Practical - 4

Generate the Golomb Codes for input parameter m=5 and for the set of integers n={0,1,2,…,15}. Display the result as follows.

```
#include <bits/stdc++.h>

using namespace std;

void findGolomb(int n,int m);
int main()
{
    int m;
    cout<<"180420116026 : Gautam Makavana "<<endl;
    cout<<"Practical 4 : Generate the Golomb Codes for input parameter m=5
and for the set of integers "
          "n={0,1,2,...,15}."<<endl;
    cout<<"Enter m : ";
    cin>>m;
    cout<<"n\tq\tr\tcodeword"<<endl;
    for(int i=0;i<=15;i++)
        findGolomb(i,m);
    return 0;
}

void findGolomb(int n,int m){
    int c,k,q,r;
    string x="";
    string y = "";
    q = n/m;

    for(int i=1;i<=q;i++)
        x=x+"1";

    x = x+"0";

    k = ceil(log2(m));
    c = pow(2,k)-m;
    r = n%m;
    int r1=r;
    if(r>=0 && r<c){
        k = k-1;
    }
    else{
```

```
        r1=r1+c;
    }

    while(r1>0){
        y.push_back((char)((char)((int)('0')+(r1%2))));
        r1=r1/2;
    }

    string y1 ="";

    for(int i=y.length()-1;i>=0;i--)
        y1=y1+y.at(i);

    int len = k-y.length();
    string temp = "";
    while(len > 0){
        temp+="0";
        len--;
    }
    x=x+temp;
    x = x+y1;

    cout<<n<<"\t"<<q<<"\t"<<r<<"\t"<<x<<endl;
}
```

Output :

```
180420116026 : Gautam Makavana
Practical 4 : Generate the Golomb Codes for input parameter m=5 and for the set of integ
ers n={0,1,2,´┐¢,15}.
Enter m : 5
n       q       r       codeword
0       0       0       000
1       0       1       001
2       0       2       010
3       0       3       0110
4       0       4       0111
5       1       0       1000
6       1       1       1001
7       1       2       1010
8       1       3       10110
9       1       4       10111
10      2       0       11000
11      2       1       11001
12      2       2       11010
13      2       3       110110
14      2       4       110111
15      3       0       111000
```

Practical - 5

Encode a given string “BILL GATES” using Arithmetic Encoding/Decoding scheme.

```
#include <bits/stdc++.h>

using namespace std;

int main()
{
   cout<<"180420116026 : Gautam Makavana "<<endl;
   cout<<"Practical 5 : Encode a given string “BILL GATES” using Arithmetic
Encoding/Decoding"
          "scheme"<<endl;
   string s;
   string s1;
   cout<<"Enter String : ";
   getline(cin,s);

   map<char, int> order;
   map<char, int> map;

   for(int i=0;i<s.length();i++){

      map[s.at(i)]++;
   }

   double prob[map.size()+1];
   int i=1;

   for(auto m:map)
        {
      prob[i] = (m.second/(double)s.length());
      i++;
   }

   i=1;


   for(auto m:map)
        {
      order[m.first] = i;
      i++;
```

```
}

double F[map.size()+1];

F[0] = 0;
for(int j=1;j<=map.size();j++){

    F[j] = prob[j]+F[j-1];
}
double l=0;
double u = 1;

cout<<endl;

for(int j=0;j<s.size();j++){

    double l1= l + ((u-l)*F[order[s.at(j)]-1]);

    u = l + (u-l)*F[order[s.at(j)]];

    l=l1;

}
double tag = (l+u)/2;

cout<<"Tag is : "<<tag<<endl;

string decode = "";

l=0;
u=1;
double l1 = 0;
double u1 = 1;
for(int j=0;j<s.size();j++){

    for(auto m : order){

        l1 = l + (u-l)*F[m.second-1];

        u1 = l + (u-l)*F[m.second];


        if(tag>=l1 && tag<u1){


            decode.push_back(m.first);
```

```
            l=l1;

            u=u1;
            break;
        }
    }
}

    cout<< "Decode string is : "<<decode<<endl;
    return 0;
}
```

