Ex.no:

Date: **CONCATENATION**

Aim:

To develop the Perl program to concatenate two sequences.

Algorithm:

Step 1: Start the program.

Step 2: Get two DNA sequences separately.

Step 3: Join the sequences.

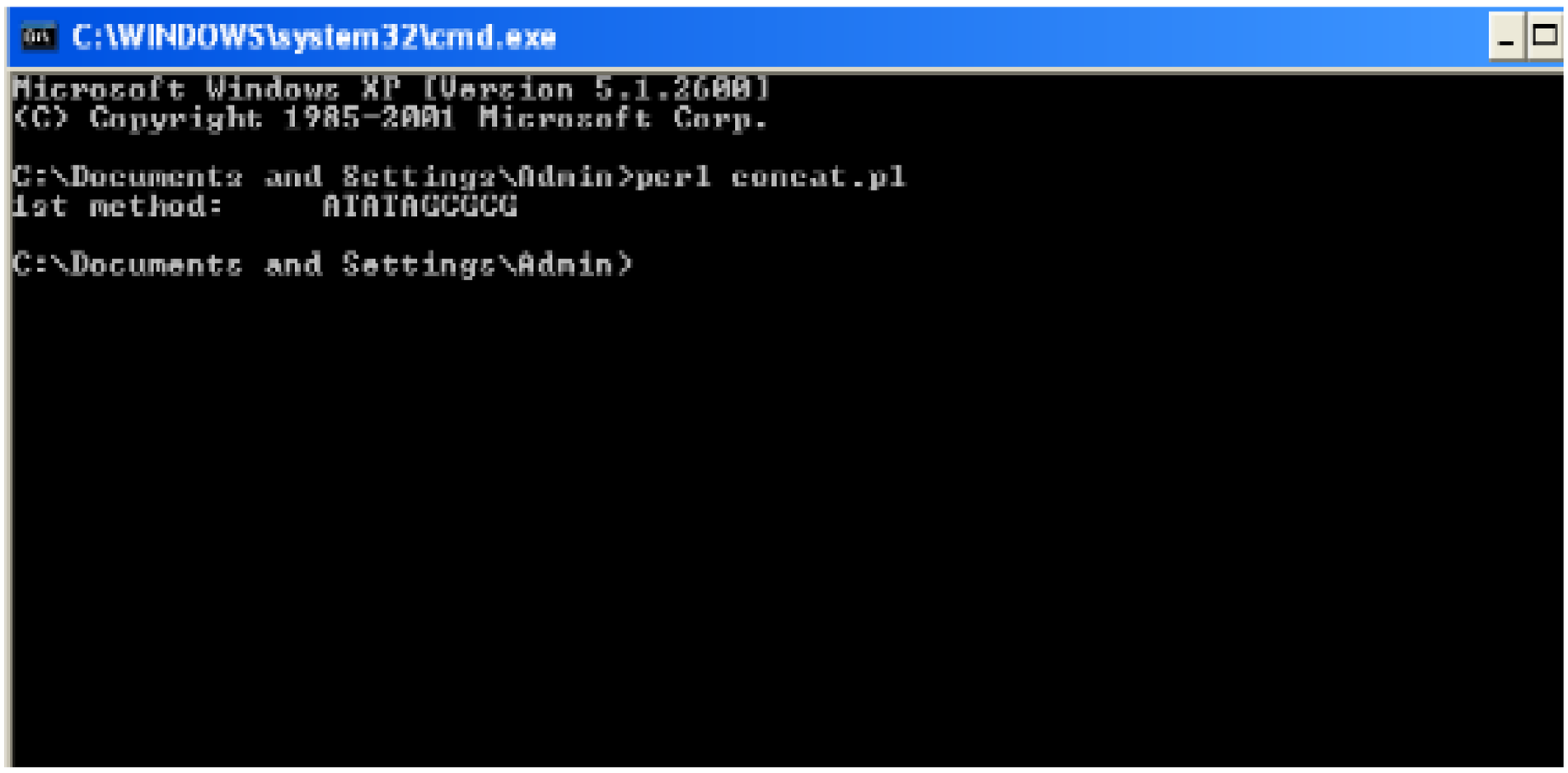
Step 4: Display the Result.

Step 5: End the program.

Program coding:

```
$seqA = "ATATA";
$seqB = "GCGCG";
print "1st method:\t", $seqA.$seqB, "\n";
$joinedSeq = "$seqA$seqB";
```

Result:



Output:

Ex.no:

Date: **DNA to RNA (Transcription)**

Aim:

To develop the Perl program to translate the given DNA sequence into RNA sequence.

Algorithm:

Step 1: Start the program

Step 2: Get input for DNA sequence

Step 3: Store the input sequence into the variable " DNA" .

Step 4: Transcript the input Sequence and Store it again in " DNA" .

Step 5: Translate the Transcriptsequence into RNA sequence and store it into ' RNA' .

