C++ Intro Catchup

Victor Eijkhout, Susan Lindsey

Fall 2022

last formatted: August 31, 2022

Basics

1.

In programming you have two kinds of files:

- *source files*, which are understandable to you, and which you create with an editor such as `vi` or `emacs`; and
- *binary files*, which are understandable to the computer, and unreadable to you.

Your source files are translated to binary by a compiler, which 'compiles' your source file.

Exercise 1

Make a file `zero.cc` with the following lines:

```
#include <iostream>
using std::cout;

int main() {
  return 0;
}
```

and compile it. Intel compiler:

```
icpc -o zeroprogram zero.cc
```

Run this program (it gives no output):

```
./zeroprogram
```

2.

- `icpc` : compiler. Alternative: use `g++` or `clang++`
- `-o zeroprogram` : output into a binary name of your choosing
- `zero.cc` : your source file.

Input/Output and strings

3.

You have already seen cout:

```
float x = 5;
cout << "Here is the root: " << sqrt(x) << "\n";
```

4.

```
string name; int age;
cout << "Your name?\n";
cin >> name;
cout << "age?\n";
cin >> age;
cout << age << " is a nice
    age, "
     << name << "\n";
```

```
> ./cin
Your name?
Victor
age?
18
18 is a nice age, Victor
> ./cin
Your name?
THX 1138
age?
1138 is a nice age, THX
```

5.

- Add the following at the top of your file:

```
#include <string>
using std::string;
```

- Declare string variables as

```
string name;
```

- And you can now `cin` and `cout` them.

Exercise 2

Write a program that asks for the user's first name, uses `cin` to read that, and prints something like `Hello, Susan!` in response.

What happens if you enter first and last name?

Conditionals

6. If-then-else

A conditional is a test: 'if something is true, then do this, otherwise maybe do something else'. The C++ syntax is

```
if ( something ) {
  // do something;
} else {
  // do otherwise;
}
```

- The 'else' part is optional
- You can leave out braces in case of single statement.

7. Complicated conditionals

Chain:

```
if ( /* some test */ ) {
  ...
} else if ( /* other test */ ) {
  ...
}
```

Nest:

```
if ( /* some test */ ) {
  if ( /* other test */ ) {
    ...
  } else {
    ...
  }
}
```

8. Local variables in conditionals

The curly brackets in a conditional allow you to define local variables:

```
if ( something ) {
  int i;
  .... do something with i
}
// the variable ‘i’ has gone away.
```

Good practice: only define variable where needed.

Braces induce a scope.

Exercise 3

Read in a positive integer. If it's a multiple of three print 'Fizz!'; if it's a multiple of five print 'Buzz!'. It it is a multiple of both three and five print 'Fizzbuzz!'. Otherwise print nothing.

Note:

- Capitalization.
- Exclamation mark.
- Your program should display at most one line of output.

For loops

9.

The loop variable is usually an integer:

```
for ( int index=0; index<max_index; index=index+1) {
  ...
}
```

But other types are allowed too:

```
for ( float x=0.0; x<10.0; x+=delta ) {
  ...
}
```

Beware the stopping test for non-integral variables!

10. Nested loops

Traversing a matrix
(we will discuss actual matrix data structures later):

```
for (int row=0; row<m; row++)
  for (int col=0; col<n; col++)
    ...
```

This is called 'loop nest', with
row: outer loop
col: inner loop.

11. Indefinite looping

Sometimes you want to iterate some statements not a predetermined number of times, but until a certain condition is met. There are two ways to do this.

First of all, you can use a 'for' loop and leave the upperbound unspecified:

```
for (int var=low; ; var=var+1) { ... }
```

12. Break out of a loop

This loop would run forever, so you need a different way to end it. For this, use the `break` statement:

```
for (int var=low; ; var=var+1) {
  statement;
  if (some_test) break;
  statement;
}
```

Exercise 4

The integer sequence

$$u_{n+1} = \begin{cases} u_n/2 & \text{if } u_n \text{ is even} \\ 3u_n + 1 & \text{if } u_n \text{ is odd} \end{cases}$$

leads to the Collatz conjecture: no matter the starting guess u_1, the sequence $n \mapsto u_n$ will always terminate at 1.

