Assignment 2

Message Encoding/Decoding Scheme

Due: 17 **Nov. 2024, 23:59**

1. This assignment contains **ONE** question with **Two** parts. You are required to **submit** a complete C++ program (.cpp only) **for each part** on **PASS** before the deadline (submission will still be allowed one week after the deadline, but a score penalty of 50% will be imposed).
2. You can submit as many times as you want before the deadline, and we will grade your latest version.
3. Only a small set of the test cases are visible for the testing, and a more comprehensive set of test cases will be used for grading. In other word, passing all the visible test cases does not mean that you can get full mark for this assignment. Try your best to thoroughly test your program. Hidden test cases will not be released.
4. The marking of each question is based on the percentage of the total test cases that your solution can pass. If your submitted solution leads to a compilation error on PASS, zero mark will be given to that question and no manual checking to your solution is provided in such a case.
5. No late submission will be accepted. ALL submission should be on **PASS**.
6. Plagiarism check will be performed.
7. You should **ONLY** use the library <iostream> and <cstring>. You are **NOT ALLOWED** to use objects from <string>.

In this assignment, you will write a C++ program that can decode a message under the following ***encoding scheme***.

There are two parts in this encoding scheme. The first part is a character set (header) that covers all possible characters of the messages to be decoded. The second part is an encoded message to be decoded based on the keys in the encoding scheme of the first part.

Part A. Header Encoding

Write a program that reads a line of *printable* characters and encodes with a sequence of keys with each **key** is a binary ***string*** of '0's and '1's. The sequence of keys starts with a key with length 1, followed by three keys with length 2, seven keys of length 3, fifteen keys of length 4, etc. The keys of the same length are in sequence of its binary value but not with all '**0**'s.

The encoding scheme is as follows. (You can get a clearer understanding in this example).

Assume the input header is: **n(X+# $90\"?** ...

	n	(	X	+	#		$	9	0	\	"	?	…
key string:	1	01	10	11	001	010	011	100	101	110	111	0001	…

There is one char "n" encoded by "1", the length is 1;

There are three chars "(", "X", "+" encoded by "01","10","11", respectively. The length of each key is 2;

The same goes for the rest of the other characters...

To verify the correctness of the encoding, write a function to read in a character and print the key(s) of that character.

Notes:

1. The length of the header will **NOT** exceed 256.
2. The **maximum** length of the key string is seven. That is, the longest key string is "**1111111**". *Hint: A character string should have a '\0' at the end of the string. Otherwise, the string will not display its normal content.*
3. The header contains only printable characters including "*space*", i.e. any characters in the ASCII Table with values **from 32 to 126**.
4. We assume the input is valid, i.e., **no need** to check the correctness of input.

5. The header may have repeated characters that lead to different keys. For instance, as shown in "Example 2", there is a user input "`n(X+# $90\"?#`", the first "#" and the second "#" here are both encoded but with different keys. Your program needs to print out all the encoded value with respect to char "#", such as `001` and `0010`.

Sample Input and Output

Example 1

```
Enter Header:
n(X+# $90\"?
Character?$
011
```

Example 2

```
Enter Header:
n(X+# $90\"?#
Character?#
001
0010
```

Example 3

```
Enter Header:
n(X+# $90\"?
Character?_
010
```

input is a "space"

Part B. Message Decoding

Extend your program in **Part A** to decode a message based on the keys generated in Part A.

The encoded message is a sequence of binary digits. The message should be decoded in **multiple** segments. **Each** segment is a subsequence of the encoded message which **starts** with the first 3 digits to represent the length of keys to be decoded in the subsequent digits. The **end** of the segment is signified by a sequence of '`0`'s of that length.

With the example header in Part A, if the segment is `010100100`, the segment would be decoded as follows.

Segment	`010`	`10`	`01`	`00`
	length of key	decoded to be	decoded to be	end of segment
	2	X	(	

One message may have multiple segments. For example, `0101000011011000` is a message that contains two segments, which can be divided into `0101000` and `011011000`. The length of keys in the former is 2, while the latter's is 3. Your program should be able to automatically identify and segment the correct subsequences.

Your program should read in the header and an encoded message; then decode the message and output the decoded message.

Notes:

1. Refer to the Notes of **Part A**.
2. The encoded message is assumed to be in one line.
3. We assume all the input is valid. So, you don't need to consider unreasonable or invalid user input, and we don't include such cases in our test cases, either.

Sample Input and Output

Example 1

```
Enter Header:
n(X+# $90\"?
Enter code:
01100101010000000110011111000010100000
# 9n"X
```

Example 2

```
Enter Header:
:-+ lkC
Enter code:
011011000010101000000
C++
```

Example 3

```
Enter Header:
=>?@A pqrstuv !"#$EFSTCabgh673rs-./01cde92ieo83r
Enter code:
10011001010000010110000001000101101011000001000011000010110001001100000010000111101000001101000010011110000101101100111000000100110100000111110001000011000010101100100110000010000010000101101100000001111000010101110000001000100000000
CS2311 is a great course!
```