

Assignment #5 [8 Marks]

Submission Date / Time	5 June 2022 23:59
Course	[M1522.000600] Computer Programming
Instructor	Jae W. Lee

- You are allowed to discuss with fellow students to understand the questions better, but your code must be **SOLELY YOURS**. You **MUST NOT** show your code to anyone else, or vice versa.
- We will use the automated copy detector to check the possible plagiarism of the code between the students. The copy checker is reliable so that it is highly likely to mark a pair of code as the copy even though two students quickly discuss the idea without looking at each other's code. Of course, we will evaluate the similarity of a pair compared to the overall similarity for the entire class.
- We will do the manual inspection of the code. In case we doubt that the code may be written by someone else (outside of the class like github), we reserve the right to request an explanation about the code. We will ask detailed Problems that cannot be answered if the code is not written by yourself.
- If any of the above cases happens, you will **get 0 marks** for the assignment and may get **a further penalty**.
- Download and unzip "HW5.zip" file from the NeweTL. "HW5.zip" file contains skeleton codes for Question 1 and 2 (in "problem1" and "problem2" directory, respectively).
- Do not modify the overall directory structure after unzipping the file, and fill in the code in appropriate files. It is okay to add new directories or files if needed.
- When you submit, compress the "HW5" directory which contains "problem1" and "problem2" directories in a single zip file named "{studentID}.zip" (e.g. 2022-12345.zip) and upload it to NeweTL. Contact the TA if you are not sure how to submit. Double-check if your final zip file is properly submitted. You will **get 0 marks** for the wrong submission format.
- CLion sometimes successfully completes the compilation even without including some header files. Nevertheless, you **MUST include all the necessary header files in the code** to make it successfully compile in our server. Otherwise, you may not be able to get the points.
- C++ Standard Library (including Standard Template Library) is allowed.
- Do not use any external libraries. If you are not sure, contact TAs to clarify.

- Regarding variable names and function names in skeleton code, Google C++ coding conventions are used (<https://google.github.io/styleguide/cppguide.html>). Whether you follow these coding conventions or not will not affect your score.

Contents

Question 1. C++ Basics Exercise [4 Marks]

- 1-1. Palindrome String [0.5]
- 1-2. Hamming Distance [0.5]
- 1-3. Merge Arrays [0.5]
- 1-4. Pascal's Triangle [0.5]
- 1-5. Connected Component Labeling [2]

Question 2. Data Compression [4 Marks]

- 2-1. Printing Run-Length Encoded Data [0.5]
- 2-2. Run-Length Encoding [1]
- 2-3. Printing Delta Encoded Data [0.5]
- 2-4. Delta Encoding [1]
- 2-5. Estimating Compression Performance [1]

Submission Guidelines

- You should submit your code on the NeweTL.
- After you extract the zip file, you must have an "HW5" directory. The submission directory structure should be as shown in the table below.
- You can create additional files in each "src" directory. Make sure to update CMakeLists.txt to reflect your code structure changes.
- You can add additional methods or classes, but do not remove or change signatures of existing methods.
- Compress the "HW5" directory and name the file "20XX-XXXXX.zip" (your student ID).

Submission Directory Structure (Directories or Files can be added)

- Inside the "HW5" directory, there should be "problem1" and "problem2" directories.

Directory Structure of Question 1	Directory Structure of Question 2
<pre> problem1/ ├── CMakeLists.txt ├── src │ ├── main.cpp │ ├── p1-1.cpp │ ├── p1-2.cpp │ ├── p1-3.cpp │ ├── p1-4.cpp │ └── p1-5.cpp </pre>	<pre> problem2/ ├── CMakeLists.txt ├── src │ ├── main.cpp │ ├── RunLength.cpp │ ├── RunLength.h │ ├── Delta.cpp │ ├── Delta.h │ └── TestHelper.cpp </pre>

└─ header.h └─ stb_image.h (for test) └─ stb_image_write.h (for test)	└─ TestHelper.h
---	-----------------

Question 1: C++ Basics Exercise [4 Marks]

For the sub-questions of Question 1, there will be NO partial points.

Question 1-1: Palindrome String [0.5 Marks]

Objective: Write a function that checks whether the given string is a palindrome or not.

