Functions

Abhiram Ranade

Announcements about the quiz

- Aug 31, 8:25-9:25. Please arrive at 8:20.
- Closed book, only bring pens.
- Do not bring cell phones. If detected, you may get 0 on quiz.
- You may bring a bottle of water, but no sharing.
- No water/bathroom breaks. If you have a medical condition, please bring a certificate.
- Please write your lab timings on top right hand corner of page 1 of answer book, e.g. “Tuesday 9am-11am”.
- Read announcements on moodle newsgroup.

Portion for the quiz

- Everything covered before today. Chapters 1-7 of the book.
- Basic data types, repeat, if, while, for. Basic operators and assignment statement.
- Basic elements of graphics. But you don't need to remember commands exactly.
- Algorithmic techniques learnt:
  - Generating sequences, accumulation
  - Fitting curves to plotted points
  - Newton-Raphson method.
- Writing, invariants, comments.

Functions

- Official name for what we have been calling "command".
- Examples: sqrt, sin, tan, asin, abs : already defined in C++.
- Function = a separate, nearly independent program that runs "on demand" to do some specific work.
- Function = contractor who swings into action when a job is to be done. Must be given details of the job (arguments), returns the result when finished.

Topics for today

- How to define new functions.
- How defined functions execute.
- Examples.

If we had a function gcd

- We would be able to write:

```
main_program{
  cout << gcd(36,24) << endl;  // should print 12
  cout << gcd(99,47) << endl;  // should print 1
}
```

- gcd(36,24) : function call or function invocation.
- 36, 24 : arguments to the invocation.

To define gcd, write the following before main_program

```
int gcd(int Large, int Small){
  int Remainder;
  while(true){
    Remainder = Large % Small;
    if(Remainder == 0) break;
    Large = Small;     Small = Remainder;
  }
  return small;
}
```

Defining functions in general

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

Definition must appear before use.

Parts of a function definition

return-type : The type of the value that will be returned by the function.

gcd: int

parameters: variables for holding values of arguments. gcd: Large, Small.

parameter-type: Designer of the function gets to say what type the corresponding argument must have. gcd: both arguments must be int.

How a function executes

main_program executes and reaches gcd(36,24)

main_program suspends.

Preparations made to run “program” 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.

Execution of function-body starts.

(contd.)

Execution ends when “return” statement is encountered.

Value after return is copied back to the calling program, to 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 execution.

More complex mechanisms possible, study later.

Remarks

Set of variables in main_program is completely disjoint from the set in called function, gcd.

Both may contain same name, e.g. Large. main_program will reference the variables in its activation frame, and gcd in its activation frame. Change in Large in gcd will not affect Large in main_program.

New variables can be created in called function.

Arguments to calls/invocations can be expressions, which are first evaluated before called function executes.

Functions can be called while executing functions.

Example 2

We will write a function prime which will return true iff argument is a prime.

```
main_program{
  int x; cin >> x;
  if(prime(x) && prime(x+2))
    cout << “First of prime pair.\n”;
}
```

Function prime

```
bool prime(int x){
  bool found = false;
  for(int i=2; i < x && !found; i++)
     found = found || (x % i) == 0;
  return !found;  // found true if factor found
}
// write before main_program
```

Example 3: Least common multiple

```
int lcm(int m, int n){
   return m*n/gcd(m,n);
}
// must come after gcd definition
// but before main_program, or wherever it
// used.
```

Example 4

```
void printlcm(int m, int n){
   cout << lcm(m,n);
   return;
}
```

void : does not return anything. Hence nothing follows return.

must follow lcm definition in our file, but come before main_program.

Example 4 (contd.)

```
#include <simplecpp>
int gcd(int Large, int Small){….}
int lcm(int m, int n){…}
void printlcm(int m, int n){…}
main_program{
  int m, n; cin >> m >> n;
  printlcm(m,n);
}
```

Example 4 (execution)

- main_program executes. Suspends when printlcm encountered.
- printlcm executes. Suspends when lcm encountered.
- lcm executes. Suspends when gcd encountered.
- gcd executes.

At this point memory holds 4 activation frames.

(contd.)

observation: 3 variables named m,n in memory.

- gcd finishes. result copied. Frame destroyed.
- lcm finishes. result copied. Frame destroyed.
- printlcm finishes. No result to copy, however calling program told to resume, and frame of printlcm is destroyed.

Contract View of Functions

- Contractors in everyday life: tailor, lawyer, doctor, …
- We give them work: we don't worry how they do it. They explicitly or implicitly make a promise to us, and we trust that promise.
- We like it that we don't need to know how a tailor stitches, or a doctor diagnoses.

Something similar for functions.

Contract view (contd.)

The user of a function needs to know:

- return-type
- type of each parameter
- Possible additional constraints on what kinds of arguments are allowed, e.g. arguments to gcd and lcm must be > 0. PRE-CONDITION
- The promise made by the function developer about what will happen. POST-CONDITION

- Does not need to know function-body

Typical coding style

```
return-type function-name (parameter-type
    parameter-name …)
// specification: what the function does. pre/post
// conditions.
{
   … function body “implementation” …
   // comments explaining “how”
}
```

Uses of functions

- If same operation is to be performed several times, put it in a function and call it several times.
- Break up large program into small pieces. Just as you break up a book into chapters.
- Different functions can go into different files. Different programmers can work on different functions.

Breaking code into many files

main.cpp

```
int gcd(int m, int n);  // declaration
main_program{ … gcd(24,36) …}
```

gcd.cpp

```
int gcd(int Large, int small){  //definition
  … }
```

Function declaration

return-type name(param1type, param2type, ..)

int gcd(int, int);

- Declaration says:
  - what is the type of gcd: it is a function returning int and taking two ints as arguments.
  - The full definition will come later, possibly in another file.
- Declaration needed to compile file containing invocation/ call, e.g. main.cpp.

int gcd(int m, int n); // also allowed. “definition without body”

m, n in above only for readability. only gcd being declared.

Compiling the two files

s++ main.cpp gcd.cpp

will produce a.out

s++ -c gcd.cpp

will produce “object file” gcd.o

object file: machine language but cannot be executed, because usually it is incomplete in some way e.g. because there is no main_program

(contd.)

s++ main.cpp gcd.o

will generate executable if both files are available.

s++ main.o gcd.o

also allowed.

- Object files (.o) can be distributed without source files (.cpp). Receiver can execute but not read program.

Another use of declarations

- If you declare first, then definitions can come later in any order. Useful if you like the main program to come first in the file.

```
int gcd(int m, int n);
int lcm(int m, int n);
main_program{ cout << lcm(24,36);}
int lcm(int m, int n){ … }
int gcd(int m, int n){ … }
```

main_program

abbreviation created in simplecpp for:

int main()

Your main program is also a function!

When a.out is executed, the Operating system asks that the function main be executed.

int type: historical significance.

return int?: C++ compiler treats main specially.