UNIVERSITY OF LIVERPOOL

Application Development with C++ (ELEC362)

Lecture 2: Basics of C++ programmes

mihasan@liverpool.ac.uk

Previous lecture

- An overview of C++
- Difference between complied and interpreted languages.
- The role of compliers and how executable files are created.
- Introduction to Visual Studio.
- Basic rules of C++ coding.

This lecture

- What is covered in this lecture?

  1. Header inclusions and their roles in a code 2. Some common libraries

- Why it is covered?

  1. Headers files allow codes to make use of other parts of codes written already.

  2. Libraries allow for high-level functionality to be introduced in a programme.

- How are topics covered in this lecture:

  3 source codes and error type demonstration.

Parts of a simple code

```
# include <iostream>
```

Header section
(this lecture)

```
int main(){

  int x = 0;

  std::cout << "x = " << x;

}
```

Variable (lecture 3)

Function (lecture 5)

C++ components

- Generally there are three core components of C++, these are:

1. Keywords (analogous to alphabets in human language).

2. Syntax rules (analogous to grammar in human language).

3. Developer-written statements (analogous to words in human language).

- All C++ keywords are given here: https://en.cppreference.com/w/cpp/keyword
- QUESTION: Are these keywords enough to build a highly functional programme?

Pre-processor directives and header files

- Early C++ developers thought of the most common requirements in any programme, creating “libraries” to implement those requirements.
- Developers can use libraries using header inclusion.

```
# include <iostream>

int main(){

  int x = 0;
  std::cout << “x = ” << x;

return 0;
}
```

- `# include` is a pre-processor directive, which is a command that tells the compiler to include a file in pre-processor stage before compilation stage.
- `<iostream>` is the name of the file/library to be included in the pre-processing stage.

Pre-processor directives and header files

- "`#include`" is one of many pre-processor directives, including:

  `#define` : used to define a macro (set of simple statements).

  `#undef` : used to undefine a macro that was defined earlier in the code.

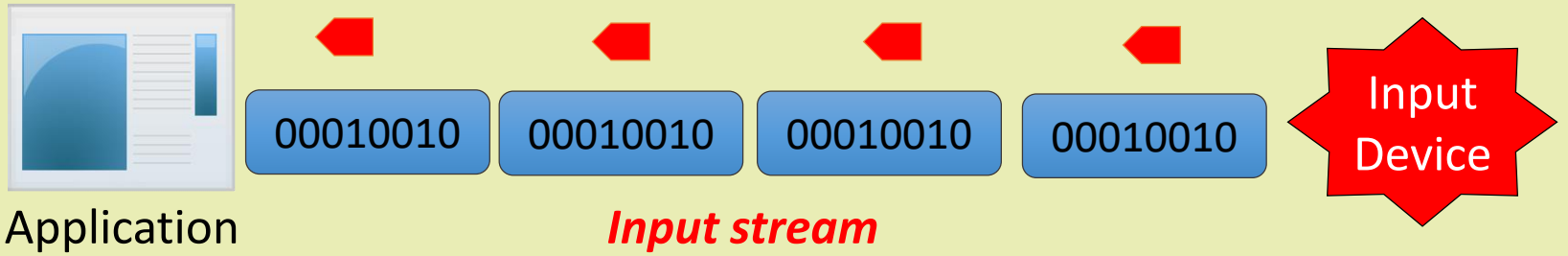
  `#ifdef` : used to define a macro if some conditions are met.

