0. Annotated slides from Wednesday

CS319: Scientific Computing

Projects; Strings, and Files and Streams (draft)

Dr Niall Madden

Week 8: 6 + 8 March, 2025

Slides and examples: https://www.niallmadden.ie/2324-CS319

0. Outline

1. Projects!
2. Recall: objects
3. Strings
   - Operator overloading
4. I/O streams as objects
   - manipulators
5. Files
   - ifstream and ofstream
   - open a file
   - Reading from the file
   - Tip: working with files
6. Portable Bitmap Format (pbm)

Slides and examples:

https://www.niallmadden.ie/2425-CS319

1. Projects!

Notes for this part are at:

https://www.niallmadden.ie/2425-CS319/2425-CS319-Projects.pdf

2. Recall: objects

Last week we learned that

- A **class** is a general form of data type that we can create;
- An **object** is an instance of a particular class. E.g,

  ```
  MyStack  s1, s2;
  ```

  Here s1 & s2 are objects of type MyStack.
- A **method** is a member of a class that is a function. E.g.,

  s1.pop()   s2.push();

Note: The notes for last week were updated after the class to give a more coherent view on constructors and destructors. See https://www.niallmadden.ie/2425-CS319/Week07/CS319-Week07.pdf

. . . .

Before we continue with writing our own classes, we can now visit some important related topics in C++:

`string`s   *   **input and output streams**   *   **files**

3. Strings

A **string** is a collection of characters representing, for example, a word or a sentence.

In C++, a `char` array can be used to store a string. That approach is called a “C string”, since it is inherited from an older language, C.

Such “C strings” are not so easy to work with, so C++ provides its own `string` class. The class can be accessed once the `string` header file is included. It is part of the `std` namespace.

```
#include <string>
.
.
.
std::string name;
```

3. Strings

We have used `string` before (Weeks 1 and 2), but have not thought of it as a class.

Since it is a class, it has some methods, including:

- `length()` and `size()` which both return the number of characters in the string; Eg: name.lengt();
- `substr(i,l)` with returns a substring of length `l`, starting at position `i`. name.substr(3,4)
- `find()` which finds the first occurence of one substring in another.
- `c_str()` return the "C string" version. (Need this when working with files).

3. Strings

Example

Write a short C++ program that defines a `string` containing a sentence, and then extract the first word as another `string`.

00substring.cpp

```cpp
#include <iostream>
#include <string>

int main(void)
{
  std::string
    sentence="Ada Lovelace was the first programmer",
    first_word;
  int space_loc = sentence.find(" ");     // Find first space
  first_word = sentence.substr(0,space_loc); // extract substring

  std::cout << "sentence is: " << sentence << std::endl;
  std::cout << "first word is: '" << first_word << "'\n";
  return(0);
}
```

3. Strings

Expected output:

```
sentence is: Ada Lovelace was the first programmer
first word is: ‘Ada’
```

Operator overloading

With numbers, we are used to working with special functions called **operators**, which are usually represented by a mathematical symbol, such as `+`, `-`, `=`, `*`, `/`, etc.

When writing our own `class`, we can overload some of these (more about the details later).

The `string` class overloads several operators:

- Assignment: `=`

we'll learn we should always code this.

- Relational: `==`, `>`, `<`, etc;
- Arithmetic: `+`, `+=`

01string-operators.cpp

```cpp
#include <iostream>
#include <string>

int main(void)
{
  std::string name[3], // array of names
    long_name="";
  name[0]="Augusta";
  name[1]="Ada";
  name[2]="King";

  long_name = name[0] + " " + name[1] + " " + name[2];

  std::cout << "long_name: " << long_name << std::endl;
  return(0);
}
```

Output

```
  long_name: Augusta Ada King
```

4. I/O streams as objects `cout, cin`

`I/O` means “Input/Output. So far, we have taken input from the keyboard, typically using `cin`, and sent output to a terminal window, using `cout`.

These are examples of **streams**: flows of data to or from your program. Moreover, they are examples of **objects** in C++.

In fact `cout` and `cin` are **objects** and are manipulated by their **methods**, i.e., public member functions and operators. (We saw this in Week 3)

Methods:

- `width(int x)` – minimum number of characters for next output,
- `fill(char x)` – character used to fill with in the case that the width needs to be elongated to fill the minimum.
- `precision(int x)` – sets the number of significant digits for floating-point numbers.

4. I/O streams as objects   cout, cin

Code – width, fill

```
std::cout.fill('0');
for (int i=0; i<8; i++)
{
  std::cout.width(6);
  std::cout << rand()%200000
     << std::endl;
}
```

"Pad with zeros"

use at least 6 characters.

Output

```
089383
130886
092777
036915
147793
038335
085386
160492
```

4. I/O streams as objects

cout, cin

Code – precision

```
double Pi=3.1415926535;
for (int i=1; i<=8; i++)
{
  std::cout.precision(i);
  std::cout << "Pi (correct to "<< i << " digits) is "
            << Pi << std::endl;
}
```

Output

```
Pi (correct to 1 digits) is 3
Pi (correct to 2 digits) is 3.1
Pi (correct to 3 digits) is 3.14
Pi (correct to 4 digits) is 3.142
Pi (correct to 5 digits) is 3.1416
Pi (correct to 6 digits) is 3.14159
Pi (correct to 7 digits) is 3.141593
Pi (correct to 8 digits) is 3.1415927
```

4. I/O streams as objects manipulators

Finised here Wednesday

- `setw` – like `width`
- `left` – Left justifies output in field width. Used after `setw(n)`.
- `right` – right justify.
- `endl` – inserts a newline into the stream and calls flush.
- `flush` – forces an output stream to write any buffered characters
- `dec` – changes the output format of number to be in decimal format
- `oct` – octal format
- `hex` – hexadecimal format
- `showpoint` – show the decimal point and some zeros with whole numbers

Others: setprecision(n), fixed, scientific, boolalpha, noboolalpha, ...

Need to include `iomanip`