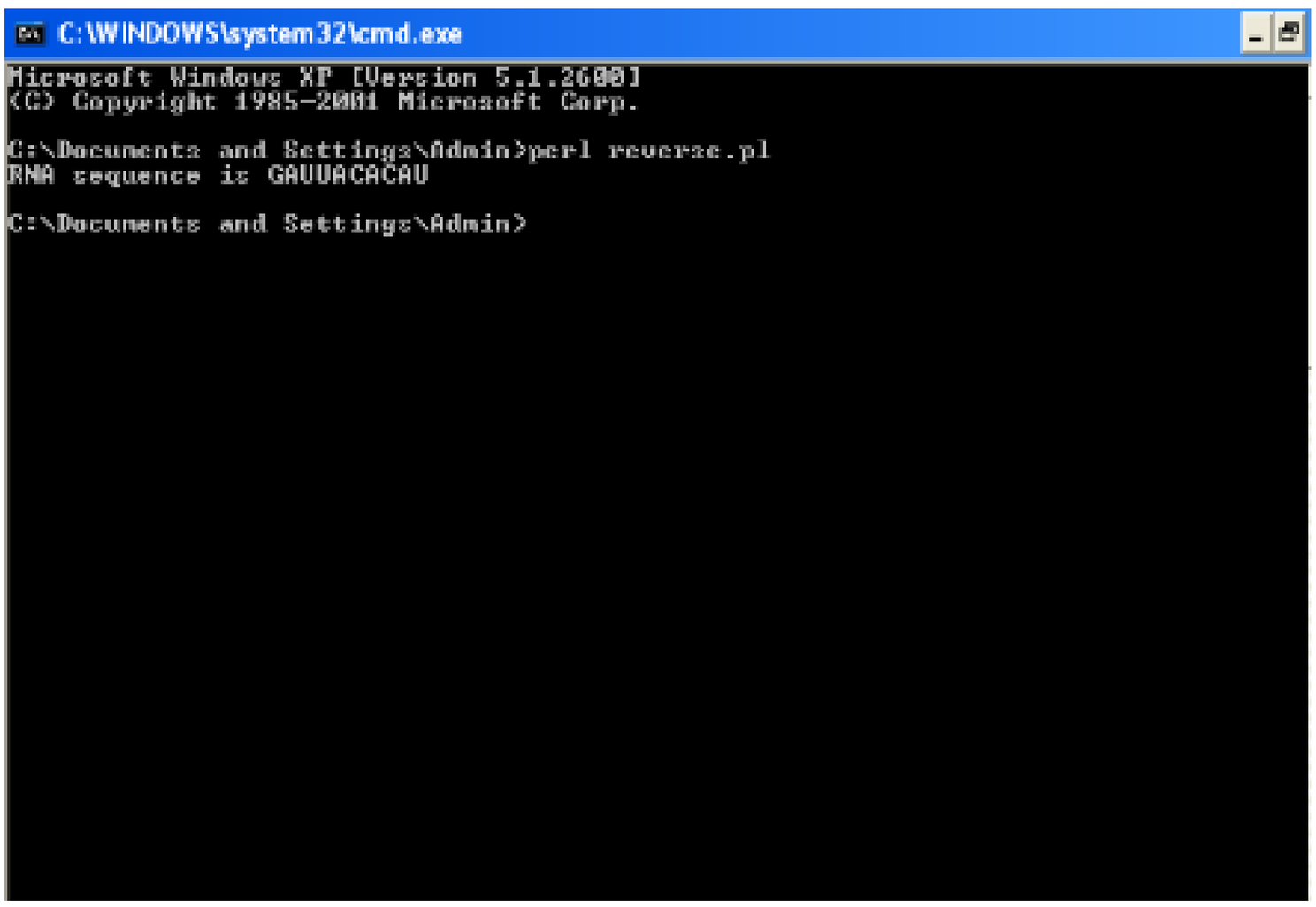
Step 6: Display the Result.

Step 7: End the program.

Program coding:

```
my $DNA = "GATTACACAT";
my $RNA = $DNA;
$RNA =~ s/T/U/g;
print "RNA sequence is $RNA\n";
```

Result:



Output:

Ex.no:

Date: **COMPLEMENT**

Aim:

To develop the Perl program into complement for the given DNA sequence.

Algorithm:

Step 1: Start the program

Step 2: Get input for DNA sequence

Step 3: Store the input sequence into the variable " DNA" .

Step 4: The input Sequence store again in " DNA" .

Step 5: Given sequences are print as complement.

Step 6: Display the complement DNA.

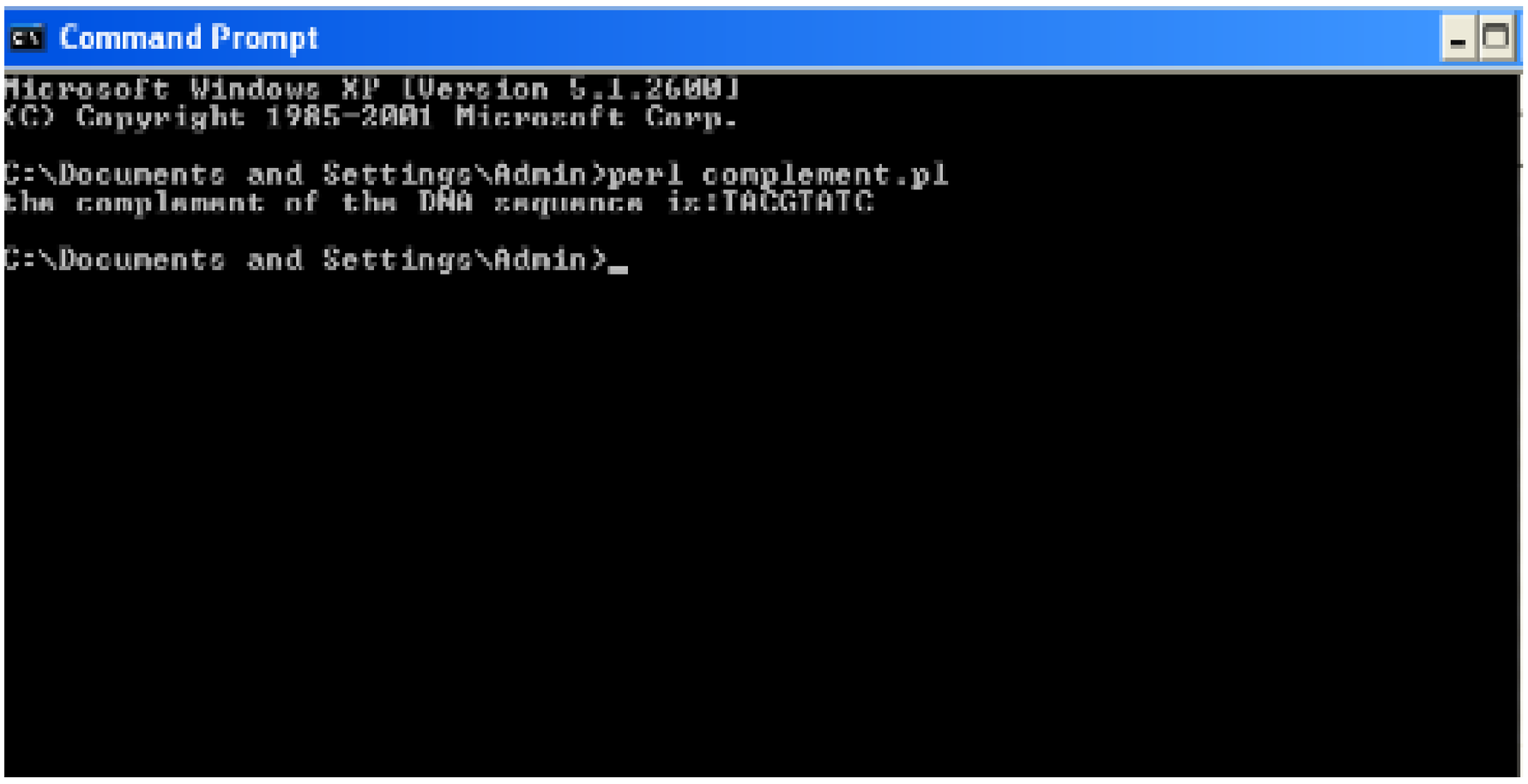
Step 7: Display the Result.