Description:

- Palindrome string is a string that is the same when written forwards or backwards.
- Note that it should be case sensitive and spaces are also treated as characters.

Target Function: `bool IsPalindrome(string s)` (in 1-1.cpp)

Parameter: string s (0 <= length <= 200)

Return value: True if s is a palindrome, false otherwise.

Input (The actual input is the contents inside "")	Output
s = ""	true
s = "ab"	false
s = "aaabbb"	false
s = "abba"	true
s = "abbA"	false
s = "12321a12321"	true
s = "12321a 12321"	false (due to the presence of space)

Question 1-2: Hamming Distance [0.5 Marks]

Objective: Write a function that calculates the hamming distance between two numbers.

Description:

- The Hamming distance between two integers is the number of positions at which the corresponding bits are different.
- For example, given 1 and 4, the binary format of 1 is “001” and the binary format of 4 is “100”. Then you can see that the leftmost bit and the rightmost bit are different. Therefore, the hamming distance is 2.

1 (001)
4 (100)
 ↑ ↑

Figure 1. Hamming Distance Example

Target Function: `int HammingDistance(int x, int y)` (in 1-2.cpp)

Parameter: positive integer `x`, `y` ($0 \leq x, y \leq 2^{31}-1$)

Return value: The hamming distance between `x` and `y`.

Input	Output
<code>x = 1, y = 4</code>	2
<code>x = 3, y = 1</code>	1
<code>x = 33, y = 15</code>	4

Question 1-3: Merge Arrays [0.5 Marks]

Objective: Write a function that merges two arrays that are sorted into a single sorted array.

Description:

- Two arrays, `arr1` and `arr2`, that are sorted in decreasing order are given. You should merge the two arrays into a single array in a decreasing order.
- The final sorted array should NOT be returned by the function, but instead be stored inside the first array, `arr1`. To accommodate this, `arr1` has a length of `len1 + len2`, where the first `len1` elements denote the sorted elements that should be merged, and the last `len2` elements are set to 0 and should be ignored. `arr2` has a length of `len2`.

Target Function: `void MergeArrays(int* arr1, int len1, int* arr2, int len2)` (in 1-3.cpp)

Parameter:

- `arr1` and `arr2` are the two arrays that are sorted in decreasing order.
- `len1 + len2` and `len2` are the lengths of `arr1` and `arr2`, respectively.
- `1 <= len1, len2 <= 200`
-

Return value: There is no return value. You should modify the `arr1`.

Input	Output
<code>arr1={3,2,1,0,0,0}</code> , <code>len1=3</code> , <code>arr2={6,5,4}</code> , <code>len2=3</code>	<code>arr1</code> modified to <code>{6,5,4,3,2,1}</code>
<code>arr1={5,3,1,0,0}</code> , <code>len1=3</code> , <code>arr2={5,3}</code> , <code>len2=2</code>	<code>arr1</code> modified to <code>{5,5,3,3,1}</code>
<code>arr1={16,15,10,-3,0,0,0}</code> , <code>len1=4</code> , <code>arr2={50,-5,-99}</code> , <code>len2=3</code>	<code>arr1</code> modified to <code>{50,16,15,10,-3,-5,-99}</code>

Question 1-4: Pascal's Triangle [0.5 Marks]

Objective: Write a function that returns the N-th row of the Pascal's triangle

Description:

- Pascal's triangle is a triangular array of the binomial coefficients that arises in probability theory, combinatorics, and algebra. It is named after the French mathematician Blaise Pascal.
- For each row, the leftmost number and the rightmost number are always 1.
- Other numbers can be calculated by summing the two numbers directly above them as shown in the figure below
- Code that directly assigns constants to array values is not allowed, as in the example below.
 - ex if(N==5) {arr[0]=1; arr[1]=4; arr[2]=6; arr[3]=4; arr[4]=1;}

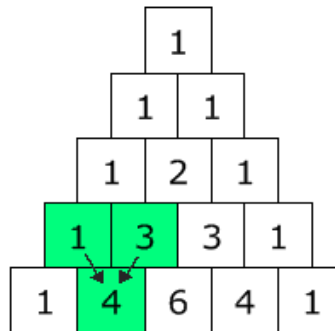


Figure 2. Pascal Triangle

Target Function: `int* PascalTriangle(int N)` (in 1-4.cpp)

Parameter:

- N indicates which row of the triangle to return. ($1 \leq N \leq 20$)

Return value: Array of length N. You should allocate the proper amount of memory space for the array.