$$5 \to 16 \to 8 \to 4 \to 2 \to 1$$

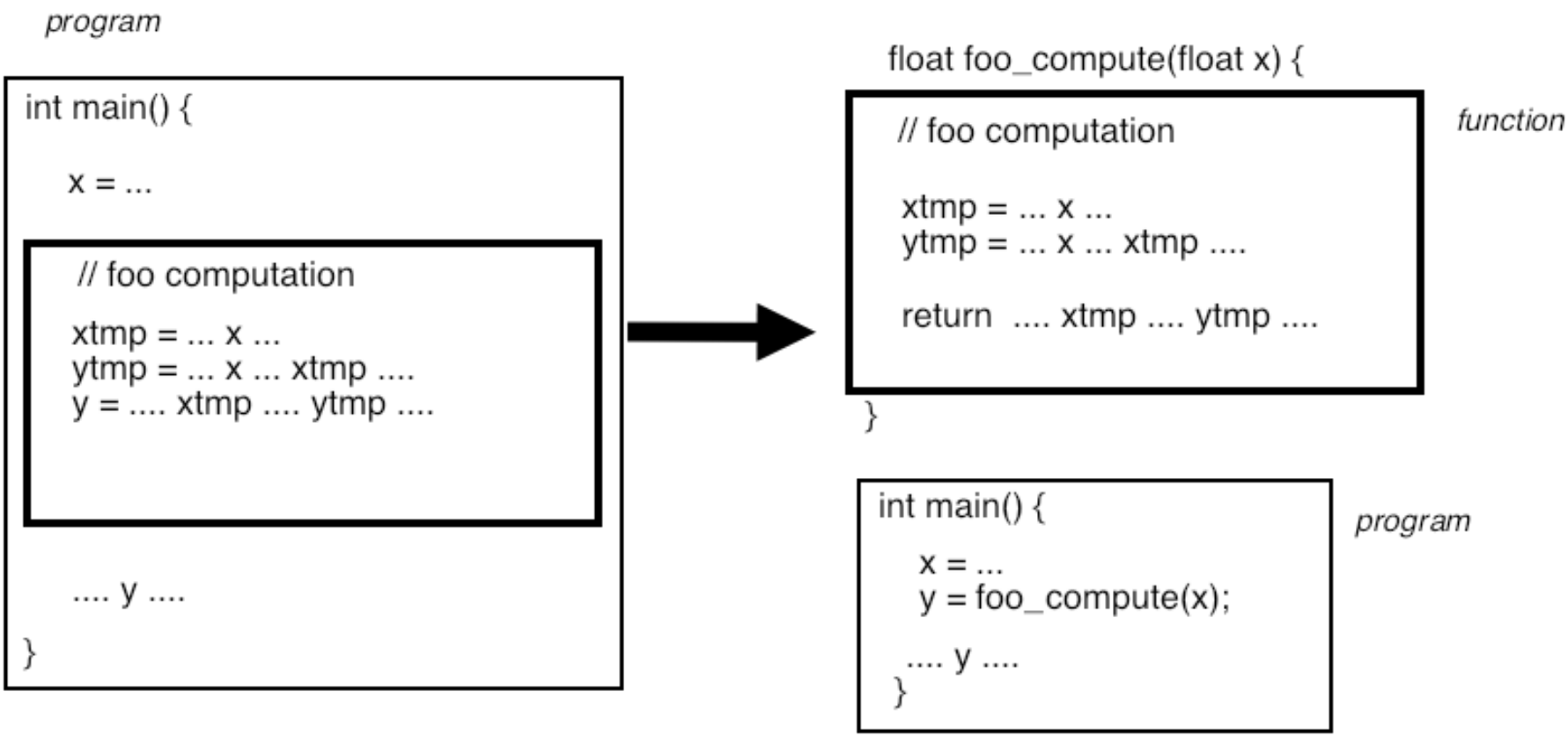
$$7 \to 22 \to 11 \to 34 \to 17 \to 52 \to 26 \to 13 \to 40 \to 20 \to 10 \to 5 \cdots$$

(What happens if you keep iterating after reaching 1?)

Try all starting values $u_1 = 1, \ldots, 1000$ to find the values that lead to the longest sequence: every time you find a sequence that is longer than the previous maximum, print out the starting number.

Function basics

13.


program
int main() {
x = ...
// foo computation
xtmp = ... x ...
ytmp = ... x ... xtmp ....
y = ... xtmp .... ytmp ....
.... y ....
}
float foo_compute(float x) {
// foo computation
xtmp = ... x ...
ytmp = ... x ... xtmp ....
return .... xtmp .... ytmp ....
}
function
int main() {
x = ...
y = foo_compute(x);
.... y ....
}
program

14.

