An Introduction to Programming though C++

Abhiram G. Ranade

Lecture 5.1

Ch. 9: Functions

Can we define new commands?

- 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”

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

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;
}
```

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;
}
```

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;
}
```

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;
}
```

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;
}
```

Remarks

- Set of variables in calling program e.g. main_program is completely disjoint from the set in called function, e.g. gcd.
- Both may contain same name.
  - Calling program will refer to the variables in its activation frame.
  - Called program will refer to the variables in its activation frame.
- Arguments to calls/invocations can be expressions
  - Evaluated, then copied to parameters of called function.
- Function definition must appear before any call to it in the program file.

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.

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);
}
```

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

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.

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.

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.

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.

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

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,

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.

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