Input	Output
N = 1	{1}
N = 3	{1,2,1}
N = 5	{1,4,6,4,1}

Question 1-5: Connected Component Labeling [2 Marks]

Objective: Write a function that labels separate objects in an image with different labels.

Description:

- An image can be treated as a two-dimensional array of height and width.
(e.g. `image[height][width]`)
- The coordinates of the upper-left corner of the image are (0,0), and the width increases toward the right side of the image, and the height increases as it goes down the image.

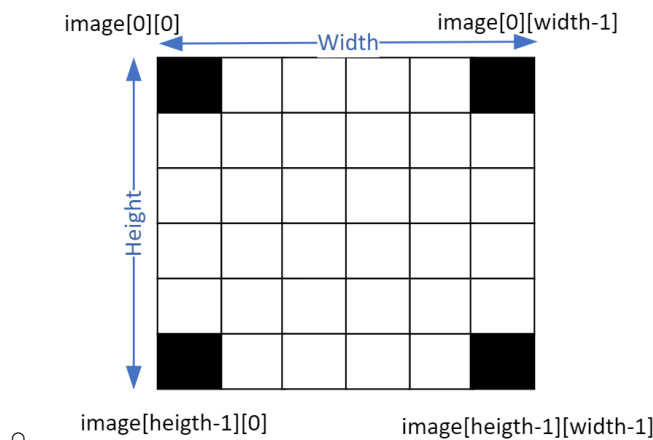


Figure 3. Coordinates of the Image

- The value stored in each array element is a pixel value, ranging from 0 to 255, where 0 represents black and 255 represents white.
- In this question, it is assumed that the input image is a binary image with only pixel values of 0 or 255.
- An object is defined as a pixel with 0 value.
- Based on a pixel with 0 value, if there is a pixel with 0 value among the 8 surrounding pixels, these pixels are regarded as *connected*. These connected pixels constitute the same object, and hence get assigned the same label.

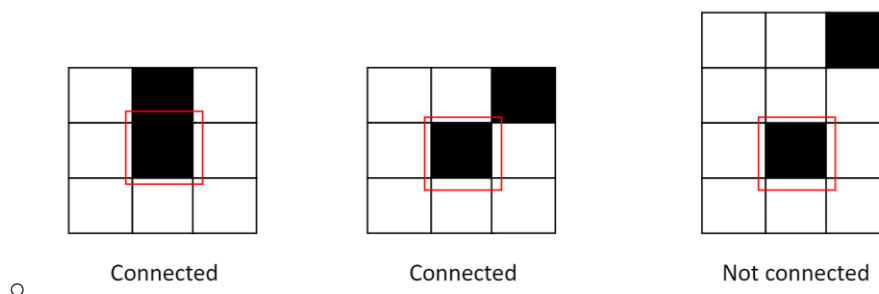


Figure 4. Connection Example

- The order of searching objects is as in the figure below. Starting from the top left of the image, width pixels are sequentially searched from 0. Repeat this by increasing the

height sequentially from 0.

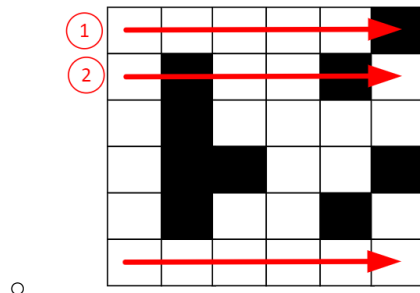


Figure 5. Object Search Order

- Labels are assigned incrementally starting from 0, and labels are assigned from the object that is searched first based on the aforementioned search order. Please see the example in the figure below.