Output

```
180420116026 : Gautam Makavana
Practical 5 : Encode a given string ÔÇ£BILL GATESÔÇØ using Arithmetic Encoding/Decodings
cheme
Enter String : BILL GATES

Tag is : 0.257217
Decode string is : BILL GATES
```

Practical - 6

Write a program to implement digram coding for given text file.

```
#include <bits/stdc++.h>
#include <cmath>

using namespace std;

int main()
{
    int n;

    int totalChar = 0;
    string s = "";
    map<string,string> map;

    cout<<"180420116026 : Gautam Makavana "<<endl;
    cout<<"Practical 6 : Write a program to implement digram coding for given
text file" <<endl;

    ifstream myFile("test.txt");

    if(myFile.fail()){
        cout<<"Error"<<endl;
        return 0;
    }

    char c;
    int count = 0;
    while(myFile.get(c)){

       s.push_back(c);
    }
    myFile.close();

    map["a"]  = "000";
    map["b"]  = "001";
    map["c"]  = "010";
    map["d"]  = "011";
    map["r"]  = "100";
    map["ab"]  = "101";
    map["ac"]  = "110";
    map["ad"]  = "111";

    int i=0;
```

```
    string ans;
    while(i<s.length()){

        string s1 = "";

        s1.push_back(s.at(i));

        if(i!=s.length()-1)
            s1.push_back(s.at(i+1));

        if(map.find(s1)!=map.end()){
            ans = ans + map[s1];
            i = i+2;
        }
        else{
            s1 = "";
            s1.push_back(s.at(i));
            ans = ans+map[s1];
            i++;
        }
    }

    cout<<"Input String : " <<s<<endl;
    cout<<"Encoded String is : "<<ans<<endl;

}
```

Output

```
180420116026 : Gautam Makavana
Practical 6 : Write a program to implement digram coding for given text file
Input String : ancacacannafaafaafc
Encoded String is : 00001011011000000000000000000010
```

Practical - 7

Encode a given input file using dictionary method “LZ77” and store the result in file. Also perform decoding on the same file.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

// windowSize = Size of dictionary
// bufferSize = Size of lookahead buffer
// Important: windowSize < 255 & windowSize > bufferSize!
#define windowSize 60
#define bufferSize 40
#define arraySize bufferSize + windowSize

typedef enum { false, true } bool;

char file_name_path_[1000];
char file_output_name_path_[1000];

//
==========================================================
================

// This method searches for a match from str[] in window[] of strLen length.
// Returns the position of the match starting from the beginning of window[],
// or -1 if no match is found.
// Is invoked during every iteration of the compression algorithm.
int findMatch(unsigned char window[], unsigned char str[], int strLen) {
    int i, j, k, pos = -1;

    for (i = 0; i <= windowSize - strLen; i++) {
        pos = k = i;

        for (j = 0; j < strLen; j++) {
            if (str[j] == window[k])
                k++;
            else
                break;
        }
        if (j == strLen)
            return pos;
```

```
    }

    return -1;
}

//
================================================================
================

// This method contains the logic of the compression algorithm.
// Is invoked when "-c" option is specified in launch command, followed by file
name.
int compress() {
    FILE *fileInput;
    FILE *fileOutput;
    int k, j, l;
    bool last = false;
    int inputLength = 0;
    int outputLength = 0;
    int endOffset = 0;
    int pos = -1;
    int i, size, shift, c_in;
    size_t bytesRead = (size_t) -1;
    unsigned char c;
    unsigned char array[arraySize];
    unsigned char window[windowSize];
    unsigned char buffer[bufferSize];
    unsigned char loadBuffer[bufferSize];
    unsigned char str[bufferSize];

    // Open I/O files
    char path[30] = "input/";
    //strcat(path, inputPath);
    fileInput = fopen(file_name_path_, "rb");
    fileOutput = fopen(file_output_name_path_, "wb");

    // If unable to open file, return alert
    if (!fileInput) {
        fprintf(stderr, "Unable to open fileInput %s", file_name_path_);
        return 0;
    }

