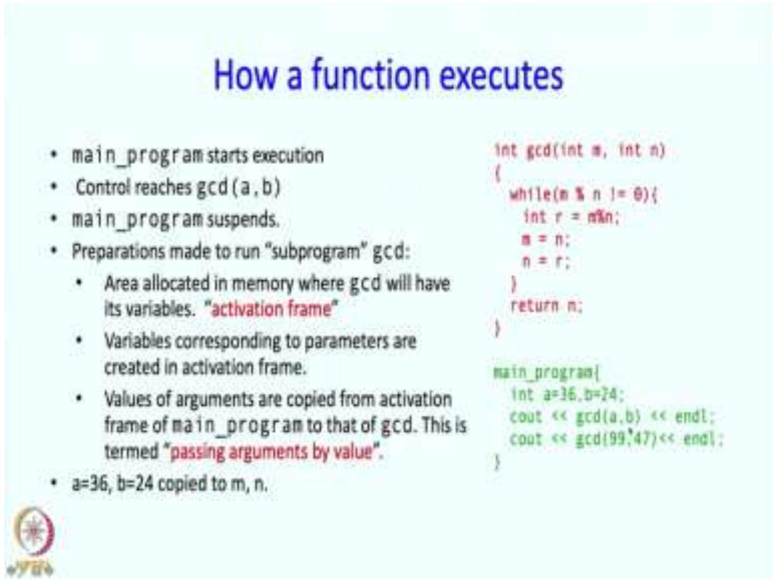
```
return-type name-of-function(
   parameter1-type parameter1-name,
   parameter2-type parameter2-name,
   …){
  function-body
}
```

- `return-type` : the type of the value returned by the function, e.g. `int`.
- Some functions may not return anything, discussed later.
- `name-of-function`: e.g. gcd
- parameter : variables that will be used to hold the values of the arguments to the function. `m`, `n` in `gcd`.
- `function-body` : code that will get executed.

```
int gcd(int m, int n)
{
  while(m % n != 0){
   int r = m%n;
   m = n;
   n = r;
  }
  return n;
}

main_program{
  int a=36,b=24;
  cout << gcd(a,b) << endl;
  cout << gcd(99,47)<< endl;
}
```

Now, I have given an example over here. But I just now want to state what is going to happen in general, or how would you write functions in general? So, in general you would have a return type. The first word of the function definition is the return type. So indeed this int is the return type. It tells you the type of the value that is going to be the result of the function execution. Then there is the name of the function over here and then the parameter names. So in the calling program these are called arguments and in the function these are called the parameters. So the parameters, you can give a list of parameters but each list has two parts. First is the parameter type, parameter one type, parameter one name, then parameter two type, parameter two name and so on as many parameters you want. So, in this case we only give two parameters and then, there is the body. So the function body has to be placed inside braces. So, in our code this is the function body. The explanations I have already given you the return type is the type of the value returned by the function. So, for example, int in our case and sometimes functions may not return anything at all and this is discussed later but in this case the return type will be written as void. We will see an example soon enough. The name of the function could be any identifier name. So, for example, gcd as we have used over here and parameters are variables. So that will be used to hold the values of the arguments to the function. So, in this case these are the parameters m and n and these will hold the values of the argument. So, we will see the exact execution process in a minute and the function body is the code that gets executed.

(Refer Slide Time: 11:31)



