Statements and expressions

Kevin Schmidt, Susan Lindsey, Charlie Dey

Spring 2019

Basics

Two kinds of files

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 file(s) is/are translated by binary by a *compiler*.

Exercise 1

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

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

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

and compile it. Intel compiler:

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

- `icpc` : compiler. Alternative Gnu: use `g++` instead of `icpc`.
- `-o programname` : output into a binary name of your choosing
- `something.cc` : your source file.

Run this program (it gives no output):

```
./zeroprogram
```

Exercise 2

Add this line:

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

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

Compile and run again.

(Did you indent the 'hello world' line? Did you editor help you with the indentation?)

Review quiz 1

True or false?

1. The programmer only writes source files, no binaries.
2. The computer only executes binary files, no human-readable files.

Errors

There are two types of errors that your program can have:

1. *Syntax* or *compile-time* errors: these arise if what you write is not according to the language specification. The compiler catches these errors, and it refuses to produce an *executable*.
2. *Run-time* errors: these arise if your code is syntactically correct, and the compiler has produced an executable, but the program does not behave the way you intended or foresaw. Examples are divide-by-zero or indexing outside the bounds of an array.

File names

File names can have extensions: the part after the dot.

- `program.cxx` or `program.cc` are typical extensions for C++ sources.
- `program.cpp` is sometimes used, but your instructor does not like that.
- `program` without extension usually indicates an *executable*.

Statements

Program statements

- A program contains statements, each terminated by a semicolon.
- 'Curly braces' can enclose multiple statements.
- A statement can correspond to some action when the program is executed.
- Some statements are definitions, of data or of possible actions.
- 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 3

Take the 'hello world' program you wrote above, 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`; you can not change their definition.
- Curly braces and parentheses need to be matched.
- There has to be a `main` keyword.
- The `iostream` and `std` are usually needed.

Exercise 4

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 the computation $1/3$,

with the same `cout` statement?

Variables

What's a variable?

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*.

Typical variable lifetime

```
int i,j; // declaration
i = 5; // set a value
i = 6; // set a new value
j = i+1; // use the value of i
i = 8; // change the value of i
       // but this doesn't affect j:
       // it is still 7.
```

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 matters whether you use upper or lowercase: the language is *case sensitive*.
- Words such as `main` or `cout` are *reserved words*.
- Usually `i` and `j` are not the best variable names.

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;
```

Where do declarations go?

Declarations can go pretty much anywhere in your program, but they need to come before the first use of the variable.

Note: it is legal to define a variable before the main program but that's not a good idea. Please only declare *inside* main (or inside a function et cetera).

Review quiz 2

Which of the following are legal variable names?

1. `mainprogram`
2. `main`
3. `Main`
4. `1forall`
5. `one4all`
6. `one_for_all`
7. `onefor{all}`.

Datatypes

Variables come in different types;

- We call a variable of type `int`,`float`,`double` a *numerical variable*.
- *Complex numbers* will be discussed later; section **??**.
- For characters: `char`. Strings are complicated.
- Truth values: `bool`
- 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;
```

Variable of the left-hand side gets value of the right-hand side.

You see that you can assign both a simple value or an *expression*.

Assignments

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

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

These are not math equations: variable on the lhs gets the value of the rhs.

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;
```

Review quiz 3

Which of the following are legal?

1. `n = n;`
2. `n = 2n;`
3. `n = n2;`
4. `n = 2*k;`
5. `n/2 = k;`
6. `n /= k;`

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};
```

Do not use uninitialized variables! Doing so is legal, but there is no guarantee about the initial value. Do not count on it being zero...

Exercise 5

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

Review quiz 4

```
#include <iostream>
using std::cout;
using std::endl;
int main() {
  int i;
  int j = i+1;
  cout << j << endl;
  return 0;
}
```

What happens?

1. Compiler error
2. Output: 1
3. Output is undefined
4. Error message during running the program.

Integer constants

Integers are normally written in decimal, and stored in 32 bits. If you need something else:

```
int d = 42;
int o = 052; // start with zero
int x = 0x2a;
int X = 0X2A;
int b = 0b101010; // C++14
long ell = 42L;
```

Floating point constants

- Default: `double`
- Float: `3.14f` or `3.14F`
- Long double: `1.22l` or `1.22L`.

This prevents numerical accidents:

```
double x = 3.;
```

converts float to double, maybe introducing random bits.

Warning: floating point arithmetic

Floating point arithmetic is full of pitfalls.

- Don't count on `3*(1./3)` being exactly 1.
- Not even associative.

(See Eijkhout, Introduction to High Performance Computing, chapter 3.)

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`.

```
bool found{false};
found = true;
```

Quick intro to strings

- 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.

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.

```
// add at the top of your program:
using std::cin;

// then in your main:
int i;
cin >> i;
```

There is also `getline`, which is more general.

Exercise 6

Write a program that :

- displays the message `Type a number`,
- accepts an integer number from you (use `cin`),
- makes another variable that is three times that integer plus one,
- and then prints out the second variable.

Turn it in!

- If you have compiled your program, do:

  `sdstest3np1 yourprogram.cc`

  where '`yourprogram.cc`' stands for the name of your source file.
- Is it reporting that your program is correct? If so, do:

  `sdstest3np1 -s yourprogram.cc`

  where the `-s` flag stands for 'submit'.

Exercise 7

Write a program that asks for the user's first name, and prints something like `Hello, Victor!` in response.

What happens if you enter first and last name?

Expressions

Arithmetic expressions

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

  ```
  #include <cmath>
  ```
- Modulus: %

Math library calls

Math function in `cmath`:

```
#include <cmath>
.....
x = pow(3,.5);
```

For squaring, usually better to write `x*x` than `pow(x,2)`.

Boolean expressions

We'll do that in the lecture on conditionals.

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.
For floating point result do:

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

Exercise 8

Write a program that asks for two integer numbers `n1,n2`.

- Assign the integer ratio n_1/n_2 to an integer variable.
- Can you use this variable to compute the modulus

$$n_1 \mod n_2$$

  (without using the % modulus operator!)
  Print out the value you get.
- Also print out the result from using the modulus operator:%.
- Investigate the behaviour of your program for negative inputs.
  Do you get what you were expecting?

Optional exercise 9

Write two programs, one that reads a temperature in Centigrade and converts to Fahrenheit, and one that does the opposite conversion.

$$C = (F - 32) \cdot 5/9, \qquad F = 9/5\, C + 32$$

Check your program for the freezing and boiling point of water. (Do you know the temperature where Celsius and Fahrenheit are the same?)

Can you use Unix pipes to make one accept the output of the other?

Review quiz 5

True of false?

1. Within a certain range, all integers are available as values of an integer variable.
2. Within a certain range, all real numbers are available as values of a float variable.
3. `5(7+2)` is equivalent to 45.
4. `1--1` is equivalent to zero.
5. `int i = 2./3.;` The variable `i` is 1.
6. `float x = 2/3;` The variable `x` is approximately `0.6667`.