    // Get fileInput length
    fseek(fileInput, 0, SEEK_END);
    inputLength = ftell(fileInput);
    fseek(fileInput, 0, SEEK_SET);
```

```
fprintf(stdout, "Input file size: %d bytes", inputLength);

// If file is empty, return alert
if (inputLength == 0)
    return 3;

// If file length is smaller than arraySize, not worth processing
if (inputLength < arraySize)
    return 2;

// Load array with initial bytes
fread(array, 1, arraySize, fileInput);

// Write the first bytes to output file
fwrite(array, 1, windowSize, fileOutput);

// LZ77 logic beginning
while (true) {
    if ((c_in = fgetc(fileInput)) == EOF)
        last = true;
    else
        c = (unsigned char) c_in;

    // Load window (dictionary)
    for (k = 0; k < windowSize; k++)
        window[k] = array[k];

    // Load buffer (lookahead)
    for (k = windowSize, j = 0; k < arraySize; k++, j++) {
        buffer[j] = array[k];
        str[j] = array[k];
    }

    // Search for longest match in window
    if (endOffset != 0) {
        size = bufferSize - endOffset;
        if (endOffset == bufferSize)
            break;
    }
    else {
        size = bufferSize;
    }

    pos = -1;
    for (i = size; i > 0; i--) {
```

```
            pos = findMatch(window, str, i);
            if (pos != -1)
                break;
        }

        // No match found
        // Write only one byte instead of two
        // 255 -> offset = 0, match = 0
        if (pos == -1) {
            fputc(255, fileOutput);
            fputc(buffer[0], fileOutput);
            shift = 1;
        }
        // Found match
        // offset = windowSize - position of match
        // i = number of match bytes
        // endOffset = number of bytes in lookahead buffer not to be considered
(EOF)
        else {
            fputc(windowSize - pos, fileOutput);
            fputc(i, fileOutput);
            if (i == bufferSize) {
                shift = bufferSize + 1;
                if (!last)
                    fputc(c, fileOutput);
                else
                    endOffset = 1;
            }
            else {
                if (i + endOffset != bufferSize)
                    fputc(buffer[i], fileOutput);
                else
                    break;
                shift = i + 1;
            }
        }

        // Shift buffers
        for (j = 0; j < arraySize - shift; j++)
            array[j] = array[j + shift];
        if (!last)
            array[arraySize - shift] = c;

        if (shift == 1 && last)
            endOffset++;
```

```
        // If (shift != 1) -> read more bytes from file
        if (shift != 1) {
            // Load loadBuffer with new bytes
            bytesRead = fread(loadBuffer, 1, (size_t) shift - 1, fileInput);

            // Load array with new bytes
            // Shift bytes in array, then splitted into window[] and buffer[] during
next iteration
            for (k = 0, l = arraySize - shift + 1; k < shift - 1; k++, l++)
                array[l] = loadBuffer[k];

            if (last) {
                endOffset += shift;
                continue;
            }

            if (bytesRead < shift - 1)
                endOffset = shift - 1 - bytesRead;
        }
    }

    // Get fileOutput length
    fseek(fileOutput, 0, SEEK_END);
    outputLength = ftell(fileOutput);
    fseek(fileOutput, 0, SEEK_SET);

    fprintf(stdout, "\nOutput file size: %d bytes\n", outputLength);

    // Close I/O files
    fclose(fileInput);
    fclose(fileOutput);

    return 1;
}

//
================================================================
================

// This method contains the logic of the inverse algorithm, used to
decompress.
// Is invoked when "-d" option is specified in launch command.
int decompress() {
    FILE *fileInput;
    FILE *fileOutput;
    int shift, offset, match, c_in;
```

```
bool done = false;
int i, j;
unsigned char c;
unsigned char window[windowSize];
unsigned char writeBuffer[windowSize];
unsigned char readBuffer[2];

// Open I/O files
fileInput = fopen(file_name_path_, "rb");
fileOutput = fopen(file_output_name_path_, "wb");

if (!fileInput) {
    fprintf(stderr, "Unable to open fileInput %s", file_name_path_);
    return 0;
}