Example: zero-finding through bisection.

$$\underset{x}{?}: f(x) = 0, \qquad f(x) = x^3 - x^2 - 1$$

(where the question mark quantor stands for 'for which x').

First attempt at coding this: everything in the main program.

Code:

```
float left{0.},right{2.},
  mid;
while (right-left>.1) {
  mid = (left+right)/2.;
  float fmid =
    mid*mid*mid - mid*mid-1;
  if (fmid<0)
    left = mid;
  else
    right = mid;
}
cout << "Zero happens at: " << mid <<
    "\n";
```

Output

```
[func] bisect1:

Zero happens at:
     1.4375
```

15. Why functions?

- Easier to read: use application terminology
- Shorter code: reuse
- Cleaner code: local variables are no longer in the main program.
- Maintainance and debugging

Project Exercise 5

Write a function `test_if_prime` that has an integer parameter, and returns a boolean corresponding to whether the parameter was prime.

```
int main() {
  bool isprime;
  isprime = test_if_prime(13);
```

Read the number in, and print the value of the boolean.

Does your function have one or two `return` statements? Can you imagine what the other possibility looks like? Do you have an argument for or against it?

Project Exercise 6

Take your prime number testing function `test_if_prime`, and use it to write a program that prints multiple primes:

- Read an integer `how_many` from the input, indicating how many (successive) prime numbers should be printed.
- Print that many successive primes, each on a separate line.
- (Hint: keep a variable `number_of_primes_found` that is increased whenever a new prime is found.)

Parameter passing

16. Mathematical type function

Pretty good design:

- pass data into a function,
- return result through `return` statement.
- Parameters are copied into the function. (Cost of copying?)
- pass by value
- 'functional programming'

17. Pass by value example

Note that the function alters its parameter `x`:

Code:

```
double squared( double x ) {
  double y = x*x;
  return y;
}
  /* ... */
  number = 5.1;
  cout << "Input starts as: "
       << number << "\n";
  other = squared(number);
  cout << "Output var is: "
       << other << "\n";
  cout << "Input var is now: "
       << number << "\n";
```

Output

```
[func] passvalue:

Input starts as: 5.1
Output var is: 26.01
Input var is now: 5.1
```

but the argument in the main program is not affected.

18.

A reference is indicated with an ampersand in its definition, and it acts as an alias of the thing it references.

Code:

```
int i;
int &ri = i;
i = 5;
cout << i << "," << ri << "\n";
i *= 2;
cout << i << "," << ri << "\n";
ri -= 3;
cout << i << "," << ri << "\n";
```

Output
[basic] ref:

```
5,5
10,10
7,7
```

(You will not use references often this way.)

19.

The function parameter `n` becomes a reference to the variable `i` in the main program:

```
void f(int &n) {
  n = /* some expression */ ;
};
int main() {
  int i;
  f(i);
  // i now has the value that was set in the function
}
```

20.

Code:

```
void f( int &i ) {
  i = 5;
}
int main() {

  int var = 0;
  f(var);
  cout << var << "\n";
```

Output

```
[basic] setbyref:

5
```

Compare the difference with leaving out the reference.

21.

```
bool can_read_value( int &value ) {
  // this uses functions defined elsewhere
  int file_status = try_open_file();
  if (file_status==0)
    value = read_value_from_file();
  return file_status==0;
}

int main() {
  int n;
  if (!can_read_value(n)) {
    // if you can't read the value, set a default
    n = 10;
  }
  ..... do something with 'n' ....
```

Exercise 7

Write a `void` function `swap` of two parameters that exchanges the input values:

Code:

```
cout << i << "," << j << "\n";
swap(i,j);
cout << i << "," << j << "\n";
```

Output

```
[func] swap:

1,2
2,1
```

More about functions

22.

Functions can have default argument(s):

```
double distance( double x, double y=0. ) {
  return sqrt( (x-y)*(x-y) );
}
  ...
  d = distance(x); // distance to origin
  d = distance(x,y); // distance between two points
```

Any default argument(s) should come last in the parameter list.

23. Polymorphic functions

You can have multiple functions with the same name:

```
double average(double a,double b) {
  return (a+b)/2; }
double average(double a,double b,double c) {
  return (a+b+c)/3; }
```

Distinguished by type or number of input arguments: can not differ only in return type.

```
int f(int x);
string f(int x); // DOES NOT WORK
```