How a function executes

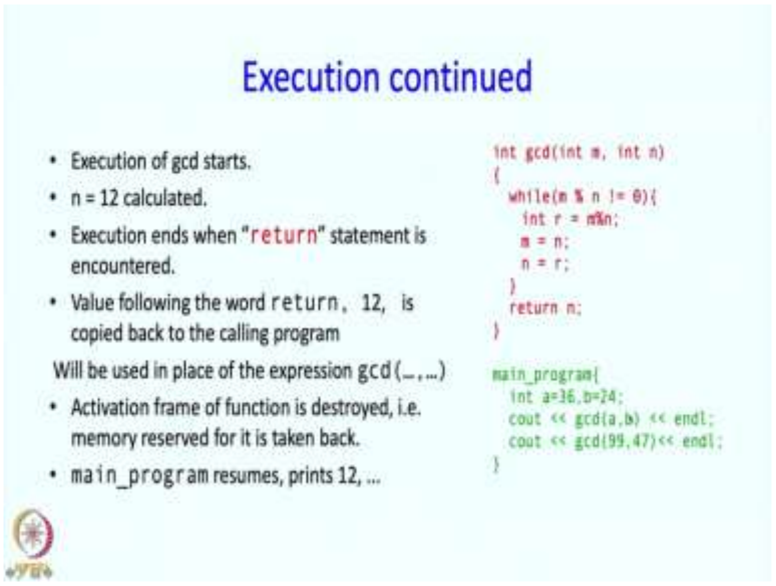
- `main_program` starts execution
- Control reaches `gcd(a,b)`
- `main_program` suspends.
- Preparations made to run "subprogram" `gcd`:
  - Area allocated in memory where `gcd` will have its variables. "activation frame"
  - Variables corresponding to parameters are created in activation frame.
  - Values of arguments are copied from activation frame of `main_program` to that of `gcd`. This is termed "passing arguments by value".
- a=36, b=24 copied to m, n.

```
int gcd(int m, int n)
{
  while(m % n != 0){
    int r = m%n;
    m = n;
    n = r;
  }
  return n;
}

main_program{
  int a=36,b=24;
  cout << gcd(a,b) << endl;
  cout << gcd(99,47)<< endl;
}
```

So, how does a function execute? So let us say you compile this and you load it and now you are executing the main program. So, the main program starts execution. Then the main program executes, you execute this. So, a gets the value 36, b gets the value 24 and then the main program or the control, the control in the main program reaches this statement and you see that there is a call over here. So at this point the main program is going to suspended. So, this execution of the main program is going to stop over here. But the main program is not going to go away, it is going to wait. So, suspend means that it is going to stop but it will wait. We can ask it to start again at some later time. So the main program has stopped at this point it is said that look I need this expression to be evaluated.

So now you make now C++ makes preparations to run 'gcd' and in some languages this is actually called a sub program, so it really is pretty much like another program but it is not the main program and so you may call it you think of it as a subprogram. So some preparations are made in particular, an area is allocated in memory where the GCD will have its variables. So, of course, the program needs some area in which its variables are to be stored and main program also has it and main program will have an activation frame in which these variables will be stored. So, similarly, C++ will create an activation frame for gcd as well. So gcd's variables will somehow is told over there as and when needed. Now, immediately as soon as this activation frame is created C++ also creates variables m and n. So, these parameters actually become variables during the executions of gcd. Not only that the values of the arguments from this call are copied into m and n. So, the value of this first argument is copied into m and the value of the second argument is copied into n and because the values are copied, so it is the value that is copied, this is called passing arguments by value. So, you will see there are other ways of passing, so right now we will just say that this is passing arguments by value. So, a, which has the value 36, so 36 will get copied in m over here and b which has the value 24 and therefore, 24 will get copied in n.

(Refer Slide Time: 14:42)



Execution continued

- Execution of gcd starts.
- n = 12 calculated.
- Execution ends when "`return`" statement is encountered.
- Value following the word `return`, 12, is copied back to the calling program

