Statements and expressions

Victor Eijkhout and Charlie Dey

spring 2017

Basics

Edit, compile, run

Exercise: make a file `zero.cc` with the following lines:

```
#include <iostream>
using namespace std;

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

and compile it:

```
# Intel:
icpc -o zeroprogram zero.cc
# Gnu:
g++ -o zeroprogram zero.cc
```

Run this program. It doesn't do anything...

Add this line:

```
cout << "Hello world!" << endl;
```

(copying from the pdf file is dangerous! please type it yourself)

Compile and run again.

Statements

Program statements

- A program contains statements, each terminated by a semicolon.
- 'Curly braces' can enclose multiple statements.
- A statement corresponds to some action when the program is executed.
- Comments are 'Note to self', short:

```
cout << "Hello world" << endl; // say hi!
```

and arbitrary:

```
cout << /* we are now going
            to say hello
         */ "Hello!" << /* with newline */ endl;
```

Exercise 1

Take the 'hello world' program you wrote earlier, and duplicate the hello-line. Compile and run.

Does it make a difference whether you have the two hellos on the same line or on different lines?

Experiment with other changes to the layout of your source. Find at least one change that leads to a compiler error.

Fixed elements

You see that certain parts of your program are inviolable:

- There are *keywords* such as `return` or `cout`.
- Curly braces and parentheses need to be matched.
- There has to be a `main` keyword.
- The `iostream` and `namespace` are usually needed.

Exercise 2

Experiment with the `cout` statement. Replace the string by a number or a mathematical expression. Can you guess how to print more than one thing, for instance the string `One third is` and the result of 1/3, with the same `cout` statement?

Variables

Variable declarations

Programs usually contain data, which is stored in a *variable*.
A variable has

- a *datatype*,
- a name, and
- a value.

These are defined in a *variable declaration* and/or *variable assignment*.

Variable names

- A variable name has to start with a letter,
- can contains letters and digits, but not most special characters (except for the underscore).
- For letters it matter whether you use upper or lowercase: the language is *case sensitive*.

Declaration

There are a couple of ways to make the connection between a name and a type. Here is a simple *variable declaration*, which establishes the name and the type:

```
int n;
float x;
int n1,n2;
double re_part,im_part;
```

Declarations can go pretty much anywhere in your program.

Datatypes

Variables come in different types;

- We call a variable of type `int`,`float`,`double` a *numerical variable*.
- For characters: `char`. Strings are complicated.
- You can make your own types. Later.

Assignments

Assignment

Once you have declared a variable, you need to establish a value. This is done in an *assignment* statement. After the above declarations, the following are legitimate assignments:

```
n = 3;
x = 1.5;
n1 = 7; n2 = n1 * 3;
```

You see that you can assign both a simple value or an *expression*; see section **??** for more detail.

Assignments

A variable can be given a value more than once. You the following sequence of statements is a legitimate part of a program:

```
int n;
n = 3;
n = 2*n + 5;
n = 3*n + 7;
```

Special forms

Update:

```
x = x+2; y = y/3;
// can be written as
x += 2; y /= 3;
```

Integer add/subtract one:

```
i++; j--;  /* same as: */ i=i+1; j=j-1;
```

Pre/post increment:

```
x = a[i++]; /* is */ x = a[i]; i++;
y = b[++i]; /* is */ i++; y = b[i];
```

Initialization

You can also give a variable a value a in *variable initialization*. Confusingly, there are several ways of doing that. Here's two:

```
int n = 0;
double x = 5.3, y = 6.7;
double pi{3.14};
```

Exercise 3

Write a program that has several variables. Assign values either in an initialization or in an assignment. Print out the values.

Truth values

So far you have seen integer and real variables. There are also *boolean values* which represent truth values. There are only two values: `true` and `false`.

Exercise 4

Print out `true` and `false`. What do you get?

Input/Output

Terminal output

You have already seen `cout`:

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

Terminal input

There is also a *cin*, which serves to take user input and put it in a numerical variable.

```
int i;
cin >> i;
```

However, this function is somewhat tricky.
`http://www.cplusplus.com/forum/articles/6046/`.

Better terminal input

It is better to use `getline`. This returns a string, rather than a value, so you need to convert it with the following bit of magic:

```
#include <iostream>
#include <sstream>
using namespace std;
  std::string saymany;
  int howmany;

  cout << "How many times? ";
  getline( cin,saymany );
  stringstream saidmany(saymany);
  saidmany >> howmany;
```

Exercise 5

Read some variables, and print them out again. Use both `cin` and `getline`, but not in the same program.

Expressions

Arithmetic expressions

- Expression looks pretty much like in math. `2+3` `3.2/7`
- Use parentheses to group `25.1*(37+42/3.)`
- Careful with types.
- There is no 'power' operator: library functions.

Boolean expressions

- Not, and, or: `!  &&  ||`
- Bitwise: `&  |  ^`
- Shortcut operators:

```
if ( x>=0 && sqrt(x)<5 ) {}
```

Conversion and casting

Real to integer: round down:

```
double x,y; x = .... ; y = .... ;
int i; i =  x+y:
```

Dangerous:

```
int i,j; i = ... ; j = ... ;
double x ; x = 1+i/j;
```

The fraction is executed as integer division. Do:

```
(double)i/j /* or */ (1.*i)/j
```

Exercise 6

Compute some arithmetic expressions and print them out.
Experiment with conversions. Also boolean expressions.

Project Exercise 7

Read two integers into two variables, and print their sum, product, quotient, modulus.

Project Exercise 8

Read two numbers and print out their modulus. Two ways:

- use the `cout` function to print the expression, or
- assign the expression to a variable, and print that variable.