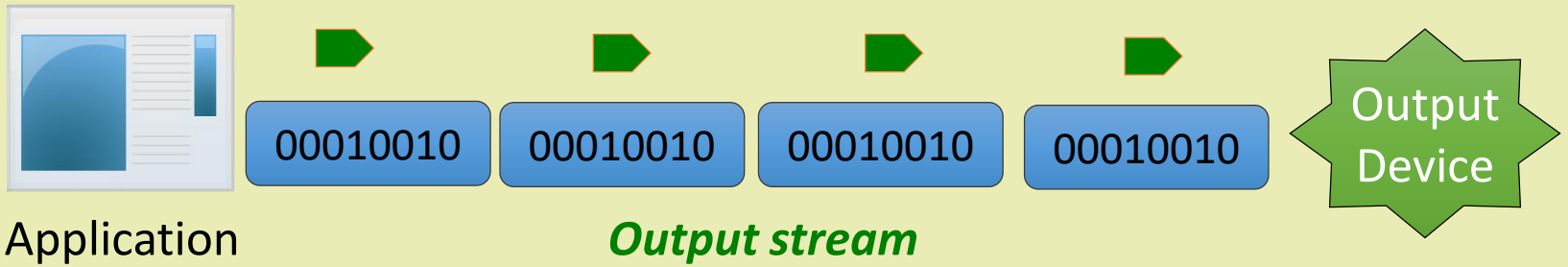
  `#pragma` : used to switch on / off some features in the compiler (highly dependent on compiler used).

- Many of these pre-processor directives are important when dealing with classes and multiple header codes.

Standard Input / Output Streams library

- This is the `<iostream>` library that deals with input / output to the user.
- Full description is found here: http://www.cplusplus.com/reference/iostream/
- Stream: a flow of bytes from the programme to the user or vice versa.





Standard Input / Output Streams library

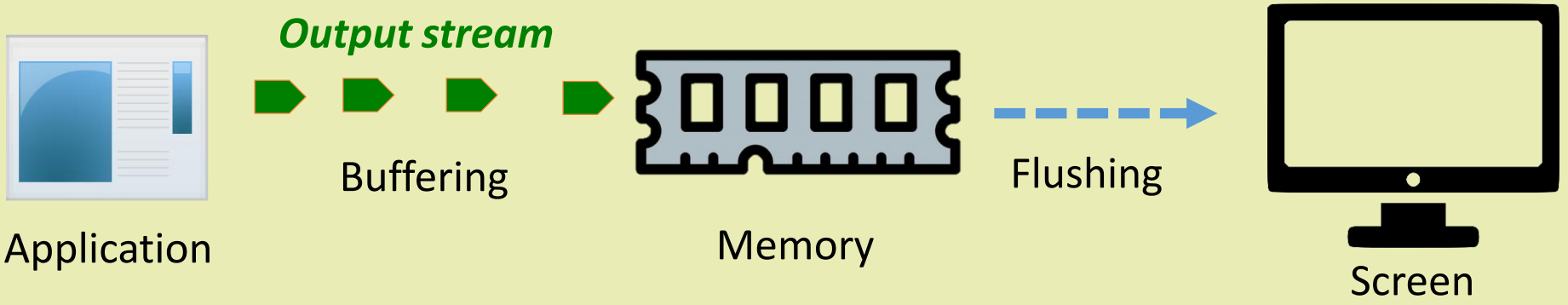
- There are four streams in this standard library (std):

Stream	Description
`cin`	Input stream
`cout`	Output stream
`cerr`	Error stream
`clog`	Log stream

- Characters are passed to the streams using the “<<“ operator for output streams and the “>>” operator for input streams.
- Go to L2D1.cpp

Buffering of Output Streams

- To handle streams faster in code execution, streams are buffered.



- **`std::endl`** flushes the stream every time, while **`\n`** only does it when necessary.
- Output streams **`cout`** and **`clog`** are buffered while **`cerr`** is not.

Practical notes:

For speed use "**`\n`**" instead of **`std::endl`** to start a new line in an output stream.

Other standard libraries

- A list of standard libraries available in C++ can be found in: http://www.cplusplus.com/reference/
- Another library is `<string>` which handles sequences (or 1D arrays) of characters.
- Go to L2D2.cpp
- Example functions in <string> library (full list can be found here: http://www.cplusplus.com/reference/string/string/)

Function	Description
`length()`	Returns the length of the string calling the function
`at(i)`	Returns the character at the position “i” of the string calling the function
`find(“a”)`	Returns the position of the first “a” in the string (there are many options)

Files Input / Output streams

- The `<iostream>` library deals with direct input / output to the user.
- If we want the input of the programme to be a file or its output to be a file, the library `<fstream>` should be used (http://www.cplusplus.com/reference/fstream/fstream/).
- There are two main streams in `<fstream>` library which are:

Stream	Description	Used for
`ifstream`	Input File stream	Loading a file into the programme
`ofstream`	Output File stream	Writing a file by the programme

Files Input / Output streams

- Before using the streams of `<fstream>` , the file to be loaded / written should be opened at first!! And must be closed at the end of the stream.