// Load array with initial bytes and write to file
fread(window, 1, windowSize, fileInput);
fwrite(window, 1, windowSize, fileOutput);

// Inverse algorithm beginning
while (true) {
    // Read file by couples/triads to reconstruct original file
    size_t bytesRead = fread(readBuffer, 1, 2, fileInput);

    if (bytesRead >= 2) {
        offset = (int) readBuffer[0];
        match = (int) readBuffer[1];

        // If first byte of readBuffer is 255 -> offset = 0, match = 0
        if (offset == 255) {
            offset = 0;
            c = (unsigned char) match;
            match = 0;
            shift = match + 1;
        }
        else {
            shift = match + 1;
            c_in = fgetc(fileInput);
            if (c_in == EOF)
                done = true;
            else
                c = (unsigned char) c_in;
        }

        // Load and write occurrence to file
```

```
            for (i = 0, j = windowSize - offset; i < match; i++, j++)
                writeBuffer[i] = window[j];
            fwrite(writeBuffer, 1, (size_t) match, fileOutput);

            if (!done)
                fputc(c, fileOutput);

            // Shift window
            for (i = 0; i < windowSize - shift; i++)
                window[i] = window[i + shift];

            for (i = 0, j = windowSize - shift; i < match; i++, j++)
                window[j] = writeBuffer[i];
            window[windowSize - 1] = c;
        }
        else {
            break;
        }
    }

    // Close I/O files
    fclose(fileInput);
    fclose(fileOutput);

    return 1;
}

//
================================================================
================

// This method is the controller, reads user inputs.
// Is invoked on program launch.
int main(int argc, char* argv[]) {
    clock_t begin;
    char d_or_c;
    printf("\n180420116026 : Gautam Makavana \n\nPractical No. 7: Encode a
given input file using directory method \"LZ77\" and store the result in file. Also
perform decoding on the same file.\n\n");
    printf("\n\nThis is file compression and decompression LZ77
algorithm.\n\n");
    printf("Enter your file name or file path(if in different directory): ");
    scanf("%s", file_name_path_);

    again1:
    printf("\n\nDo you want to compress or decompress?('c' or 'd'): ");
```

```
    scanf(" %c", &d_or_c);

    //printf("\nThe File name is: %s", file_name_path_);


    if (d_or_c!='c' && d_or_c!='d') {
        printf("Please enter a correct choice('c' or 'd'): ");
        goto again1;
    } else {
        // Start decompression
        printf("Enter your file name or file path(if in different directory) for
OUTPUT: ");
          scanf("%s", file_output_name_path_);
          begin = clock();
        if (d_or_c=='d') {
            int result = decompress();

            if (result == 0) {
                fprintf(stderr, "\nDecompression FAIL");
            } else if (result == 1) {
                printf("\nDecompression OK");
            }
        }
        // Start compression
        else if (d_or_c == 'c') {
            int result = compress();
            if (result == 0) {
                fprintf(stderr, "\nCompression FAIL\n");
            } else if (result == 1) {
                printf("\nCompression OK");
            } else if (result == 2) {
                fprintf(stderr, "\nFile too small\n");
            } else if (result == 3) {
                fprintf(stderr, "\nFile is EMPTY\n");
            }
        } else {
            printf("Invalid arguments");
        }
    }

    // Print execution time
    clock_t end = clock();
    printf("\n\nExecution time: ");
    printf("%f", ((double) (end - begin) / CLOCKS_PER_SEC));
    printf(" [seconds]");
```

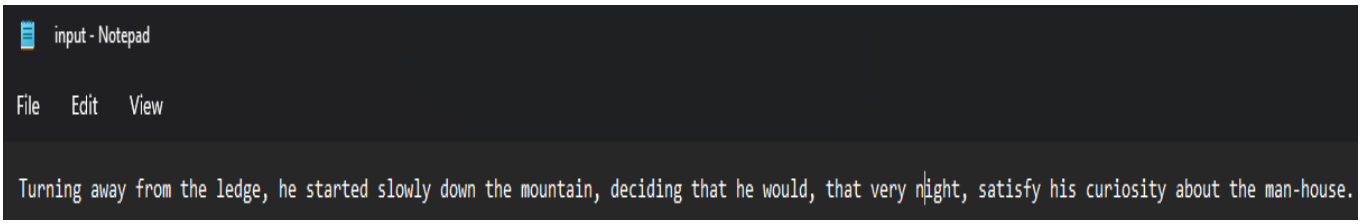
```
    return 0;
}
```

Output :

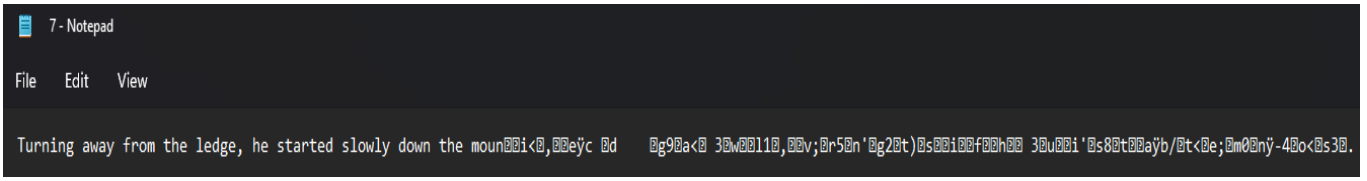
Compression

```
180420116026 : Gautam Makavana

Practical No. 7: Encode a given input file using directory method "LZ77" and store the r
esult in file. Also perform decoding on the same file.



This is file compression and decompression LZ77 algorithm.

Enter your file name or file path(if in different directory): input.txt


Do you want to compress or decompress?('c' or 'd'): c
Enter your file name or file path(if in different directory) for OUTPUT: 7.txt
Input file size: 149 bytes
Output file size: 162 bytes

Compression OK

Execution time: 0.005000 [seconds]
```

input.txt



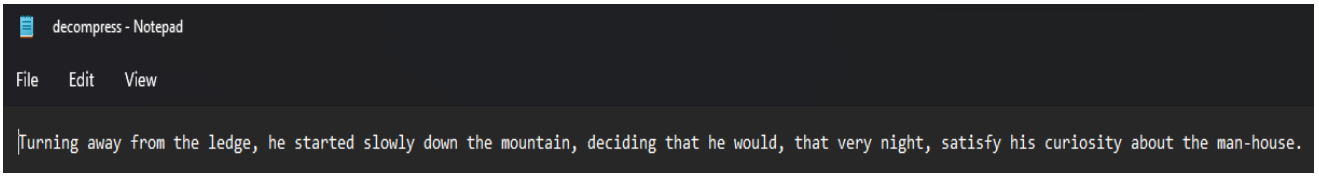
7.txt



Decompression

```
180420116026 : Gautam Makavana

Practical No. 7: Encode a given input file using directory method "LZ77" and store the r
esult in file. Also perform decoding on the same file.



This is file compression and decompression LZ77 algorithm.

Enter your file name or file path(if in different directory): 7.txt


Do you want to compress or decompress?('c' or 'd'): d
Enter your file name or file path(if in different directory) for OUTPUT: decompress.txt

Decompression OK

Execution time: 0.001000 [seconds]
```

decompress.txt



Practical - 8

Encode a given input file using dictionary method “LZ78” and store the result in file.

```
#include <bits/stdc++.h>
#include <list>
#include <bitset>
#include <iostream>
#include <fstream>
#include <string>
#include <algorithm>
#include <queue>
#include<time.h>

using namespace std;

string file_input_name_, file_output_name_;

string encode_int(int in)
{
  return bitset<8>(in).to_string();
}

int decode_int(char out)
{
  return bitset<8>(out).to_ulong();
}

string encode_char(string in)
{
  return bitset<8>(in[0]).to_string();
}

char decode_char(string out)
{
  return (char)bitset<8>(out).to_ulong();
}

struct Dict
{
  string label; // dictionary entry string
  char output;  // first non-matching symbol
  int entry;    // longest matching dictionary entry

