Data Structures and Algorithms

UNIT 1.3

Polynomial, String-ADT, Pattern Matching

Polynomials ADT

A polynomial in mathematics is an expression consisting of variables and coefficients, combined using addition, subtraction, and multiplication. Polynomials can have one or more terms, and the exponents on the variables must be non-negative integers. Here are some examples of polynomials:

1. Linear Polynomial:
   - An expression with a single term of the form **ax + b**, where **a** and **b** are constants.
   - Example: **3x + 2** or **-2x - 7**.
2. Quadratic Polynomial:
   - An expression with a single term of the form **ax^2 + bx + c**, where **a**, **b**, and **c** are constants, and **a** is not equal to zero.
   - Example: **2x^2 - 5x + 3** or **x^2 + 4x + 1**.
3. Cubic Polynomial:
   - An expression with a single term of the form **ax^3 + bx^2 + cx + d**, where **a**, **b**, **c**, and **d** are constants, and **a** is not equal to zero.
   - Example: **4x^3 - 3x^2 + 2x - 1** or **x^3 + 6x^2 + 12x + 8**.
4. Polynomial with Multiple Terms:
   - Polynomials can have more than one term. For example:
   - **2x^3 + 5x^2 - 3x + 7** (a cubic polynomial with four terms).
   - **x^4 - 2x^3 + 4x^2 - 5x + 1** (a quartic polynomial with five terms).
5. Polynomial with Multiple Variables:
   - Polynomials can involve more than one variable. For example:
   - **3x^2y + 2xy^2 - 5x^2 + 7** (a polynomial involving variables **x** and **y**).
   - **2a^2b - 4ab^2 + 3ab + 1** (a polynomial involving variables **a** and **b**).

Polynomials are used in various branches of mathematics and have numerous applications in solving equations, graphing functions, and modeling real-world phenomena. The highest power of the variable(s) in a polynomial is known as the degree of the polynomial, and it provides important information about its behavior.

A Polynomial Abstract Data Type (ADT) is a mathematical construct used to represent and manipulate polynomial expressions in a way that abstracts away the implementation details and focuses on the key properties of polynomials. Polynomials are mathematical expressions consisting of variables raised to non-negative integer powers, multiplied by coefficients. They are used in various fields, including mathematics, engineering, physics, and computer science.

Here are the key components and operations associated with the Polynomial ADT:

Components of a Polynomial ADT:

1. Coefficients: The numerical values that multiply each term in the polynomial.
2. Variables: The variables, often represented by a single letter like "x," raised to various powers in each term.
3. Exponents: The non-negative integer powers to which the variables are raised in each term.

Operations on a Polynomial ADT:

i. Initialization: Create a polynomial by specifying its terms, coefficients, and exponents.

ii. Addition and Subtraction: Perform addition and subtraction operations on two polynomials of the same variable.

iii. Multiplication: Multiply two polynomials to create a new polynomial.

iv. Evaluation: Evaluate the polynomial for a specific value of the variable(s).

v. Differentiation: Compute the derivative of the polynomial with respect to a variable.

vi. Integration: Compute the integral of the polynomial with respect to a variable.

vii. Simplification: Simplify the polynomial by combining like terms.

viii. String Representation: Convert the polynomial to a human-readable string for display and analysis.

A Polynomial ADT abstracts these operations and properties, allowing you to work with polynomials without worrying about the underlying mathematical or computational details. It's particularly useful in computer programs where you need to perform polynomial-related calculations.

For example, a Polynomial ADT in a programming language might consist of a class or structure with methods for creating, adding, multiplying, and evaluating polynomials. These methods would take care of the actual mathematical computations, allowing you to work with polynomials in a more user-friendly way.

In C, you can perform various polynomial operations, such as addition, subtraction, multiplication, and evaluation. Below are examples of these operations using simple functions and data structures. For more complex or performance-critical tasks, consider using dedicated libraries or more optimized algorithms.