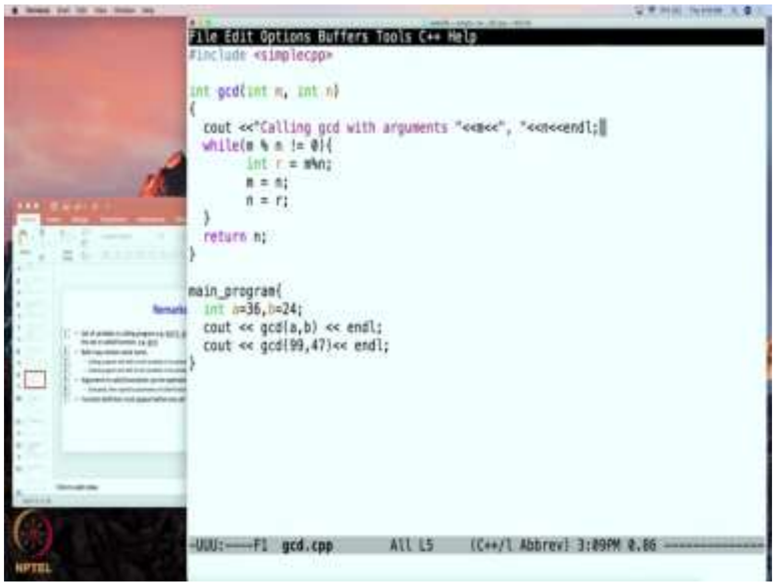
Will be used in place of the expression `gcd(…,…)`

- Activation frame of function is destroyed, i.e. memory reserved for it is taken back.
- `main_program` resumes, prints 12, …

```
int gcd(int m, int n)
{
  while(m % n != 0){
    int r = m%n;
    m = n;
    n = r;
  }
  return n;
}

main_program{
  int a=36,b=24;
  cout << gcd(a,b) << endl;
  cout << gcd(99,47)<< endl;
}
```

Now, at this point execution of gcd will start. So what happens? So, this is exactly like the execution of the main program. So, these are the variables so far and now we execute this code. So, you have seen how this code executes. So, 36 and 24 we know that this is going to calculate the correct GCD. So, n at the so then all of this is executed ‘n’ will have the value 12 at the end of this. So, all, how to execute all of this is very clear to you, so when you come to this statement this is a new statement, so let us explain, what that return means. So, n equal to 12 is calculated and the execution comes to an end essentially when we encounter a return statement and the return statement is going to somehow send back the value. So, what value it sends is this the word following return, the expression following return. So, this value is going to be sent, but this value n has value 12. So, somehow we are going to send back 12 to the calling program and where does this value go? So it goes and effectively sits instead of the call. So, we are going to essentially have a 12 coming over here. So at this point the purpose for which we started off running the GCD sub program is done. Its code has been executed, it has returned its value. So, the activation frame that we created for it has been will be destroyed by C++ and, in fact, these variables m and n will also go away at this point. So, this memory will be taken back by C++, and at this point we will resume the suspended execution of main program, remember we suspended the execution when we were, when they encountered this call. So, now that value has been evaluated the value of GCD of a, b has been evaluated as 12 we resume the execution. So, when we resume the execution what happens, well if there were 12 over here we just want to bring 12. So, 12 would get printed, sorry 12 would get printed and after that this main program execution would continue and everything would start with GCD of 99, 47 so the same thing will happen. Again 99 and 47 would be copied to m and n and again every so in gcd, the first an activation would be a frame would be created, m and n, GCD of m and n would be copied and so on. So, this is how this means this program is going to resume and it is going to print values or it is going to calculate values and send them back to the main program and the main program can do whatever it wants with the values. So, maybe before I go to the remarks let me stop here and let me just show you this program in action.

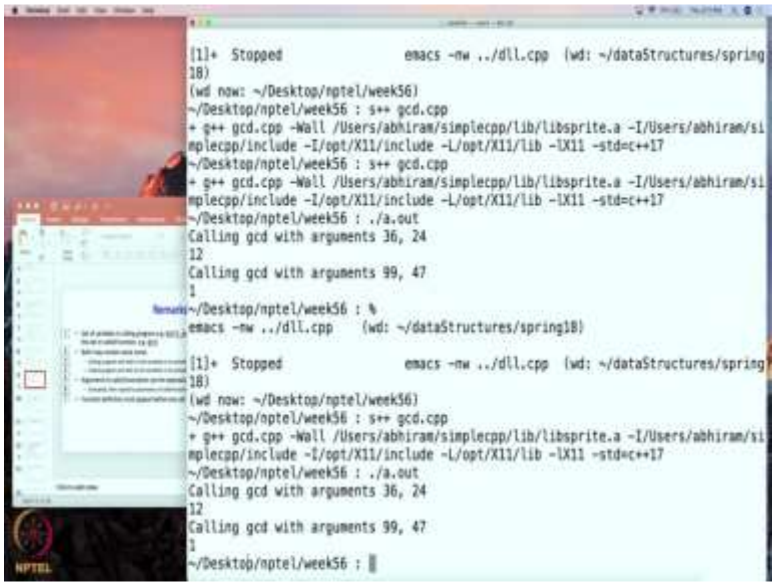
(Refer Slide Time: 18:04)



```
File Edit Options Buffers Tools C++ Help
#include <simplecpp>

int gcd(int m, int n)
{
  cout <<"Calling gcd with arguments "<<m<<", "<<n<<endl;
  while(m % n != 0){
        int r = m%n;
        m = n;
        n = r;
  }
  return n;
}

main_program{
  int a=36,b=24;
  cout << gcd(a,b) << endl;
  cout << gcd(99,47)<< endl;
}

-UUU:----F1  gcd.cpp       All L5     (C++/l Abbrev) 3:09PM 0.86 -------------
```

