# Assignment 2 [8 marks]

## Lucky Number for Date and Digital Number Segmentation

Due: 12th Nov. 2023, 23:59 [Week 11]

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 due date.

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 test cases is visible for the testing, and a more comprehensive set will be used for grading. In other words, passing all the visible test cases does not mean you can get full marks for this assignment. Try your best to test your program thoroughly. 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 marks will be given to that question, and no manual checking of your solution is provided in such a case.

5. Late submissions will not be accepted. ALL submissions should be on PASS.

6. Plagiarism check will be performed.

7. You only need to use the material from Lectures 1 to 5. It is NOT necessary to include any other library, except <iostream>.

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. [4 mark] 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 **one** 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 '1's.

The encoding scheme is as follows.

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

| | n | ( | X | + | # | | $ | 9 | 0 | \ | " | ? | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **key string:** | 0 | 00 | 01 | 10 | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 0000 | … |

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

Notes:

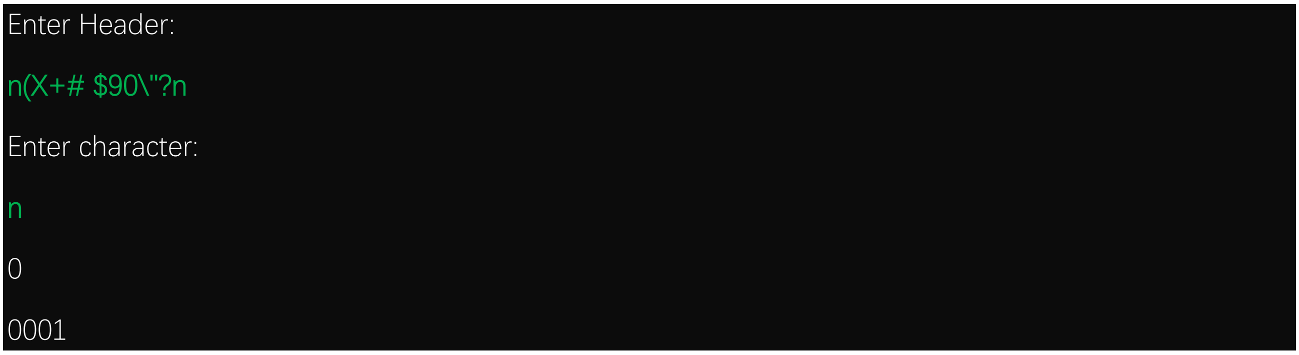
1. You may not know the length of the header in advance.
2. The maximum length of the key string is seven. That is, the longest key string is "1111110". Hint: A character string should have a '\0' at the end of the string.
3. The header contains only printable characters including "space", i.e. any characters 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.

### Sample Input and Output

Example 1

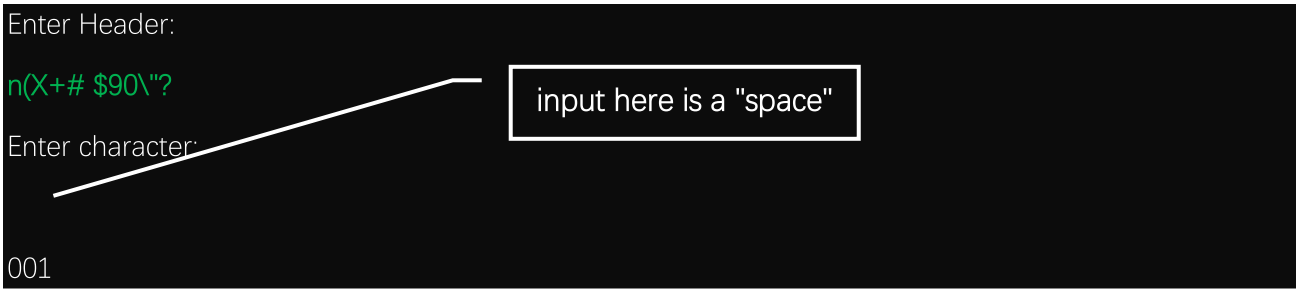
```
Enter Header:

n(X+# $90\"?

Enter character:

$

010
```

Example 2



```
Enter Header:

n(X+# $90\"?n

Enter character:

n

0

0001
```

Example 3



```
Enter Header:

n(X+# $90\"?

Enter character:



001
```

input here is a "space"

## Part B. [4 marks] 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 segments. Each segment starts with the **first 3 digits to represent the length of keys** to be decoded in the subsequent digits. **The end of each segment is signified by a sequence of '1's** of that length.

With the example header in Part A, the segment would be decoded as follows.
The message may have **multiple segments** and ends with the binary string "000".

**Example 1: message 010010011 (1 segment) → decoded message X(**

| 010 | 01 | 00 | 11 | 000 |
|---|---|---|---|---|
| length of key | decoded to be | decoded to be | end seg.1 | end of string |
| 2 | X | ( | | |

**Example 2: message 100 0100 111 (2 segments) → decoded message #$9n**

| 011 | 000 | 010 | 011 | 111 | 001 | 0 | 1 | 000 |
|---|---|---|---|---|---|---|---|---|
| length of key | decoded to be | decoded to be | decoded to be | end seg.1 | length of key | Decoded to be | end seg.2 | end of string |
| 3 | # | $ | 9 | | 1 | n | | |

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 the input is valid.

## Sample Input and Output

Example 1

```
Enter Header:

n(X+# $90\"?

Enter message:

0110000010111110010101111011101001110000

# 9n"X
```

Example 2

```
Enter Header:

:-+ lkC

Enter message:

0110101110100101111000

C++
```

Example 3

```
Enter Header:

=>?@A pqrstuv !"#$EFSTCabgh673rs-./01cde92ieo83r

Enter message:

1001011100111111010111110100010100100111111011001111101100001111101110111110000010110
0001011101111110110101011011111110011001111011110111100001011111010101110010111111000
0001111101101011111101110111110110001111111100001111111000

CS2310 is a great course!
```

Example 4

```
Enter Header:

gi0Astd!2..mt-e+n

Enter message:

0101011011000000111010001100101100010100000011010100011111011100111000

Assignment2
```