**Scientific Report**



**C code to implement the BWT algorithm (Encoding/decoding):**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_LENGTH 100

char stop\_char;

void findBWT(char \*S);

char \*inverseBWT(char bwt\_arr[]);

int compare(const void \*a, const void \*b);

void menu();

// Comparison function for qsort

int compare(const void \*a, const void \*b) {

return strcmp(\*(const char \*\*)a, \*(const char \*\*)b);

}

// Function to find Burrows-Wheeler Transform (BWT)

void findBWT(char \*S) {

int len = strlen(S);

char \*\*rotations = (char \*\*)malloc(len \* sizeof(char \*));

if(rotations == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

// Generate all rotations of S

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

rotations[i] = (char \*)malloc((len + 1) \* sizeof(char));

if(rotations[i] == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

strcpy(rotations[i], S + i);

strncat(rotations[i], S, i);

}

// Sort rotations lexicographically

qsort(rotations, len, sizeof(rotations[0]), compare);

// Print BWT matrix

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

printf("%s\n", rotations[i]);

}

// Extract encoded message

printf("The Encoded message is: ");

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

printf("%c", rotations[i][len - 1]);

}

printf("\n");

// Free memory

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

free(rotations[i]);

}

free(rotations);

}

// Function to sort strings lexicographically

void sortStrings(char (\*strings)[MAX\_LENGTH], int size) {

qsort(strings, size, MAX\_LENGTH, (int (\*)(const void \*, const void \*))strcmp);

}

// Function to append characters to strings

char (\*appendCharacter(char (\*strings)[MAX\_LENGTH], char \*characters, int size))[MAX\_LENGTH] {

char (\*result)[MAX\_LENGTH] = malloc(size \* sizeof(\*result));

if (!result) {

fprintf(stderr, "Memory allocation failed\n");

exit(EXIT\_FAILURE);

}

for (int i = 0; i < size; i++) {

memmove(strings[i] + 1, strings[i], strlen(strings[i]) + 1);

strings[i][0] = characters[i];

strcpy(result[i], strings[i]);

}

return result;

}

// Function to search for strings ending with a specific character

void searchAndDisplay(char (\*strings)[MAX\_LENGTH], int size, char ch) {

printf("Strings ending with '%c':\n", ch);

for (int i = 0; i < size; i++) {

if (strlen(strings[i]) == size && strings[i][strlen(strings[i]) - 1] == ch) {

printf("Decoded String %d: %s\n", i + 1, strings[i]);

break;

}

}

}

// Menu function to display options

void menu() {

printf("\n");

printf("Menu:\n");

printf("1. Encode String\n");

printf("2. Decode String\n");

printf("3. Exit\n");

printf("Enter your choice: ");

}

int main() {

char input[MAX\_LENGTH];

int choice;

while (1) {

menu();

scanf("%d", &choice);

switch(choice) {

case 1:

char inp[MAX\_LENGTH];

printf("Enter the string: ");

scanf("%s", inp);

printf("Enter the end of file character: ");

scanf(" %c", &stop\_char);

int len = strlen(inp);

char S[MAX\_LENGTH];

strcpy(S, inp);

S[len] = stop\_char;

S[len + 1] = '\0';

printf("%s",S);

findBWT(S); //encoding in case 1

break;

case 2:{

char input[MAX\_LENGTH];

printf("Enter a string: ");

scanf("%s", input);

int size = strlen(input);

char strings[size][MAX\_LENGTH];

char characters[size];

char endchr;

printf("Enter end char: ");

scanf(" %c", &endchr);

for (int i = 0; i < size; i++) {

strings[i][0] = input[i];

strings[i][1] = '\0';

characters[i] = input[i];

}

printf("\n");

printf("Original strings:\n");

for (int i = 0; i < size; i++) {

printf("String %d: %s\n", i + 1, strings[i]);

}

printf("\n");

for (int i = 0; i < size-1; i++) {

sortStrings(strings, size);

printf("Sorted strings:\n");

for (int i = 0; i < size; i++) {

printf("String %d: %s\n", i + 1, strings[i]);

}

printf("\n");

// decoding in case 2

char (\*appended)[MAX\_LENGTH] = appendCharacter(strings, characters, size);

printf("After appending character '%c':\n", characters[i]);

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

printf("String %d: %s\n", j + 1, appended[j]);

}

if(strlen(appended[0])==size){

searchAndDisplay(appended,size, endchr);

break;

}

printf("\n");

free(appended);

}

break;}

case 3:{

printf("Exiting program.\n");

exit(0); // exit application

}

default:

printf("Invalid choice. Please enter a valid option.\n");

break;

}

}

return 0;

