Lecture 4
Strings, more I/O, and file handling

Dr David Harle
Dr Christos Tachtatzis

The course so far…

- As your programs get more complex, good structure and even good typesetting become more important
- Get used to the 'integrated development environment' (IDE) for managing lots of code
- The lecture slides are really just a summary of ideas and do not replace a book
- A *compiler* (provided within the IDE)
  1. turns the *source code* into *object code*
  2. then links the objects together into an *executable*
- The *executable* is what you run
- Procedural code comprises statements within *functions*
  - statements within functions are executed in the order in which they appear

Structure of a source code module

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

Headers

```
using namespace std;
```

Any other stuff ‘global’ across the module

```
<type of return value> <functionName> (<arguments>)
{
    …
}
```

Zero, one or many functions
(In an object oriented program, the functions are all part of a class and do things to objects)

```
int main (int argc, char *argv[])
{
    …
}
```

`main`’s arguments
Can be empty, i.e. ( )

`main` is a function like any other – just its name and arguments are particular

`main`’s return type

Exactly one `main` function per program

Gathering code within { … }

```
int testuser (int target)
```

A function. It 'knows about' `target`, `guess` and `n`

```
{
    int guess, n;
```

What the function does is inside { … }

```
    for (n=0; n < 10; n++) {
        cout << "Enter a guess at the random number." << endl;
        cin >> guess;
        if (guess == target){
            cout << "Well done! It was " << target << endl;
            return 1;
        }
        else if (guess > target)
            cout << guess << " is too big." << endl;
        else cout << guess << " is too small." << endl;
    }
    cout << "You failed. It was " << target << endl;
    return 0;
}
```

for loop – code done each time inside { … }

if – code done if condition met inside { … }

Using the Integrated Development Environment (IDE) – lab highlights

- A 'solution' (i.e. a project) can comprise
  - a number of different *sources* that you have written
    - for simple programs, one is enough
  - a number of different *headers* that you have written
    - we will come on to use of these later
- A 'solution' should only have one `main` function
- Avoid confusing the 'solution' with old sources!
  - You could end up with more than one `main` – not allowed!
  - Remove sources you are not using from the 'solution'
- The easiest way to run your program is with 'Start without debugging' (or 'Start with debugging' – see a later lecture)
  - The executable doesn't need the IDE in order to run
    - use the Windows Accessories 'command prompt'

Using the IDE

string_ex - Visual C++ 2005 Express Edition

File Edit View Project Build Debug Tools Window Community Help

Debug Win32

Solution Explorer - Solution 's...

Display Word Completion.

Solution 'string_ex' (1 project)
- string_ex
  - Header Files
  - Resource Files
  - Source Files
    - string_ex_main.cpp

string_ex_main.cpp | Start Page

(Global Scope) | main()

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

using namespace std;

int main()
{
    string my_string;

    cout << "Type in a word\n";
    cin >> my_string;
    cout << "The word is " << my_string << endl;
    system ("PAUSE");
}
```

Current line number in code editing

Messages from the compiler (including errors and warnings)

Output

Show output from: Build

```
1>Compiling...
1>string_ex_main.cpp
1>Linking...
1>Embedding manifest...
1>Build log was saved at "file://h:\Teaching\19.267 - C++\Visual cpp express code\string_ex\string_ex\Debug\BuildLog.htm
1>string_ex - 0 error(s), 0 warning(s)
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Code Definition Window | Call Browser | Output

Build succeeded | Ln 10 | Col 32 | Ch 29 | INS

Strings

- A `string` variable (in C++, actually a `string` *object*) is used to store a string of characters, e.g.
  - `"Hello world"`
  - `"3.1415"`

Note the use of quotes!

- Very useful for manipulation of text!
- The equivalent in C was an *array* of type `char`
  - We will do arrays next week
  - Don't worry too much about `char`
    - Just notice that some standard library functions have `char` arguments
    - You can do a conversion from `string` to a `char` array using the `c_str()` function – see later

The `string` class

- The variable type `string` can be accessed by including the C++ `string` 'standard template library'

```
#include <string>
```

  - This defines a string *class*
  - It contains *variables* and *functions*
  - In object oriented programming,
    - a variable of type `class` is known as an *object*
    - functions in classes are sometimes known as *methods*
  - The `string` class includes functions and operators:
    - If you have `s1`, `s2` and `s3` each of type `string`, and then do

```
s3 = s1 + s2;
```

the program would know what to do (concatenate `s1` and `s2` and copy the result to `s3`)

'Operator overloading' – see later in course

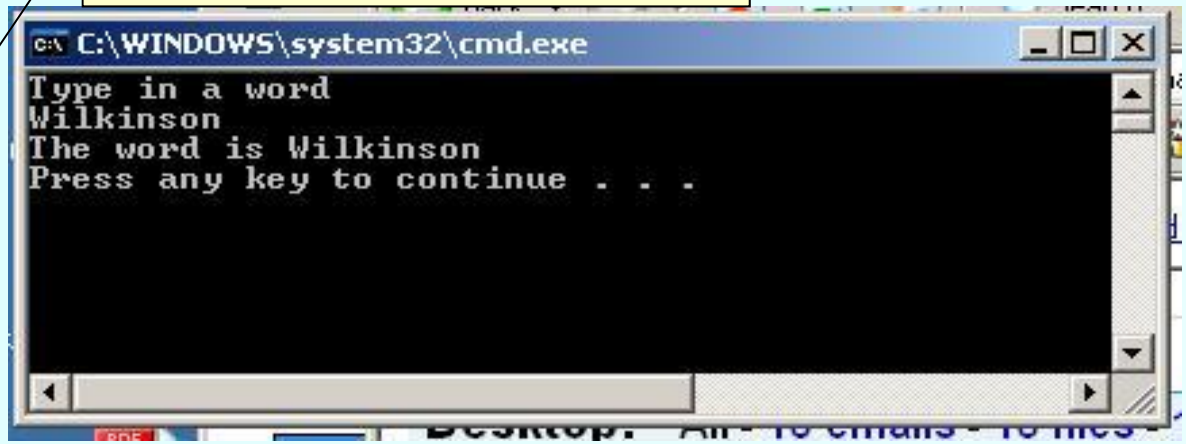
More on objects later in the course

A simple string example

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

using namespace std;

int main()
{
    string my_string;

    cout << "Type in a word\n";
    cin >> my_string;
    cout << "The word is " << my_string << endl;
}
```

Include the `string.hpp` header

Declaration of a `string`



`\n` makes a carriage return…

Show `my_string` on the screen

…so does `<< endl;`

String manipulation functions - 1

- The `string` class makes various functions available for working with strings
  - Need to include the `string.hpp` header

```
#include <string>
```

- Examples of functions available
  - Finding the length of a string called name
    - the `name.size()` function
  - Testing for empty strings: returns TRUE or FALSE
    - `name.empty()`
  - Comparing strings
    - `name.compare()`

C++ headers are `.hpp`
- You don't need the `.hpp` in the `#include`

C headers are `.h`
- Some standard functions are normally only available in the C library, e.g. `math.h`
- You do need the `.h` in the `#include`

String manipulation functions - 2

- Some more functions available in the `string` class
- In these examples,
  - the functions are applied to a string called `name`
  - *<a_string>* could be a `string` object (i.e. a variable that is declared to be a `string` and has a value) or a 'literal', e.g. "`some text`"
- e.g. copy *<a_string>* to `name`
  - `name.assign(`*<a_string>*`)` or simply `name =` *<a_string>*
- Finding a substring *<a_string>* within `name`
  - `int pos = name.find(`*<a_string>*`)`
- `c_str()` function
  - Converts a `string` to a C style string, i.e. array of `char`
  - Lets you use some library functions that have `char` as argument

`find` returns an integer:
tells the position at which *<a_string>* is found within *name*

Examples of use of `string` functions

```
int main()
{
    string my_string1, my_string2, my_string3;

    while (my_string1.empty()) {
        cout << "Type in a word\n";
        cin >> my_string1;
    }
    cout << "Type in another word\n";
    cin >> my_string2;

    if (my_string1.compare(my_string2) == 0)
        cout << "The two words are the same." << endl;

    my_string3 = my_string1 + my_string2;
    cout << my_string3 << endl;
}
```

`#include` statements and `using namespace std;` omitted here simply to save space on slide

Continue to prompt the user until they enter something…

Using the `empty()` function of the `string` class on the `my_string` instance of it

Strangely, `compare` returns 0 when it's true…

Assign the concatenation of `my_string1` and `my_string2` to `my_string3`

Another example of using `string` member functions

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

using namespace std;

int main()
{
    string claim = “C++ is difficult";

    cout << "The string <" << claim << "> has " <<

        claim.size() << " characters\n";

    string correction = "not ";

    claim.insert(7, correction);

    cout << "The truth is, " << claim << endl;

}
```

Create an instance of a `string` called `claim` with the initial value “C++ is difficult”

Use the `size` method on the string `claim`

Create a string called `correction`

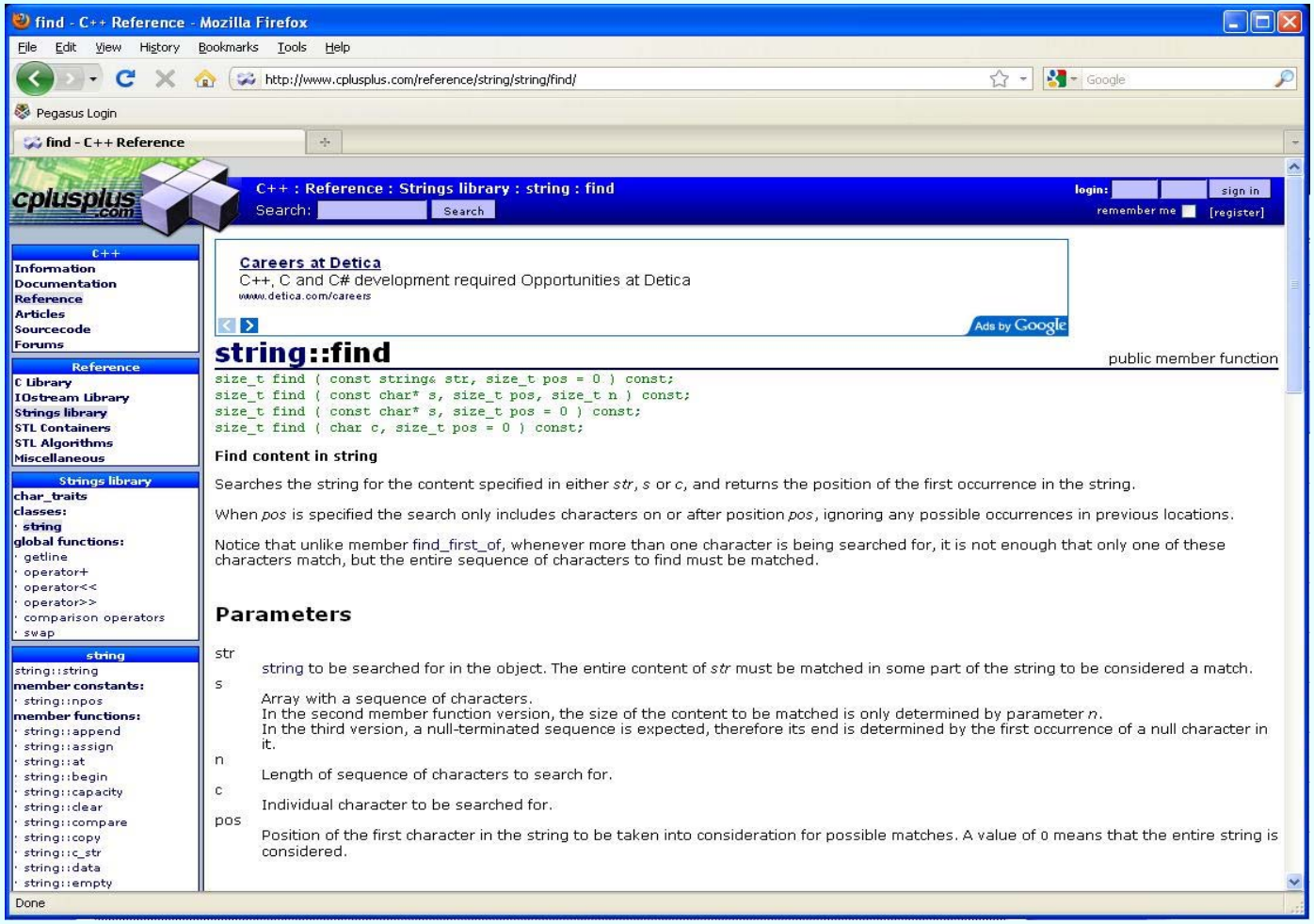
Use the `insert` method to insert `correction` into `claim`

```
The string <C++ is difficult> has 16 characters
The truth is, C++ is not difficult
```

Looking up details of functions

A reference such as cplusplus.com or a text book can tell you

- what library a function is in
- what inputs it needs
- what outputs it gives



String input problems

- The `cin` class for inputting strings can be used for single characters or single words/numbers
  - It mostly succeeds in assigning values of different types
  - However, characters after a space or tab will be
    - ignored, or
    - treated as relating to a second insertion

Spaces in inputs


```
int main()
{
    string my_string;

    cout << "Type in a sentence\n";
    cin >> my_string;

    cout << my_string << endl;
}
```

getline

- The function `getline` can be used to read from an input stream into a string
  - Reads up to the first carriage return

```
int main()
{
    string my_string;

    cout << "Type in a sentence\n";
    getline(cin, my_string);

    cout << my_string << endl;
}
```

read a sequence of characters from the console until the newline character is found;
put results into `my_string`

`Getline` can take a 3rd arguement

`cin` is actually a special stream: reads from the console

Example of using `getline`

File input and output

- It is useful to be able to read from a file, not only the 'console'
- Reading from a file or the 'console' is done using 'streams'
- The classes `ifstream` and `ofstream` can be accessed by including the C++ `fstream` library

```
#include <fstream>
```

  - Like most classes, the `ifstream` and `ofstream` *classes* contain *variables* and *functions*
  - They include the following functions:
    - `open()`
    - `close()`
  - Reading from a file is perhaps best done with `getline`

Opening and reading from a file

```
int main()
{
    string myfilename, mystring;

    cout << "Enter the name of the file to open\n";
    cin >> myfilename;

    ifstream inFile;
    inFile.open(myfilename.c_str());

    if (!inFile) {
        cout << "Error opening file " << myfi
        return -1;
    }
    while(!inFile.eof()) {
        getline(inFile, mystring);
        cout << mystring << endl;
    }
    inFile.close();
}
```

`mystring` and `myfilename` are instances of C++ 'standard template library' class `string`

Get the name of the file to open from and place it in `myfilename`

`inFile` is an instance of C++ standard class `ifstream`

`open` wants an C-style array of `char`; function `c_str` in class `string` does that conversion

`open` is a function that is defined as part of class `ifstream`. Its result is assigned to another part of the `inFile` object

Close `inFile` when finished

Read from `inFile` until the end of the file is reached

Writing to a file

```
int main ()
{
    ofstream outFile("myoutfile.txt");


    if (!outFile) {
        cout << "Error opening file" << endl;
        return -1;
    }
    outFile << "This is my output file." << endl;
    outFile.close();
}
```

We're going to use an output file stream

In this example, the name of the file is set 'at compile time'

Use the inserter to write lines of text to the output file

Don't forget to close the file when you've finished!

Appending to an existing file

Additional argument to specify opening in append mode

```
int main ()
{
    ofstream outFile("myoutfile.txt", ios::app);

    if (!outFile) {
        cout << "Error opening file" << endl;
        return -1;
    }
    outFile << "This is my output file." << endl;
    outFile.close();
}
```

The ‘format string’

- The format string contains 3 types of things.
  - Characters that we want to print out. Similar to the **string literal** used in the `cout` examples
  - Special characters: always proceeded by a `\` character.
    - `\t` – tab
    - `\n` – newline
    - `\0` – teminator character

    e.g. `cout << “My name is Fred\n”;`
  - **Place holders.** These tell the function where you want to print the values of the variables in the parameter list.

Formatting output using inserters, or 'modifiers'

Examples:

```
const double pi = 3.1415926;
string mystring = "What fun to learn C++";
```

Pad out with spaces if necessary so that 30 characters are written

```
cout << setw(30) << mystring << endl;
cout << setprecision(5) << pi << endl;
```

Write to 5 significant figures

Here, sending to the `cout` stream but could be to an already open file stream

`setw()` and `setprecision()` are modifiers available in the `iostream` library

To find out more…

- You can find more details on the `string` class including string manipulation functions, for example, in
  - a text book
  - http://www.cplusplus.com/reference/string/string/

Next time

- Arrays
- Dynamic memory allocation
- Arrays as arguments in functions
- Brief introduction to pointers