So, this is our program. So, the only change I have made to the code is that inside GCD I have put in a line saying calling GCD with arguments m and n. So, what we are expecting over here is that when we execute the main program gcd will execute, but as it executes everything that we said a moment ago will happen but in addition it will also print this message. So, the first time we should see a message 36, 24 the next time we should see a message 99, 47 and in between we should see the GCD is calculated, so let us see that.

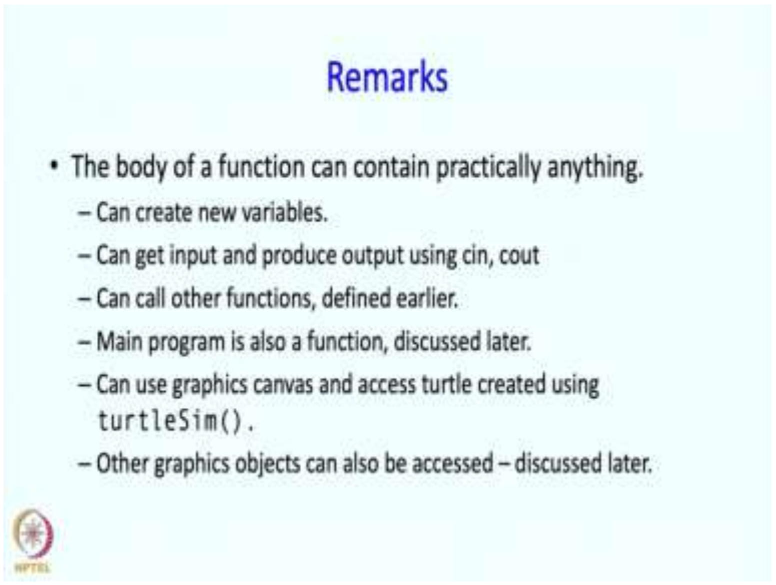
(Refer Slide Time: 18:47)



```
[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 : 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 : %
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 :
```

So, let me just compile this code and let me execute it. So, what has happened is that it indeed printed the message saying calling gcd with arguments 36, 24. 12 got printed because we were printing out the GCD and calling gcd with arguments 99, 47. 1 got printed because here the GCD is one okay. So you could ask what if I print the value to be returned before I return it yes you could do that. So you should try it out. Take the program, add that cout statement and you will see the value 12 printed twice and also the value one printer twice and maybe you could even print a message saying returning value so and so from gcd.

(Refer Slide Time: 19:42)



Remarks

- The body of a function can contain practically anything.
  - Can create new variables.
  - Can get input and produce output using cin, cout
  - Can call other functions, defined earlier.
  - Main program is also a function, discussed later.
  - Can use graphics canvas and access turtle created using `turtleSim()`.
  - Other graphics objects can also be accessed – discussed later.