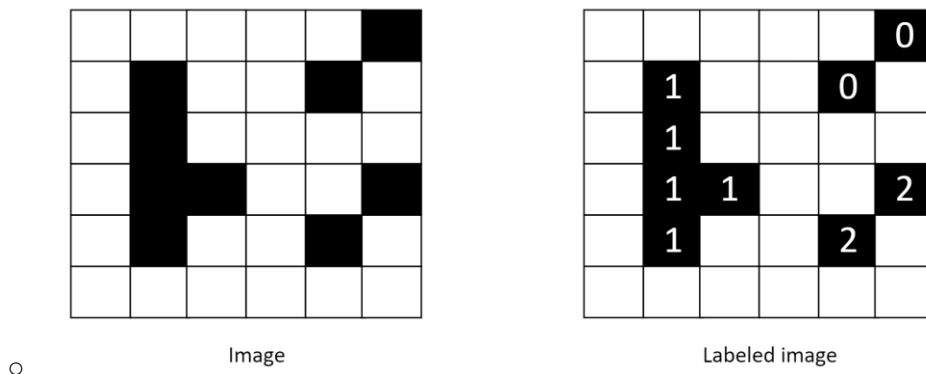


Figure 6. Example of Labeling





- The width and height of the output image are the same as those of the input image. However, for the pixel values of the output image, you should set the pixel values of the labeled object to "label number * 50". Since the maximum number of objects is assumed to be 5, the pixel value will not exceed 255. We will not test for input images with more than 6 objects.
- Implement the following target function based on what has been described so far.

Target Function: `void Labeling (uint8_t* input_image, uint8_t* output_image, int width, int height)` (in 1-5.cpp)

Parameter:

- `input_image` is the start address of the input image array.
- `width` and `height` refer to the width and height of the input image.
- `output_image` is the start address of the output image array. You have to do connected component labeling based on the input image and set the pixel value of the output image based on each label. The details are the same as described in the description. Note that

this function is called after memory allocation for the output image array. uint8_t is an unsigned 8-bit data type with values from 0 to 255.

Input	Output	Explanation
		<p>■ is labeled with label 0. Therefore, pixel values are 0 ($= 0*50$).</p> <p>■ is labeled with label 1. Therefore, pixel values should be 50 ($= 1*50$).</p> <p>■ is labeled with label 2. Therefore, pixel values should be 100 ($= 2*50$).</p> <p>■ is labeled with label 3. Therefore, pixel values should be 150 ($= 3*50$).</p>
		<p>⚡ is labeled with label 0. Therefore, pixel values are 0 ($=0*50$).</p> <p>★ is labeled with label 1. Therefore, pixel values should be 50 ($= 1*50$).</p> <p>♥ is labeled with label 2. Therefore, pixel values should be 100 ($= 2*50$).</p> <p>⊕ is labeled with label 3. Therefore, pixel values should be 150 ($= 3*50$).</p>

Question 2: Data Compression [4 Marks]

Objective: Develop a Compressor to support Run-Length Encoding and Delta Encoding. Then evaluate the compression performance of each algorithm for a given input. Do not worry about the terms like Run-Length Encoding or Delta Encoding for now; we will explain the necessary details shortly.

Description: Data compression is a method used to reduce data size, and there are various compression algorithms depending on the compression method. In this question, we will implement Run-Length Encoding and Delta Encoding, which are representative compression algorithms. Then, the performance of each algorithm is evaluated for a given input. This allows you to check which compression algorithm is effective for a given input pattern.

Note:

- For this question, you only need to modify RunLength.cpp(.h) and Delta.cpp(.h). The test codes are in main.cpp.
- We provide the basic test cases, and we will use a richer set of test cases for evaluation. We strongly encourage you to add more diverse test cases to make sure your application works as expected.

Background on compression algorithms.

- **Run-Length Encoding** (https://en.wikipedia.org/wiki/Run-length_encoding)
Run-Length encoding (RLE) is a form of lossless data compression in which runs of data (sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run. This is most efficient on data that contains many such runs, for example, simple graphic images such as icons, and line drawings. For files that do not have many runs, RLE could increase the file size as shown in the third example below.

