Swinburne University Of Technology

Faculty of Science, Engineering and Technology

ASSIGNMENT COVER SHEET

Subject Code:	COS30008
Subject Title:	Data Structures and Patterns
Assignment number and title:	3, Simple Text Processing in C++
Due date:	April 8, 2014, 10:30 am
Lecturer:	Dr. Markus Lumpe

Your name:_______________________________ **Your student id:**_________________

Check Tutorial	Fri 10:30	Fri 12:30	Fri 14:30

Marker's comments:

Problem	Marks	Obtained
1	8+1+44 = 53	
Total	53	

Extension certification:

This assignment has been given an extension and is now due on ____________________

Signature of Convener:____________________________

Problem Set 3: Simple Text Processing in C++

Build the C++ console application, called `HexDump`, that takes one argument (i.e., a file name), reads the corresponding file in binary mode, and output a hex dump of the input file to the console screen. For example, when started in a terminal, then the invocation

```
C:\DSP2014\PS3> HexDump ShiftValue.class
```

reads the file `ShiftValue.class` and produces a console screen hex dump, which should be structured as follows:

```
00000000:  CA FE BA BE 00 00 00 31 | 00 2B 07 00 02 01 00 12  .......1.+......
00000010:  4C 61 6D 62 64 61 46 2F | 53 68 69 66 74 56 61 6C  LambdaF/ShiftVal
00000020:  75 65 07 00 04 01 00 13 | 4C 61 6D 62 64 61 46 2F  ue......LambdaF/
00000030:  42 69 6E 61 72 79 56 61 | 6C 75 65 01 00 06 3C 69  BinaryValue...<i

                                     …

000003C0:  01 00 00 00 39 00 0B 00 | 00 00 20 00 03 00 00 00  ....9..... .....
000003D0:  0E 00 0C 00 0D 00 00 00 | 00 00 0E 00 0E 00 0F 00  ................
000003E0:  01 00 00 00 0E 00 10 00 | 0F 00 02 00 01 00 29 00  ..............).
000003F0:  00 00 02 00 2A                                     ....*

C:\DSP2014\PS3>
```

Problem Specification

In order to organize the solution, the program `HexDump` splits the input in units of length 16 bytes (you will need to use a local array variable of `unsigned char` for this purpose and cast it to `(char *)` while reading from the input file). The output format for a unit consists of three parts:

1. Each line of the hex dump begins with the corresponding absolute start address (printed in hexadecimal format) of the unit analyzed.
2. The middle block uses two groups (separated by the character '|') of 8 two-digit hexadecimal numbers, separated by a space character, representing the value at this absolute position in the input file.
3. At the end of each line we print the graphical representation of the unit, that is, a sequence of printable ASCII characters representation of the unit. Byte values that do not possess a graphical character representation are substituted by the character '.' in the output.

In order to produce the correct output, you will need to use the following I/O manipulators defined in `iomanip` of the C++ I/O library:

- `endl` – generate platform specific end-of-line,
- `hex` – force integers to be printed in hexadecimal format,
- `setw(n)` – sets the precision of a printed integer,
- `setfill(c)` – sets the fill character for `0`s left of an integer, and
- `uppercase` – print all character in upper case format.

In addition, you may need to study the manual pages (or any appropriate C++ documentation) to learn more about the libraries `ifstream` and `string`. The solution requires a proper use of these libraries as well as a good understanding of the stream methods `open()`, `fail()`, `close()`, `read()`, and `gcount()`.

The application `HexDump` uses an input file stream object to read data and the console for output. For this reason, you need to include `iostream`, `fstream`, and `iomanip` where appropriate into the current compilation unit. These header files provide the required abstractions to work with files. When working with files remember to define appropriate guards to check for any occurrences of I/O errors, as this will prevent the application `HexDump` from working correctly.

The application should consist of two parts: a class `HexDump` that implements the desired functionality and a main function that performs the necessary argument checks, instantiates an object of class `HexDump`, and calls the appropriate methods in the right order. The specification of class `HexDump` is shown below:

```
#pragma once

#include <fstream>
#include <string>

class HexDump
{
private:
  std::ifstream fInput;

public:
  bool open( const std::string& aInputFileName );
  void close();
  void processInput();
};
```

There may also be the need to use type casts in order to adjust the actual type of an expression to the required type. A typecast is written `(typename)expression`. For example, if you wish to output a character (i.e., a variable of type `unsigned char`), say `current_char`, to the console screen, then you would normally write `cout << current_char`. Unfortunately, the selected `<<` operator in this case will render the value of `current_char` to its printable representation, which may not be desired. To print the raw value (i.e., an integer representation), you must apply a cast to `int` first. More precisely, when you write `cout << (int)current_char`, then the output is the integer representation (in hexadecimal if you have used the correct manipulator). Please note that the C++ compiler performs an "unsigned extension" when converting `unsigned char` to `int`. To illustrate the different behavior, consider the following code fragment:

```
unsigned char current_character = 'A';

cout << current_character << endl;

cout << (int) current_character << endl;

cout << hex << (int) current_character << endl;
```

The two output statements produce the following result:

```
A

65

41
```

In the first case we obtain the graphical representation of `current_character`; in the second case the integer representation is generated; and finally, we produce the hexadecimal representation of character `'A'`. The integer representation uses the default manipulator `dec`. To render it to hexadecimal, you need to use the manipulator `hex`.

If the value of `current_character` exceeds 127, then the line

```
cout << hex << (int) current_character << endl;
```

produces a two-character hex digit. For example `current_character = 200` yields `C8` in the output. In contrast, if we were to declare `current_character` to have type `char` (signed), then assigning 200 to `current_character` would result in a negative number and sending it to the console screen yields `FFFFFFC8`.

As main function use

```
#include "HexDump.h"
#include <iostream>

using namespace std;

int main( int argc, char* argv[] )
{
  if ( argc < 2 )
  {
    cerr << "Error: Argument missing!" << endl;
    cerr << "Usage: HexDump filename" << endl;
    return 1;
  }

  HexDump lProcessor;

  if ( lProcessor.open( argv[1] ) )
  {
    lProcessor.processInput();
    lProcessor.close();
    return 0;
  }

  return 2;
}
```

The final program will require approx. 80 lines of code including comments and very spacious formatting.

In order to achieve full marks, you must correctly implement the functionality including a proper handling of the last line. The separator '|' must occur if there are at least 8 characters and the printable representation must be properly aligned. Use the provided test files (on Blackboard) to check all boundary conditions.

Submission deadline: Tuesday, April 8, 2014, 10:30 a.m.

Submission procedure: on paper, code of class `HexDump`.