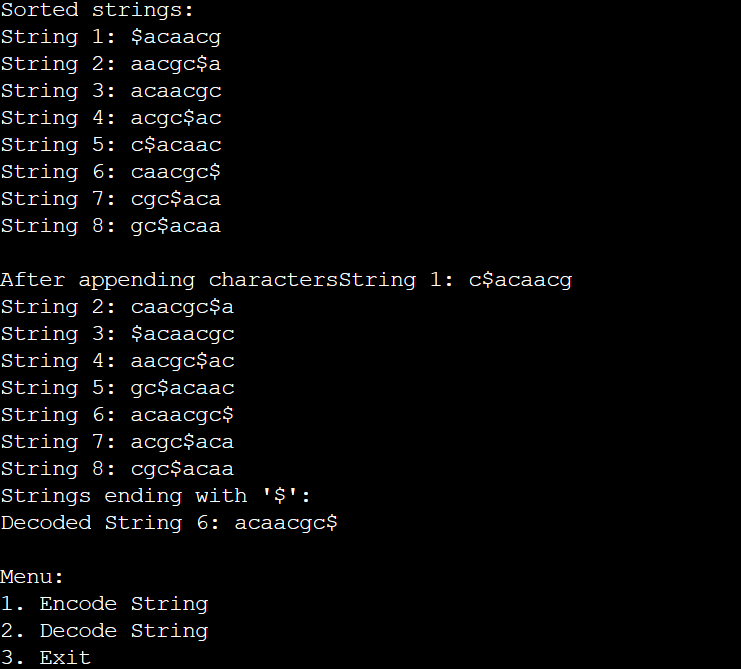
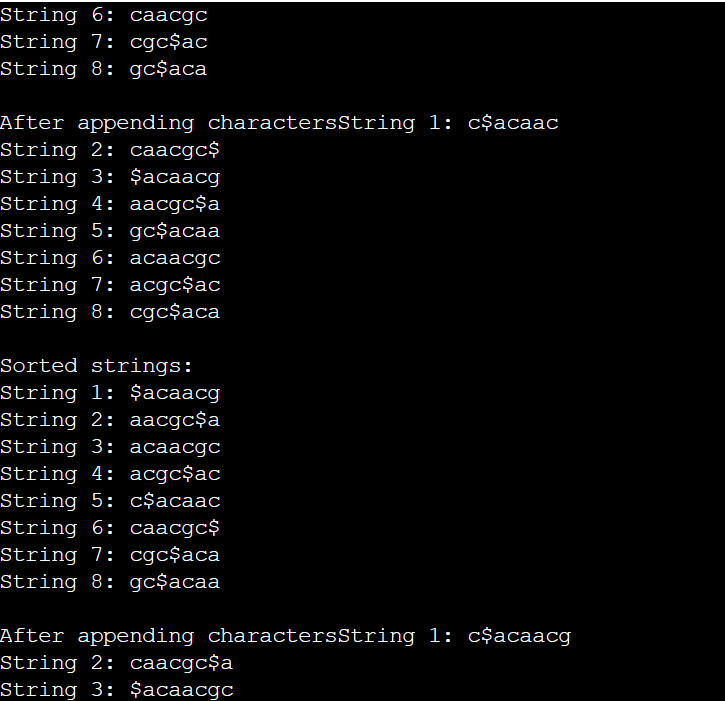
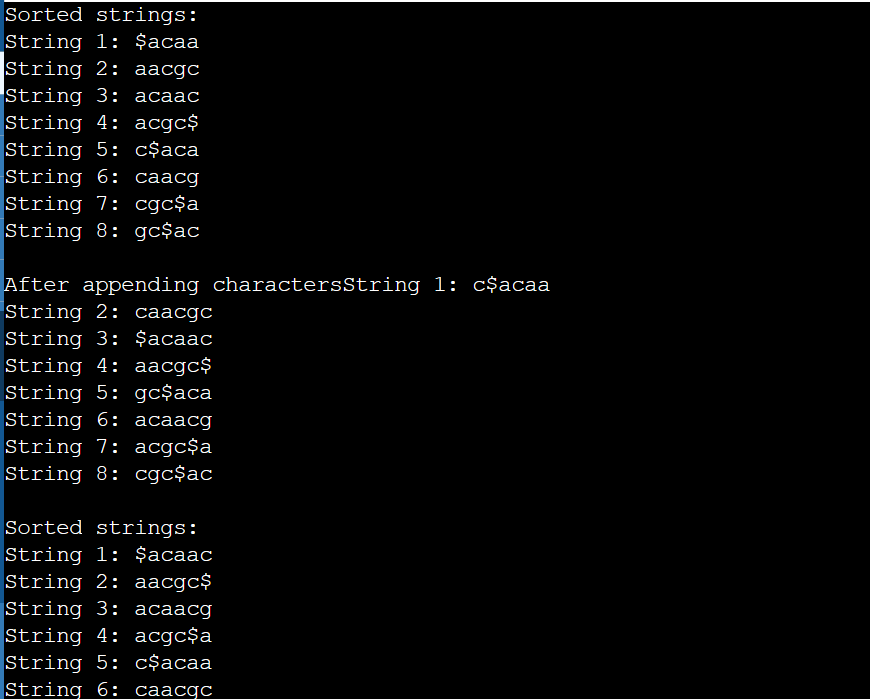
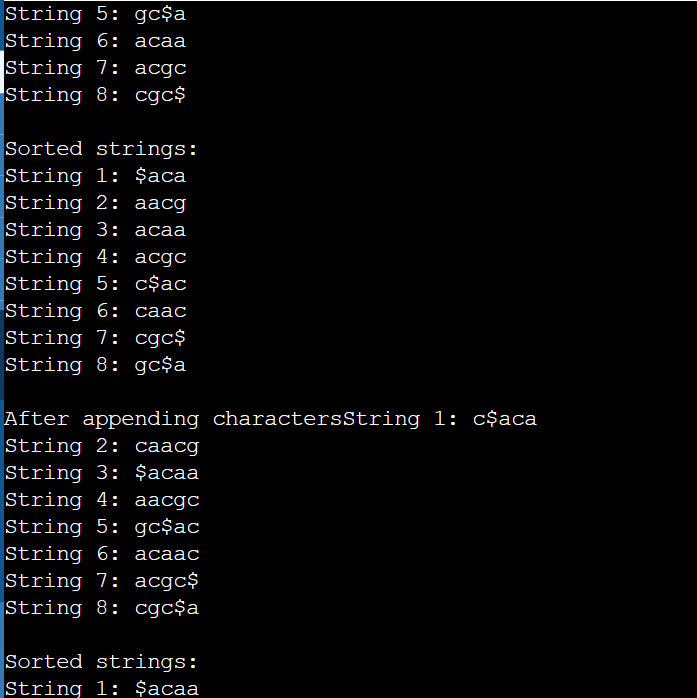
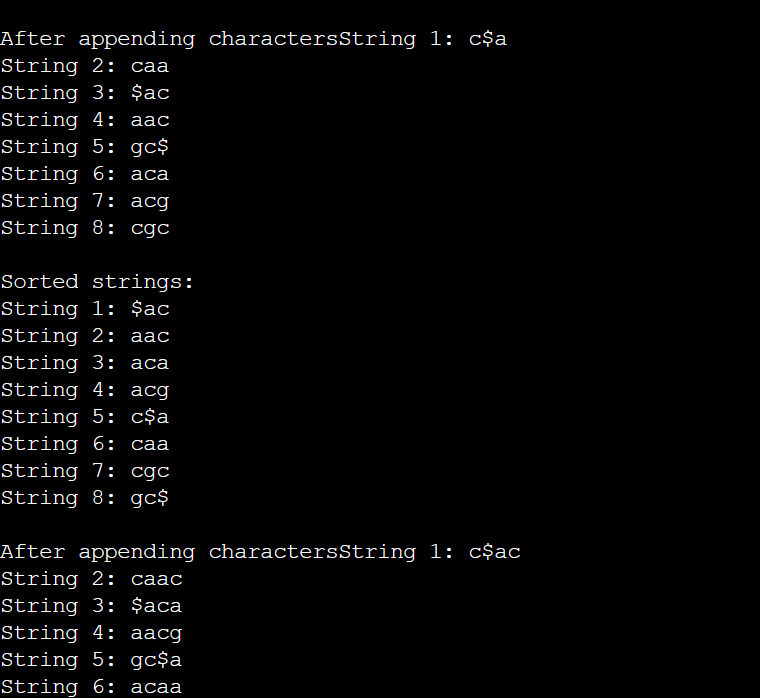
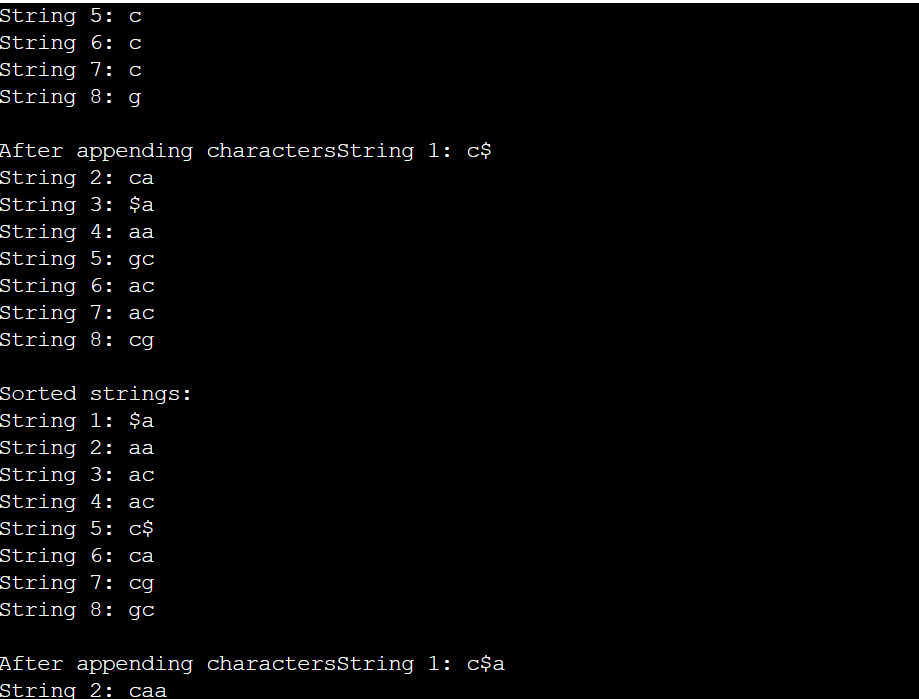
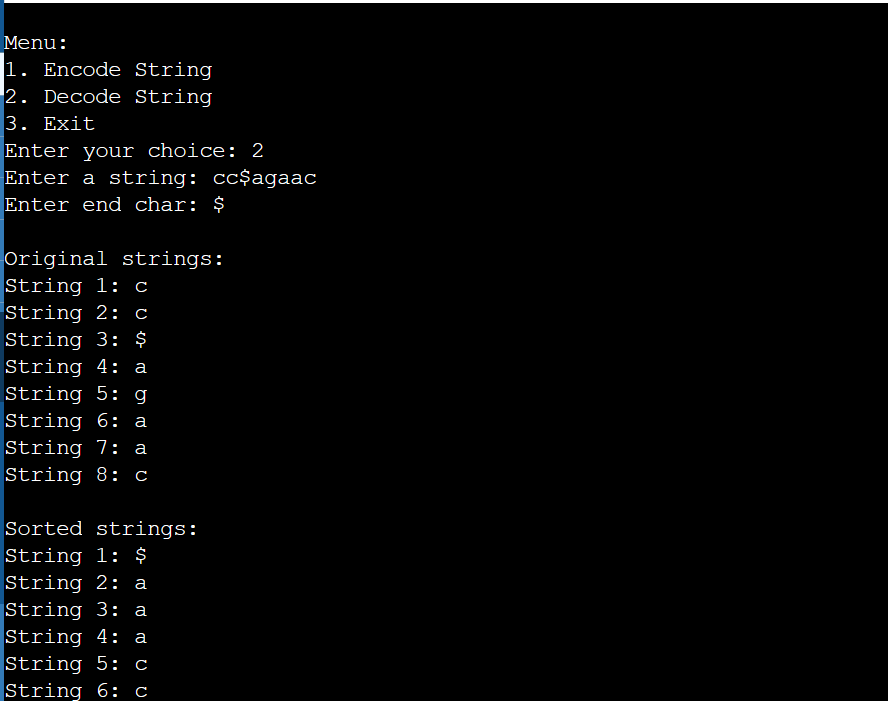
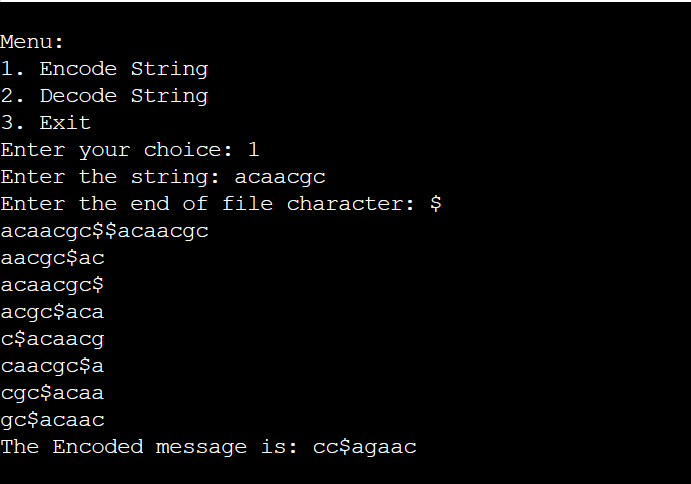
}

**Appendix B: Strings for Testing Using BWT**

**S1: acaacgc with end of file character $**

The number of sort and append operations: 7

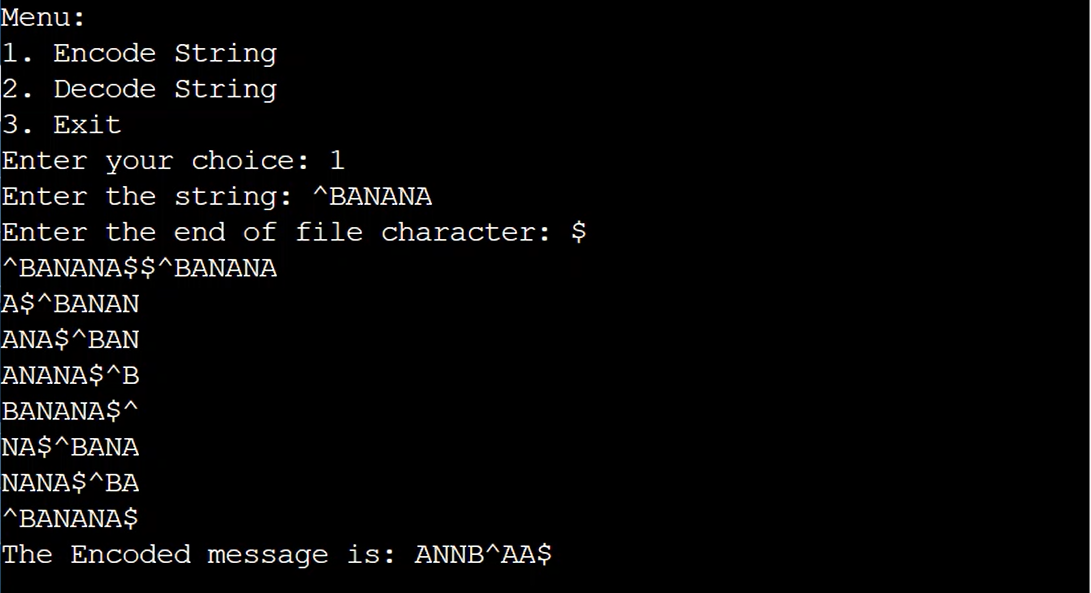
**ENCODED STRING:**



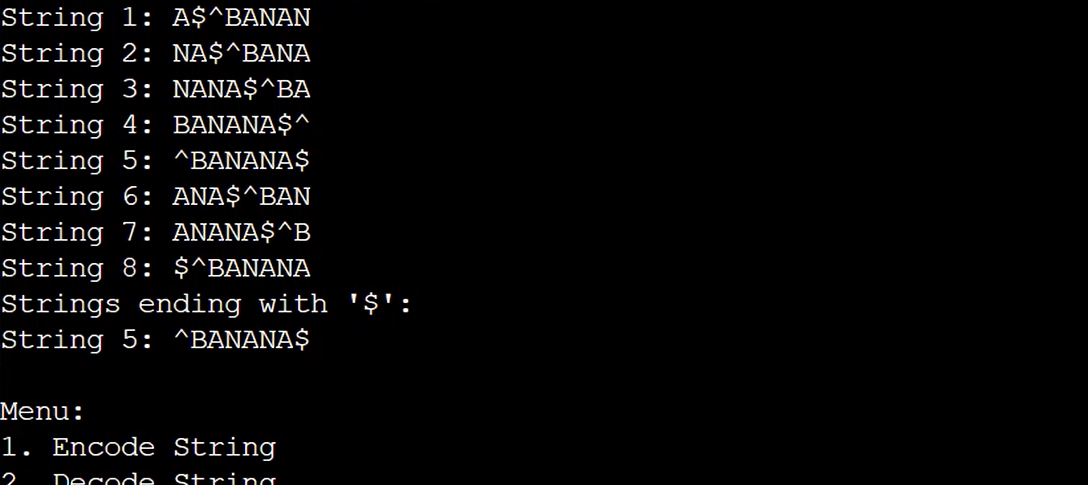
**S2: ^BANANA with end of file character $**