Step 8: End the program.

Program coding:

```
 my $DNA =’  GATTCACAT’  ;
$DNA =~tr/ATGC/TACG/;
print”  complement of DNA sequence is $DNA\n”  ;
```

Result:



Output:

Ex.no:

Date: REVERSE COMPLEMENT

Aim:

To develop the Perl program into reverseand its complement for the given DNA sequence.

Algorithm:

Step 1: Start the program

Step 2: Get input for DNA sequence

Step 3: Store the input sequence into the variable " DNA" .

Step 4: Reverse the input Sequence and Store it again in " DNA" .

Step 5: Translate the reverse of the DNA sequence.

Step 6: Convert the reverse DNA sequence in to its complement sequence.

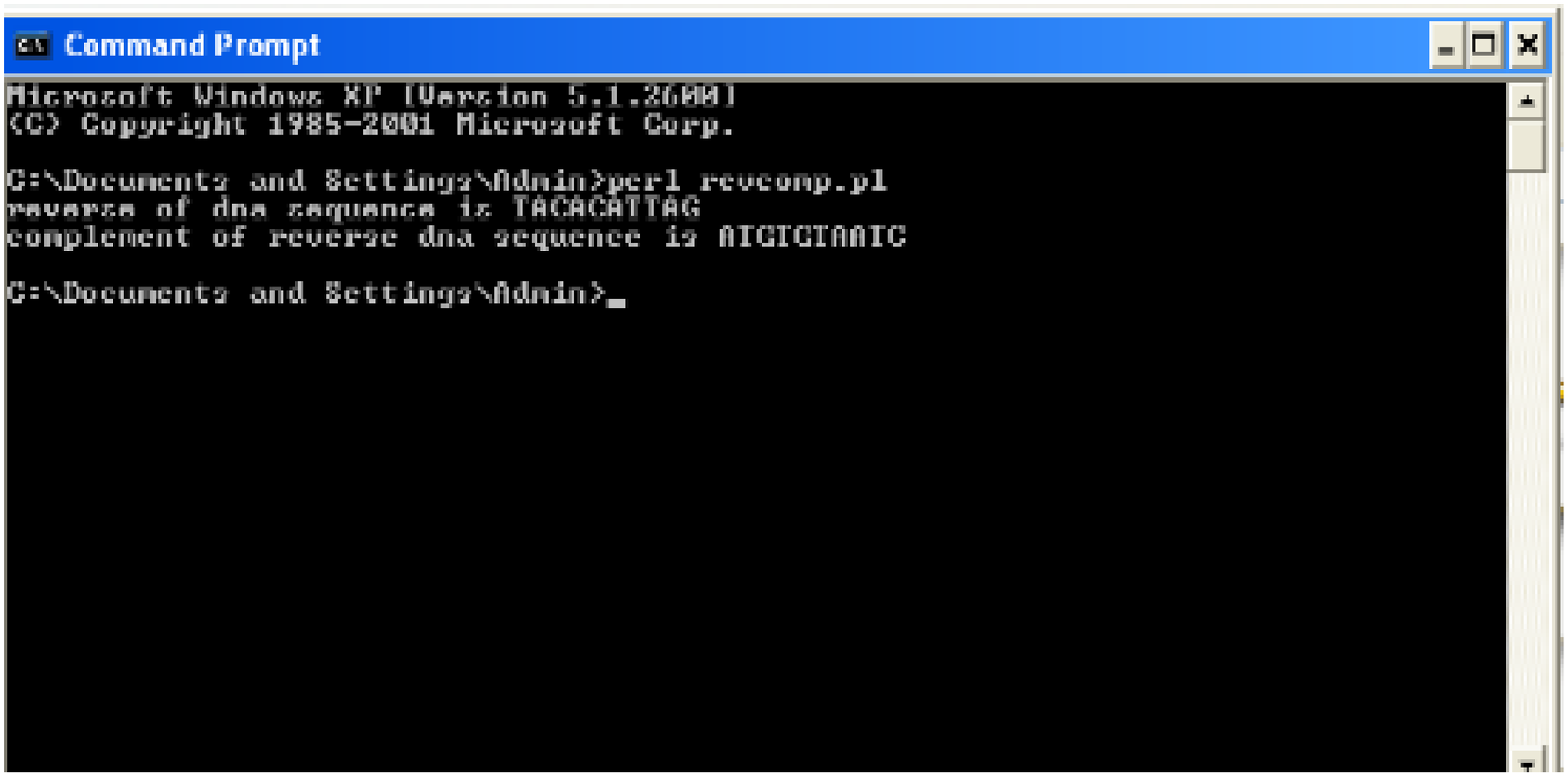
Step 7: Display the Result.

Step 8: End the program.

Program coding:

```
my $DNA = "GATTACACAT";
$rev=reverse ($DNA);
print "reverse of dna sequence is $rev\n";
$rev=~ tr/ATGC/TACG/;
print "complement of reverse dna sequence is $rev\n";
```

Result:



Output:

Ex.no:

Date: **TO CREATE SEQUENCE**

Aim:

To develop the Perl program to create sequence of DNA in the string.

Algorithm:

Step 1: Start the program

Step 2: Get input for DNA sequence

Step 3: Store the input sequence into the variable " DNA" .

Step 4: Create the input Sequence and Store it again in " DNA" .