So, let me make some remarks about what we have seen. So, we have a set of variables in the calling program. So, here the calling program was the main program and this set of variables lives in the activation frame of the main program and it is completely different, it is completely disjoint from the set of variables in the called function. So, in in this case the call function was gcd. Now both may contain the same name. So, I had a and b in the main program and m and n in GCD but that is not needed. I could have named the parameters a and b if I liked. So, that is not a problem okay. So, when I execute the calling program, the code that is written there is expected to refer to the names in its body and in its parameter list.

So, if there was m and n in both places and if there was code, the code for gcd contained m and n, that m and n would refer to whatever m and n is defined inside the body of the GCD or inside the parameter list. So, it had better be defined only there and, of course, the called program, the calling it is, so it is the same rule for the calling program and the called program.

So, whatever variables appear in gcd are accessible in the code of gcd and whatever variables appear in the main program are accessible in the code of the main program. So there is no problem using the same name in both cases. So, it is clear which, so if you refer to a name in your code which declaration, which definition of that name you are referring to.

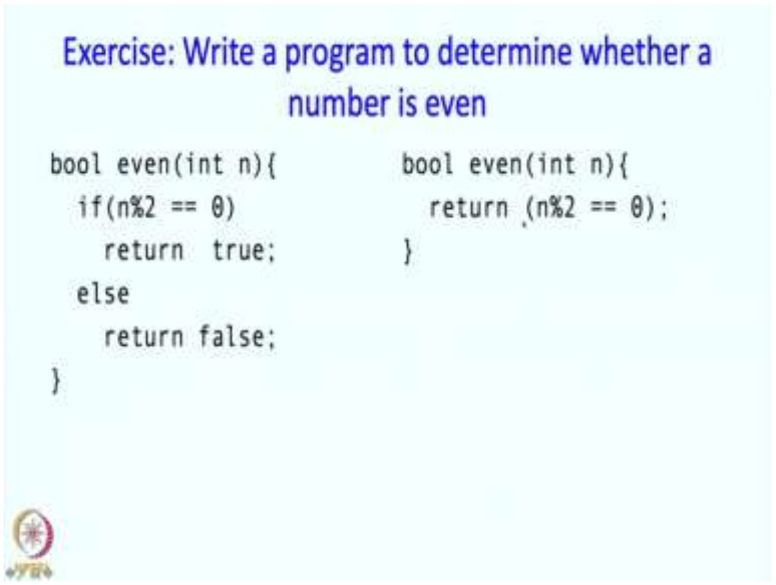
One more point, when you pass the values of arguments, here we had written either the numbers 99, 47 or names of variables but that is not necessary. You might have done some calculation and then decided what value to pass. So you could put an expression instead of either the values directly or instead of putting the names of variables which contain the values. So, if you put an expression then that expression is first evaluated and then copied to the parameters of the called function.

An important point is that in C++, there is a very general rule that if you are referring to a name, that name must be defined earlier. So, in the main program we refer to the name gcd. So the gcd has to be defined earlier. If GCD is defined later then that would not be correct. C++ compiler expects that GCD should be defined before it is actually used.

So some more remarks, the body of a function can contain practically anything. Anything that say the main program can contain. So, for example, it can create new variables internally if it wants to, and the same rules apply. So, if you create a new variable inside a block and the body of gcd is a block in fact, so those variables will go away when the execution of that body ends. Inside the body of GCD or inside the body of any function I can read values from the keyboard. I can use cin and also cout statements to produce output. So you already saw an example. We put a cout statement inside the gcd function and just to clean this up completely let me just observe right now and this will become more obvious a little bit later. The main program is actually also a function. So, gcd is also a function. So, essentially the same rules apply in both cases. Well in the main program we have not been writing return but that is something that some that is a concession which is given to the main program. So we could write return there and we will talk about that later as well. But in general for a function you really should be writing return and we have already seen in case of gcd we returned the value that was meant to be the evaluated value of the call GCD whatever our arguments being supplied.

Then inside the body of a function you can access the canvas and you can access turtle, the turtle created using turtleSim. In fact, you can make the call turtle swim also inside the body of the function that is. So, that will also create a window and that window can then be used even after you return to the main program. In fact, you can create other graphics objects also inside the body of a function okay and we will discuss that a little bit later, you can create the objects, you can pass objects to a function, a graphics objects but that will we will talk about that a little bit later.

(Refer Slide Time: 25:37)



Exercise: Write a program to determine whether a number is even

```
bool even(int n){
  if(n%2 == 0)
    return  true;
  else
    return false;
}
```

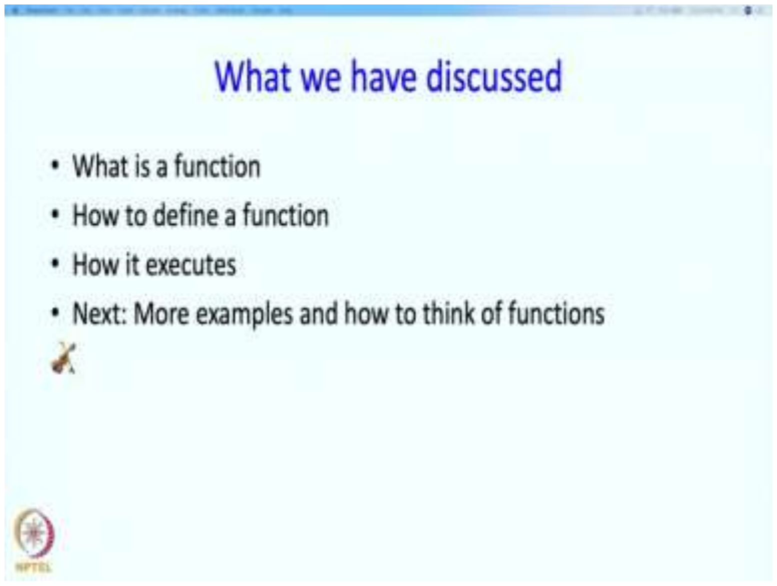
```
bool even(int n){
  return (n%2 == 0);
}
```

So, here is a quick exercise. I am going to give you the solution. So you should try and do this yourself before proceeding further okay. So, what is the exercise? Write a program to determine whether a number is even. So, I would strongly request you to pause the video here and try and write this program. So, assuming you have paused and assuming you have written the program let us now see how that program could be written. So, I am going to show you two ways. So the first way is okay, first of all we have to define the function. So, what is the value that is going to be returned? So, we are asking we want to determine whether a number is even? So, the answer expected over here is, yes, the number is even or, no, a number is even or true the number is even, false the number is not even.

So, indeed even we will write a function called even, and it is expected to be returning a true or false value and therefore, we have the return type as bool okay. So, if you remember bool is return is a type whose acceptable values are true or false. Then this is the name of the function and it takes one argument. So, this is the number or the parameter is n and this function is going to tell you whether this number is even or odd. So, if it is even it should be returning true if it is odd it should be returning false. Now, how do we determine whether a number is even? So, if you remember, we have the percent operator which gets the remainder when one number is divided by another. So, what we want to know over here is what is the remainder when n is divided by 2, is it 0 or is it not 0. So, if it is 0, if n mod 2 is equal to 0 then we want to return true because then the number is true otherwise we want to return false. So that is the function. Now, this is a perfectly fine function but I just want to alert you to the fact that you can write this much more compactly. So, here is the compact form. So, here what I have done is I have put in an expression over here. So instead of having an if statement, I have put in an expression. So, what let us just check that, so if n is n mod 2 is equal to 0, then this expression will turn out to be true, otherwise it will turn out to be false. So, depending upon what this expression evaluates to a true or false will be returned.

But when will this expression be true? It will be true only when n is even. So, when n is even it will return a true and in this expression, if n is odd, this expression will be false and therefore, a false will be returned. So, both forms are acceptable but this is a more compact way of writing it and just I just want to alert you that you can have an expression over here and, in fact, this is a Boolean expression, so Boolean expressions are also perfectly fine.

(Refer Slide Time: 29:10)



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

All right! So, what have we discussed so far? So, we discussed what is a function, we discussed how to define a function? How it executes? And next we are going to take more examples and we are going to see how we are going to think of functions. So, let us take a short break.