The number of sort and append operations: 7

**ENCODED STRING:**



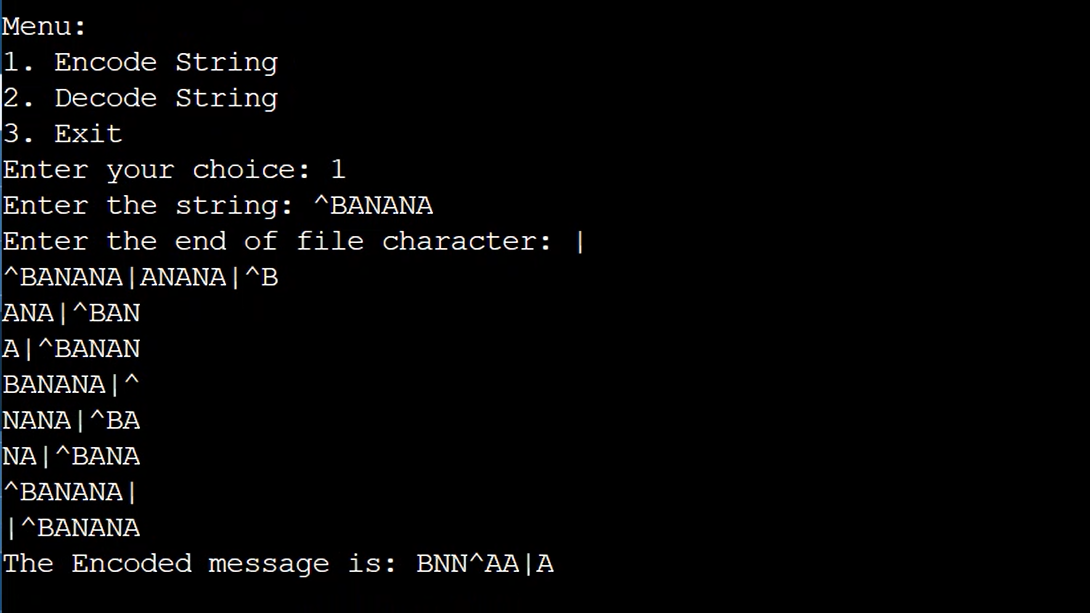
DECODED STRING:



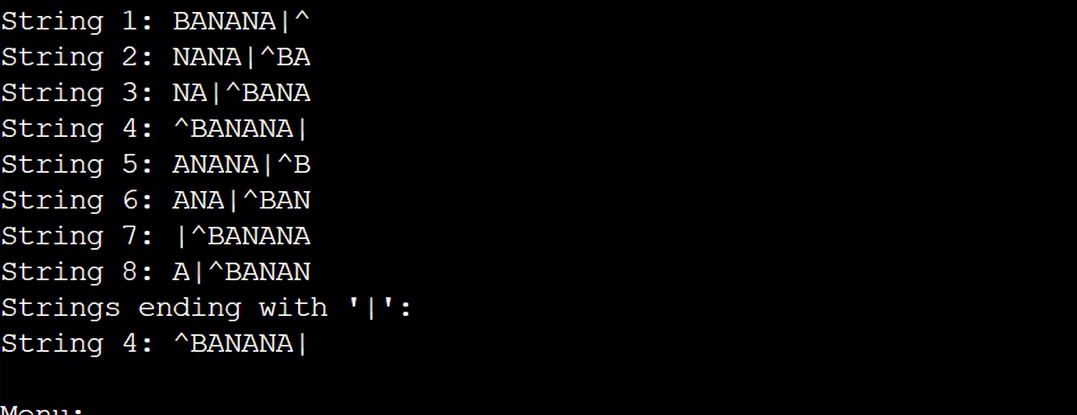
**S3: ^BANANA with end of file character |**

The number of sort and append operations: 7

**ENCODED STRING:**



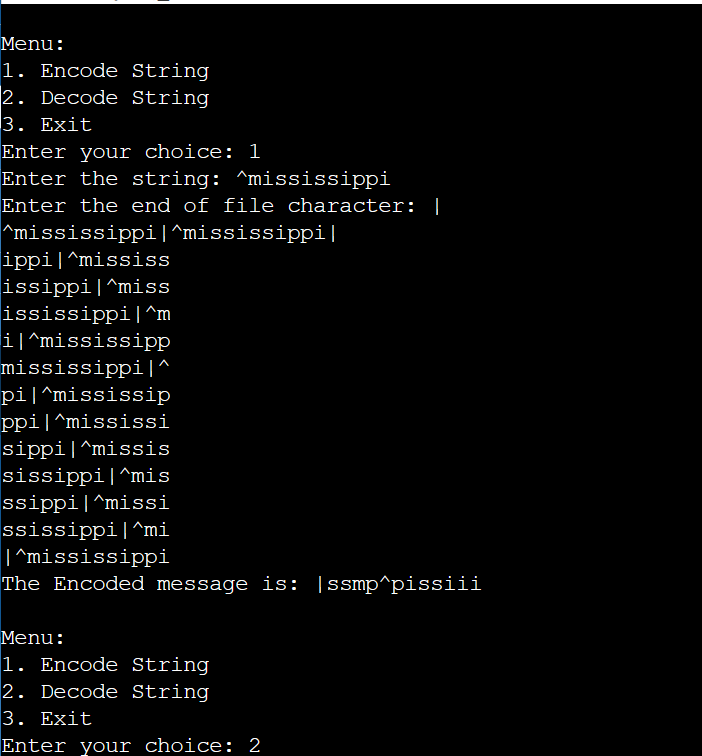
DECODED STRING:

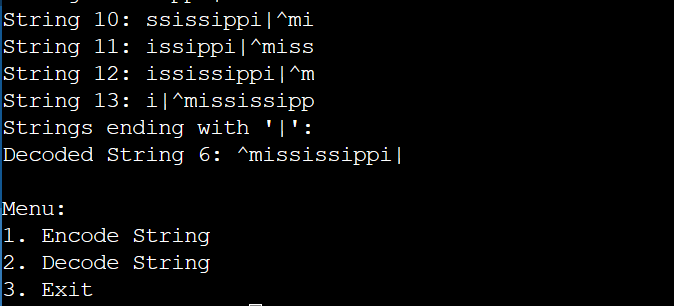
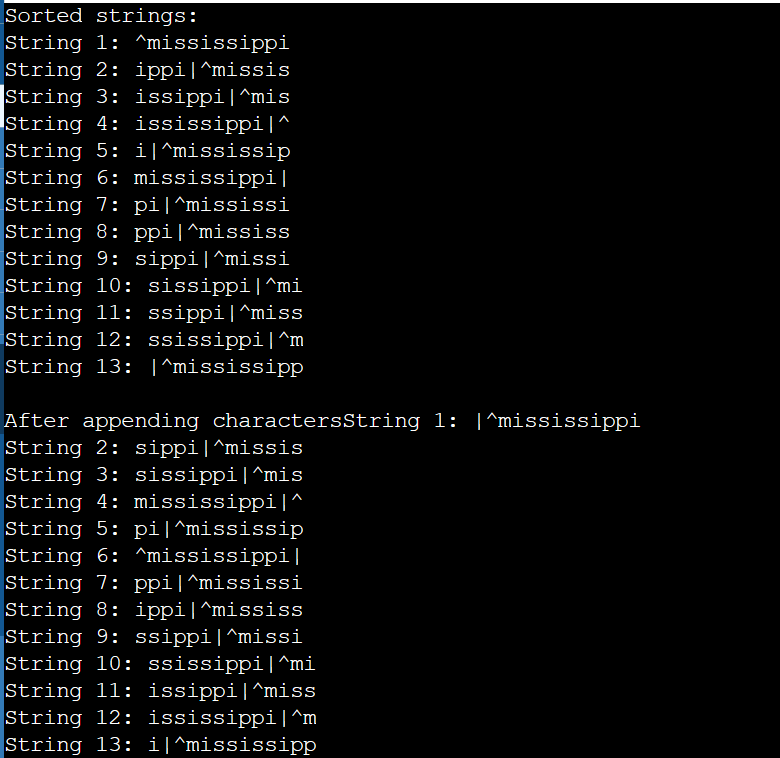
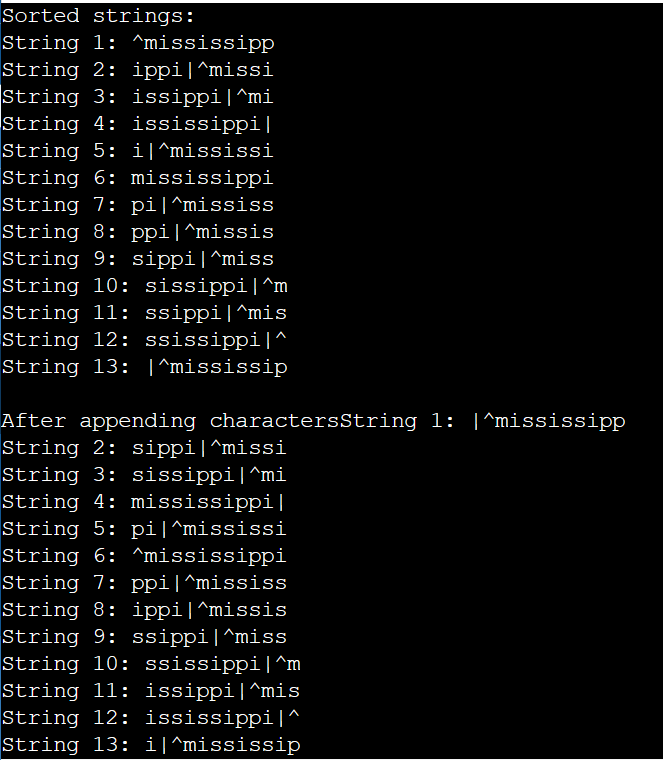
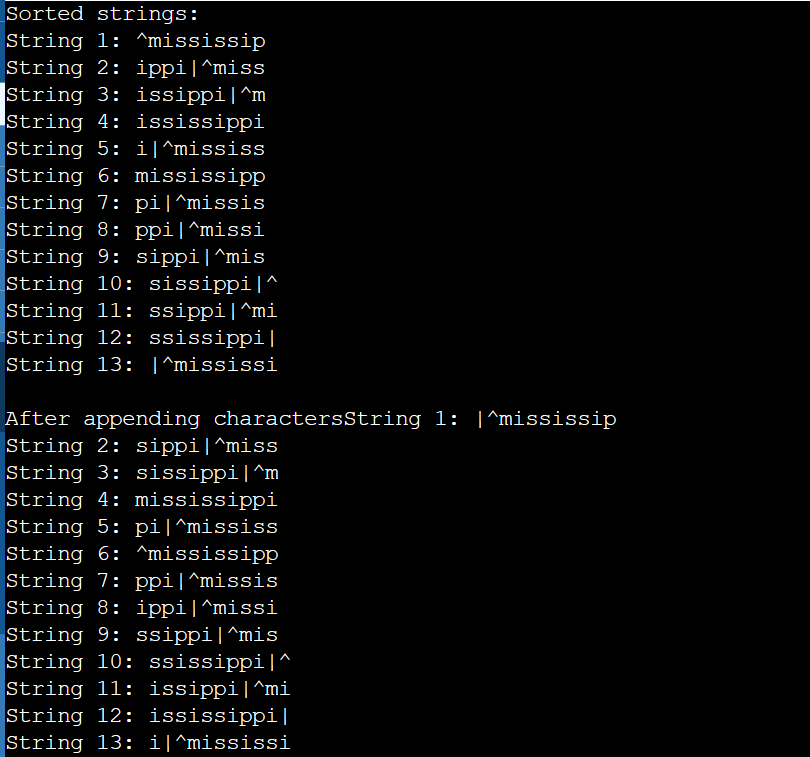
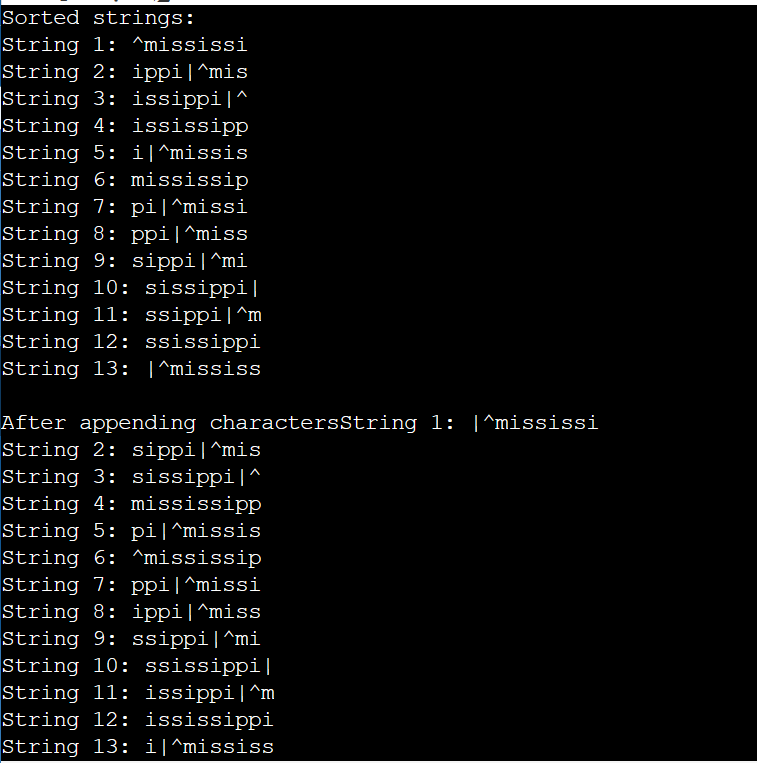
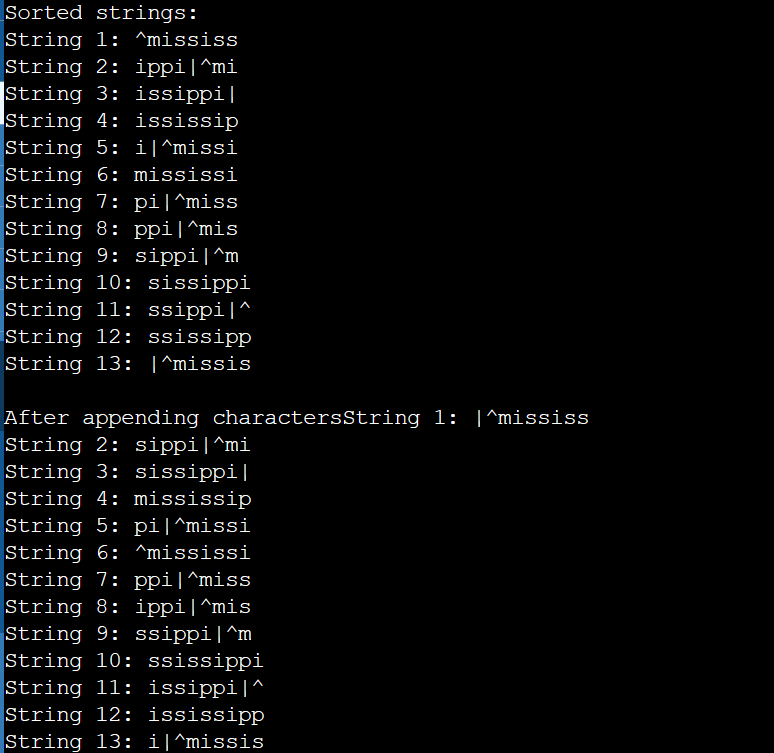
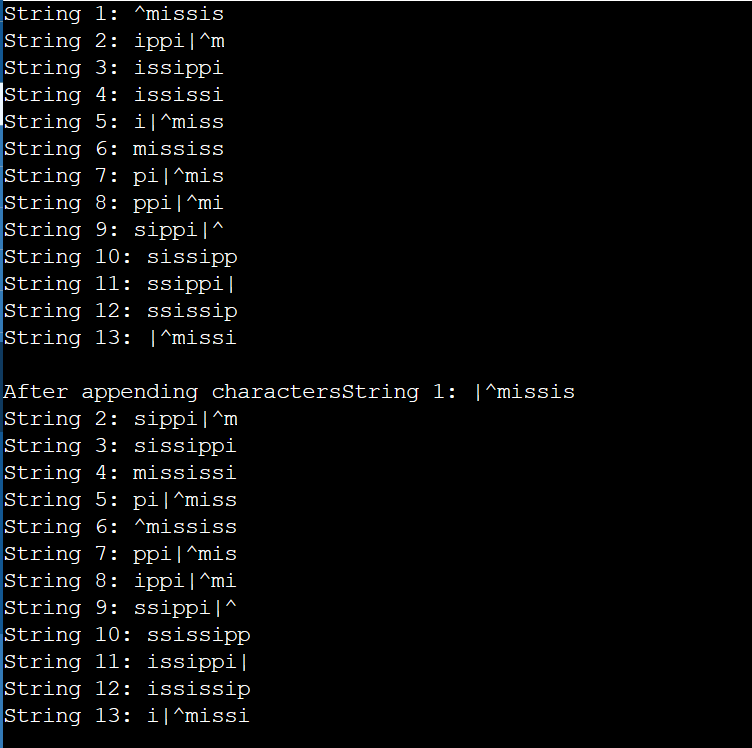
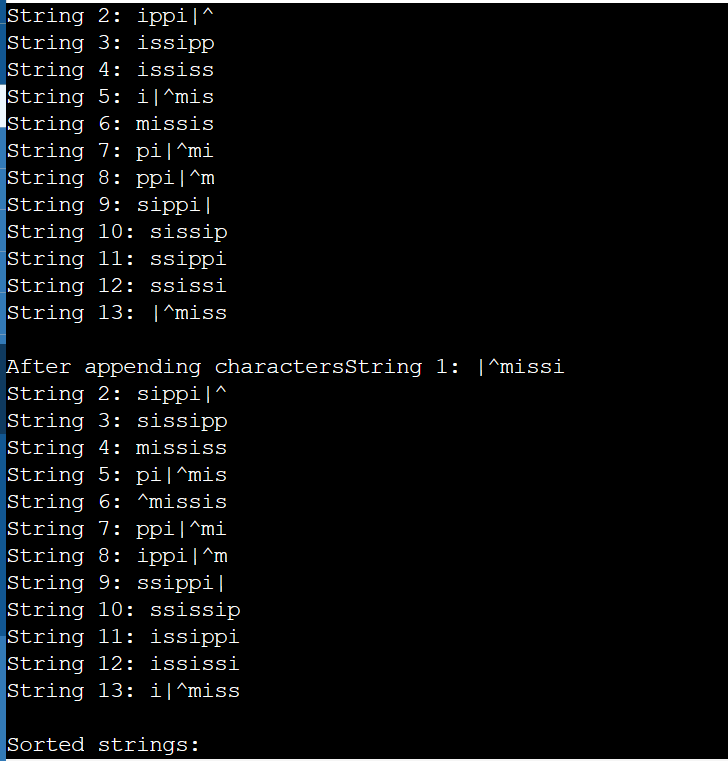
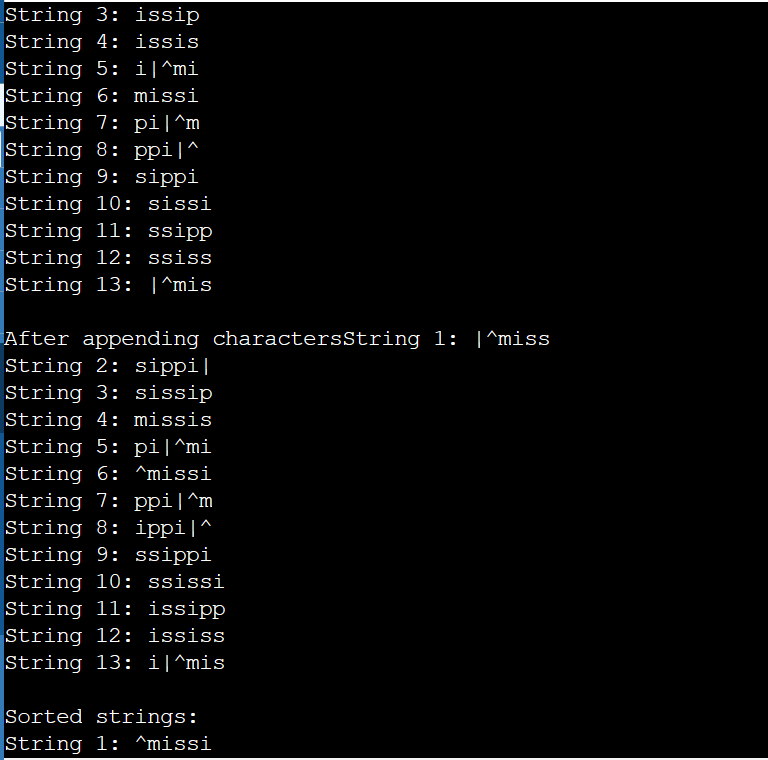
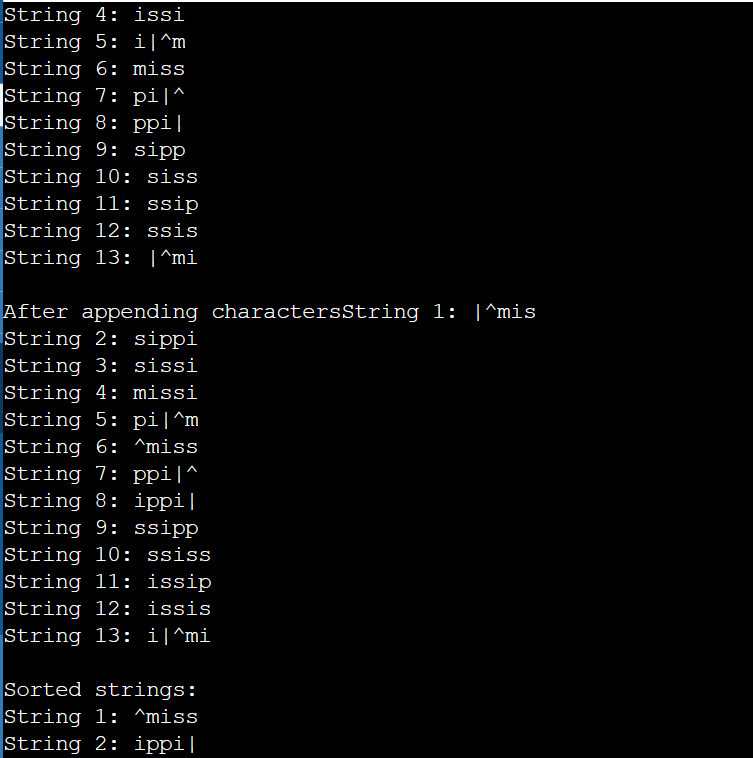
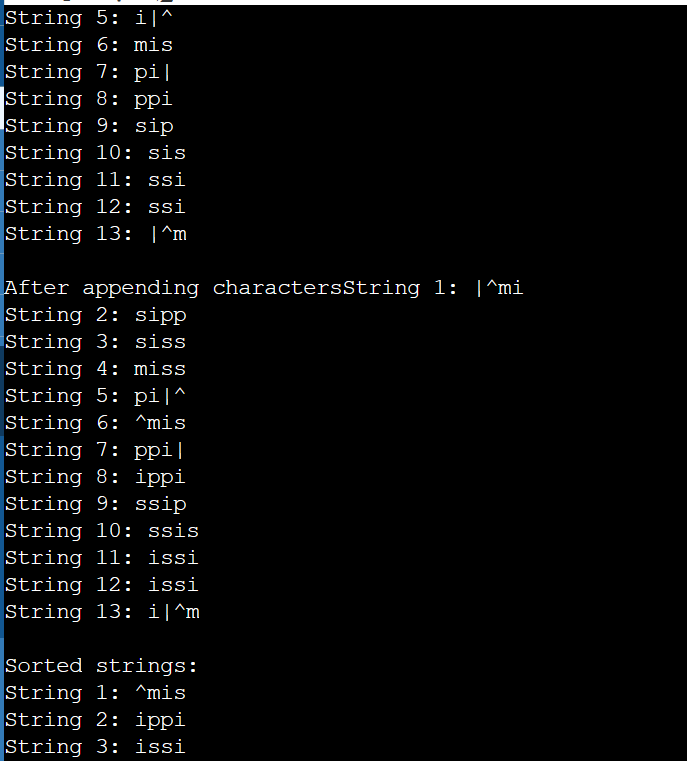
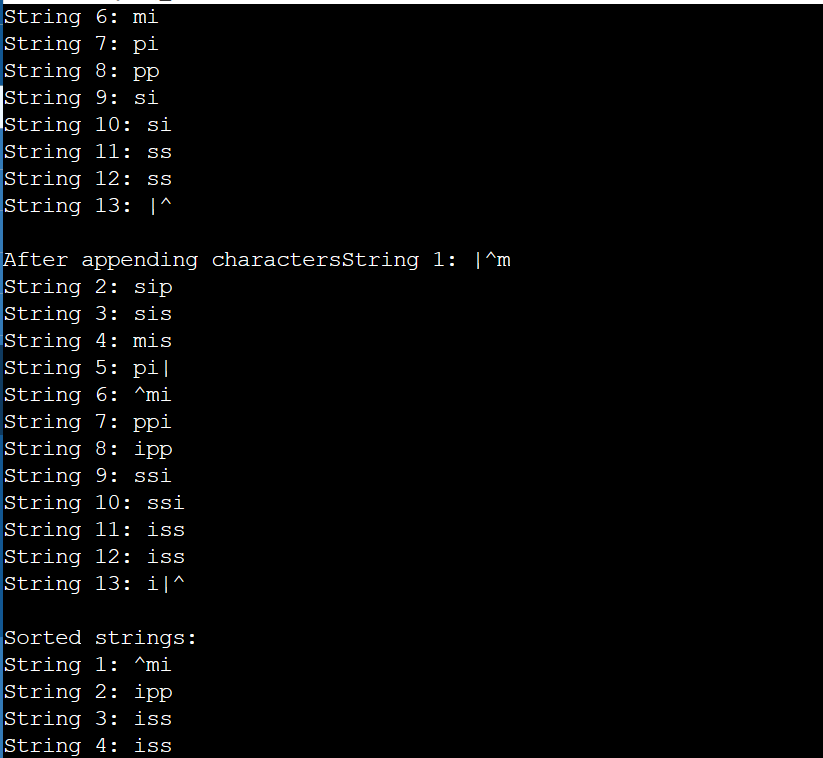
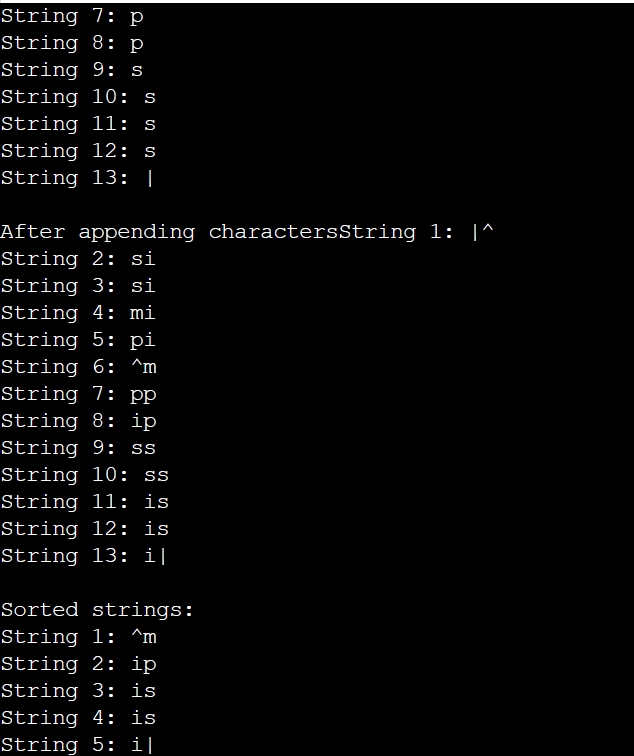
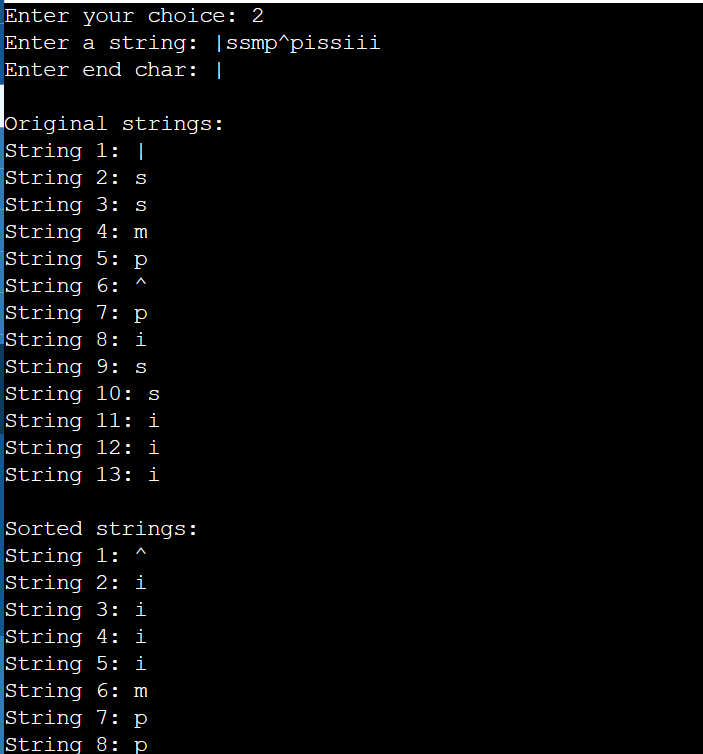


**S4: ^mississippi with end of file character |**

The number of sort and append operations: 7

**ENCODED STRING:**

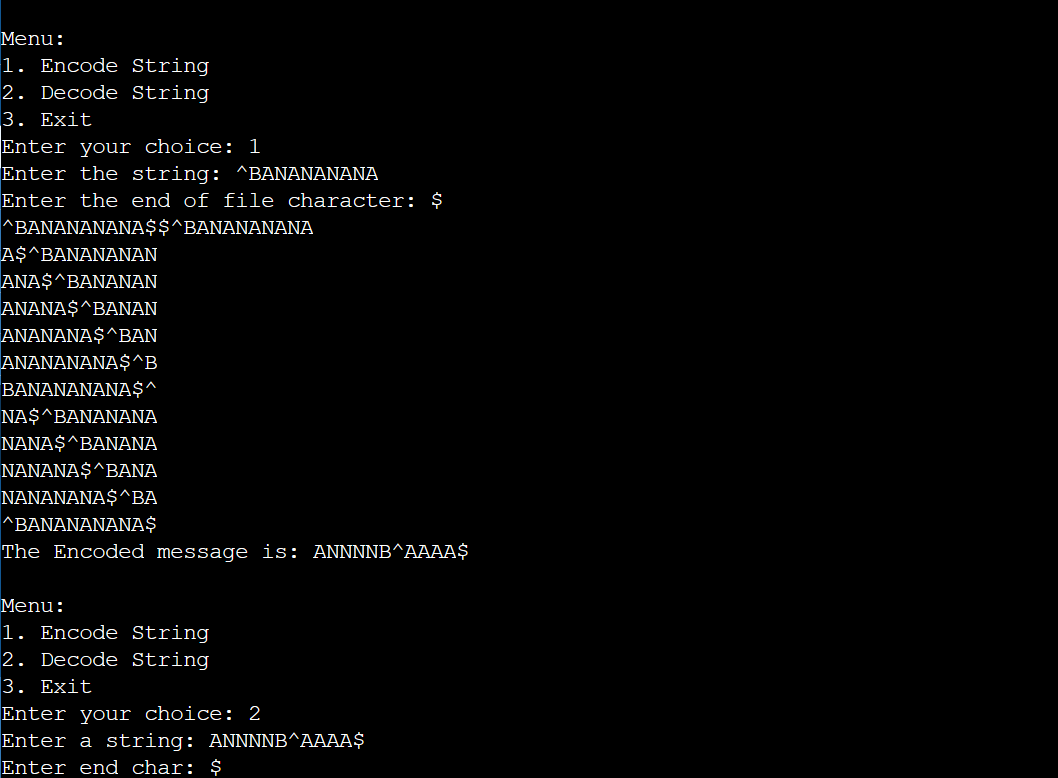


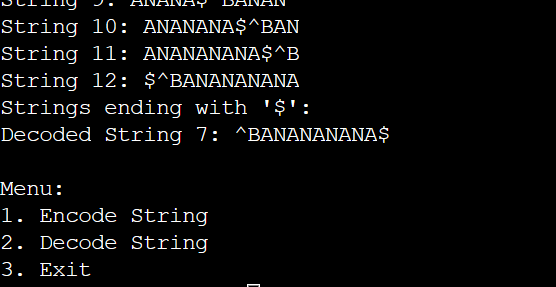
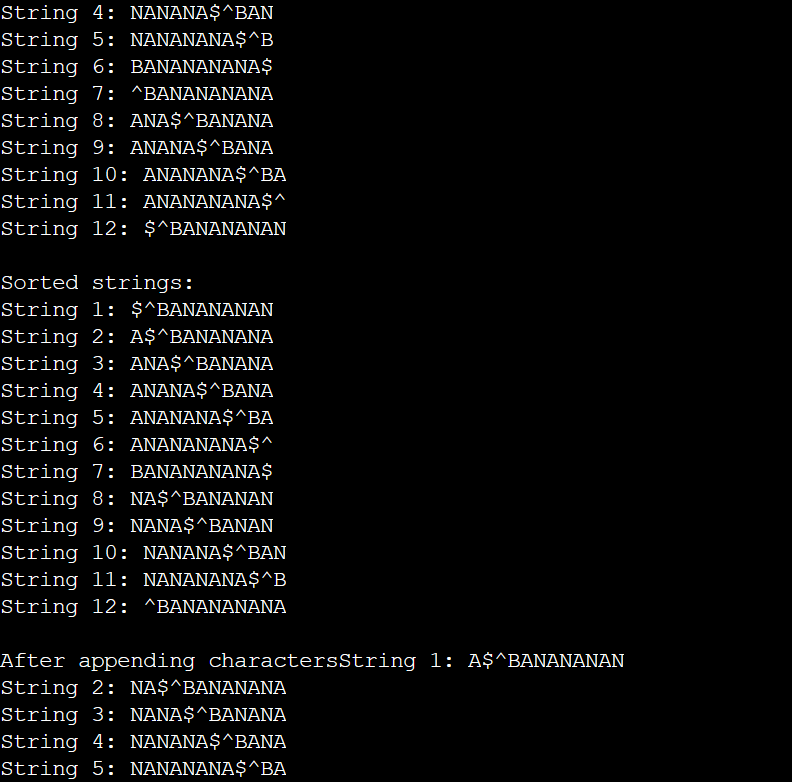
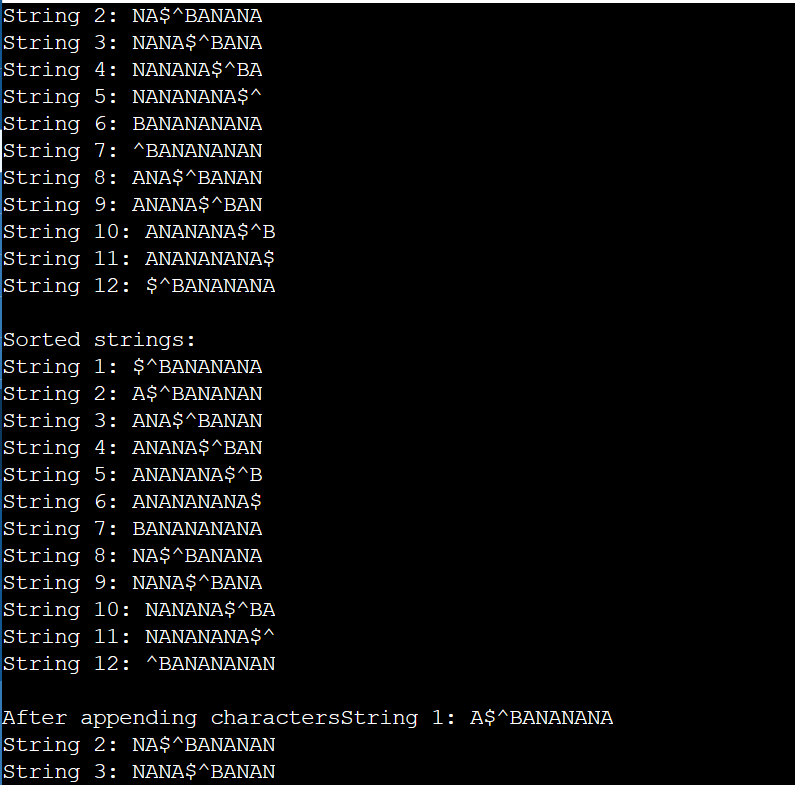
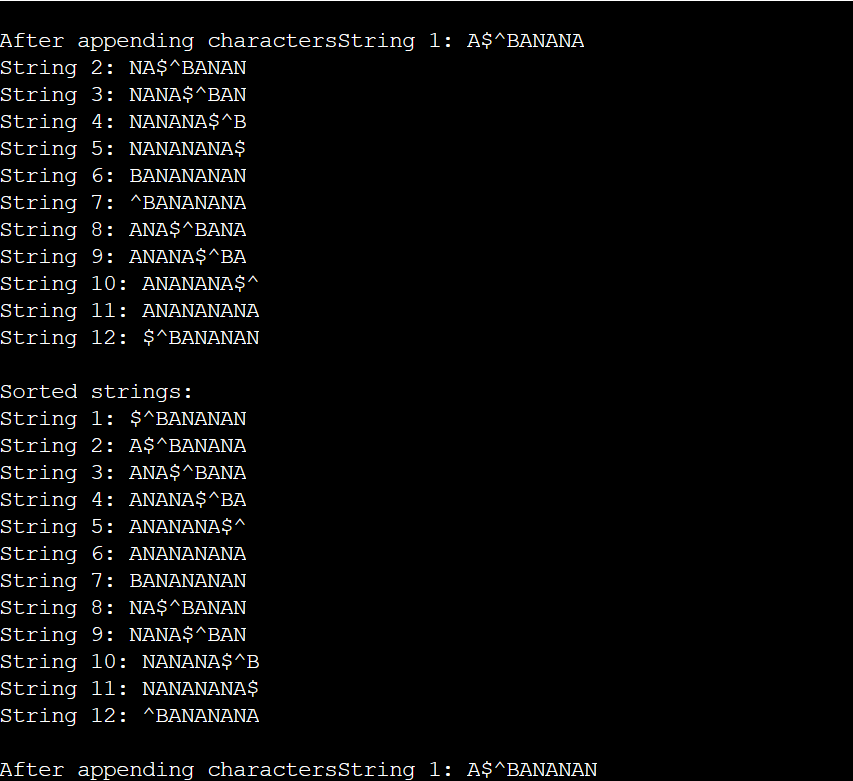
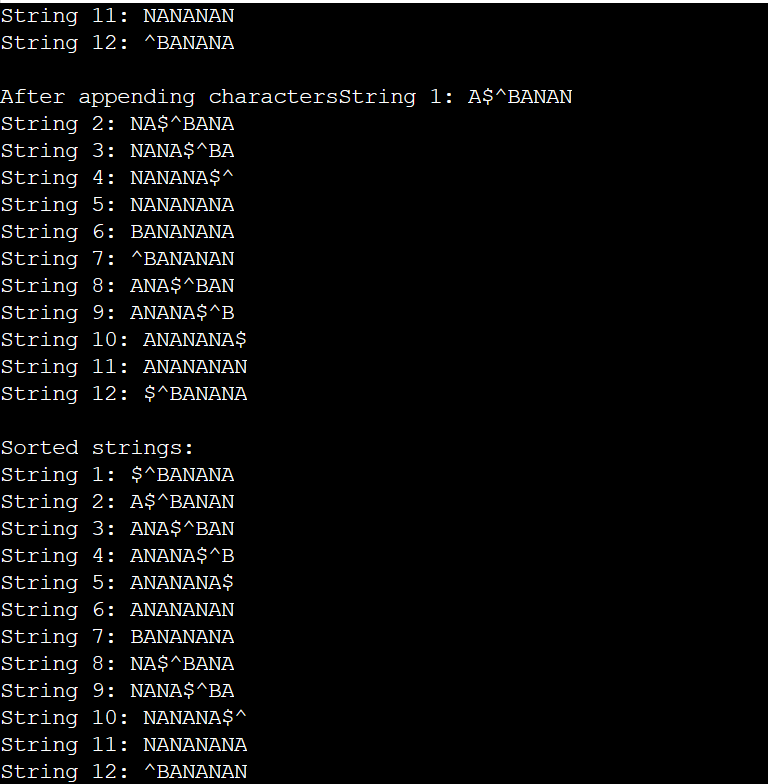
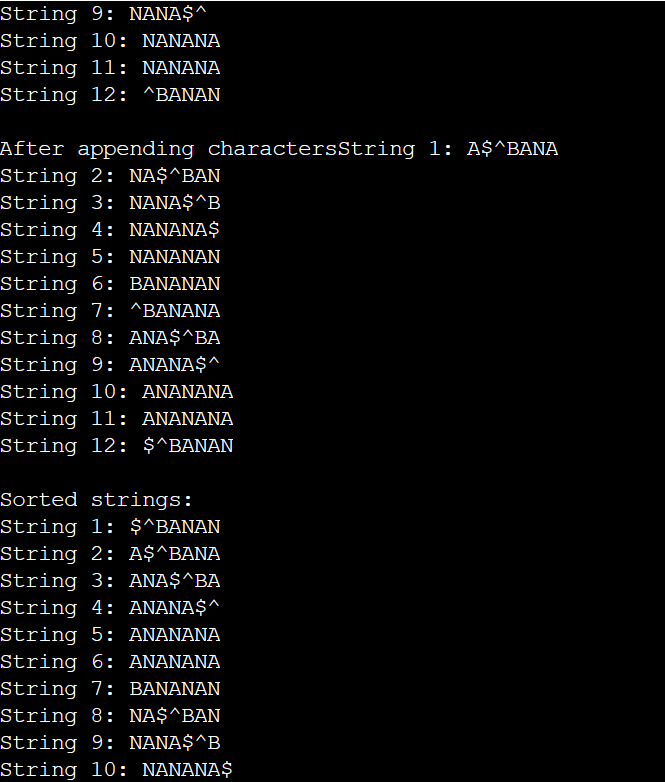
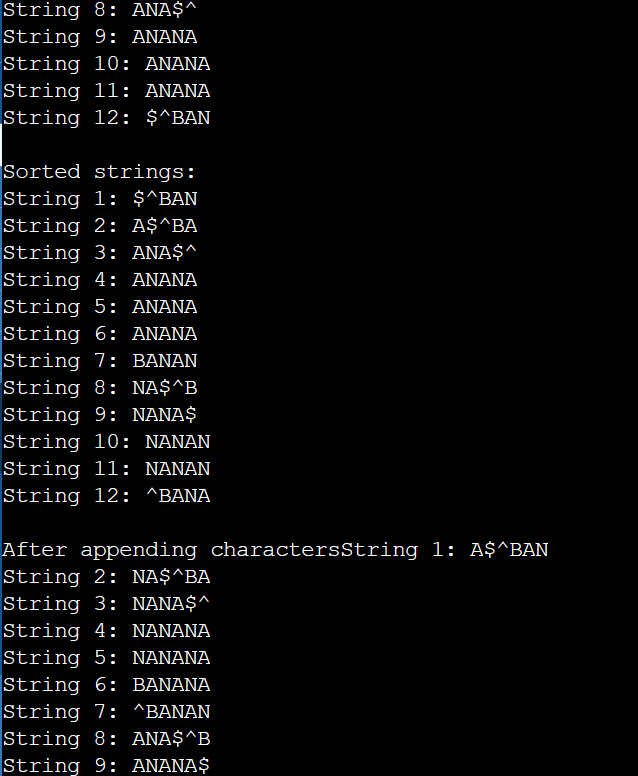
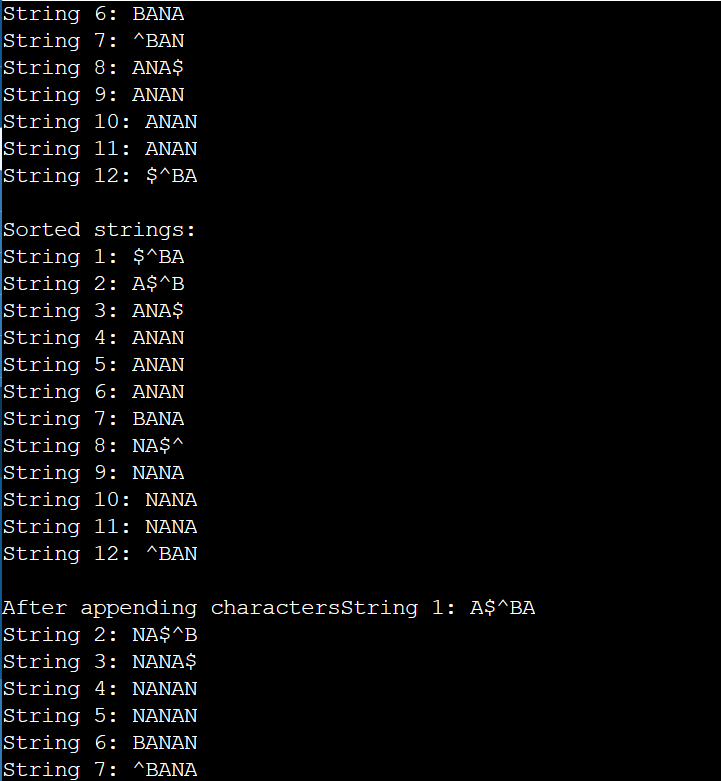
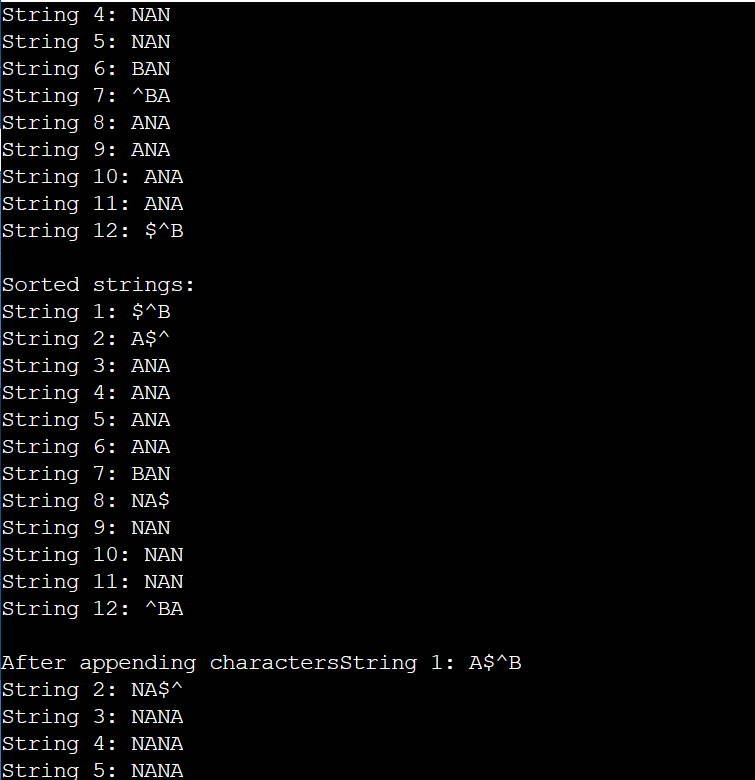
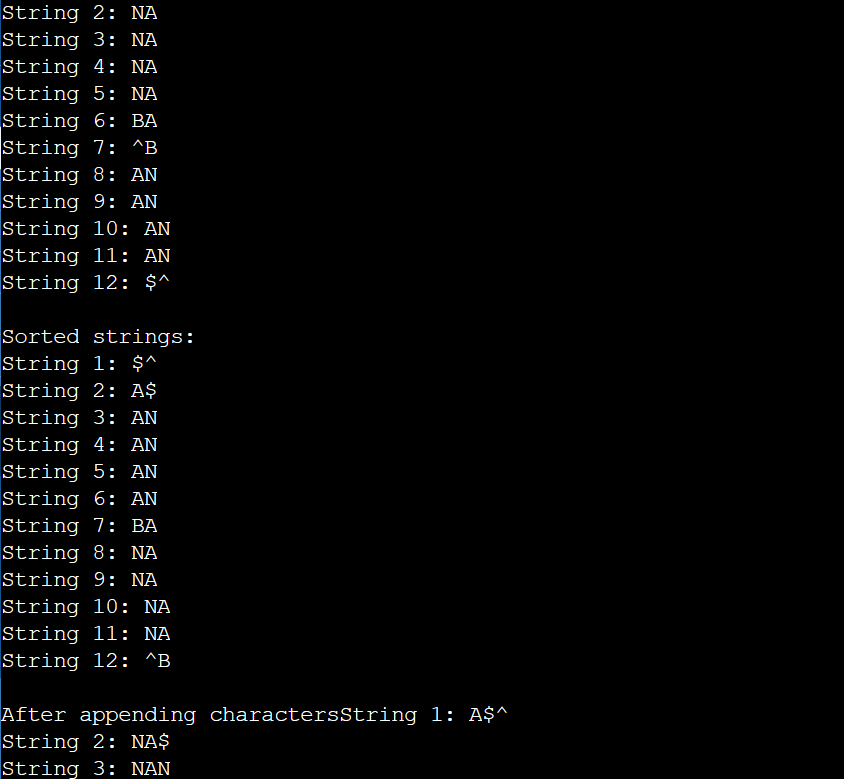
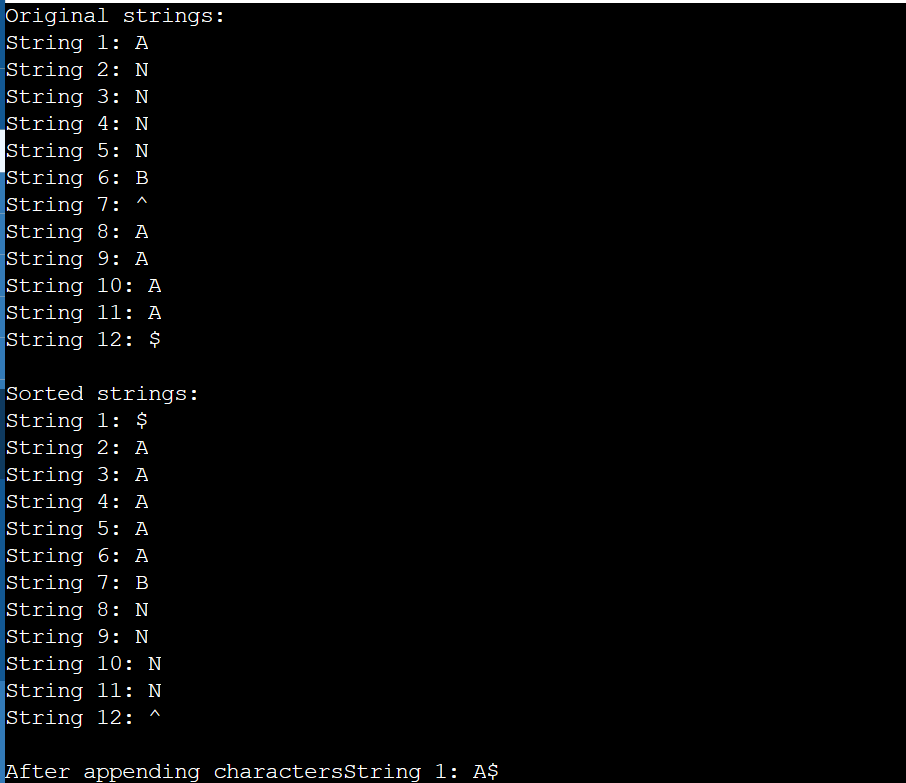