  Dict(string label, int entry, char output) // constructor
```

```
  {
    this->label = label;
    this->entry = entry;
    this->output = output;
  }
};

int find(string l, list<Dict> enc_list)
{ // change list to map

  list<Dict> temp = enc_list;
  int i = 1;

  while(!temp.empty())
  {
      if(!(l.compare(temp.front().label)))
      {
          return i;
      }
      temp.pop_front();
      i++;
  }
  return -1;
}

void write_file(string input, string output_filename)
{
  string one_byte;
  unsigned long bin_number;
  unsigned char chr;
  int i, len = input.length();

  FILE *fp;
  fp = fopen(output_filename.c_str(), "wb");

  if(fp == NULL)
  {
    printf("Unable to open output file!\n");
    return;
  }

  for (i=0; i<len; i+= 8)
  {
    one_byte = input.substr(i, 8);
    bin_number = strtol(one_byte.c_str(), NULL, 2);
```

```
    chr = bin_number;
    fprintf(fp, "%c", bin_number);
  }

  fclose(fp);
}

void LZ78_Compress(string txt, string output_filename)
{
  list <Dict> Dictionary;
  string Prefix = "", Char, compressed;

  int CodeWord, IndexForPrefix = 1, len, i;

  len = txt.length();


  for(i=0; i<len; i++){

     Char = string(1, txt[i]);

     IndexForPrefix = find((Prefix + Char), Dictionary);  // if it equals to -1, it
means (Prefix + Char) is not in the dictionary
     if(IndexForPrefix != -1)
     {
        Prefix = Prefix + Char; // if  Prefix + Char already exists, append Char
     }

     else
     {
       if(Prefix.empty())
       {
         CodeWord = 0;           // if Prefix is empty, a new letter was processed
         compressed += "00000000";
       }
       else
       {
         CodeWord = find(Prefix, Dictionary);     // search Prefix index
         compressed += encode_int(CodeWord);       // encode index
       }

       compressed += encode_char(Char);                              // encode char
       Dictionary.push_back(Dict((Prefix + Char), CodeWord, txt[i])); // add new
entry to the dictionary
       Prefix.clear();
     }
```

```
  }


        write_file(compressed, output_filename);
}

void LZ78_Decompress(string input_filename, string output_filename)
{
  // Decompression Variables
  string dict = "";
  string decompressed_text;      // the the decomressed string
  string compressed_text;        // the compressed input
  string character;              // the character immediately after the current
codeword
  string temp;

  unsigned char ch;
  unsigned int codeword, l = 0, i, len;          // the current dictionary entry being
processed

  FILE *fp;
  fp = fopen(input_filename.c_str(), "rb");

  if(fp == NULL)
  {
    printf("Unable to open compressed file!\n");
    return;
  }


  while(fscanf(fp, "%c", &ch) == 1)
  {
    compressed_text += ch;
  }
  len = compressed_text.length();

  fclose(fp);

  ofstream outfile(output_filename.c_str(), ios::binary);

  int *idx = new int[len]; // used for storing the index of the i-th dictionary entry

  for (i=0;i<len;i+=2)
  {
    codeword = compressed_text[i];                    // longest matching dictionary
entry
```

```
   character = compressed_text.substr(i + 1, 1);       // first non-matching
symbol
   dict += character;
   idx[l] = codeword;
   l++; // idx size

   // let's say l = 0
   // then (idx[0], dict[0]) represents the first dictionary entry

   if(codeword == 0)
   {
      decompressed_text += character; // new letter, just append
   }

   else
   {
     while(codeword > 0)  // go back in the dictionary string, adding each letter
until you get one with codeword = 0
      {
       temp += dict[codeword-1];
       codeword = idx[codeword-1];
      }
      reverse(temp.begin(), temp.end()); // restore correct order
      decompressed_text += temp;         // append string and char
      decompressed_text += character;
      temp.clear();
   }
  }


        outfile << decompressed_text;
        outfile.close();

}


void Compress(string input_filename, string output_filename)
{
  ifstream in(input_filename.c_str());
  string line, txt;

  while(getline(in, line))
  {
    txt += line;
    txt += "\n";
  }
```

```
  in.close();

  LZ78_Compress(txt, output_filename);
}

int main()
{
	char d_or_c_;
	clock_t begin;

	cout<<"\n180420116026 : Gautam Makavana\n\nPractical No. 8: 
Encode a given input file using dictionary method \"LZ78\" and store the result 
in file.\n\n";
	cout<<"This is LZ78 File compression and Decompression 
Algorithm.\n\n";

	cout<<"Enter your file name or file path(if in different directory): ";
	cin>>file_input_name_;

	again1:
	cout<<"\nDo you want to compress or decompress file?('c' or 'd'): ";
	cin>>d_or_c_;

	if(d_or_c_=='d'){

		cout<<"\nEnter your file name or file path(if in different directory) 
for OUTPUT: ";
		cin>>file_output_name_;

		begin = clock();

		LZ78_Decompress(file_input_name_, file_output_name_);

		cout<<endl<<"\nDecompression is completed.\n";


	}
	else if(d_or_c_=='c'){

		cout<<"\nEnter your file name or file path(if in different directory) 
for OUTPUT: ";
		cin>>file_output_name_;

		begin = clock();

		Compress(file_input_name_, file_output_name_);
```

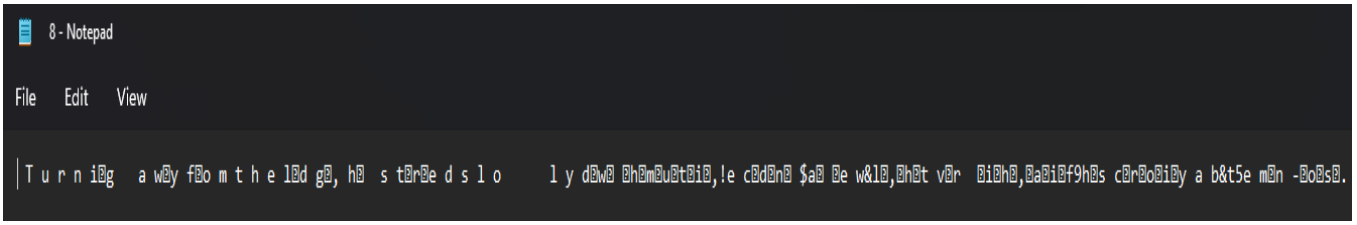
```
            cout<<endl<<"\nCompression is completed.\n";

        }
        else{
            cout<<endl<<"Please enter correct choice 'c' or 'd'.";
            goto again1;
        }


        clock_t end = clock();

        cout<<"\n\nExecution time: ";
    cout<<((double) (end - begin) / CLOCKS_PER_SEC);
    cout<<" [seconds]";

        return 0;

}
```

Output :

Compression

```
180420116026 : Gautam Makavana

Practical No. 8: Encode a given input file using dictionary method "LZ78" and store the
result in file.

This is LZ78 File compression and Decompression Algorithm.

Enter your file name or file path(if in different directory): input.txt

Do you want to compress or decompress file?('c' or 'd'): c

Enter your file name or file path(if in different directory) for OUTPUT: 8.txt


Compression is completed.


Execution time: 0.001 [seconds]
```

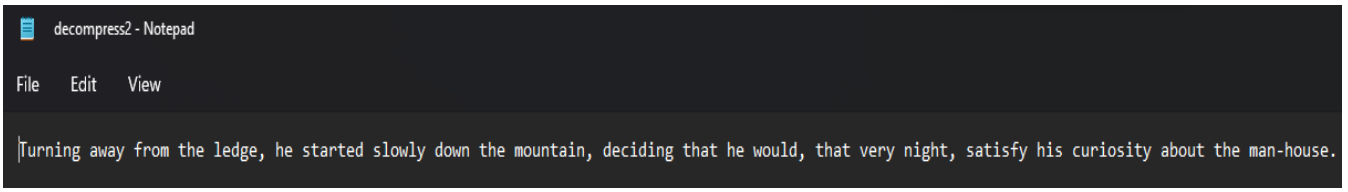
8.txt



Decompression

```
180420116026 : Gautam Makavana

Practical No. 8: Encode a given input file using dictionary method "LZ78" and store the
result in file.

This is LZ78 File compression and Decompression Algorithm.

Enter your file name or file path(if in different directory): 8.txt

Do you want to compress or decompress file?('c' or 'd'): d

Enter your file name or file path(if in different directory) for OUTPUT: decompress2.txt



Decompression is completed.


Execution time: 0.001 [seconds]
```

decompress2.txt



Practical - 9

Encode a given input file using dictionary method “LZW” and store the result in file.

```
#include<stdio.h>
#include<string.h>
char dict[100][100]={{"\0"}};
int lr=-1; //it keeps count on which the last string was added.
FILE *fout;
int findIndex(char *s){
 int i=0;
 while(i<=lr){
 if(strcmp(dict[i],s)==0){
 return i;
 }
 i++;
 }
 return -1;
}
void displayDict(){
 int i=0;
 printf("Code\tword\n");
 while(i<=lr){
 printf("%d\t%s\n",i,dict[i]);
 i++;
 }
}
void makeInitialDict(char *s){
 int i=0;
 while(s[i] !='\0'){
 if(isalpha(s[i])){
 lr++;
 dict[lr][0] = s[i];
 dict[lr][1] = '\0';
 }
 i++;
 }
}
void insertInDict(char *s){
 lr++;
 strcpy(dict[lr],s);
}
void insertInFile(int m,int coma){
 int r=0;
 if(m==0){
```

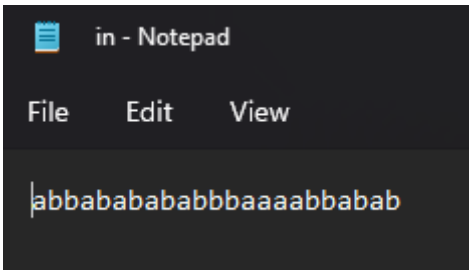
```
 fputc('0',fout);
 fputc(',',fout);
 return;
 }

 while (m!=0){
 r = r*10 + (m%10);
 m/=10;
 }
 while(r!=0){
 m = r%10;
 fputc(m+48,fout);
 r/=10;
 }
 if(coma == 1){
 fputc(',',fout);
 }
}
int main(){
 char s[100],charSet[26],in[100],out[100],w[20],t;
 int m,i=0;
 printf("180420116026 : Gautam Makavana \n");
 printf("Enter character set for initial dict(seperated by space) : ");
 fgets(s,100,stdin);
 makeInitialDict(s);
 printf("Enter input file name (press enter to use in.txt): ");
 fgets(in,100,stdin);
 if(strcmp(in,"")){
 strcpy(in,"in.txt");
 }
 printf("Enter output file name (press enter to use out.txt): ");
 fgets(out,100,stdin);
 if(strcmp(out,"")){
 strcpy(out,"out.txt");
 }
 FILE *fin = fopen(in,"r");
 if(fin == NULL){
 printf("Please create a input file name %s\n\n",in);
 }
 else{
 fout = fopen(out,"w");
 w[0] = '\0';
 while((t=fgetc(fin))!=EOF){
 w[i++] = t;
 w[i] = '\0';
 m = findIndex(w);
```

```
 if(m == -1){
 insertInDict(w);
 w[strlen(w)-1] = '\0';
 m = findIndex(w);
 insertInFile(m,1);
 w[0]=t; //set the last
 i=1;
 w[1] = '\0';
 }
 }

 m = findIndex(w);
 if(m != -1){
 insertInFile(m,0);
 }
 }
 fclose(fin);
 fclose(fout);
}
```

Output

```
180420116026 : Gautam Makavana
Enter character set for initial dict(seperated by space) : a b
Enter input file name (press enter to use in.txt): in.txt
Enter output file name (press enter to use out.txt): out.txt
```

in.txt



out.txt