Input (Original data)	Output (Encoded Data)
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}	{0,16}
{1,2,2,3,3,3,4,4,4,4}	{1,1,2,2,3,3,4,4}
{1,2,3,4}	{1,1,2,1,3,1,4,1}

- **Delta Encoding** (https://en.wikipedia.org/wiki/Delta_encoding)

Like RLE, Delta Encoding is also a form of lossless data compression. As you might have guessed from its name, Delta Encoding stores the differences (deltas) between consecutive data rather than complete data. However, the first data stores the same values as the original data, not the delta. Compression gain occurs because it is assumed that the bit-width for storing delta is smaller than the bit-width for storing the original data.

Input (Original data)	Output (Encoded Data)
{40001,40002,40003,40004,40005,40000}	{40001,1,1,1,1,-5}
{40001,40001,40001,40001,40001,40001}	{40001,0,0,0,0,0}

Question 2-1: Printing Run Length Unit Data [0.5 Marks]

Objectives: Implement the following function in RunLength.cpp to print the object of the custom struct RunLengthUnit in a designated format.

- `std::ostream& operator<<(std::ostream &os, const RunLengthUnit &rlu)`
 - This method overloads the insertion operator (<<) of this struct so that we can print out a RunLengthUnit variable rlu using `std::cout << rlu`. The output should be: (data length).
 - The purpose of this function is similar to overriding the `toString` method in Java!

Description: We will fill up a convenience struct called RunLengthUnit, which represents a single Run Length Unit. As you are aware, in RLE, data is expressed as a value of data itself and its running length, and Run Length Unit is a basic unit composed of the value and length. The RunLengthUnit struct stores the data and length parts (data, length) as the integer type public member variables data and length. All the other parts of the RunLengthUnit struct are given.

- $-10^6 < \text{data} < 10^6$ (It is assumed that all input values are within this range)
- $1 \leq \text{length} \leq 2^{\text{length_bit_width_}}$
- $1 \leq \text{length_bit_width_} \leq 8$
- If data is x and length is y, it should be represented as (x y).
- ex) If data is 8 and length is 4, it should be represented as (8 4).

Question 2-2: Run Length Encoding [1 Marks]

Objectives: Implement the following function in RunLength.cpp to encode the input data.

- `void RunLength::Encode(const char* file_name)`
 - It reads the input file (with the `file_name`) containing the header and data, then encodes data with the Run Length Encoding algorithm based on a description below.
 - Encoded data is stored as an array of `RunLengthUnit` and then the start address of the array is assigned to the `RunLengthUnit* run_length_unit` which is the member variable of the `RunLength` class.
 - You should allocate the proper amount of memory space to store Run Length encoded data. There must be no empty arrays. The proper amount size depends on the compression ratio.

Description: The format of the input file is shown in the figure below. All the values are integers. As you can see in the figure, a header exists at the beginning of the file. The first value of the header is the number of bits required to store one data. That is, in the example below, one data occupies 32 bits. The second value of the header is the total number of data excluding the header. In the example below, there are a total of ten data, so this value is 10.

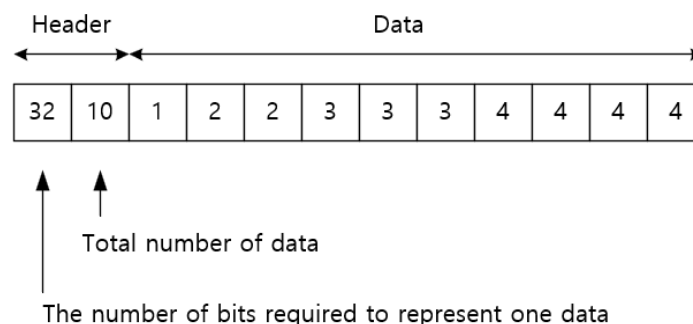


Figure 7. Input data example

The result corresponding to the figure above is as follows.