Appendix C: Strings for Testing Using BWT

S5: ^BANANANANA with end of file character $

The number of sort and append operations: 11

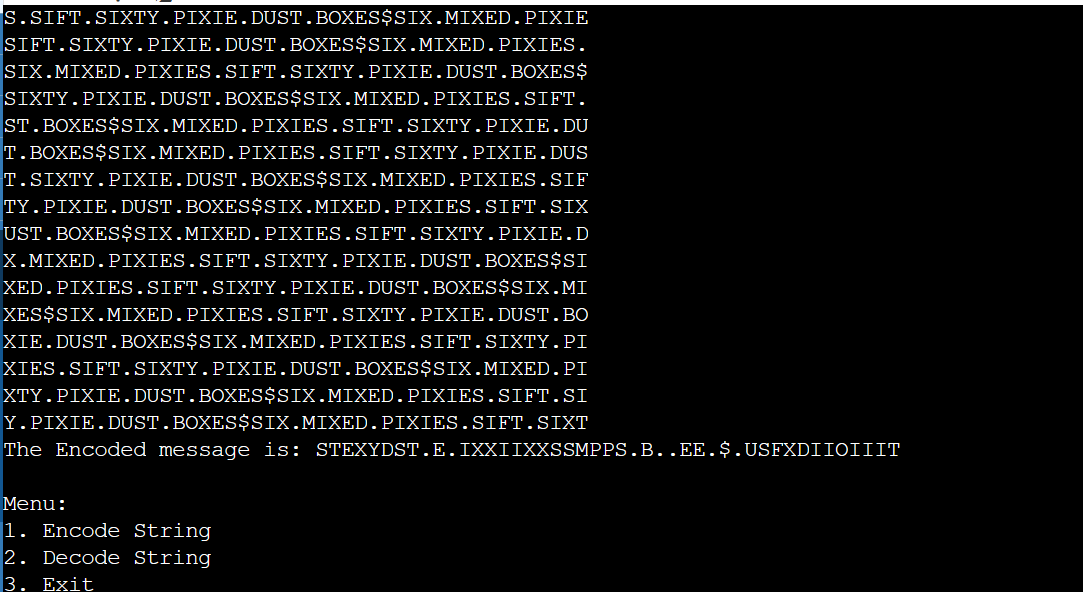




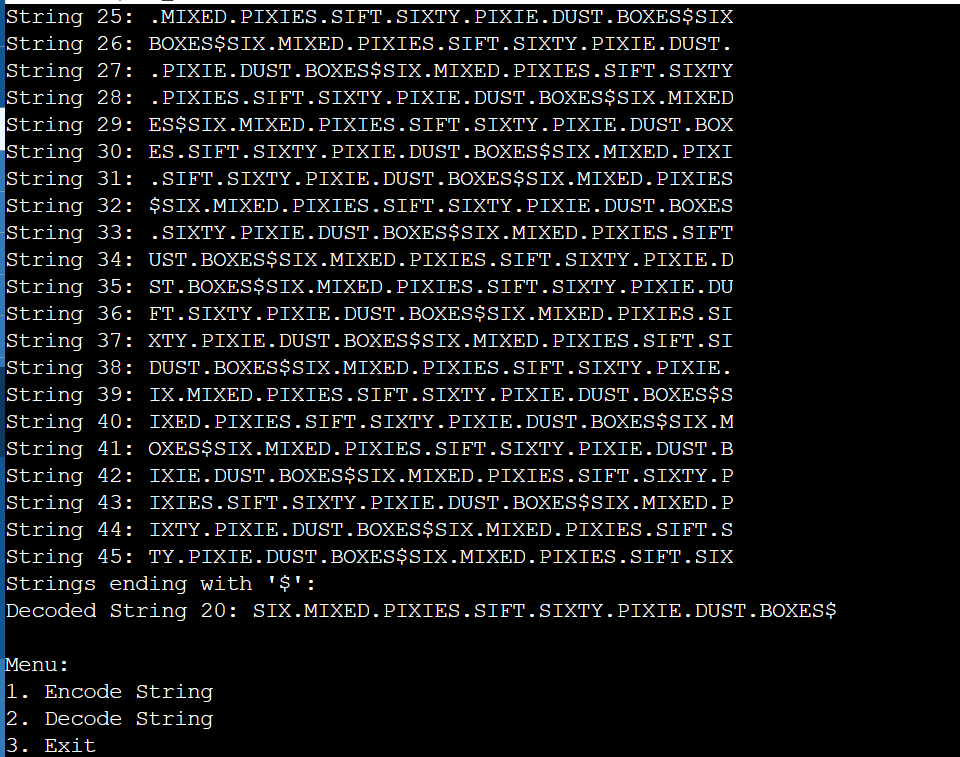
S6: SIX.MIXED.PIXIES.SIFT.SIXTY.PIXIE.DUST.BOXES with end of file character $

The number of sort and append operations: 44

**ENCODED STRING:**



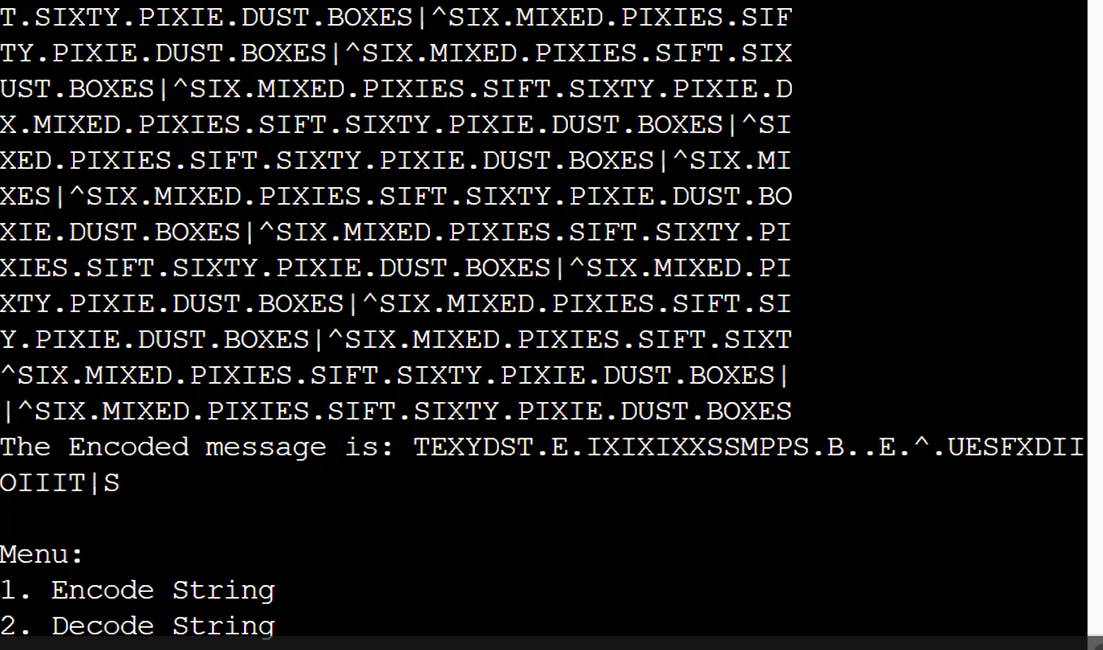
DECODED STRING:



S7: ^SIX.MIXED.PIXIES.SIFT.SIXTY.PIXIE.DUST.BOXES with end of file character |

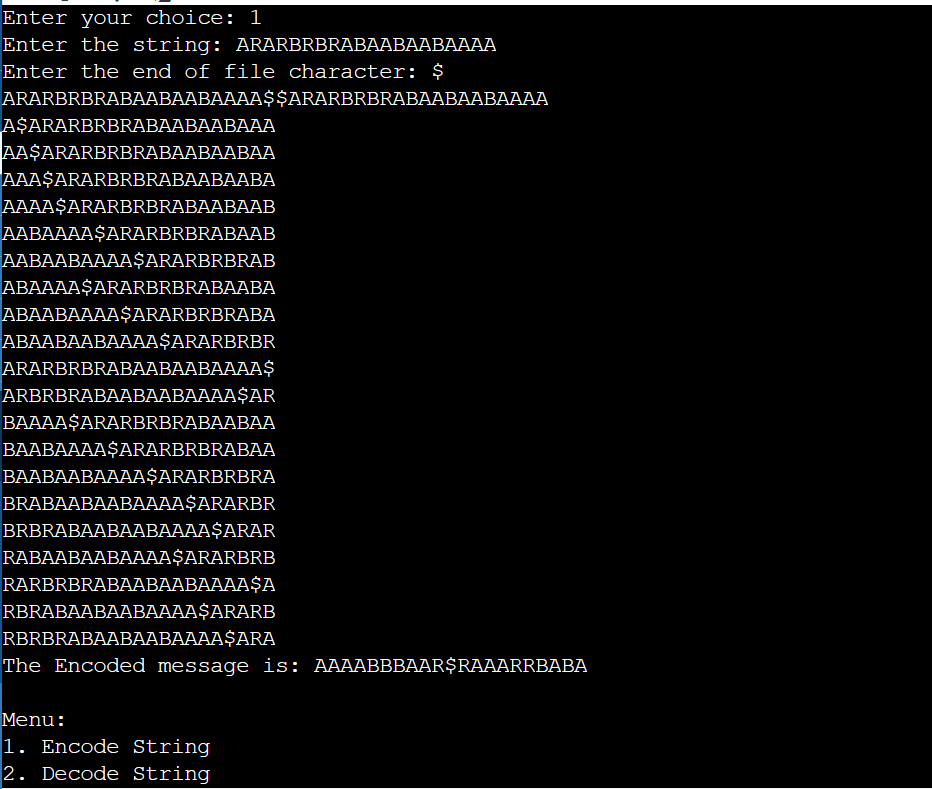
The number of sort and append operations: 45

**ENCODED STRING:**



S8: ARARBRBRBRABAABAABAAAA with end of file character $

The number of sort and append operations: 22



DECODED STRING:

