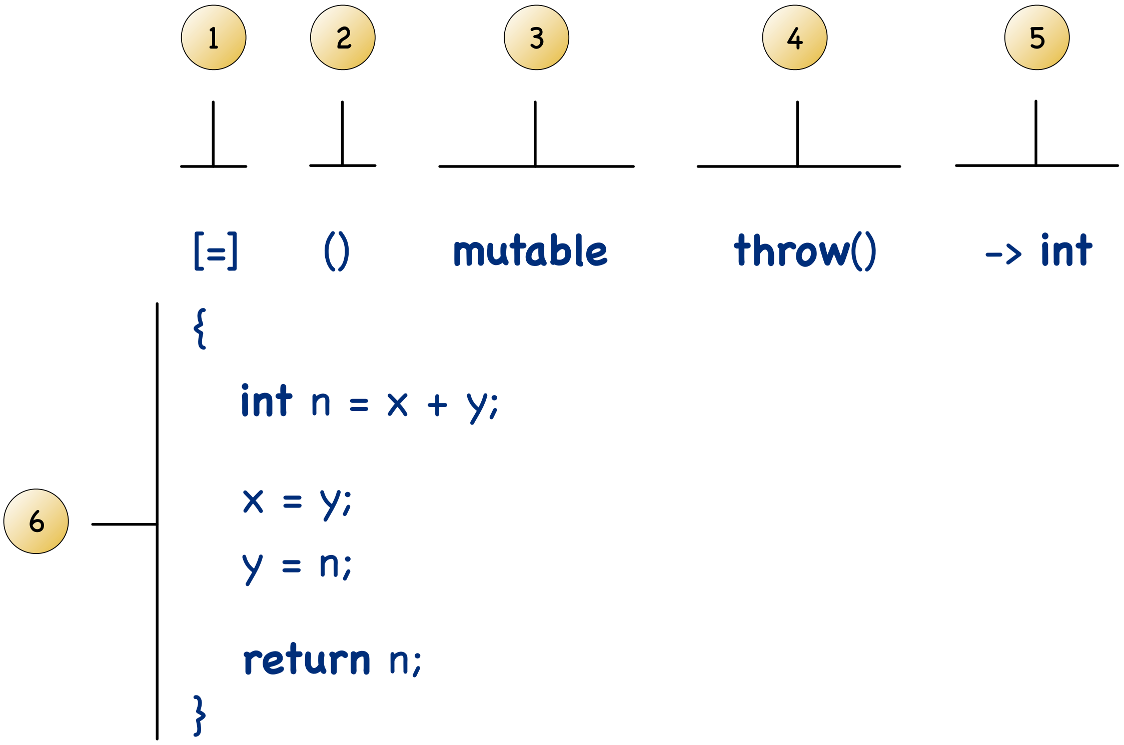
Swinburne University of Technology

Faculty of Information and Communication Technologies

LABORATORY COVER SHEET

Subject Code:	COS30008
Subject Title:	Data Structures and Patterns
Lab number and title:	5, Indexers & Lambdas
Lecturer:	Dr. Markus Lumpe



```
[=]  ()  mutable  throw()  -> int
{
   int n = x + y;

   x = y;
   y = n;

   return n;
}
```

Figure 1: Lambda Expression in C++-11.

Preliminaries

This tutorial is concerned with the definition of indexers and lambda expressions. For this purpose, we define a small auxiliary data type which allows us to read data from an input file. Once we have loaded the data into memory, we can use an indexer to provide random access to the data, or use a lambda, to systematically traverse/access the data.

The input data has been randomized. The actual information entailed in the data is hidden. When we use the indexer, we only obtain the raw information at a given index. This is how the task is set up. When using a lambda, we will employ an additional process that performs *sorting on-the-fly*. The process that we target, in combination with the lambda, is similar to *Bubble Sort*, a classic educational sorting algorithm with quadratic running time complexity (i.e., O(n^2) which means that we check every element in the input against all other elements in two for-loops). The actual sorting is split in two parts: an outer loop that runs through all possible indices, and an inner loop (implemented in a lambda expression) that performs linear search to map the out index to a corresponding datum.

Format of Input File

The input data is a sequence of decimal numbers stored in a text file. The first number represents the number of value pairs following. Every value pair consists of an index and a datum separated by a whitespace character. Only one value pair occurs per line. The name of the input file is `Data.txt`:

```
1050
738 46
667 96
545 32
549 10
793 32
…
663 32
565 46
630 32
```

The file `Data.txt` contains `1050` value pairs, each on a separate line (every line ends with a newline character). The indices in the first column range between 0 and 1049. The values in the second column range between 0 and 255. We use an array of type `DataMap` to store the value pairs:

```
struct DataMap
{
  size_t fIndex;
  size_t fDatum;

  const char getAsChar() const;
};
```

`DataMap` is a public class, that is, all its members have implicit public access. C++ treats structures (aka records) like classes. Hence, we can also define member functions for **`struct`** classes.

The array has to be dynamically allocated at runtime using a **`new`** expression. This means, we also have to explicitly free the memory at the end using a **`delete`** expression.

The solution requires two classes: `DataMap` and `DataMapper`. In addition, we will have to define to lambdas: one that implements plain random access semantics and one that orders access, that is, which maps a given index to the corresponding datum in the data map.

Problem 1

We start with the auxiliary class DataMap and payload class DataWrapper:

```
#pragma once

#include <string>
#include <functional>

struct DataMap
{
  size_t fIndex;
  size_t fDatum;

  const char getAsChar() const;
};

using Callable = std::function<const char(size_t)>;

class DataWrapper
{
private:
  size_t fSize;
  DataMap* fData;

public:

  DataWrapper();
  ~DataWrapper();

  bool load( const std::string& aFileName );

  size_t size() const;

  const DataMap& operator[]( size_t aIndex ) const;

  const char get( size_t aIndex, Callable aSelector );
};
```

The class DataWrapper depends on class DataMap. Hence, we need to specify class DataMap before class DataWrapper.

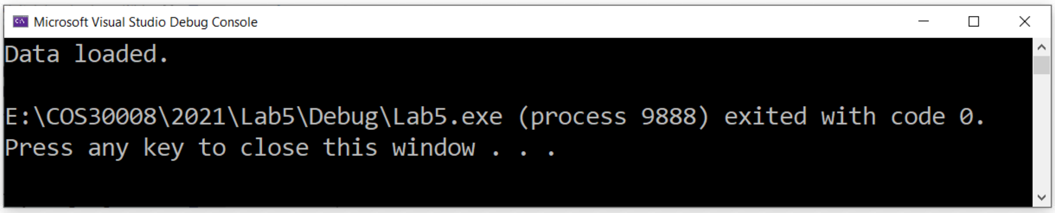
The class DataMap defines a getter that has to return the value of fDatum as constant character. C++11 offer a corresponding cast template for this purpose: static_cast. This cast template performs the required type conversion, but cannot be used to remove "constness" of a type. We write static_cast<**const char**>(fDatum) to convert fDatum from size_t to **const char**.

Class DataWrapper defines two member variables to represent a dynamic array of type DataMap. The constructor has to initialize both values to 0 and nullptr, respectively. The destructor has to release the memory using a **delete** expression.

The load function takes a file name string reference and returns true, if load succeeds. Within function load, we need to create a file input stream, read the size, and fetch all value pairs. The array elements are records (C++ struct). Record fields can be accessed in the usual way.

Finally, the size function returns the size of the data array.

Enable `#define P1` in `Main.cpp` to compile and run your code. It should work. You are not using the indexer or the lambdas yet. If a feature is specified but not used anywhere, then the C++ compiler ignores it.



```
Data loaded.

E:\COS30008\2021\Lab5\Debug\Lab5.exe (process 9888) exited with code 0.
Press any key to close this window . . .
```

Problem 2

Enable `#define P1` and `#define P2` in `Main.cpp`.

Implement the indexer **`operator[]`**`( size_t aIndex )`. The indexer does not change the underlying object. It returns a reference to the `DataMap` entry at `aIndex`. You need to perform a range check. If the index is out of bounds, you cannot return a value. Instead, you need to throw a `out_of_range` exception which is defined in `stdexcept`.

At this point the `getAsChar()` for class `DataMap` must have been implemented for the program to compile.

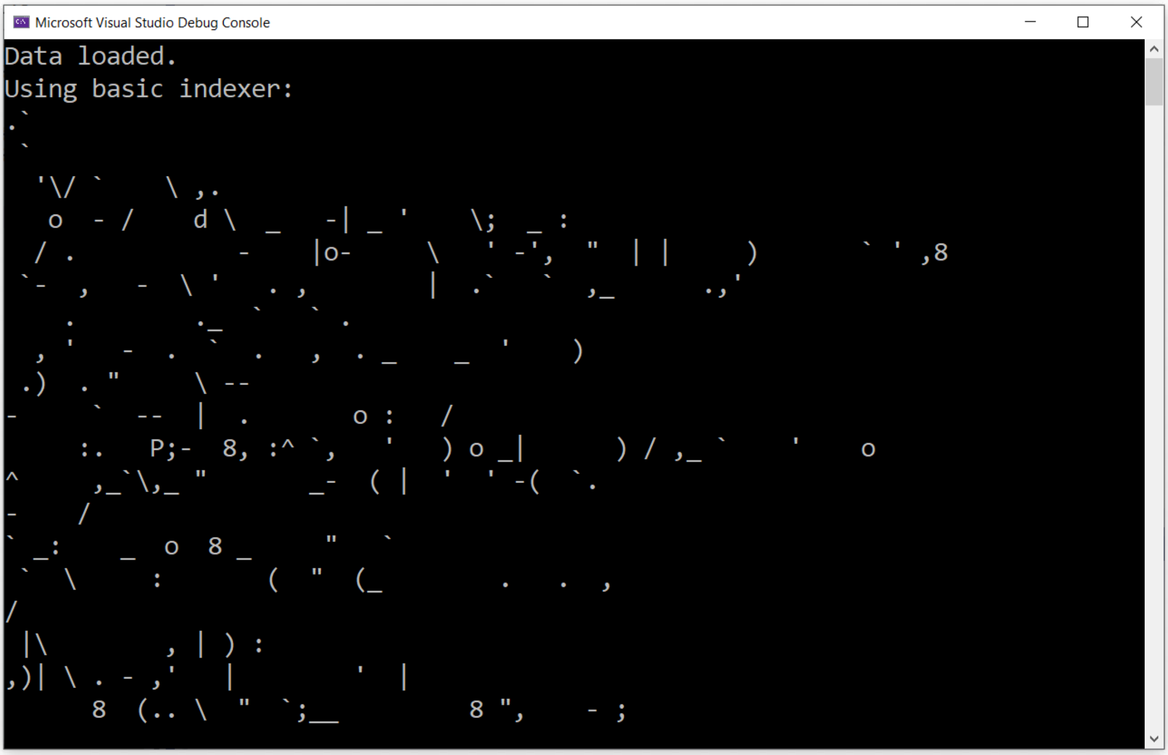
The test code in main.cpp

```
cout << "Using basic indexer: " << endl;

for ( size_t i = 0; i < lData.size(); i++ )
{
    cout << lData[i].getAsChar();
}

cout << endl;
```

should produce the following output



```
Data loaded.
Using basic indexer:
.`
 `
  '\/ `    \ ,.
   o  - /    d \  _   -| _ '    \;  _ :
  / .           -    |o-     \   ' -',  "  | |     )       ` ' ,8
 `-  ,   -  \ '   . ,        |  .`   `  ,_      .,'
    .        ._  `   ` .
  , '   -  .  `  .   ,  . _    _  '    )
 .)  . "     \ --
-     `  --  |  .       o :   /
     :.   P;-  8, :^ `,   '   ) o _|      ) / ,_ `    '    o
^     ,_`\,_ "       _-  ( |  '  ' -(  `.
-    /
` _:    _  o  8 _     "   `
 `  \     :       (  "  (_        .   .  ,
/
 |\        , | ) :
,)| \ . - ,'   |        '  |
      8  (.. \  "  `;__         8 ",    - ;
```

The indexer test code prints some characters, but it is not legible. The program works. Our data is just not properly sorted.

Problem 3

Enable `#define P1`, `#define P2`, and `#define P3` in `Main.cpp`.

Define the lambda expression for `lIdentityMapper`:

```
auto lIdentityMapper = [&lData] (size_t aIndex) throw(out_of_range) -> const char
{
        // Implementation
};
```

This lambda expression has to implement the same functionality as the indexer. That is, it provides random access to a `DataMap` item. In contrast to the indexer, however, this lambda returns the payload value `fDatum` as constant character.

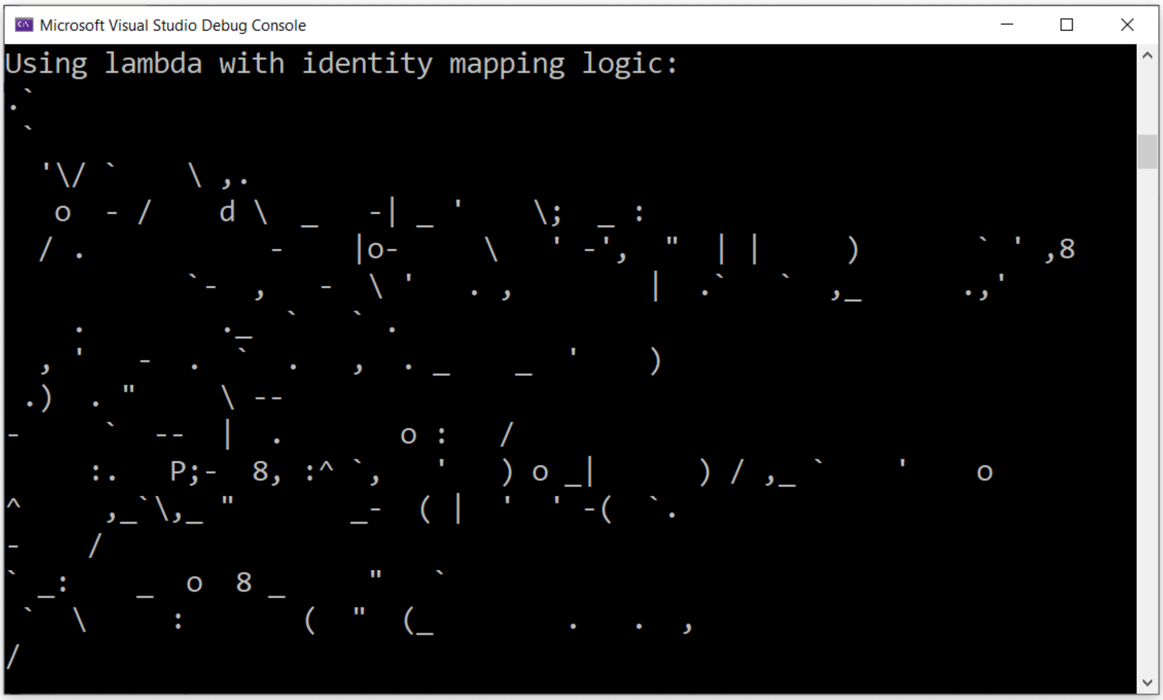
The lambda captures the local variable `lData` by reference. This allows the lambda to access it. In addition, the lambda can throw an `out_of_range` exception. Hence, we also use the **`throw`** clause for this lambda.

The for-loop

```
    for ( size_t i = 0; i < lData.size(); i++ )
    {
        cout << lData.get( i, lIdentityMapper );
    }

    cout << endl;
```

should produce output that looks the same as the one for the indexer test. We still use random access. No ordering occurs.



```
Using lambda with identity mapping logic:
.`
 `
  '\/ `    \ ,.
   o  - /    d \  _   -| _ '    \;  _ :
  / .           -    |o-     \   ' -',  "  | |     )       ` ' ,8
           `-  ,   -  \ '   . ,        |  .`   `  ,_      .,'
    :         ._ `   `  .
  , '   _  .  `  .   ,  . _    _  '    )
 .)  . "     \ --
-     `  --  |  .       o :   /
     :.   P;-  8, :^ `,   '   ) o _|      ) / ,_ `    '    o
^     ,_`\,_ "       _-  ( |  '  ' -(  `.
-    /
` _:    _  o  8 _     "   `
 `  \     :       (  "  (_        .   .  ,
/
```

The lambda test code for `lIdentityMapper` prints some characters, but it is not legible. The program works. Our data is just not properly sorted.

Problem 4

Enable `#define P1`, `#define P2`, `#define P3`, and `#define P4` in `Main.cpp`.

Define the lambda expression for `lOrderedMapper`:

```
auto lOrderedMapper = [&lData] (size_t aIndex) throw(out_of_range) -> const char
{
       // Implementation
};
```

This lambda expression has to implement a linear search that maps `aIndex` to the corresponding payload datum in the container array. This linear search in combination with a for-each loop over the iterator achieves sorting on-the-fly in a manner similar to *Bubble Sort*. For example, if `aIndex` is `738`, then the payload index is `0` (see Format of Input File). The lambda `lOrderedMapper` would have to return the constant character that corresponds to the unsigned value `46`.

The lambda captures the local variable `lData` by reference. This allows the lambda to access it. In addition, the lambda can throw an `out_of_range` exception. Hence, we also use the **`throw`** clause for this lambda.

The for-loop

```
for ( size_t i = 0; i < lData.size(); i++ )
{
    cout << lData.get( i, lOrderedMapper );
}

cout << endl;
```

should produce legible output when the lambda `lOrderedMapper` is correctly implemented. The output is the "Easter Egg" here.

You may need to **develop a plan**, that is, analyze the problem in depth, identify the unknowns, check the C++ reference and DSP lecture notes for suitable solution scenarios. You must not write a single line of code prior finishing the problem analysis.

Sketch out a plan/solution on **paper**. There might be hidden issues.

Once we understand all the requirements and possible issues of the project, we can start building the solution.

This is a rather complex tutorial. The solutions require approx. 170 lines of low density C++ code. Use this tutorial to practice the idioms of C++ and the proper coding of them.

Completing this tutorial is crucial for succeeding in the upcoming assignments and tests.

Main.cpp

```
#ifdef _MSC_VER
// VS 2019 does not implement exception specification
#pragma warning( disable : 4290 )
#endif

#include <iostream>

#define P1
#define P2
#define P3
#define P4

#include "DataWrapper.h"

using namespace std;

int main( int argc, char* argv[] )
{
  if ( argc != 2 )
  {
    cerr << "Arguments missing." << endl;
    cerr << "Usage: DataWrapper <filename>" << endl;

    return 1;
  }

#ifdef P1

  DataWrapper lData;

  if ( !lData.load( argv[1] ) )
  {
    cerr << "Cannot load data file " << argv[1] << endl;

    return 2;
  }

  cout << "Data loaded." << endl;

#endif

#ifdef P2

  cout << "Using basic indexer: " << endl;

  for ( size_t i = 0; i < lData.size(); i++ )
  {
    cout << lData[i].getAsChar();
  }

  cout << endl;

#endif

#ifdef P3

  cout << "Using lambda with identity mapping logic: " << endl;

  auto lIdentityMapper = [&lData] (size_t aIndex) throw(out_of_range) -> const char
  {
       // Implementation
  };

  for ( size_t i = 0; i < lData.size(); i++ )
  {
    cout << lData.get( i, lIdentityMapper );
  }

  cout << endl;

#endif
#ifdef P4

  cout << "Using lambda with ordered mapping logic: " << endl;

  auto lOrderedMapper = [&lData] (size_t aIndex) throw(out_of_range) -> const char
  {
       // Implementation
  };

  for ( size_t i = 0; i < lData.size(); i++ )
  {
    cout << lData.get( i, lOrderedMapper );
  }

  cout << endl;

#endif

  return 0;
}
```