C Program for Polynomial Addition Using Structure

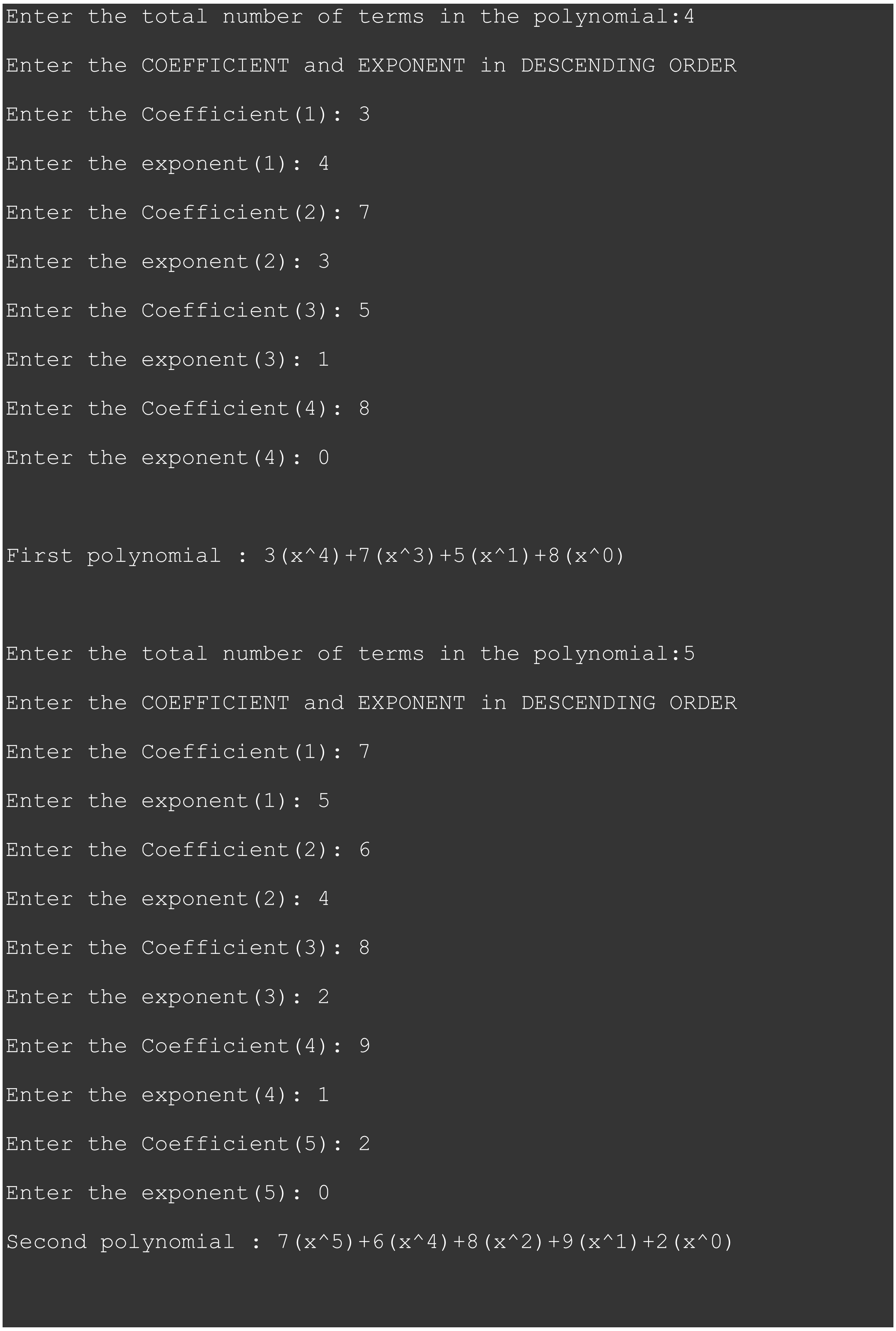
```c
/* program for addition of two polynomials

 polynomial are stored using structure

 and program uses array of structure

*/

#include<stdio.h>


/* declare structure for polynomial */

struct poly {

    int coeff;

    int expo;

};
```

```
/* declare three arrays p1, p2, p3 of type structure poly.

each polynomial can have maximum of ten terms

addition result of p1 and p2 is stored in p3*/


struct poly p1[10], p2[10], p3[10];


/* function prototypes */

int readPoly(struct poly[]);

int addPoly(struct poly[], struct poly[], int, int, struct
poly[]);

void displayPoly(struct poly[], int terms);


int main(){

    int t1, t2, t3;


    /* read and display first polynomial */

    t1 = readPoly(p1);

    printf(" \n First polynomial : ");

    displayPoly(p1, t1);

    /* read and display second polynomial */

    t2 = readPoly(p2);

    printf(" \n Second polynomial : ");

    displayPoly(p2, t2);


    /* add two polynomials and display resultant polynomial */

    t3 = addPoly(p1, p2, t1, t2, p3);

    printf(" \n\n Resultant polynomial after addition : ");
    displayPoly(p3, t3);

    printf("\n");


    return 0;

}


int readPoly(struct poly p[10])

{

    int t1, i;


    printf("\n\n Enter the total number of terms in the 
polynomial:");

    scanf("%d", &t1);


    printf("\n Enter the COEFFICIENT and EXPONENT in DESCENDING 
ORDER\n");

    for (i = 0; i < t1; i++) {

        printf("   Enter the Coefficient(%d): ", i + 1);

        scanf("%d", &p[i].coeff);

        printf("      Enter the exponent(%d): ", i + 1);

        scanf("%d", &p[i].expo); /* only statement in loop */

    }

    return (t1);

}


int addPoly(struct poly p1[10], struct poly p2[10], int t1, int 
t2, struct poly p3[10])

{

    int i, j, k;
```

```
i = 0;
j = 0;
k = 0;

while (i < t1 && j < t2) {
    if (p1[i].expo == p2[j].expo) {
        p3[k].coeff = p1[i].coeff + p2[j].coeff;
        p3[k].expo = p1[i].expo;

        i++;
        j++;
        k++;
    }
    else if (p1[i].expo > p2[j].expo) {
        p3[k].coeff = p1[i].coeff;
        p3[k].expo = p1[i].expo;
        i++;
        k++;
    }
    else {
        p3[k].coeff = p2[j].coeff;
        p3[k].expo = p2[j].expo;
        j++;
        k++;
    }
}
```

```c
    /* for rest over terms of polynomial 1 */
    while (i < t1) {
        p3[k].coeff = p1[i].coeff;
        p3[k].expo = p1[i].expo;
        i++;
        k++;
    }
    /* for rest over terms of polynomial 2 */
    while (j < t2) {
        p3[k].coeff = p2[j].coeff;
        p3[k].expo = p2[j].expo;
        j++;
        k++;
    }

    return (k); /* k is number of terms in resultant polynomial*/
}

void displayPoly(struct poly p[10], int term)
{
    int k;

    for (k = 0; k < term - 1; k++)
        printf("%d(x^%d)+", p[k].coeff, p[k].expo);
    printf("%d(x^%d)", p[term - 1].coeff, p[term - 1].expo);
}
```

Output



```
Enter the total number of terms in the polynomial:4

Enter the COEFFICIENT and EXPONENT in DESCENDING ORDER

Enter the Coefficient(1): 3

Enter the exponent(1): 4

Enter the Coefficient(2): 7

Enter the exponent(2): 3

Enter the Coefficient(3): 5

Enter the exponent(3): 1

Enter the Coefficient(4): 8

Enter the exponent(4): 0


First polynomial : 3(x^4)+7(x^3)+5(x^1)+8(x^0)


Enter the total number of terms in the polynomial:5

Enter the COEFFICIENT and EXPONENT in DESCENDING ORDER

Enter the Coefficient(1): 7

Enter the exponent(1): 5

Enter the Coefficient(2): 6

Enter the exponent(2): 4

Enter the Coefficient(3): 8

Enter the exponent(3): 2

Enter the Coefficient(4): 9

Enter the exponent(4): 1

Enter the Coefficient(5): 2

Enter the exponent(5): 0

Second polynomial : 7(x^5)+6(x^4)+8(x^2)+9(x^1)+2(x^0)
```

```
Resultant polynomial after addition :
7(x^5)+9(x^4)+7(x^3)+8(x^2)+14(x^1)+10(x^0)
```

The STRING ADT:

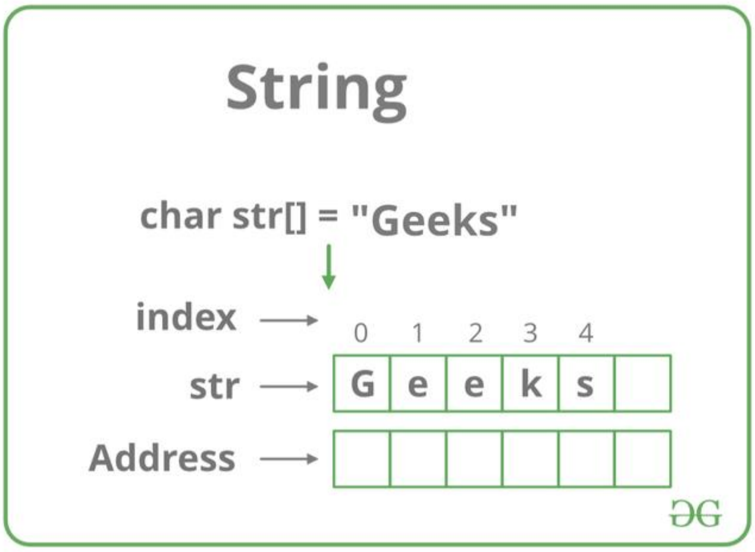
What is String?

Strings are considered a **data type** in general and are typically represented as arrays of bytes (or words) that store a **sequence of characters**.
Strings are defined as an **array of characters**.
The difference between a character array and a string is the string is terminated with a special character **'\0'**

Below are some examples of strings:
"geeks" , "for", "geeks", "GeeksforGeeks", "Geeks for Geeks", "123Geeks", "@123 Geeks"

How String is represented in Memory?

In C, a string can be referred to either using a character pointer or as a character array.
When strings are declared as character arrays, they are stored like other types of arrays in C. For example, if str[] is an auto variable then the string is stored in the stack segment, if it's a global or static variable then stored in the data segment, etc.



How to Declare Strings in C?

Below is the representation of strings in various languages:

```
// C program to illustrate strings
#include <stdio.h>

int main()
{
   // declare and initialize string
   char str[] = "Geeks";
```

```
    // print string
    printf("%s", str);

    return 0;
}
```

General Operations performed on String:
Here we are providing you with some must-know concepts of string:
1. Concatenation of Strings
The process of combining more than one string together is known as Concatenation. String Concatenation is the technique of combining two strings.

There are two ways to concatenate two strings:
a) String concatenation without using any inbuilt methods:
Below is the algorithm for the Concatenation of two strings:
Algorithm: CONCATENATE (STR1, STR2, STR3)

```
1. LEN1 = LENGTH(STR1).
2. LEN2 = LENGTH(STR2).
3. SET I = 0.
4. Repeat Steps 5 and 6 while I < LEN1-1:
5.     STR3[I] = STR1[I].
6.     SET I = I+1.
7. SET J = 0.
8. Repeat Steps 9 to 11 while I < (LEN1 + LEN2 - 2):
9.     STR3[I] = STR2[J].
10.     J = J+1.
11.     I = I+1.
12.Exit.
```

```
#include <stdio.h>
int main() {
  char s1[100] = "programming ", s2[] = "is awesome";
  int length, j;

  // store length of s1 in the length variable
  length = 0;
  while (s1[length] != '\0') {
    ++length;
  }

  // concatenate s2 to s1
  for (j = 0; s2[j] != '\0'; ++j, ++length) {
    s1[length] = s2[j];
  }

  // terminating the s1 string
  s1[length] = '\0';
  printf("After concatenation: ");
  puts(s1);

  return 0;
}
```

Output:

```
After concatenation: programming is awesome
```

b) String concatenation using inbuilt methods:

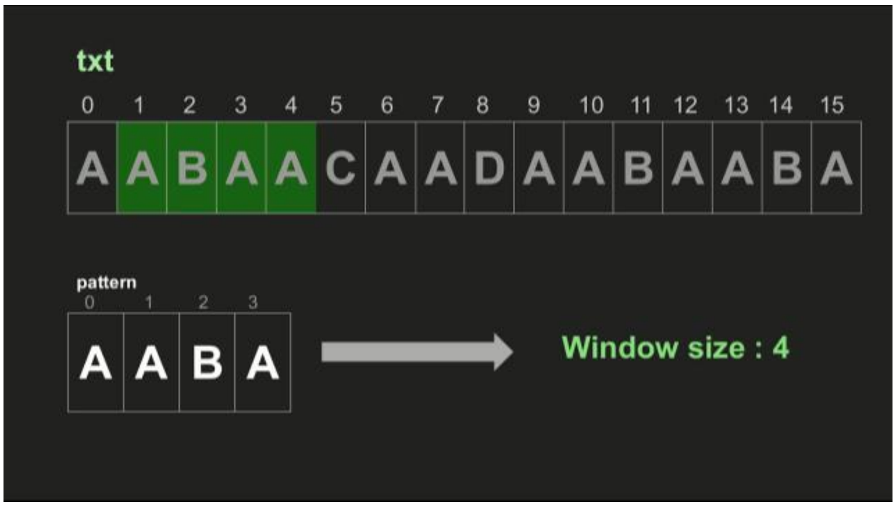
```
#include <stdio.h>
#include <string.h>
int main() {
   char str1[100] = "This is ", str2[] = "programiz.com";

   // concatenates str1 and str2
   // the resultant string is stored in str1.
   strcat(str1, str2);

   puts(str1);
   puts(str2);

   return 0;
}
```

Output:

```
This is programiz.com

programiz.com
```

2. Find in String

A very basic operation performed on Strings is to find something in the given whole string. Now, this can be to find a given character in a string, or to find a complete string in another string.



Find in String

a) Find a character in string:

Given a string and a character, your task is to find the first position of the character in the string. These types of problems are very competitive programming where you need to locate the position of the character in a string.

b) Find a substring in another string:

Consider there to be a string of length N and a substring of length M. Then run a nested loop, where the outer loop runs from 0 to (N-M) and the inner loop from 0 to M. For every index check if the sub-string traversed by the inner loop is the given sub-string or not.

An efficient solution is to use a O(n) searching algorithm like KMP algorithm, Z algorithm, etc.

3. Replace in String

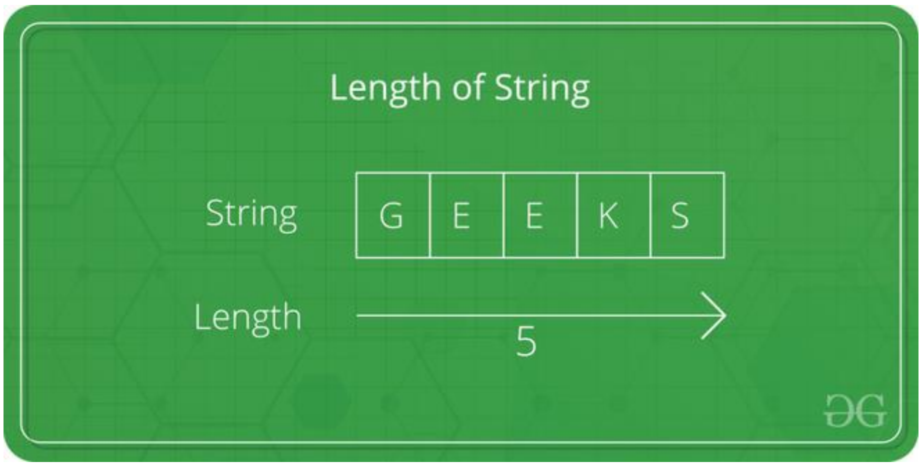
Many times, it is very important to make corrections in strings. Replacing a character, word or phrase in a String is another very common operation performed on Strings.

The simplest approach to solve the given problem is to traverse the string S and when any string S1 is found as a substring in the string S then replace it by S2. Follow the steps below to solve this problem:

- Initialize a string ans to store the resultant string after replacing all the occurrences of the substring S1 to S2 in the string S.
- Iterate over the characters of the string S using variable i and perform the following steps:
    - If the prefix substring of the string S is equal to S1 from the index i, then add the string S2 in the string ans.
    - Otherwise, add the current character to the string ans.
- After completing the above steps, print the string ans as the result.
-

4. Finding the Length of String

One of the most general operations on String is to find the length/size of a given string. Length is defined as the number of characters in a string is called the length of that string.



Finding the Length of String

There are two ways to concatenate two strings:
a) Length of string without using any inbuilt methods:
Below is the algorithm for finding the length of two strings:
1. SET LEN = 0 AND I = 0.
2. Repeat Steps 3 to 4 while STRING[I] is not NULL:
3. LEN = LEN + 1.
4. SET I = I + 1.
5. Exit.

```
// C program to find the length of string
#include <stdio.h>
#include <string.h>

int main()
{
        char Str[1000];
        int i;

        printf("Enter the String: ");
        scanf("%s", Str);

        for (i = 0; Str[i] != '\0'; ++i);

        printf("Length of Str is %d", i);

        return 0;
}
```

b) Length of string using inbuilt methods:

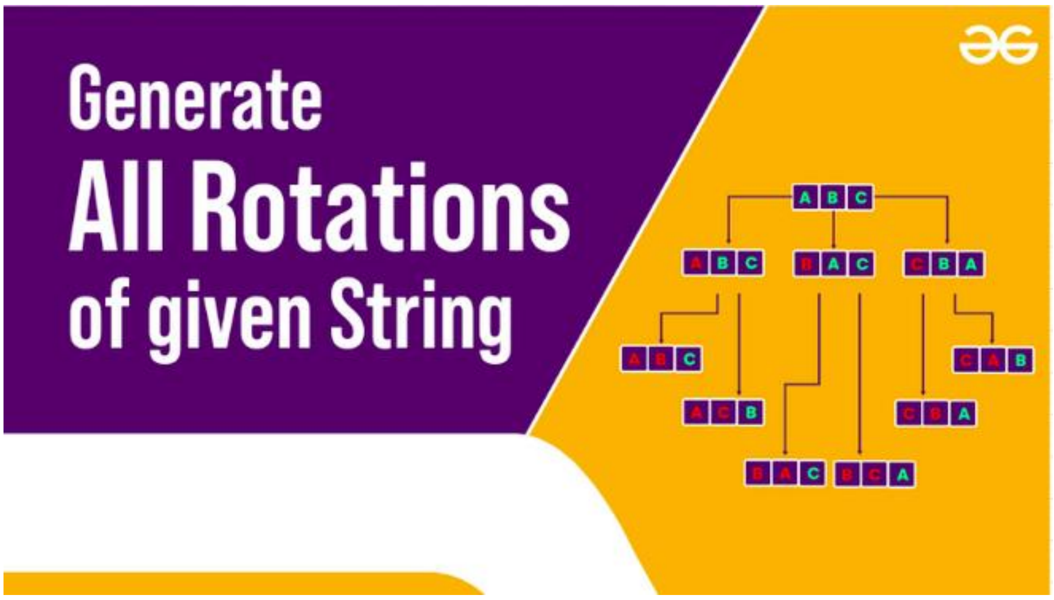
```
// C program to find the length of
// string using strlen function
```

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

int main()
{
	char Str[1000];
	int i;

	printf("Enter the String: ");
	scanf("%s", Str);

	printf("Length of Str is %ld", strlen(Str));

	return 0;
}
```

5. Reverse and Rotation of a String

Reverse operation is interchanging the position of characters of a string such that the first becomes the last, the second becomes the second last, and so on.

a) Rotations of a String**:**

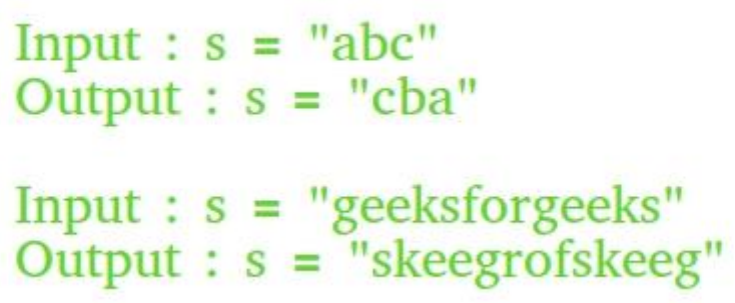


Consider a string “geeks”, now all possible rotations for this will be:

- *geeks*
- *eeksg*
- *eksge*
- *ksgee*
- *sgeek*

b) Reverse a String**:**

The reversing of a string is nothing but simply substituting the last element of a string to the 1st position of the string.



```
Input  : s = "abc"
Output : s = "cba"

Input  : s = "geeksforgeeks"
Output : s = "skeegrofskeeg"
```

6. Subsequence of a String

*A **subsequence** is a sequence that can be derived from another sequence by removing zero or more elements, without changing the order of the remaining elements.*

More generally, we can say that for a sequence of size n, we can have (2n-1) non-empty sub-sequences in total.

For example, Consider the string “geeks”, there are 15 sub-sequences.

They are:

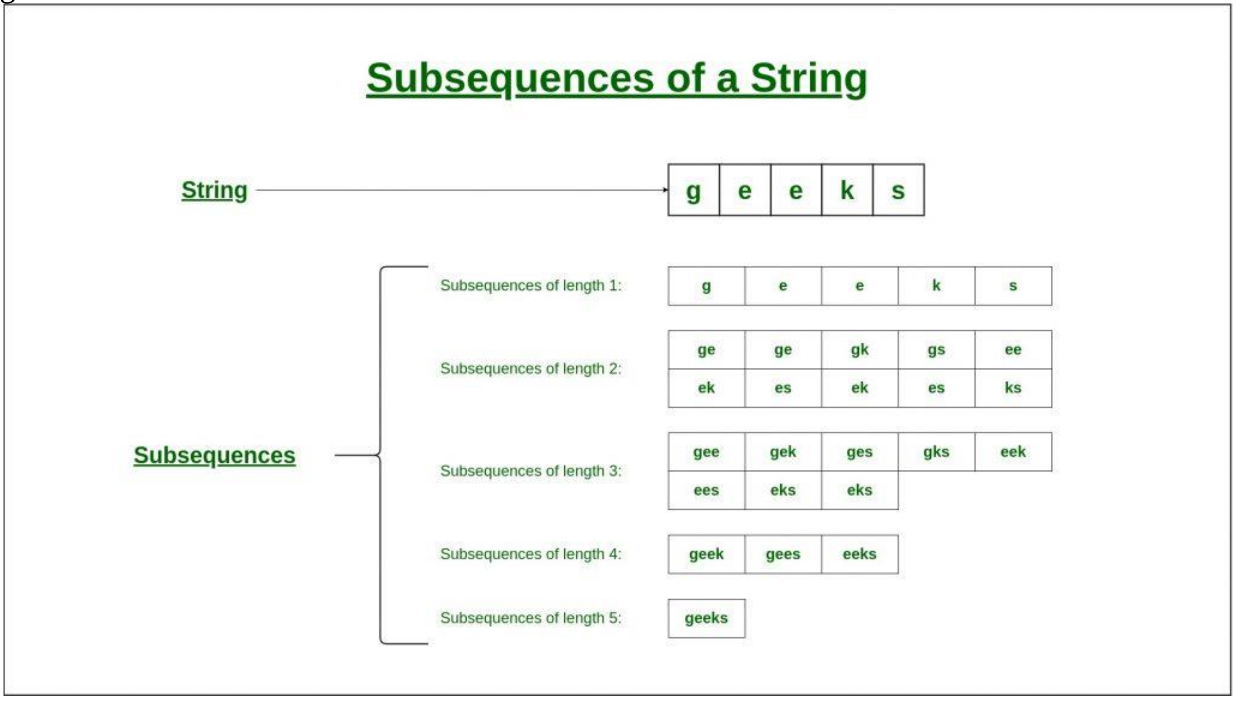
g, e, e, k, s,

ge, ge, gk, gs, ee, ek, es, ek, es, ks,

gee, gek, ges, gek, ges, gks, eek, ees, eks, eks,

geek, gees, eeks,

geeks



7. Substring of a String

*A **substring** is a contiguous part of a string, i.e., a string inside another string.*

In general, for a string of size n, there are n*(n+1)/2 non-empty substrings.

For example, Consider the string “geeks”, There are 15 non-empty substrings.

The subarrays are:

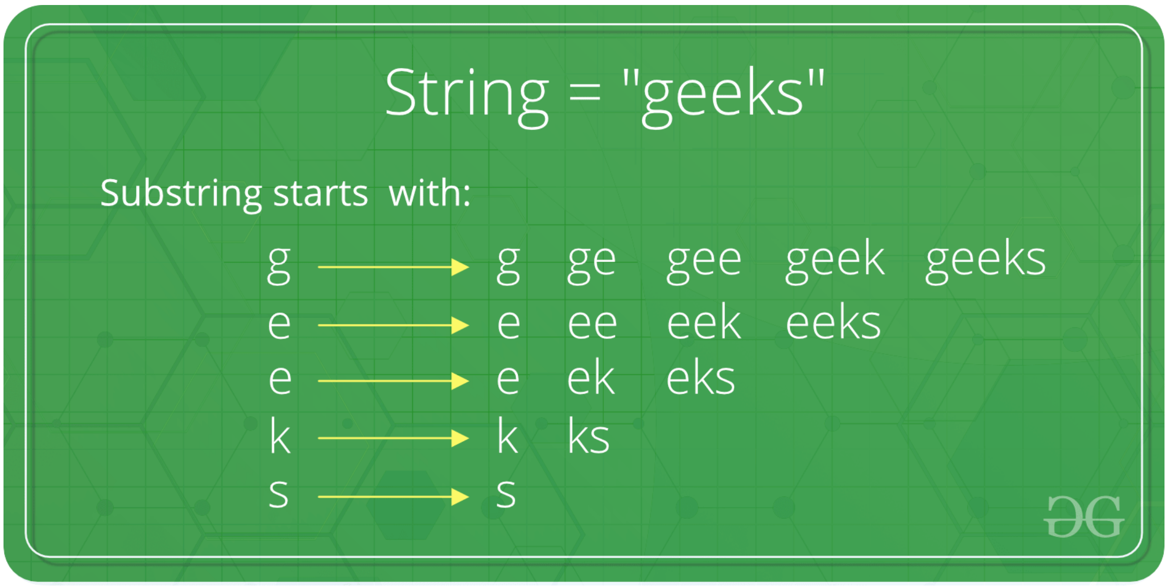
g, ge, gee, geek, geeks,

e, ee, eek, eeks,

e, ek, eks,

k, ks,

ks



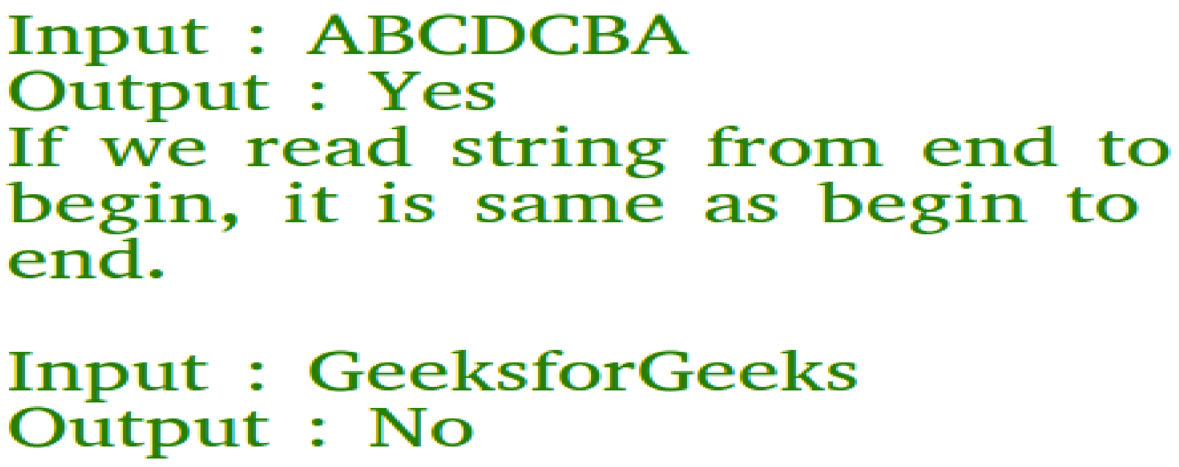
<u>8. Binary String</u>

A Binary String is a special kind of string made up of only two types of characters, such as 0 and 1.
For Example:
Input: str = "01010101010"
Output: Yes, it is a Binary String

Input: str = "geeks101"
Output: No, it is not a Binary String

<u>9. Palindrome String</u>

A string is said to be a palindrome if the reverse of the string is the same as the string.
For example,
“abba” is a palindrome, but “abbc” is not a palindrome.



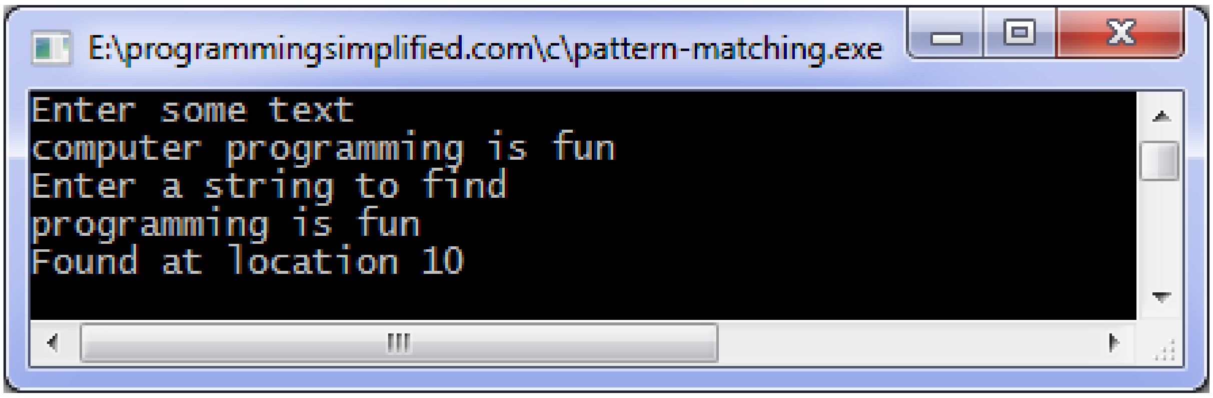
```
Input : ABCDCBA
Output : Yes
If we read string from end to
begin, it is same as begin to
end.

Input : GeeksforGeeks
Output : No
```

PATTERN MATCHING:

C program to check if a string is present in an another string, for example, the string "programming" is present in the string "C programming". If it's present, then its location (i.e. at which position it's present) is printed. We create a function match which receives two character arrays and returns the position if matching occurs otherwise returns -1. We are implementing naive string search algorithm in this program.

```
#include <stdio.h>
#include <string.h>
int match(char [], char []);

int main() {
  char a[100], b[100];
  int position;

  printf("Enter some text\n");
  gets(a);

  printf("Enter a string to find\n");
  gets(b);

  position = match(a, b);

  if (position != -1) {
    printf("Found at location: %d\n", position + 1);
  }
  else {
    printf("Not found.\n");
  }

  return 0;
}

int match(char text[], char pattern[]) {
  int c, d, e, text_length, pattern_length, position = -1;

  text_length    = strlen(text);
  pattern_length = strlen(pattern);

  if (pattern_length > text_length) {
    return -1;
  }

  for (c = 0; c <= text_length - pattern_length; c++) {
    position = e = c;
    for (d = 0; d < pattern_length; d++) {
      if (pattern[d] == text[e]) {
        e++;
      }
      else {
        break;
      }
    }
    if (d == pattern_length) {
      return position;
    }
  }

  return -1;
}
```

Output:



E:\programmingsimplified.com\c\pattern-matching.exe

```
Enter some text
computer programming is fun
Enter a string to find
programming is fun
Found at location 10
```

Using pointers:

```
#include<stdio.h>
int match(char*, char*);

int main()
{
   char a[100], b[100];
   int position;

   printf("Enter some text\n");
   gets(a);

   printf("Enter a string to find\n");
   gets(b);
```

```c
  position = match(a, b);

  if(position!=-1)
    printf("Found at location %d\n", position+1);
  else
    printf("Not found.\n");

  return 0;
}

int match(char *a, char *b)
{
  int c;
  int position = 0;
  char *x, *y;

  x = a;
  y = b;

  while(*a)
  {
    while(*x==*y)
    {
      x++;
      y++;
      if(*x=='\0'||*y=='\0')
        break;
    }
    if(*y=='\0')
      break;

    a++;
    position++;
    x = a;
    y = b;
  }
  if(*a)
    return position;
  else
    return -1;
}
```

strstr() in C for Pattern matching

In C, std::strstr() is a predefined function used for string matching. **<string.h>** is the header file required for string functions. This function takes two strings **s1** and **s2** as arguments and finds the first occurrence of the string **s2** in the string **s1**. The process of matching does not include the terminating null-characters('\0'), but function stops there.

Syntax

char ***strstr** (const char **s1*, const char **s2*);

Parameters

- **s1**: This is the main string to be examined.
- **s2**: This is the sub-string to be searched in string.

Return Value

- This function returns a pointer point to the first character of the found *s2* in *s1* otherwise a null pointer if *s2* is not present in *s1*.
- If s2 points to an empty string, s1 is returned.

Example

The below program illustrates the usage of the strstr() function.

```
// C program to illustrate strstr()
#include <stdio.h>
#include <string.h>
int main()
{
        // Take any two strings
        char s1[] = "GeeksforGeeks";
        char s2[] = "for";
        char* p;

        // Find first occurrence of s2 in s1
        p = strstr(s1, s2);

        // Prints the result
        if (p) {
          printf("String found\n");
          printf("First occurrence of string '%s' in '%s' is "
               "'%s'",
               s2, s1, p);
        }
        else
          printf("String not found\n");

        return 0;
}
```

Output

```
String found

First occurrence of string 'for' in 'GeeksforGeeks' is 'forGeeks'
```