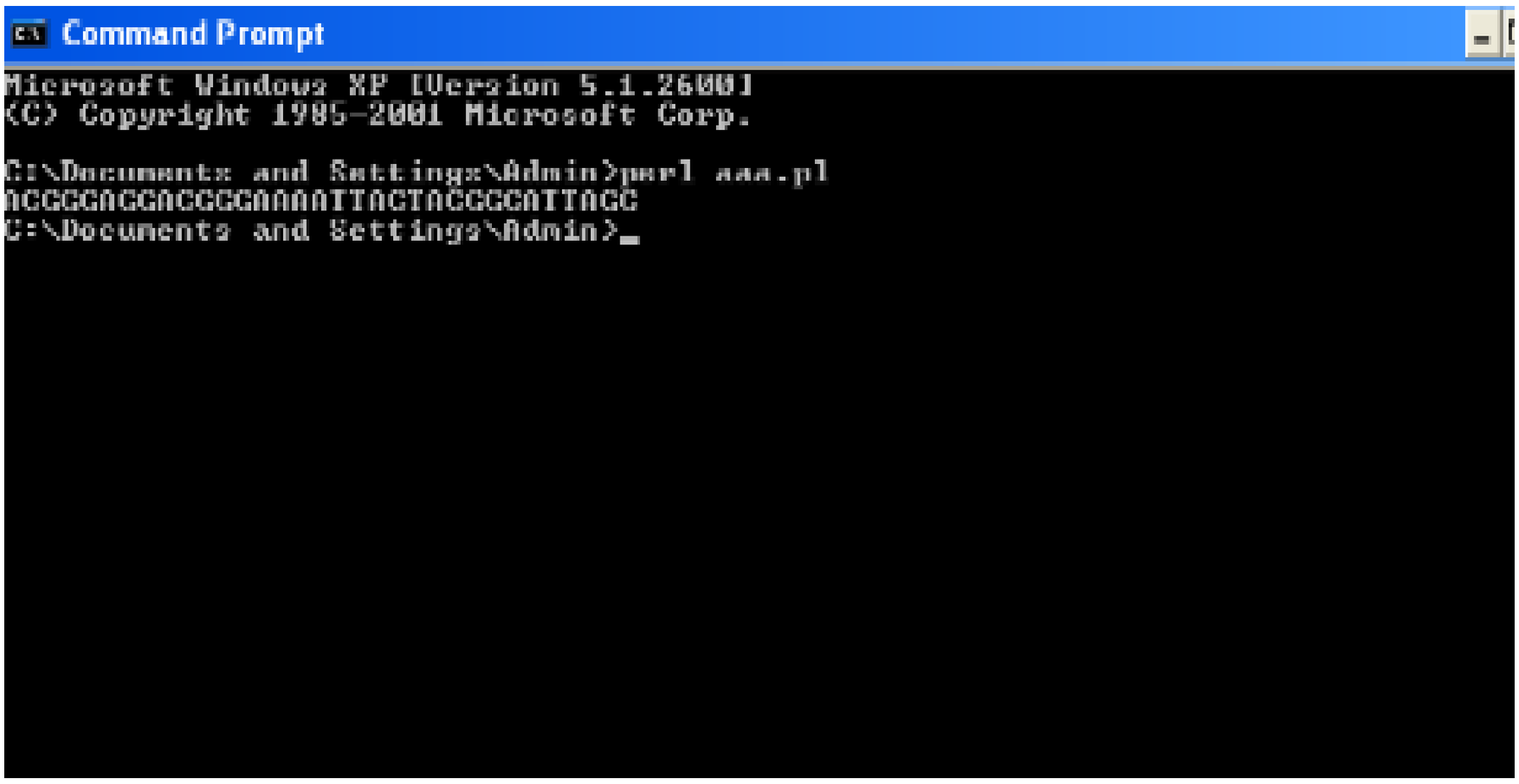
Step 5: Display the Result.

Step 6: End the program.

Program coding:

```
$DNA = ’   ACGGAGGACGGGAAAATTACTACGGCATTAGC’   ;
print $DNA;
```

Result:



Output:

Ex.no:

Date: **STRING LENGTH**

Aim:

To develop the Perl program to calculate the given string length of the DNA sequence.

Algorithm:

Step 1: Start the program

Step 2: Get input for DNA sequence

Step 3: Store the input sequence into the variable " CAPITAL" .

Step 4: The input Sequence store it in string_ len.

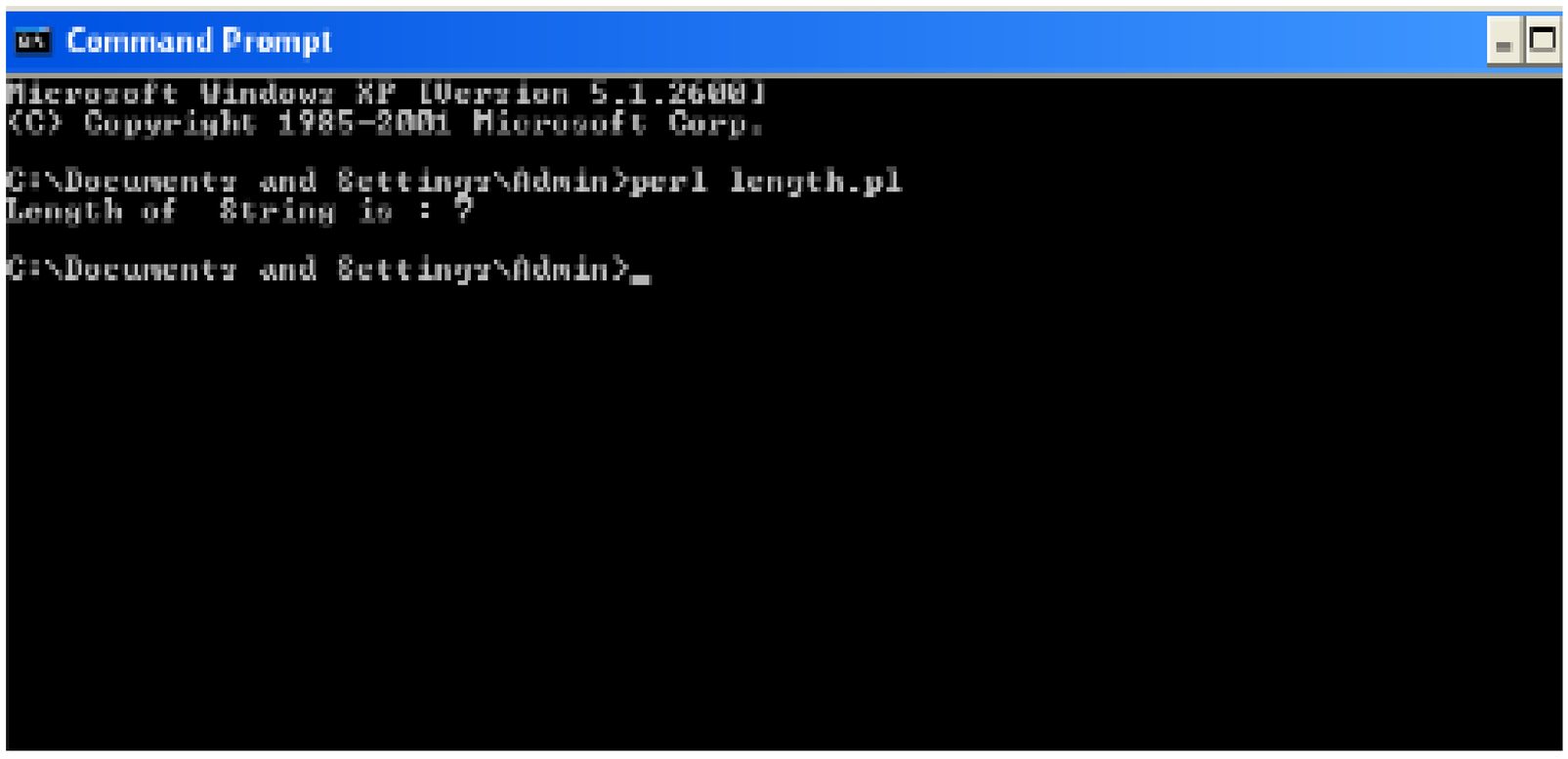
Step 5: Display the Result.

Step 6: End the program.

Program coding:

```
$orig_string ="CAPITAL";
$string_len =length( $orig_string );
print "Length of  String is : $string_len\n";
```

Result:



Output:

Ex.no:

Date: **TO REMOVE LAST CHARACTER**

Aim:

To develop the Perl program to remove the last character in the given sequence.

Algorithm:

Step 1: Start the program.

Step 2: Get the input characters.

Step 3: Retrieve the input Sequence and Store it again in string1.

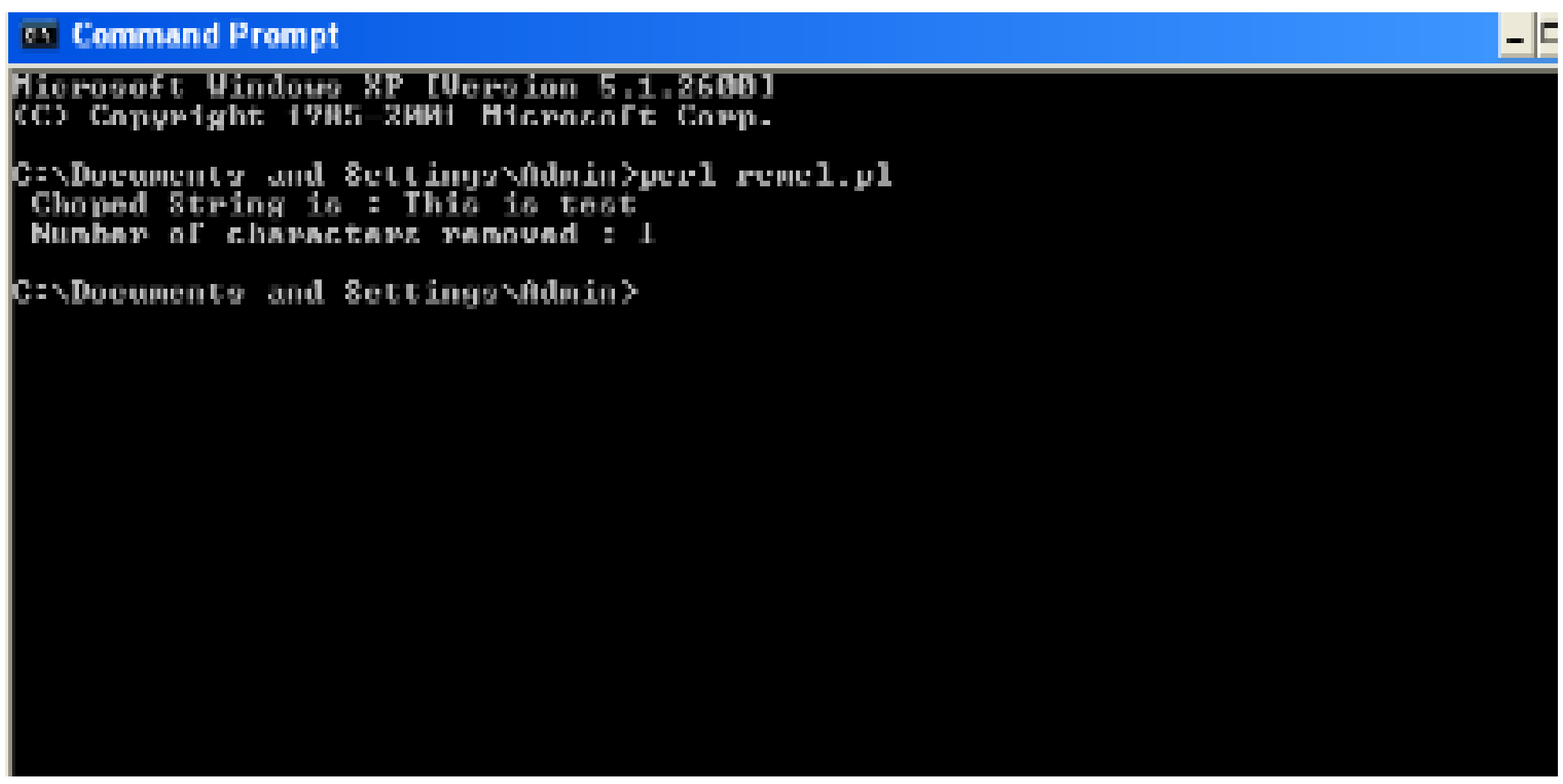
Step 4: Remove the last character by using chop and display the result.

Step 5: End the program.

Program coding:

```
$string1 = "This is test";
$retval  = chop( $string1 );
print " Choped String is : $string1\n";
print " Character removed : $retval\n";
```

Result:



Output:

Ex.no:

Date: HOW TO REMOVE THE LAST LETTER

Aim:

To develop the Perl program to remove the last letter in the given characters.

Algorithm:

Step 1: Start the program

Step 2: Get the input sequence

Step 3: Store the input sequence into the variable.

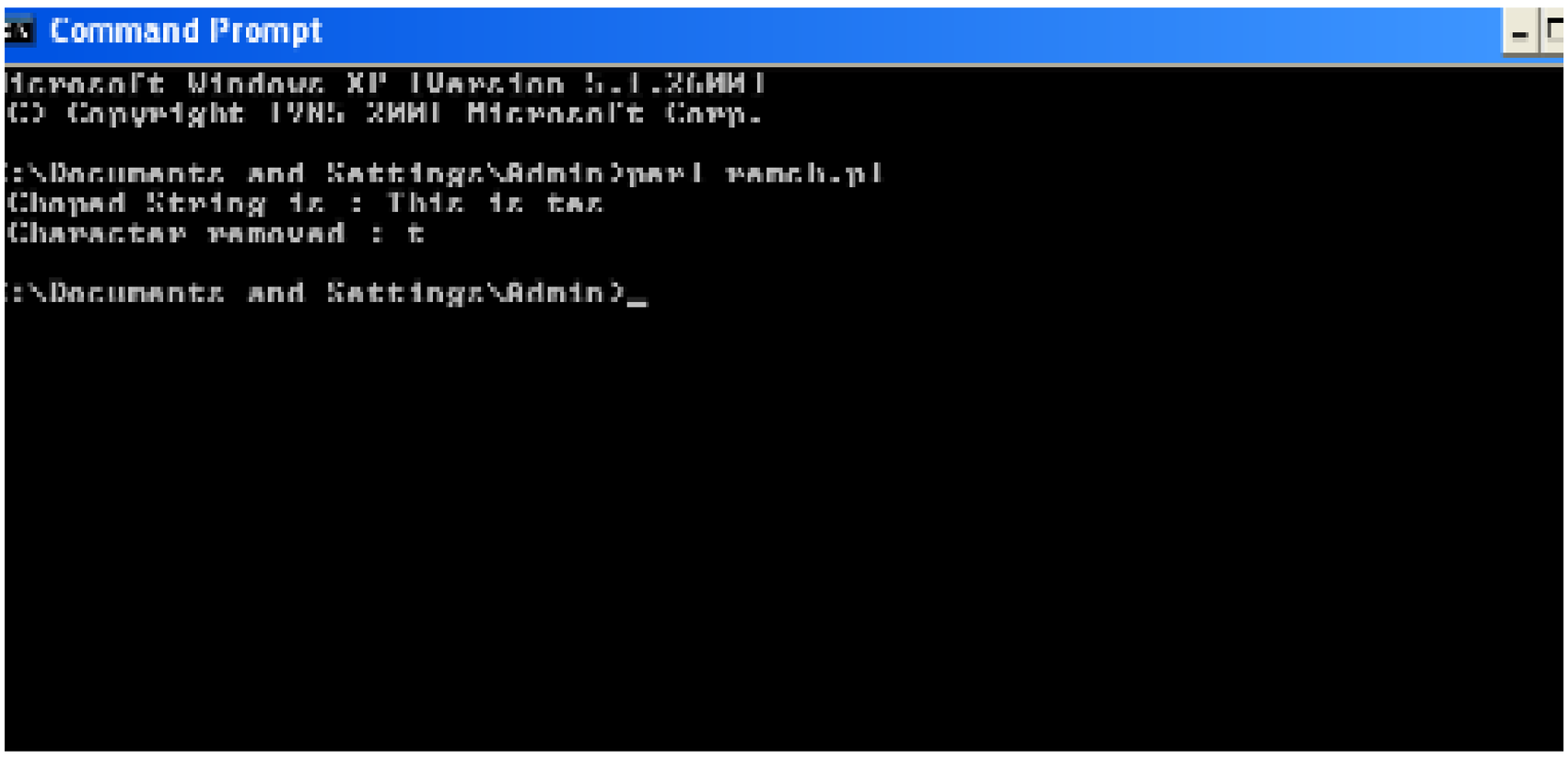
Step 4: Remove the last character by using chomp and display the result.

Step 5: End the program.

Program coding:

```
$string1 = "This is test\n";

$retval  = chomp( $string1 );

print " Choped String is : $string1\n";

print " Number of characters removed : $retval\n";
```

Result:



Output:

Ex.no:

Date: **HOW TO OPEN A FILE**

Aim:

To develop the Perl program to open a file.

Algorithm:

Step 1: Start the program

Step 2: Store a file contains DNA sequence.

Step 3: Give the input Sequence and store it again on the same file.

Step 4: Save the sequence file name as ” sasi.pep” .