- A list of important commands in `<fstream>`:

Command	Use	Example
`ifstream`	Opens a file for reading	`ifstream myfile="file.txt"`
`ofstream`	Opens a file for writing	`ofstream myfile="file.txt"`
`is_open()`	Checks if the file is open	`myfile.is_open()`
`close()`	Closes the file	`myfile.close()`
`getline(file,n)`	Writes a line into a string "n"	`getline(file,line)`

- Go to L2D3.cpp

chrono library

- The library <chrono> is used to read and manipulate time.
- It is primarily used for measuring execution time of a code.

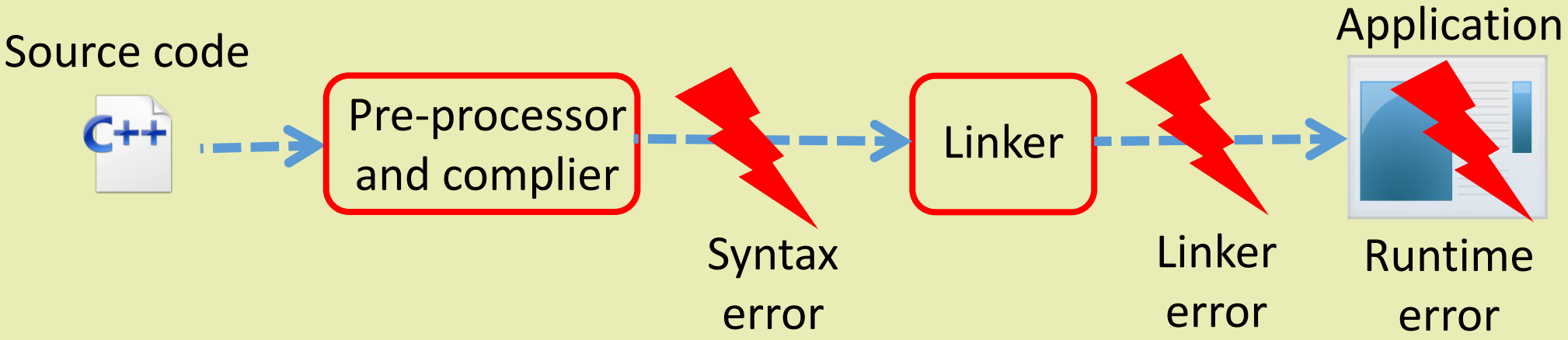
```
# include <chrono>

int main(){

 auto start = chrono::steady_clock::now();  // Starts counting

 // the rest of the code here

 auto end = chrono::steady_clock::now();  // Finishes counting

 std::chrono::duration<double> elapsed_time = end-start; //Difference

 std::cout << elapsed_time.count()<<endl;
 return 0; }
```

Error types

- Errors are illegal operations performed by the user which results in abnormal working of the programme.
- Based on the stage at which they occur, errors can be classified into 3 main categories:

1. Syntax errors (Complier errors): occur when the syntax rules of C++ are violated.
2. Linker errors: occur when the linker cannot find one or more files referred to in the source code, or a set of source codes cannot be combined for other reasons.
3. Run-time errors: occur during program execution (run-time) after successful compilation are called run-time errors.

Error types



- Syntax and Linker errors are detected by the IDE, and thus are encountered by the developer.

- Run-time errors on the other hand are not detected by the IDE, and maybe encountered by the user as well as the developer.

Summary

- C++ components were discussed, which consist of keywords, syntax rules, and developers statements.
- The role of pre-processor directives and header files in a programme was discussed.
- The libraries `<iostream>, <string>, <fstream>, and <chrono>` were discussed.
- The types of errors were discussed and examples of such errors were given.