CS 101: Computer Programming and Utilization

Shivaram Kalyanakrishnan
(Abhiram Ranade's slides, borrowed and edited)
Lecture 11

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”

Today's Lecture

- Example of defining and using functions
- How to define a function in general
- How a function executes

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;
}
```

Today's Lecture

- Example of defining and using functions
- How to define a function in general
- How a function executes

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;
}
```

Today's Lecture

- Example of defining and using functions
- How to define a function in general
- How a function executes

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.