C++ Programming

DR. YIHSIANG LIOW (FEBRUARY 4, 2025)

Contents

Chapter 0

Preamble

I hope one day I have notes (openbooks) for all my classes so that students won't have to spend 100s of dollars on books. This is one of them. It's still under heavy construction.

0.1 Goal

Learn C++ programming, learn some CS stuff, and have fun.

0.2 Audience

Anyone who can read, has access to a computer, feels comfortable using a computer, has some basic math background, and most importantly, wants to learn.

0.3 Disclaimer

I hope the number of errors in this set of notes is close to zero. Please report bugs to yliow@ccis.edu.

0.4 Acknowledgments

I want to thank students from the following classes for taking for this course.

- Fall 2003
- Spring 2004
- Fall 2004
- Spring 2005
- Fall 2005
- Spring 2006
- Spring 2007
- Spring2008
- Spring 2009
- Spring 2010
- Spring 2011
- Fall 2012

Chapter 1

Print statements and C-strings

OBJECTIVES

- Print strings and characters using `std::cout`
- Use the `\n`, `\t`, `\"`, `\'`, `\\` characters
- Use `std::endl` to force newline
- Write multiple print statements

In this set of notes, we learn to print strings and characters.

A quick advice to ease the pain of learning your first programming language: Type the program **exactly** as given. Even your spaces and blank lines must match the spaces and blank lines in my programs.

Let's begin ...

Why the big, fat right margin? For your notes. Get busy.

1.1 Hello World

Go ahead and run your first program:

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!\n";

    return 0;
}
```

... commercial break ... go to notes on software tool(s) for writing and running a program ...

TODO ... Practice doing a hello world program until you can do it in $<$ 2 minutes and without your notes.

Now let's go back to the C++ code. Right now, you should think of the stuff in bold as the program:

```
#include <iostream>

int main()
{
    *** YOUR "PROGRAM" GOES HERE ***

    return 0;
}
```

Therefore you can treat this as a template:

TODO ... Enter a single program here.

```
#include <iostream>

int main()
{

    *** PROGRAM ***

    return 0;
}
```

```
#include <iostream>

int main()
{

    *** PROGRAM ***

    return 0;
}
```

TODO ... Put your program here

I will not explain things like "`#include <iostream>`" or "`int main()`" until later. (Technically, they are also part of the program.)

Try this

```
#include <iostream>

int main()
{
    std::cout << "Hello ... world! ... mom!\n";

    return 0;
}
```

Exercise 1.1.1. Write a program that prints the following:

debug: exercises/240-0001/question.tex

```
hello columbia
```

You have 3 minutes. (Go to solution, page **??**) □

Exercise 1.1.2. (Go to Solution 1.1.2 on page 9)
Write a program that gets the computer to greet you:

```
hello dr. liow, my name is c++.
```

(replace my name with yours ... you're probably not "`dr. liow`"). 2.5 minutes! □

Exercise 1.1.3. (Go to Solution 1.1.3 on page 9)
Debug (i.e., correct) this program by hand and then verify by running it with your C++ compiler. 3 minutes!

```
#include (iostream)

int main()
(
    std;;cout < "Hello, world!\n";

    return 0;
)
```

□

SOLUTIONS

Solution 1.1.1. (Go to Exercise **??** on page **??**)

```
#include <iostream>

int main()
{
    std::cout << "hello columbia\n";

    return 0;
}
```

Solution 1.1.2. (Go to Exercise 1.1.2 on page 7)

```
#include <iostream>

int main()
{
    std::cout << "hello dr. liow, my name is c++.\n";

    return 0;
}
```

Solution 1.1.3. (Go to Exercise 1.1.3 on page 7)

```
#include ¡iostream¿

int main()
—
    std::cout <¡ "Hello, world!\n";

    return 0;
˝
-
```

1.2 Statement

Here's the first jargon. Look at our program again:

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!\n";

    return 0;
}
```

The line in bold is a **statement**. In C++, statements must terminate with a **semi-colon**. Note that the semi-colon is part of the statement.

At this point you should think of a statement as something that will cause your computer to perform some operation(s) when you run the program.

If you like, you can think of a statement as a sentence and the semicolon as a period. When you're told to write a C++ statement, don't forget the semicolon!!! That would be like writing a sentence without a period (or question mark or exclamation mark) in an english essay.

Exercise 1.2.1. (Go to Solution 1.2.1 on page 12)
Modify your program by removing the first semicolon to get this:

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!\n"

    return 0;
}
```

Run it. Does it work? Fix it. Test to make sure it works (that means: run the program.) □

Exercise 1.2.2. (Go to Solution 1.2.2 on page 12)
Replace the semicolons by periods.

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!\n".

    return 0.
}
```

Run it. Does it work? Fix it. □

See what I mean by this advice:

> *A quick advice to ease the pain of learning your first programming language: Type the program* ***exactly*** *as given. Even the spaces and blank lines must match my programs.*

Computers are dumb (and picky about details.) We are the smart ones. So . . . when you communicate with your computer, you have to be exact and explicit in your programs.

Exercise 1.2.3. (Go to Solution 1.2.3 on page 12)
Now write a program that prints any message (example: "do you want green eggs and ham?") ... close your notes first. Peek at your notes only when you have problems. □

SOLUTION

Solution 1.2.1. (Go to Exercise 1.2.1 on page 10)
No it won't work. You must have the semicolon. □

Solution 1.2.2. (Go to Exercise 1.2.2 on page 10)
Nope. It won't work either. You *must* have the semicolon. □

Solution 1.2.3. (Go to Exercise 1.2.3 on page 11)

```
#include <iostream>

int main()
{
    std::cout << "do you want green eggs and ham?\n";

    return 0;
}
```

□

1.3 C-Strings

The stuff in quotes is called a C-string or just a string:

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!\n" ;

    return 0;
}
```

You can think of a string as textual data. (Soon I'll talk about numeric data for numeric computations. Got to have that for computer games, right?)

Double quotes are used to mark the beginning and ending of the string. So technically they are not part of the string. That's why when you run the above program, you do not see double-quotes.

Exercise 1.3.1. (Go to Solution 1.3.1) Does it work without the double-quotes?

```
#include <iostream>

int main()
{
    std::cout << Hello, world!\n;

    return 0;
}
```

□

In the string `"Hello, world!\n"`, `H` is a character. The next character is `e`. Etc. When we talk about characters we enclose them with single-quotes. So I should say character `'H'` rather than `H` or `"H"`. You can think of the character as the smallest unit of data in a string.

The string `"Hello, world!\n"` contains characters `'H'` and `'e'` and `'l'` and `'l'` and ... However a character such as `'H'` can contain one and only one unit of textual data. So you cannot say that `'Hello'` is a character.

Note that a string can contain as many as characters as you like: 10,

20, 50, 100, etc. There's actually a limit, but we won't be playing around with a string with 1,000,000 characters anyway for now – that would be a pain to type!!! Note that a string can contain no characters at all: `""`. Of course a string can contain exactly one character. For instance, here's a string with one character: `"H"`.

Note that `"H"` is a string with one character whereas `'H'` is a character. You just have to look at what quotes are used to tell if the thingy is a string or a character. That's all.

Exercise 1.3.2. (Go to Solution 1.3.2) Does it work with single-quotes?

```
#include <iostream>

int main()
{
    std::cout << 'Hello, world!\n';

    return 0;
}
```

□

Exercise 1.3.3. (Go to Soluton 1.3.3) Can you also print characters?? (You did think about this, right?) Try this:

```
#include <iostream>

int main()
{
    std::cout << '?';

    return 0;
}
```

□

Exercise 1.3.4. (Go to Solution 1.3.4) Which of the following is a string, a character, or neither:

```
"I"
like
{eggs}
"and"
'ham'
```

'.'

□

SOLUTION

Solution 1.3.1. (Go to Exercise 1.3.1) No! □

Solution 1.3.2. (Go to Exercise 1.3.2) No! □

Solution 1.3.3. (Go to Exercise 1.3.3) Yes. □

Solution 1.3.4. (Go to Exercise 1.3.4) Which of the following is a string, a character, or neither:

```
"I"      string
like     neither
'green'  neither
{eggs}   neither
"and"    string
'ham'    neither
'.'      character
```

□

1.4 Case Sensitivity

Is C++ case sensitive?

Exercise 1.4.1. Modify your program by changing `std` to `Std`:

```
#include <iostream>

int main()
{
    Std::cout << "Hello, world!\n";

    return 0;
}
```

Run it. Does it work? Fix it. □

Exercise 1.4.2. Modify your program by changing `return` to `RETURN`:

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!\n";

    RETURN 0;
}
```

Does it work? Fix it. □

Now answer this question:

Is C++ case sensitive? Circle one: YES NO

(duh ... I'm not taking answers in class.)

1.5 Whitespaces

A whitespace is ... well ... a white space.

Spaces, tabs, and newlines are whitespaces.

Exercise 1.5.1. Modify your program by inserting some spaces:

```
#include <iostream>

int main()
{
    std::cout            << "Hello, world!\n";

    return 0;
}
```

Does it work? □

Exercise 1.5.2. Modify your program by inserting some newlines:

```
#include <iostream>

int main()
{
    std::cout <<




            "Hello, world!\n";

    return 0;
}
```

□

In general you can insert whitespaces between "basic words" understood by C++. These "basic words" are called **tokens**. (Oooooo another big word.)

If you think of statements as sentences, semi-colons as periods, then you can think of tokens as words.

For instance the following are some tokens from the above program: `int`, `return` and even `{`.

We say that C++ **ignores whitespace**.

Exercise 1.5.3. Try this:

```
#include <iostream>

int main()
{
    std :: cout << "Hello, world!\n";

    return 0;
}
```

Does it work? Now try this:

```
#include <iostream>

int main()
{
    std:     :cout << "Hello, world!\n";

    return 0;
}
```

Does it work? □

The above shows you that **::** is a token – you cannot break it down.

Exercise 1.5.4. Try this:

```
#include <iostream>

int main()
{
    std    ::    cout << "Hello, world!\n";

    return 0;
}
```

Does it work? □

Try this:

```
#include <iostream>

int main()
{
    std::cout << "Hello,          world!\n" ;

    return 0;
}
```

The whitespaces between tokens are removed when your computer

runs your C++ program. So, to the computer, it doesn't matter how much whitespace you insert between tokens – it still works.

However spaces in a ***string*** are not removed before the program runs. The space characters ' ' (there are 10 in the above string) are actually characters within the string.

Note that although you can insert whitespaces, in general, good spacing of a program makes it easier to read. Therefore you must follow the style of spacing shown in my notes. In particular, I **don't** want to see monstrous programs like this (although the program does work):

```
#include <iostream>

int                          main(){std
::
                     cout<<

"Hello, world!\n"; return 0; }
```

or this:

```
#include <iostream>
int main(){std::cout<<"Hello, world!\n";return 0;}
```

And don't try to be cute this like ...

```
#include <iostream>
int
 main
  ()
   {
    std
     ::
      cout
       <<
        "Hello, world!\n";
         return 0;
          }
```

The above examples are excellent candidates for the F grade. The correct coding style produces this:

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!\n";

    return 0;
}
```

The left trailing spaces of a statement is called the **indentation** of the statement:

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!\n";
    TODO: left-right red arrow for indentation

    return 0;
}
```

It's a common programming style to use 4 spaces for indentation.

TODO ... Indentation.

1.6 Special characters

Hmmmm ... you don't see the `\n` printed out. In fact what's it for???

Try this:

```
#include <iostream>

int main()
{
    std::cout << "Hello, \nworld!\n";

    return 0;
}
```

And this

```
#include <iostream>

int main()
{
    std::cout << "Hel\nlo, \nworld!\n";

    return 0;
}
```

And finally this:

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!";

    return 0;
}
```

So you can think of the print cursor as jumping to the next line whenever a `\n` is encountered.

As a matter of fact you cannot separate the `\` from the `n`. `\n` is consider ***one*** single character. So technically I should write `'\n'`. `'\n'` is called the **newline character**.

Exercise 1.6.1. Write a program that prints this:

```
^ ^
0 0
 |
---
```

□

Try this:

```
#include <iostream>

int main()
{
    std::cout << "1\t2\tbuckle my shoe\n3\t4\t...\n";

    return 0;
}
```

The `'\t'` move the print cursor to the next tab position. `'\t'` is also a character. It's called the **tab character**.

Special characters like the above (the newline and the tab characters) are not printable and so we have to use some special notation to say "the newline character" or the "the tab character". Computer scientists decided (long time ago) to use \ to indicate a special character. These are called **escape characters**.

Now recall that `"` is used to mark the beginning and end of a string. What if you want to print something like this:

.....".....

Try this:

```
#include <iostream>

int main()
{
    std::cout << ".....".....";

    return 0;
}
```

Doesn't work right? Do you see why?

Now try this:

```
#include <iostream>

int main()
{
    std::cout << "He shouted, \"42!\"\n";

    return 0;
}
```

So in a string, `\"` will let you print `"`.

`'\"'` is actually a character. There are two `'\"'` characters in the above string `"He shouted, \"42!\"\n"`.

Exercise 1.6.2. Write a program that prints this:

```
She said, "I would rather marry a pig."
```

□

Exercise 1.6.3. Write a program that prints this: Here's a standard formula:

```
       2      2          2
(x + y)   =  x  + 2xy + y
```

There are two `' '` characters on each side of the `'='` character. □

Exercise 1.6.4. Write a program that prints this:

```
 d  3     2
-- x  = 3x
dx
```

□

Exercise 1.6.5. How would you print the character `'`. Let me tell you that

```
std::cout << ''';
```

won't work. □

Exercise 1.6.6. Write a program that print the backslash character, i.e. `\`. □

Exercise 1.6.7. Write a program that prints this:

```
He said, "I am! I'm a pig! Really!"
```

□

Exercise 1.6.8. Write a program that prints this:

```
Get the Gold!
+--------------+
|              |
| --------+ | |
|         | | |
| +---+ | | | |
| |G  | | |   |
| +-+ +-+ | | |
| | | |       | |
| | +-----+-+ |
|               <---
+--------------+
```

□

Exercise 1.6.9. Write a program that prints this:

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!\n";

    return 0;
}
```

□

The number of characters is a string is called **length** of the string. For instance the string

```
"Hello, world!\n"
```

has a length of 14.

Exercise 1.6.10. What is the number of characters (i.e., the length) of the following strings?

- `"columbia"`
- `""`
- `"columbia,mo"`
- `"columbia, mo"`
- `"ma ma mia!"`
- `"ma ma mia!\n"`
- `"ma\tma\tmia!\n\n"`

□

(You'll see much later that in every string, there's actually an extra secret character. This is however not included in the count of the length of the string. Don't worry about this for now. I will be coming back to this later.)

1.7 Printing More than One Strings or Characters

Try this

```
#include <iostream>

int main()
{
    std::cout << "Hello, " << "world!\n";

    return 0;
}
```

And this:

```
#include <iostream>

int main()
{
        std::cout << "Hello, " << "world!\n"
                  << "Spam, \n";

        return 0;
}
```

And this:

```
#include <iostream>

int main()
{
        std::cout << "Hello, " << "world!\n"
                  << "Ham, " << "eggs! \n";

        return 0;
}
```

And this:

```
#include <iostream>

int main()
{
        std::cout << "Hello, " << "world!\n"
                  << "Ham, " << "e" << "ggs! \n";

        return 0;
}
```

By the way you should try this:

```
#include <iostream>

int main()
{
    std::cout << "a" << "" << "b";

    return 0;
}
```

"" is an **empty string** (also called a null string): it's a string with no characters.

Exercise 1.7.1. The output of the following program:

```
#include <iostream>

int main()
{
    std::cout << "ham", "and", "eggs"
              << "\n";

    return 0;
}
```

is

```
ham and eggs
```

True or false? (Verify with your C++ compiler.) □

Exercise 1.7.2. The output of the following program:

```
#include <iostream>

int main()
{
    std::cout << "ham" << "and" << "eggs"
              << "\n";

    return 0;
}
```

is

```
ham and eggs
```

True or false? (Verify with your C++ compiler.) □

Of course you can also print multiple characters. Try this:

```
#include <iostream>

int main()
{
    std::cout << 'H' << 'a' << 'm' << '\n';

    return 0;
}
```

Here's a tiring hello world program:

```
#include <iostream>

int main()
{
    std::cout << 'H' << 'e' << 'l' << 'l' << 'o'
              << ',' << ' '
              << 'w' << 'o' << 'r' << 'l' << 'd'
              << '!'
              << '\n';

    return 0;
}
```

You can also mix printing strings and characters in a single statement:

```
#include <iostream>

int main()
{
    std::cout << "Hel" << 'l' << "o, "
              << 'w' << "orld!\n';

    return 0;
}
```

1.8 Another Way to Go to the Next Line, pingas

Let's go back to our hello world program:

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!\n";

    return 0;
}
```

Run yours and make sure there are no errors.

First let's make a separate string out of the newline character:

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!" << "\n";

    return 0;
}
```

Run it and make sure the effect is still the same.

Now do this:

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!" << std::endl;

    return 0;
}
```

Run it.

The `std::endl` acts like a newline character. It's actually more than that. We'll talk about this when we talk about files. Anyway for the time being just remember that `std::endl` forces a newline when you print it.

Exercise 1.8.1. Read this program and, without running it, write down the output in the grid provided below. Once you're done writ-

ing down the output, run the program and verify that you output from reading the program matches the output you get when running the program.

```
#include <iostream>

int main()
{
    std::cout << "Lives: 3" << std::endl
              << "Energy: 15000" << std::endl
              << "Score: 0" << std::endl;

    return 0;
}
```

Output (one per character):

1.9 Multiple Statements

Exercise 1.9.1. Run this program

```
#include <iostream>

int main()
{
    std::cout << "1!\n";
    std::cout << "2!\n";

    return 0;
}
```

This tells you that you can have **more than one statement** in your program. (Actually the `return 0;` is also a statement, but we'll ignore that for now.)

Furthermore C++ executes from **top to bottom** (at least right now!)

Exercise 1.9.2. The following program has three print statements:

```
#include <iostream>

int main()
{
    std::cout << "Hello World! ";
    std::cout << "I'm the queen of England. ";
    std::cout << "I have 3 arms.";

    return 0;
}
```

Run it. Rewrite it so that it has only one print statement. □

Exercise 1.9.3. True or False? The output of this program (when you run it of course)

```
#include <iostream>

int main()
{
    std::cout << "this is the last line\n";
    std::cout << "this is the second line\n";
    std::cout << "this is the first line\n";

    return 0;
}
```

is

```
this is the first line
this is the second line
this is the last line
```

□

Exercise 1.9.4. Continuing with the previous program, rewrite it so that each sentence is printed on a separate line; use only one statement. □

1.10 The `using namespace std` business

Instead of our hello world program:

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!\n";

    return 0;
}
```

in many books you will see this:

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

int main()
{
    cout << "Hello, world!\n";

    return 0;
}
```

All the

```
using namespace std;
```

does is that after it, instead of

```
std::cout
```

you can write

```
cout
```

i.e. without the

```
std::
```

That's all. I'll talk about "namespaces" later (actually much, much, much later ... in CISS245). So don't worry too much about it.

Exercise 1.10.1. Does the following program work?

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

int main()
{
    cout << "Hello, world!" << endl;

    return 0;
}
```

□

So not only can you replace `std::cout` with cout after doing the

```
using namespace std;
```

you can also replace `std::endl` with `endl`.

1.11 Summary

A **statement** is something that will cause your computer to perform some operation(s). In C++ statements must terminate with a semi-colon.

C++ is case-sensitive.

C++ ignores whitespace (between tokens).

Something that looks like `"Hello, World!\n"` (i.e. characters within double quotes) is called a **string** or a **C-string**.

A **string** is either empty (the null string), i.e. `""`, or it is made up of characters. For instance the first character of the string

```
"Hello, World!\n"
```

is `'H'`.

There are special characters: `'\n'` causes the cursor to jump to a new line while `'\t'` moves to the next tab position, `'\"'` is the double-quote character, and `'\''` is the single-quote character. The backslash character is `'\\'`. These are called **escape characters**.

To print a string, you execute the following statement:

```
std::cout << [some string];
```

You can print more than one string in a single print statement:

```
std::cout << [string1] << [string2]
          << [string3] << [string4];
```

Printing one or more characters is the same:

```
std::cout << [some character];
std::cout << [character1] << [character2]
          << [character3] << [character4];
```

You can mix print strings and characters. For instance

```
std::cout << [some character]
          << [some string];
```

or

```
std::cout << [some string]
          << [some character];
```

You can force a newline by printing `std::endl`:

```
std::cout << std::endl;
```

If your program has two or more statements like this:

```
[statement 1];
[statement 2];
[statement 3];
```

then C++ executes top to bottom (for now).

1.12 Exercises

1. Write a C++ program that produces the following output:

```
"Prediction is very difficult,
especially about the future."
    Niels Bohr
    Danish physicist (1885 - 1962)
```

Use one print statement.

2. True or false: The output of

```
#include <iostream>

int main[]
{
    std::cout << "To be ..."
              << "or not to be ..."
              << "That" << "is the question."
              << std::endl

    return 0
}
```

is

```
To be ...
or not to be ...
That is the question.
```

Verify with your C++ compiler.

3. Find and correct all the error in this program (without using your C++ compiler):

```
#include <<iostream>>

int main[]
{
    std::cout >> "Hello!/n"
    std::cout >> "I'm the king of Spain./n"
    std::cout >> "And I have 3 arms." >> std::end

    return 0
}
```

The expected output is

```
Hello!
I'm the king of Spain.
And I have 3 arms.
```

When you're done, verify your correction with your C++ compiler. If

your program does not compile, continue correcting the program until the program is error-free and runs correctly.

4. Will this program get an A from the instructor?

```
#include <iostream>

int main()
{
    std::cout<<"Hello, world!"<<std::endl;

    return 0;
}
```

Why? What about this:

```
#include <iostream>

int main()
{


    std::cout<<"Hello, world!"<<std::endl;


    return 0;
}
```

5. What is the length of the following string:

```
"Ablee, \tAblee, \tAblee, \t... That's all folks!!!\n"
```

6. A string with length 0 is called an empty string or a _____ string.

7. All C++ statements must end with a _____.