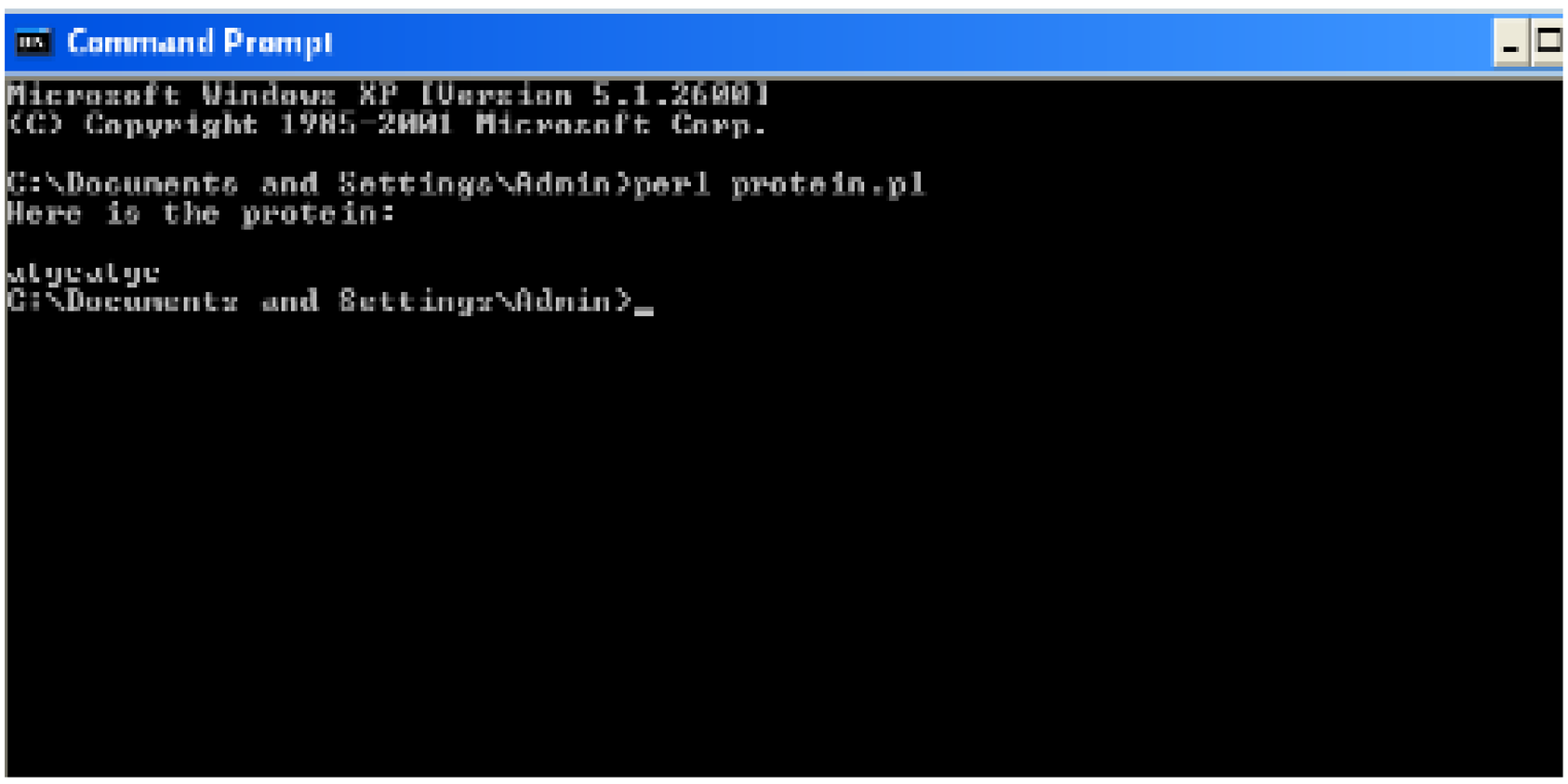
Step 5:Use open file option and display the Result.

Step 6: End the program.

Program coding:

```
$proteinfilename = 'sasi.pep';

open(PROTEINFILE, $proteinfilename);

$protein = <PROTEINFILE>;

print "Here is the protein:\n\n";

print $protein;
```

Result:



Output:

Ex.no:

Date: **SUBROUTINE**

Aim:

To develop the perl program to use add the new sequence by using subroutine for the given DNA sequence.

Algorithm:

Step 1: Start the program

Step 2: Enter the sequence.

Step 3: Input the subroutine to develop.

Step 5: Add new sequence " ATGC" .

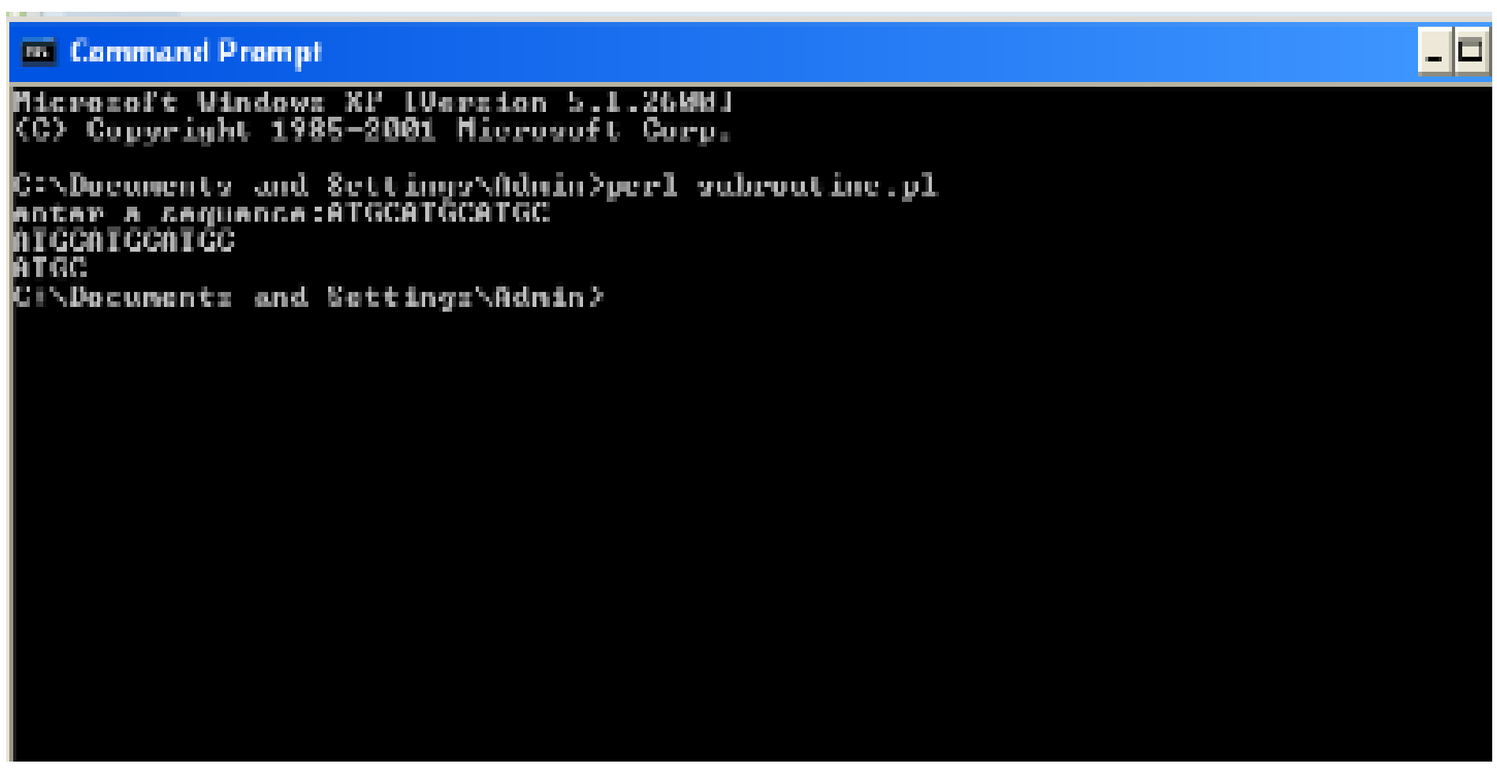
Step 6: Display the Result.