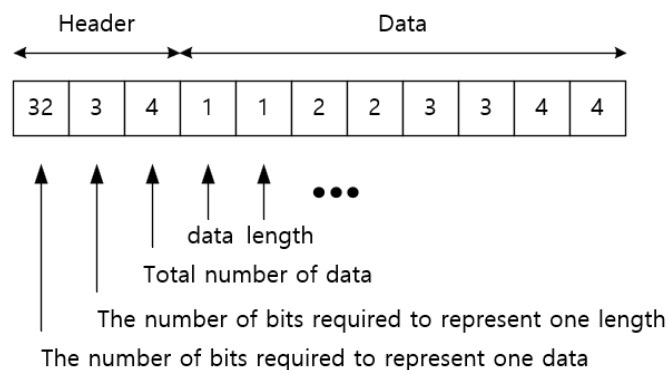


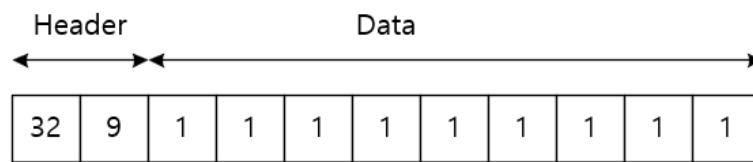
Figure 8. Output Data Example

The first header value should be stored in `data_bit_width`, a member variable of the `RunLength` class, using the first value of the input data header. The second header value should be stored

in `length_bit_width_`, but you do not need to set this value because this value is set in the main function before calling this function. The third header value means the number of `RunLengthUnit` after encoding and you need to set it yourself.

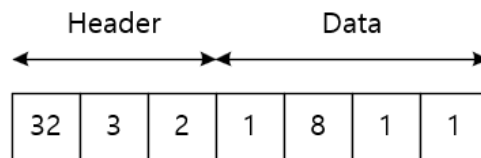
Note that the max value of length is determined by the second header value. If the `length_bit_width_` is set to 3, the max value of length becomes 8 ($= 2^3$), and in this case, if the same data is repeated 9 or more times, it is separated into two data and stored. Note that the relationship between expressible length and `length_bit_width_` is as follows.

- $1 \leq \text{length} \leq 2^{\text{length_bit_width_}}$



↑
↑
Total number of data
The number of bits required to represent one data

Input (Original Data)



↑ ↑ ↑ ↑ ↑ ...
data length
Total number of data
The number of bits required to represent one length
The number of bits required to represent one data

Output (Encoded Data)

Figure 9. An Example in which data is separated by the maximum value of length

Hint: Use `std::stoi(std::string s)` to convert an integer string into an integer.

Question 2-3: Printing Delta Encoded Data [0.5 Marks]

Objectives: Implement the following function in Delta.cpp to print the object of the custom struct DeltaUnit in a designated format.

- `std::ostream& operator<<(std::ostream &os, const DeltaUnit &du)`
 - This method overloads the insertion operator (<<) of this struct so that we can print out a DeltaUnit variable du using `std::cout << du`. The output should be: (dt delta).
 - The purpose of this function is similar to overriding the toString method in Java!

Description: We will fill up a convenience struct called DeltaUnit, which represents a single Delta Unit. As you are aware, in Delta Encoding, data is expressed as delta between consecutive data, and Delta Unit is a basic unit composed of one delta. The DeltaUnit struct stores the delta as the integer type public member variable delta. All the other parts of the DeltaUnit struct are given.

- $-10^6 < \text{delta} < 10^6$ (It is assumed that all input values are within this range)
- If delta is x, it should be represented as (dt x)
- ex1) If delta is 4, it should be represented as (dt 4)
- ex1) If delta is -10, it should be represented as (dt -10)

Question 2-4: Delta Encoding [1 Marks]

Objectives: Implement the following function in Delta.cpp to encode the input data.

- `void Delta::Encode(const char* file_name)`
 - It reads the input file (with the `file_name`) containing the header and data, then encodes data with the Delta Encoding algorithm based on a description below.
 - Encoded data is stored as an array of `DeltaUnit` and then the start address of the array is assigned to the `DeltaUnit* delta_unit_` which is the member variable of the Delta class.
 - You should allocate the proper amount of memory space to store Delta encoded data. There must be no empty arrays. The proper amount size depends on the compression efficiency.

Description: The format of the input file is shown in the figure below. All the values are integers. As you can see in the figure, a header exists at the beginning of the file. The first value of the header is the number of bits required to store one data. The second value of the header is the total number of data excluding the header. In the example below, there are a total of five data, so this value is 5.

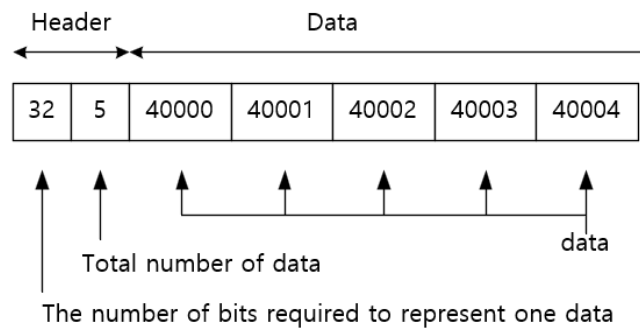


Figure 10. Input data example

The result corresponding to the figure above is as follows.

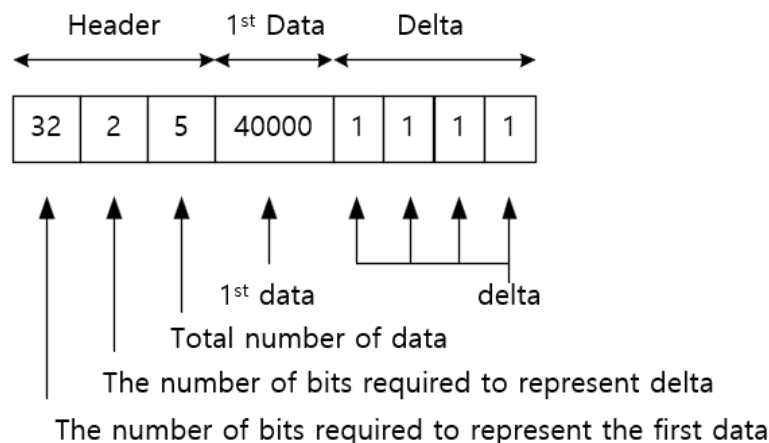
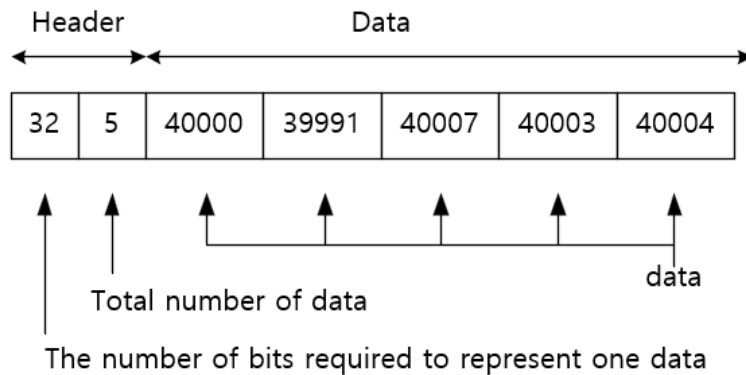


Figure 11. Output Data Example

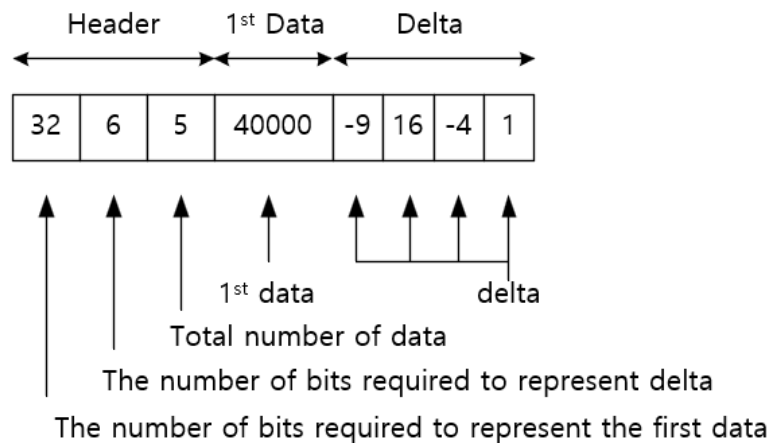
The first header value should be stored in `data_bit_width_`, a member variable of the `Delta` class, using the first value of the input data header. The second header value should be stored in `delta_bit_width_` which is a member variable of the `Delta` class and you should set this value based on the range of delta. This value means the minimum bit-width that can express all of the delta range, and for convenience, it is assumed that `delta_bit_width_` is determined by the following formula.

$$\text{delta_bit_width_} = \text{ceil}(\log_2(\max(\text{abs}(\text{delta}))) + 1) + 1$$

For example, if the maximum value of the absolute value of delta is 8, the `delta_bit_width_` becomes 5. It is assumed that no input is given whose absolute value of delta is greater than 10^6 . The third header value means the number of total data. This value is equal to the number of all deltas plus 1 (plus 1 occurs because of the first data).



Input (Original Data)



Output (Encoded Data)

Figure 12. An example of Delta Encoding

Question 2-5: Estimating Compression Performance [1 Marks]

Objectives: Implement the following function in RunLength.cpp and Delta.cpp to estimate the compression performance.

- `double RunLength::Evaluate(const char* file_name)`
 - It returns the compression ratio of the Run Length Encoding after calculation based on the description.
- `double Delta::Evaluate(const char* file_name)`
 - It returns the compression ratio of the Delta Encoding after calculation based on the description.

Description: The compression ratio is calculated by dividing the size before compression by the size after compression. That is, the higher the compression ratio, the better the compression performance.

The data size before compression can be obtained by the following formula.

The size of Input data =
 $\# \text{ of data} * \text{data bit-width} + \# \text{ of header element} * \text{data bit-width}$

In this case, you can use each value of the input header. It is assumed that the size of each element of the header is equal to the data bit-width.

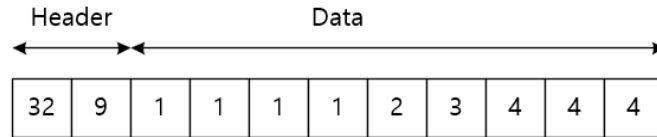
The size of RL-encoded data can be obtained by the following equation. As in the case of input, it is assumed that the size of each element of the header is equal to the size of the data.

The size of RL encoded data =
 $\# \text{ of data} * (\text{data bit-width} + \text{length bit-width}) + \# \text{ of header element} * \text{data bit-width}$

The size of Delta-encoded data can be obtained by the following equation. As in the previous case, the size of each element of the header is assumed to be the same as the data size.

The size of Delta encoded data =
 $\text{data bit-width} + (\# \text{ of data} - 1) * \text{delta bit-width} + \# \text{ of header element} * \text{data bit-width}$

The figure below shows an example of calculating the size of encoded data for each algorithm. The compression ratio of RL is $352/236 = 1.49$, and the compression ratio of Delta encoding is $352/144 = 2.44$. Note that the result is rounded to 2 decimal places.

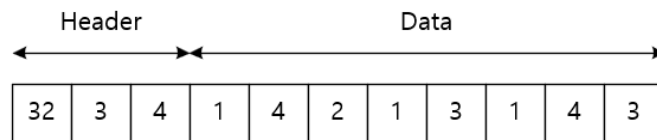


↑ ↑
Total number of data

The number of bits required to represent one data

$$9 \times 32 + 2 \times 32 = 352 \text{ b}$$

Input (Original Data)



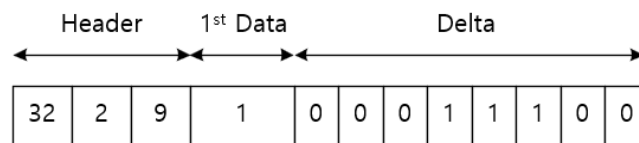
↑ ↑ ↑ ↑ ↑ ...
data length
Total number of data

The number of bits required to represent one length

The number of bytes required to represent one data

$$4 \times (32 + 3) + 3 \times 32 = 236 \text{ b}$$

RL Encoded Data



↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
1st data
Total number of data

The number of bits required to represent delta

The number of bytes required to represent the first data

$$\text{delta_bit_width_} = \text{ceil}(\log_2(1+1)) + 1 = 2$$

$$32 + (9-1) \times 2 + 3 \times 32 = 144 \text{ b}$$

Delta Encoded Data

Figure 13. Compression Ratio Calculation Example