Annotations from Tuesday

CS319: Scientific Computing (with C++)

Week 5: Streams and files

9am, 09 March, and 4pm, 10 March, 2021

1. Part 1: Review of classes
   - Constructors
2. Part 2: Destructors & Constructors
   - Destructors
   - Constructor again
3. Part 3: I/O streams as objects

Today

   - manipulators
4. Part 4: Files
   - ifstream and ofstream
   - open a file
   - Reading from the file

Tomorrow

5. Part 5: Portable Bitmap Format (pbm)
6. Part 6: Templates

Usual reminders...

	Mon	Tue	Wed	Thu	Fri
9 – 10		LECTURE	✗		
10 – 11		LAB			
11 – 12					
12 – 1					
1 – 2		LAB			
2 – 3					
3 – 4					
4 – 5			LECTURE		

1. We'll have recorded classes on We~~dnesday~~s Tue at 9.00 and ~~Thurs~~days wed at 16.00.
2. **Lab times: Tuesday 10.00-10:50, and 13.00-13.50**. You should try to attend at least one of these.
3. A short introduction to the lab will be recorded.

CS319 – Week 5
Week 5: Streams and files

Start of ...

PART 1: Review of classes

Part 1: Review of classes

class

In C++, we defined new classed with the `class` keyword.
An instance of the class is called an "*object*".
A `class` combines by data and functions.

Within a class, code and data may be either

- **Private**: accessible only to another part of that object, or
- **Public**: other parts of the program can access it.

Roughly,

- keep data elements `private`,
- make function elements `public`.

Part 1: Review of classes

The basic syntax for defining a class:

Keywords are class, private, public

```
class class-name {
private:
     ...      // private functions and variables
public:
     ...      // public functions and variables
};
```

class-name becomes a new object type—one can now declare objects to be of type *class-name*.

This is only a declaration. Therefore,

- functions are not defined, though the prototype is given,
- variables are declared but are not initialised,
- the declaration block is delineated by { and }, and terminated with a semicolon.
- use *scope resolution operator* `::` to combine a class name and element/member name.

CONSTRUCTOR

A **Constructor** is a public member function of a class.

- It has the same name as the class.
- It's return type is not specified explicitly.
- It is executed whenever a new instance of that class is created.

Constructors may contain any code you like; but it is good practice to only use them for initialization and, especially **Dynamic memory allocation** (see Part 7 of Week 4).

CS319 – Week 5
Week 5: Streams and files

END OF PART 1

CS319 – Week 5
Week 5: Streams and files

Start of ...

PART 2: Destructors and Constructors

Part 2: Destructors & Constructors

Destructors

Complementing the idea of a constructor is a **destructor**. This function is called

- for a local object – whenever it goes out of scope,
- for a global object – when the program ends.

The name of the destructor is the same as the class, but preceded by a tilde.

Recall the MyStack example from last week:

```
class MyStack {
private:
  char *contents;
  int top;
public:
  MyStack(void );
  ~MyStack(void );
  void push(char c);
  char pop();
};
```

same as class name, but starts with ~

```
MyStack::~MyStack()
{
  delete [] contents;
}
```

de-allocates memory allocated using "new" in the constructor

Part 2: Destructors & Constructors Constructor again

The example we had earlier of a constructor was particularly basic, not least because is its parameter list is `void`. More commonly, one passes arguments to the constructor that can be used, e.g.,

- to set the value of a data member;
- dynamically size an array using `new`.

However, one should still provide a default constructor (i.e., one with no arguments), or one with a default argument list.

Overloaded constructor (ie 2 versions)

```
class MyStack
{
private:
  char *contents;
  int top;
public:
  MyStack(void);
  MyStack(unsigned int MyStackSize);
  void push(char c);
  char pop(void );
};
```

```
MyStack::MyStack(void)
{
  top=0;
  contents = new char[MAX_STACK];
}
```

```
MyStack::MyStack(unsigned int StackSize)
{
  top=0;
  contents = new char[StackSize];
}
```

Part 2: Destructors & Constructors Constructor again

CS319 – Week 5
Week 5: Streams and files

END OF PART 2

Part 3: I/O streams as objects

CS319 – Week 5
Week 5: Streams and files

Start of ...

PART 3: I/O streams as objects

Part 3: 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 this section, we'll study how to manipulate these streams in C++, including writing to and reading from files.

But first, some more information about cout and cin.

"Eye - Oh"

Part 3: I/O streams as objects cout, cin

The objects `cout` and `cin` are objects and are manipulated by their **methods**, i.e., public member functions and operators.

Methods: (for cout).

- `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.

Last week: we called methods pop() and push() for stacks.

~>

Part 3: I/O streams as objects

cout, cin

Called ASCII ↘

Code – width

```
for (int i=65; i<123; i++)
{
  std::cout.width(8);
  std::cout << i;
  std::cout.width(3);
  std::cout << (char) i;
  if ( (i%5) == 4)
    std::cout << std::endl;
}
```

Output

```
 65  A     66  B     67  C ...
 70  F     71  G     72  H ...
 75  K     76  L     77  M ...
 80  P     81  Q     82  R ...
 85  U     86  V     87  W
 90  Z     91  [     92  \
 95  _     96  ‘     97  a
100  d    101  e    102  f
105  i    106  j    107  k
110  n    111  o    112  p
115  s    116  t    117  u
120  x    121  y    122  z
```

The int 65 corresponds to "A".

Extra 5 spaces to bring up to 8.

Part 3: 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;
}
```

Output

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

Part 3: I/O streams as objects

Code – precision

```
double Pi=3.1415926535;
for (int i=1; i<=10; i++)
{
  std::cout.precision(i);
  std::cout << "Pi (correct to "<< i << " digits) is "
            << Pi << std::endl;
}
```

std::cout << precision(i)
<< . . .

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
Pi (correct to 9 digits) is 3.14159265
Pi (correct to 10 digits) is 3.141592654
```

Part 3: I/O streams as objects

manipulators

- `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`

Part 3: I/O streams as objects

All of the C++ programs we have looked at so far took their input from the *standard input stream*: this was usually the keyboard.

Example:

```
std::cout << "Enter an inteter: ";
std::cin >> i;
```

Although, for example, the *standard input stream* can be redirected to a file, it is usually necessary to open a file **from within the program** and take the data from there.

The same is true for writing to a file.

To do either of these takes in C++ we create a **file stream** and use it just as we would `cin` or `cout`.

CS319 – Week 5
Week 5: Streams and files

END OF PART 3

CS319 – Week 5
Week 5: Streams and files

Start of ...

PART 4: Files

Recorded towards the end of Tuesday's class.

Part 4: Files

All of the C++ programs we have looked at so far take their input from the *standard input stream*, which is usually the keyboard. Example:

```
std::cout << "Enter an inteter: ";
std::cin >> i;
```

Although the *standard input stream* can be redirected to be, for example, a file (easily done on a Mac and on Linux), it is usually necessary to open a file **from within the program** and take the data from there. The data is then processed and written to a new file.

Part 4: Files

To achieve either of these tasks in C++, we create a **file stream** and use it just as we would `cin` or `cout`.

We'll start by looking at a simple example:

(i) open a file,

(ii) count the number of characters,

(iii) save this number to a new file.

Once we have the basic idea, we'll take a closer look at each operation (opening, reading, writing).

ifstream and ofstream

Also download "CPlusPlusTerms.txt".

When working with files, we need to include the *fstream* header file.

To **read** from a file, declare an object of type ifstream; to **write** to a file, declare an object of type ofstream.

Open the file by calling the open() member function.

To read a single character, can use *InFile.get()*

01CountChars.cpp

```
#include <iostream>
#include <fstream>
#include <cstdlib>

int main(void )
{
  std::ifstream InFile;
  std::ofstream OutFile;
  char c;

  std::cout << "Processing ..."
      << " CPlusPlusTerms.txt";
  std::cout << "See file Output.txt for"
      << " more information.";
  InFile.open("CPlusPlusTerms.txt");
  OutFile.open("Output.txt");

  int i=0;
  InFile.get( c );
```

"file stream".

if = "input file".

"output file" — where I write to.

Part 4: Files

close a file

Eof = "End of file" | "not"

If there are no more characters left in the input stream, then `InFile.eof()` evaluates as *true*.

Use the steam objects just as you would use `cin` or `cout`:

`InFile >> data` or
`OutFile << data`.

Close the files:
`InFile.close()`,
`OutFile.close()`

01CountChars.cpp

```
  while( ! InFile.eof() ) {
    i++;
    InFile.get( c );
  }

  OutFile <<
    "CPlusPlusTerms.txt contains "
    << i << " characters \n";

  InFile.close();
  OutFile.close();

  return(0);
}
```

using this just like cout

Finished here Tue @ 9·54 (sorry!)