Step 7: End the program.

Program coding:

```
print"enter a sequence:";
$DNA=<STDIN>;
$newseq=addseq($DNA);
sub addseq
{
$DNA.='ATGC';
return$DNA;
}
print"$newseq";
```

Result:



Output:

Ex.no:

Date:

GC CONTENT CALCULATION

Aim:

To develop a Perl program to calculate the GC content in a DNA sequence.

Algorithm:

Step 1: Start the program.

Step 2: Enter the DNA sequence.

Step 3: Count the occurrences of "G" and "C" in the sequence.

Step 4: Calculate the GC content percentage by dividing the total "G" and "C" count by the length of the sequence, then multiply by 100.

Step 5: Display the GC content percentage.

Step 6: End the program.

Program coding:

```
print "Enter the DNA sequence:\n";

my $DNA = <STDIN>;

chomp($DNA);


my $G_count = ($DNA =~ tr/G/G/);

my $C_count = ($DNA =~ tr/C/C/);

my $GC_content = (($G_count + $C_count) / length($DNA)) * 100;

print "GC content percentage is: $GC_content%\n";
```

Result:

Output:

Ex.no:

Date:

COUNTING SUBSTRING OCCURRENCES

Aim:

To develop a Perl program to find the number of times a specific motif appears in a DNA sequence.

Algorithm:

Step 1: Start the program.

Step 2: Enter the DNA sequence.

Step 3: Enter the motif to search for within the DNA sequence.

Step 4: Use a loop or regex to count the number of occurrences of the motif in the sequence.

Step 5: Display the count of motif occurrences.

Step 6: End the program.

Program coding:

```
print "Enter the DNA sequence:\n";

my $DNA = <STDIN>;

chomp($DNA);


print "Enter the motif to search for:\n";

my $motif = <STDIN>;
```

```
chomp($motif);

my $count = () = $DNA =~ /$motif/g;
print "The motif '$motif' appears $count times in the DNA sequence.\n";
```

Result:

Output:

Ex.no:
Date:

IDENTIFYING PALINDROMIC SEQUENCES

Aim:
To develop a Perl program to identify palindromic sequences in a DNA string.

Algorithm:

Step 1: Start the program.

Step 2: Enter the DNA sequence.

Step 3: Define a minimum length for palindromic sequences to search for.

Step 4: Use a loop to examine each substring of the DNA sequence.

Step 5: For each substring, check if it reads the same forwards and backwards.

Step 6: If a palindromic sequence is found, display it.

Step 7: End the program.

Program coding:

```
print "Enter the DNA sequence:\n";
my $DNA = <STDIN>;
chomp($DNA);
```

```
my $min_length = 4; # Set minimum length for palindromic sequences

for my $start (0 .. length($DNA) - $min_length) {
   for my $len ($min_length .. length($DNA) - $start) {

      my $substring = substr($DNA, $start, $len);

      if ($substring eq reverse($substring)) {

         print "Found palindromic sequence: $substring\n";

      }

   }

}
```

Result:

Output:

Ex.no:
Date:

TRANSLATING DNA TO PROTEIN (Using Partial Codon Table)

Aim:
To develop a Perl program to translate a DNA sequence into an amino acid sequence, focusing on the most common codons.

Algorithm:

Step 1: Start the program.

Step 2: Enter the DNA sequence.

Step 3: Define a hash table with a partial codon-to-amino acid mapping for key amino acids (e.g., Methionine, Leucine, Serine).

Step 4: Split the DNA sequence into codons.

Step 5: Use the hash table to translate each codon into an amino acid and concatenate them to form the protein sequence.

Step 6: Display the resulting protein sequence.

Step 7: End the program.

Program coding:

```
print "Enter the DNA sequence:\n";

my $DNA = <STDIN>;

chomp($DNA);


# Partial codon to amino acid mapping for common amino acids

my %codon_table = (

    'ATG' => 'M', # Methionine (Start)

    'TGG' => 'W', # Tryptophan

    'TAA' => '*', 'TAG' => '*', 'TGA' => '*', # Stop codons

    'TTT' => 'F', 'TTC' => 'F', # Phenylalanine

    'GTT' => 'V', 'GTC' => 'V', 'GTA' => 'V', 'GTG' => 'V', # Valine

    'GCT' => 'A', 'GCC' => 'A', 'GCA' => 'A', 'GCG' => 'A'  # Alanine

);


# Translate DNA to protein using partial codon table

my $protein = '';

for (my $i = 0; $i < length($DNA) - 2; $i += 3) {

    my $codon = substr($DNA, $i, 3);

    $protein .= exists $codon_table{$codon} ? $codon_table{$codon} : 'X';

}


print "The protein sequence (partial translation) is: $protein\n";
```

Result:

Output:

Ex.no:

Date:

EXTRACTING UNIQUE WORDS FROM A FILE

Aim:

To develop a Perl program to extract and display all unique words from a text file.

Algorithm:

Step 1: Start the program.

Step 2: Open the text file containing the words.

Step 3: Read the file content and split it into individual words based on whitespace.

Step 4: Store each word in a hash to ensure uniqueness.

Step 5: Display all unique words.

Step 6: Close the file and end the program.

Program coding:

```
print "Enter the filename:\n";
my $filename = <STDIN>;
chomp($filename);

open(my $file, '<', $filename) or die "Could not open file '$filename': $!";

my %unique_words;

while (my $line = <$file>) {
    chomp($line);
    # Split line into words
    my @words = split(/\s+/, $line);
    # Add each word to the hash
    $unique_words{$_} = 1 for @words;
}

print "Unique words in the file:\n";
print "$_\n" for keys %unique_words;
```

```
close($file);
```

Result:

Output: