Strings

1. Reverse a String

Write a function void reverseString(char *str) that takes a pointer to a string and reverses the string in place.

```
#include <stdio.h>

// Function prototype
void reverseString(char *str);

int main()
{
   char str[100];

   printf("Enter a string: ");
   scanf("%[^\n]", str);

   reverseString(str);
   printf("Reversed string: %s\n", str);
   return 0;
}

/*
Name: reverseString()
Return Type: void
Parameter:(data type of each parameter): char*
Short description: it is used to reverse the string
```

```
*/

// Function to reverse the string
void reverseString(char *str)
{
   if (str == NULL)
      return;

   char *start = str;
   char *end = str;

   while (*end != '\0')
      end++;
   end--;

   // Swap characters from start to end
   while (start < end)
   {
      char temp = *start;
      *start = *end;
      *end = temp;
      start++;
      end--;
   }
}
```

O/P:

```
Enter a string: Hello world
Reversed string: dlrow olleH
```

2. Concatenate Two Strings

Implement a function void concatenateStrings(char *dest, const char *src) that appends the source string to the destination string using pointers.

```
#include <stdio.h>

// Function prototype
void concatenateStrings(char *dest, const char *src);

int main()
{
   char src[50], dest[50];

   printf("Enter the destination string: ");
   scanf("%[^\n]", dest);

   printf("Enter the source string: ");
   scanf("%[^\n]", src);

   concatenateStrings(dest, src);
   printf("Concatenated string: %s\n", dest);
   return 0;
}

/*
Name: concatenateStrings()
```

```
Return Type: void
Parameter:(data type of each parameter): char* and const char*
Short description: it is used to concatenate two strings
*/

// Function to concatenate two strings
void concatenateStrings(char *dest, const char *src)
{
    while (*dest != '\0')
        dest++;

    while (*src != '\0')
    {
        *dest = *src;
        dest++;
        src++;
    }
    *dest = '\0';
}
```

O/P:

```
        Enter the destination string: good
        Enter the source string: morning
        Concatenated string: goodmorning
```

3. String Length

Create a function int stringLength(const char *str) that calculates and returns the length of a string using pointers.

```
#include <stdio.h>

// Function prototype
int stringLength(const char *str);

int main()
{
   char str[100];

   printf("Enter a string: ");
   scanf("%[^\n]", str);

   int length = stringLength(str);
   printf("Length of the string: %d\n", length);
   return 0;
}

/*
Name: stringLength()
Return Type: int
Parameter:(data type of each parameter): const char*
Short description: it is used to find the length of the string
*/

// Function to find the length of the string
```

```
int stringLength(const char *str)
{
   const char *ptr = str;
   int length = 0;

   while (*ptr != '\0')
   {
      length++;
      ptr++;
   }
   return length;
}
```

O/P:

```
        Enter a string: good afternoon
        Length of the string: 14
```

4. Compare Two Strings

Write a function int compareStrings(const char *str1, const char *str2) that compares two strings lexicographically and returns 0 if they are equal, a positive number if str1 is greater, or a negative number if str2 is greater.

```
#include <stdio.h>

//Function prototype
int compareStrings(const char *str1, const char *str2);
int main()
{
    char str1[100], str2[100];

    printf("Enter the first string: ");
    scanf("%s", str1);

    printf("Enter the second string: ");
    scanf(" %s", str2);

    int result = compareStrings(str1, str2);
    if (result == 0)
        printf("Strings are equal\n");
    else if (result > 0)
        printf("First string is greater than second string\n");
    else
        printf("Second string is greater than first string\n");
    return 0;
}

/*
Name: compareStrings()
Return Type: int
Parameter:(data type of each parameter): const char* and const char*
Short description: it is used to compare two strings
*/
```

```
// Function to compare two strings
int compareStrings(const char *str1, const char *str2)
{
   while (*str1 != '\0' && *str2 != '\0')
   {
      if (*str1 != *str2)
         // Return the difference between the mismatched characters
         return *str1 - *str2;
      str1++;
      str2++;
   }

   // if strings have the same characters but differ in length
   return *str1 - *str2;
}
```

O/P:

```
     Enter the first string: good
     Enter the second string: bad
     First string is greater than second string
```

5. Find Substring

Implement char* findSubstring(const char *str, const char *sub) that returns a pointer to the first occurrence of the substring sub in the string str, or NULL if the substring is not found.

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

```
//Function prototype
char* findSubstring(const char *str, const char *sub);

int main()
{
    char str[100], sub[100];

    printf("Enter the main string: ");
    scanf("%s", str);

    printf("Enter the substring to find: ");
    scanf("%s", sub);

    char *result = findSubstring(str, sub);
    if (result)
        printf("Substring found at position: %ld\n", result - str);
    else
        printf("Substring not found\n");
    return 0;
}

/*
Name: findSubstring()
Return Type: char*
Parameter:(data type of each parameter): const char* and const char*
Short description: it is used to find substring
```

```
*/

// Function to find the substing from the given main string
char* findSubstring(const char *str, const char *sub)
{
    if (!*sub)
        // If the substring is empty, return the beginning of str
        return (char *)str;

    for (const char *s = str; *s != '\0'; s++)
    {
        const char *strPtr = s;
        const char *subPtr = sub;

        while (*strPtr != '\0' && *subPtr != '\0' && *strPtr == *subPtr)
        {
            strPtr++;
            subPtr++;
        }

        // If the entire substring has been matched
        if (*subPtr == '\0')
            return (char *)s;
    }

    // If no match is found, return NULL
    return NULL;
}
```

O/P:

```
        Enter the main string: good
        Enter the substring to find: od
        Substring found at position: 2
```

6. Replace Character in String

Write a function void replaceChar(char *str, char oldChar, char newChar) that replaces all occurrences of oldChar with newChar in the given string.

```
#include <stdio.h>

//Function prototype
void replaceChar(char *str, char oldChar, char newChar);

int main()
{
   char str[100];
   char oldChar, newChar;

   printf("Enter a string: ");
   scanf("%99[^\n]", str);

   printf("Enter the character to replace: ");
   scanf(" %c", &oldChar);
```

```
    printf("Enter the new character: ");
    scanf(" %c", &newChar);

    replaceChar(str, oldChar, newChar);

    printf("Modified string: %s\n", str);
    return 0;
}

/*
Name: replaceChar()
Return Type: void
Parameter:(data type of each parameter): char*, char and char
Short description: it is used to replace a character in string
*/

// Function to replace a chracter in a string
void replaceChar(char *str, char oldChar, char newChar)
{
    for (char *ptr = str; *ptr != '\0'; ptr++)
    {
        if (*ptr == oldChar)
            *ptr = newChar;
    }
}
```

O/P:

```
Enter a string: hello good afternoon
Enter the character to replace: h
Enter the new character: H
Modified string: Hello good afternoon
```

7. Copy String

Create a function void copyString(char *dest, const char *src) that copies the content of the source string src to the destination string dest.

```
#include <stdio.h>

//Function prototype
void copyString(char *dest, const char *src);

int main()
{
   char src[100], dest[100];

   printf("Enter the source string: ");
   scanf("%[^\n]", src);

   copyString(dest, src);

   printf("Copied string: %s\n", dest);

   return 0;
}
```

```
/*
Name: copyString()
Return Type: void
Parameter:(data type of each parameter): char* and const char*
Short description: it is used to copy string
*/

// Function to copy content of source string to the destination string
void copyString(char *dest, const char *src)
{
   while (*src != '\0')
   {
      *dest = *src;
      dest++;
      src++;
   }
   *dest = '\0';
}
```

O/P:

```
        Enter the source string: one good day
        Copied string: one good day
```

8. Count Vowels in a String

Implement int countVowels(const char *str) that counts and returns the number of vowels in a given string.

```
#include <stdio.h>

// Function prototype
int countVowels(const char *str);

int main()
{
    char str[100];

    printf("Enter a string: ");
    scanf("%[^\n]", str);

    int vowels = countVowels(str);
    printf("Number of vowels in the string: %d\n", vowels);

    return 0;
}

/*
Name: countVowels()
Return Type: int
Parameter:(data type of each parameter): const char*
Short description: it is used to count vowels in a string
*/

// Function to count vowels in a string
int countVowels(const char *str)
```

```
{
  int count = 0;

  while (*str != '\0')
  {
    char c = *str;
    if (c == 'a' || c == 'A' || c == 'e' || c == 'E' ||
      c == 'i' || c == 'I' || c == 'o' || c == 'O' ||
      c == 'u' || c == 'U')
        count++;
    str++;
  }
  return count;
}
```

O/P:

```
    Enter a string: Good Day
    Number of vowels in the string: 3
```

9. Check Palindrome

Write a function int isPalindrome(const char *str) that checks if a given string is a palindrome and returns 1 if true, otherwise 0.

```
#include <stdio.h>

// Function prototype
int isPalindrome(const char *str);
int main()
{
   char str[100];

   printf("Enter a string: ");
   scanf("%[^\n]", str);

   if (isPalindrome(str))
      printf("String is a palindrome\n");
   else
      printf("String is not a palindrome.\n");

   return 0;
}

/*
Name: isPalindrome()
Return Type: int
Parameter:(data type of each parameter): const char*
Short description: it is used to check if a given string is palindrome
*/

// Function to check if a given string is palindrome
int isPalindrome(const char *str)
{
   int start = 0;
   int end = 0;
```

```
    while (str[end] != '\0')
        end++;
    end--;

    while (start < end)
    {
        if (str[start] != str[end])
            return 0;  // Not a palindrome
        start++;
        end--;
    }
    return 1;  // Palindrome
}
```

O/P:

```
        Enter a string: malayalam
        String is a palindrome
```

10. Tokenize String

Create a function void tokenizeString(char *str, const char *delim, void (*processToken)(const char *)) that tokenizes the string str using delimiters in delim, and for each token, calls processToken.

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

//Function prototypes
```

```
void tokenizeString(char *str, const char *delim, void (*processToken)(const char *));
void processToken(const char *token);

int main()
{
   char str[] = "Hi, my name is Nanditha!";
   const char *delim = " ,!";

   // Call tokenizeString, passing the processToken function to process each token
   tokenizeString(str, delim, processToken);

   return 0;
}

/*
Name: tokenizeString()
Return Type: void
Parameter:(data type of each parameter): const char*
Short description: it is used to tokenize the string using delimiters
*/

// Function to tokenize the string using delimiters
void tokenizeString(char *str, const char *delim, void (*processToken)(const char *))
{
   char *token = strtok(str, delim);
```

```
    while (token != NULL)
    {
        processToken(token);
        token = strtok(NULL, delim);
    }
}

/*
Name: processToken()
Return Type: void
Parameter:(data type of each parameter): const char*
Short description: it is used to process the each token
*/

// Function to process the each token
void processToken(const char *token)
{
    printf("Token: %s\n", token);
}
```

O/P:

```
          Token: Hi
          Token: my
          Token: name
          Token: is
          Token: Nanditha
```

1. Allocate and Free Integer Array

Write a program that dynamically allocates memory for an array of integers, fills it with values from 1 to n, and then frees the allocated memory.

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

int main()
{
   int n;

   printf("Enter the size of the array: ");
   scanf("%d", &n);

   int *a = (int *)malloc(n * sizeof(int));

   if (a == NULL)
   {
      printf("Memory allocation failed.\n");
      return 1;
   }

   for (int i = 0; i < n; i++)
      a[i] = i + 1;

   printf("Array elements: ");
```

```
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");

    free(a);
    return 0;
}
```

O/P:

```
        Enter the size of the array: 5
        Array elements: 1 2 3 4 5
```

2. Dynamic String Input

Implement a function that dynamically allocates memory for a string, reads a string input from the user, and then prints the string. Free the memory after use.

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

// Function prototype
void readAndPrint();

int main()
{
    readAndPrint();
    return 0;
}
```

```
/*
Name: readAndPrint()
Return Type: void
Parameter:(data type of each parameter): no parameters
Short description: it is used to read a string and print
*/

// Function to read a string and print
void readAndPrint()
{
    int size;
    printf("Enter the size of the string: ");
    scanf("%d", &size);

    char *str = (char *)malloc(size * sizeof(char) + 1);  // +1 for the null-terminator

    if (str == NULL)
    {
        printf("Memory allocation failed.\n");
        return;
    }

    printf("Enter a string: ");
    scanf("%s", str);

    printf("String: %s\n", str);
```

```
   free(str);
}
```

O/P:

```
        Enter the size of the string: 50
        Enter a string: good
        String: good
```

3. Resize an Array

Write a program that dynamically allocates memory for an array of n integers, fills it with values, resizes the array to 2n using realloc(), and fills the new elements with values.

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

int main()
{
   int n;

   printf("Enter the number of elements for the initial array: ");
   scanf("%d", &n);

   int *arr = (int *)malloc(n * sizeof(int));

   if (arr == NULL)
   {
       printf("Memory allocation failed\n");
       return 0;
   }

   printf("Enter %d values for the array\n", n);
   for (int i = 0; i < n; i++)
   {
       printf("Enter value %d: ", i + 1);
       scanf("%d", &arr[i]);
   }

   printf("\nOriginal array: ");
   for (int i = 0; i < n; i++)
       printf("%d ", arr[i]);
   printf("\n");

   // Resize the array to 2n using realloc()
   arr = (int *)realloc(arr, 2 * n * sizeof(int));

   if (arr == NULL)
   {
       printf("Memory reallocation failed\n");
       return 0;
   }

   // Fill the new elements with values
   printf("\nEnter %d additional values for the resized array\n", n);
```

```
    for (int i = n; i < 2 * n; i++)
    {
        printf("Enter value %d: ", i + 1);
        scanf("%d", &arr[i]);
    }

    printf("\nResized array: ");
    for (int i = 0; i < 2 * n; i++)
        printf("%d ", arr[i]);
    printf("\n");

    free(arr);
    return 0;
}
```

O/P:

```
Enter the number of elements for the initial array: 4
Enter 4 values for the array
Enter value 1: 1
Enter value 2: 2
Enter value 3: 3
Enter value 4: 4

Original array: 1 2 3 4

Enter 4 additional values for the resized array
Enter value 5: 5
Enter value 6: 6
Enter value 7: 7
Enter value 8: 8

Resized array: 1 2 3 4 5 6 7 8
```

4. Matrix Allocation

Create a function that dynamically allocates memory for a 2D array (matrix) of size m x n, fills it with values, and then deallocates the memory.

```c
#include <stdio.h>
#include <stdlib.h>

// Function prototype
void createFillMatrix(int m, int n);

int main()
{
   int m, n;

   printf("Enter the number of rows and columns: ");
   scanf("%d %d", &m, &n);

   // Call the function to create and fill the matrix
   createFillMatrix(m, n);
   return 0;
}
```

```
void createFillMatrix(int m, int n)
{
    int **matrix = (int **)malloc(m * sizeof(int *));
    if (matrix == NULL)
    {
        printf("Memory allocation failed for rows\n");
        return;
    }
    for (int i = 0; i < m; i++)
    {
        matrix[i] = (int *)malloc(n * sizeof(int));
        if (matrix[i] == NULL)
        {
            printf("Memory allocation failed for columns in row %d\n", i);
            return;
        }
    }
    printf("Enter values for the %d x %d matrix:\n", m, n);
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            printf("Enter value for position (%d, %d): ", i + 1, j + 1);
            scanf("%d", &matrix[i][j]);
        }
    }
    printf("\n%d x %d matrix is:\n", m, n);
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
            printf("%d ", matrix[i][j]);
        printf("\n");
    }
    for (int i = 0; i < m; i++)
        free(matrix[i]);
    free(matrix);
}
```

O/P:

```
        Enter the number of rows and columns: 2 2
        Enter values for the 2 x 2 matrix:
        Enter value for position (1, 1): 4
        Enter value for position (1, 2): 6
        Enter value for position (2, 1): 8
        Enter value for position (2, 2): 10

        2 x 2 matrix is:
        4 6
        8 10
```

5. String Concatenation with Dynamic Memory

Implement a function that takes two strings, dynamically allocates memory to concatenate them, and returns the new concatenated string. Ensure to free the memory after use.

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

// Function prototype
char* concatenateStrings(const char *str1, const char *str2);

int main()
{
    const char *str1 = "Good ";
    const char *str2 = "Afternoon";

    // Call the function to concatenate the strings
    char *result = concatenateStrings(str1, str2);

    if (result != NULL)
    {
        printf("Concatenated string: %s\n", result);
        free(result);
    }
    return 0;
}

/*
Name: concatenateStrings()
Return Type: char*
Parameter:(data type of each parameter): const char* and const char*
```

```
Short description: it is used to concatenate two strings
*/

// Function to concatenate two strings and return the concatenated string
char* concatenateStrings(const char *str1, const char *str2)
{
    int len1 = 0, len2 = 0;
    while (str1[len1] != '\0')
        len1++;

    while (str2[len2] != '\0')
        len2++;

    char *concatenated = (char *)malloc((len1 + len2 + 1) * sizeof(char));
    if (concatenated == NULL)
    {
        printf("Memory allocation failed\n");
        return 0;
    }

    // Copy the first string into the new memory space
    int i = 0;
    for (; i < len1; i++)
        concatenated[i] = str1[i];

    // Append the second string to the new memory space
    for (int j = 0; j < len2; j++, i++)
```

```
        concatenated[i] = str2[j];
    concatenated[i] = '\0';
    return concatenated;
}
```

O/P:

```
        Concatenated string: Good Afternoon
```

6. Dynamic Memory for Structure

Define a struct for a student with fields like name, age, and grade. Write a program that dynamically allocates memory for a student, fills in the details, and then frees the memory.

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

// Function prototype
struct Student* createStudent(const char *name, int age, float grade);

// Define a structure for a student
struct Student
{
    char *name;
    int age;
    float grade;
};
```

```
int main()
{
   struct Student *student = createStudent("XYZ", 22, 78.5);

   if (student != NULL)
   {
      printf("Student name: %s\n", student->name);
      printf("Age: %d\n", student->age);
      printf("Grade: %.2f\n", student->grade);

      if (student != NULL)
      {
         free(student->name);
         free(student);
      }
   }
   return 0;
}

/*
Name: createStudent()
Return Type: struct Student*
Parameter:(data type of each parameter): const char*, int and dloat
Short description: it is used to create and initialize a student dynamically
*/
```

```
// Function to create and initialize a student dynamically
struct Student* createStudent(const char *name, int age, float grade)
{
    struct Student *student = (struct Student *)malloc(sizeof(struct Student));
    if (student == NULL)
    {
        printf("Memory allocation failed\n");
        return 0;
    }

    student->name = (char *)malloc(strlen(name) + 1);
    if (student->name == NULL)
    {
        printf("Memory allocation for name failed\n");
        free(student);
        return 0;
    }

    // Copy the name into the allocated memory
    strcpy(student->name, name);

    // Initialize the age and grade
    student->age = age;
    student->grade = grade;

    return student;
}
```

O/P:

```
        Student name: XYZ
        Age: 22
        Grade: 78.50
```

7. Dynamic Array of Pointers

Write a program that dynamically allocates memory for an array of pointers to integers, fills each integer with values, and then frees all the allocated memory.

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

int main()
{
   int n;
   printf("Enter the number of elements: ");
   scanf("%d", &n);

   int **a = (int **)malloc(n * sizeof(int *));
   if (a == NULL)
   {
      printf("Memory allocation failed\n");
      return 1;
   }
   for (int i = 0; i < n; i++)
   {
        a[i] = (int *)malloc(sizeof(int));
        if (a[i] == NULL)
        {
            printf("Memory allocation for arr[%d] failed\n", i);
            for (int j = 0; j < i; j++)
                free(a[j]);
            free(a);
            return 1;
        }
    }
    for (int i = 0; i < n; i++)
        *(a[i]) = i + 1;

    printf("Array elements:\n");
    for (int i = 0; i < n; i++)
        printf("a[%d] = %d\n", i, *(a[i]));

    for (int i = 0; i < n; i++)
        free(a[i]);
    free(a);
    return 0;
}
```

O/P:

```
Enter the number of elements: 5
Array elements:
a[0] = 1
```

```
    a[1] = 2
    a[2] = 3
    a[3] = 4
    a[4] = 5
```

8. Dynamic Memory for Multidimensional Arrays

Create a program that dynamically allocates memory for a 3D array of integers, fills it with values, and deallocates the memory.

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

int main()
{
    int x, y, z;

    printf("Enter the dimensions of the 3D array: ");
    scanf("%d %d %d", &x, &y, &z);

    int ***array = (int ***)malloc(x * sizeof(int **));
    if (array == NULL)
    {
        printf("Memory allocation failed for the first dimension\n");
        return 1;
    }

    for (int i = 0; i < x; i++)
    {
        array[i] = (int **)malloc(y * sizeof(int *));
        if (array[i] == NULL)
        {
            printf("Memory allocation failed for array[%d] in second dimension\n", i);
            for (int j = 0; j < i; j++)
                free(array[j]);
            free(array);
            return 1;
        }
    }

    for (int i = 0; i < x; i++)
    {
        for (int j = 0; j < y; j++)
        {
            array[i][j] = (int *)malloc(z * sizeof(int));
            if (array[i][j] == NULL)
            {
                printf("Memory allocation failed for array[%d][%d] in third dimension\n", i, j);
                for (int k = 0; k < j; k++)
                    free(array[i][k]);
                for (int k = 0; k < i; k++)
                {
                    for (int l = 0; l < y; l++)
                        free(array[k][l]);
                    free(array[k]);
                }
                free(array);
                return 1;
            }
        }
    }

    // Fill the 3D array with values
    int value = 1;
    for (int i = 0; i < x; i++)
    {
        for (int j = 0; j < y; j++)
        {
            for (int k = 0; k < z; k++)
                array[i][j][k] = value++;
        }
    }

    // Print the values in the 3D array
    printf("3D array elements:\n");
    for (int i = 0; i < x; i++)
    {
        for (int j = 0; j < y; j++)
        {
            for (int k = 0; k < z; k++)
                printf("array[%d][%d][%d] = %d\n", i, j, k, array[i][j][k]);
        }
    }

    for (int i = 0; i < x; i++)
    {
        for (int j = 0; j < y; j++)
            free(array[i][j]);  // Free the third dimension
        free(array[i]);  // Free the second dimension
    }
    free(array);  // Free the first dimension
    return 0;
}
```

O/P:

```
Enter the dimensions of the 3D array: 2 2 2
3D array elements:
array[0][0][0] = 1
array[0][0][1] = 2
array[0][1][0] = 3
array[0][1][1] = 4
array[1][0][0] = 5
array[1][0][1] = 6
array[1][1][0] = 7
array[1][1][1] = 8
```

1. Swap Two Numbers Using Double Pointers

Write a function void swap(int **a, int **b) that swaps the values of two integer pointers using double pointers.

```
#include <stdio.h>

//Function prototype
void swap(int **a, int **b);

int main()
{
   int x = 5, y = 4;
   int *p = &x, *q = &y;

   printf("Before swap: p = %d  q = %d\n", *p, *q);

   // Call the swap function
   swap(&p, &q);

   printf("After swap: p = %d  q = %d\n", *p, *q);

   return 0;
}

/*
Name: swap()
```

```
Return Type: void
Parameter:(data type of each parameter): int** and int**
Short description: it is used to swap two numbers
*/

// Function to swap two numbers
void swap(int **a, int **b)
{
   int *temp = *a;
   *a = *b;
   *b = temp;
}
```

O/P:

```
        Before swap: p = 5  q = 4
        After swap: p = 4  q = 5
```

2. Dynamic Memory Allocation Using Double Pointer

Implement a function void allocateArray(int **arr, int size) that dynamically allocates memory for an array of integers using a double pointer.

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

// FUnction prototype
void allocateArray(int **arr, int size);
```

```
int main()
{
    int *arr = 0;
    int size = 5;

    //Call the function to allocate array
    allocateArray(&arr, size);

    for (int i = 0; i < size; i++)
        arr[i] = i + 1;

    printf("Array elements: ");
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");

    free(arr);
    return 0;
}

/*
Name: allocateArray()
Return Type: void
Parameter:(data type of each parameter): int** and int
Short description: it is used to allocate array
*/
```

```
// Function to allocate array
void allocateArray(int **arr, int size)
{
   *arr = (int *)malloc(size * sizeof(int));
   if (*arr == NULL)
   {
      printf("Memory allocation failed\n");
      exit(1);
   }
}
```

O/P:

```
        Array elements: 1 2 3 4 5
```

3. Modify a String Using Double Pointer

Write a function void modifyString(char **str) that takes a double pointer to a string, dynamically allocates a new string, assigns it to the pointer, and modifies the original string.

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

// Function prototype
void modifyString(char **str);

int main()
```

```
{
    char *str = 0;

    // Call the function to modify the string
    modifyString(&str);

    printf("Modified string: %s\n", str);

    free(str);
    return 0;
}

/*
Name: modifyString()
Return Type: void
Parameter:(data type of each parameter): char**
Short description: it is used to modify a string using double pointer
*/

// Function to modify a string using double pointer
void modifyString(char **str)
{
    *str = (char *)malloc(50 * sizeof(char));

    if (*str == NULL)
    {
        printf("Memory allocation failed\n");
        exit(1);
    }
    strcpy(*str, "Good day");
}
```

O/P:

```
        Modified string: Good day
```

4. Pointer to Pointer Example

Create a simple program that demonstrates how to use a pointer to a pointer to access and modify the value of an integer.

```
#include <stdio.h>

int main()
{
    int a = 20;
    int *p = &a;
    int **pp = &p;

    printf("Initial value: %d\n", a);

    **pp = 50;

    printf("Modified value: %d\n", a);
    printf("Accessed value through pp: %d\n", **pp);
    printf("Accessed value through p: %d\n", *p);
    return 0;
}
```

O/P:

```
        Initial value: 20
        Modified value: 50
        Accessed value through pp: 50
        Accessed value through p: 50
```

5. 2D Array Using Double Pointer

Write a function int** create2DArray(int rows, int cols) that dynamically allocates memory for a 2D array of integers using a double pointer and returns the pointer to the array.

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

// Function prototype
int** create2DArray(int rows, int cols);

int main()
{
    int r, c;

    printf("Enter the number of rows and columns: ");
    scanf("%d %d", &r, &c);
    int** a = create2DArray(r, c);
    if (a == NULL)
        return 1;

    printf("2D array elements:\n");
    for (int i = 0; i < r; i++)
    {
        for (int j = 0; j < c; j++)
        {
            a[i][j] = i * c + j;
            printf("%d ", a[i][j]);
        }
        printf("\n");
    }
    for (int i = 0; i < r; i++)
        free(a[i]);
    free(a);
    return 0;
}

/*
Name: create2DArray()
Return Type: int**
Parameter:(data type of each parameter): int and int
Short description: it is used to create a 2D array
*/
```

```
// Function to create a 2D array
int** create2DArray(int rows, int columns)
{
   int** arr = (int**)malloc(rows * sizeof(int*));
   if (arr == NULL)
   {
      printf("Memory allocation failed for rows\n");
      return 0;
   }

   for (int i = 0; i < rows; i++)
   {
      arr[i] = (int*)malloc(columns * sizeof(int));
      if (arr[i] == NULL)
      {
         printf("Memory allocation failed for row %d\n", i);
         for (int j = 0; j < i; j++)
            free(arr[j]);
         free(arr);
         return 0;
      }
   }
   return arr;
}
```

O/P:

```
        Enter the number of rows and columns: 3 3
```

```
2D array elements:
0 1 2
3 4 5
6 7 8
```

6. Freeing 2D Array Using Double Pointer

Implement a function void free2DArray(int **arr, int rows) that deallocates the memory allocated for a 2D array using a double pointer.

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

// Function prototype
void free2DArray(int **arr, int rows);
int** create2DArray(int rows, int columns);

int main()
{
   int r, c;

   printf("Enter the number of rows and columns: ");
   scanf("%d %d", &r, &c);

   int** a = create2DArray(r, c);
   if (a == NULL)
      return 1;
```

```
    printf("2D array elements: \n");
    for (int i = 0; i < r; i++)
    {
        for (int j = 0; j < c; j++)
        {
            a[i][j] = i * c + j;
            printf("%d ", a[i][j]);
        }
        printf("\n");
    }

    // Call the function to free the 2D array
    free2DArray(a, r);

    printf("Memory has been deallocated\n");
    return 0;
}

/*
Name: free2DArray()
Return Type: void
Parameter:(data type of each parameter): int** and int
Short description: it is used to free a dynamically allocated 2D array
*/

// Function to free a dynamically allocated 2D array
void free2DArray(int **arr, int rows)
```

```
{
    if (arr == NULL)
        return;

    // Free each row
    for (int i = 0; i < rows; i++)
    {
        if (arr[i] != NULL)
            free(arr[i]);
    }

    // Free the array of row pointers
    free(arr);
}

/*
Name: create2DArray()
Return Type: int**
Parameter:(data type of each parameter): int and int
Short description: it is used to create a 2D array
*/

// Function to create a 2D array
int** create2DArray(int rows, int columns)
{
    int** arr = (int**)malloc(rows * sizeof(int*));
    if (arr == NULL)
    {
        printf("Memory allocation failed for rows\n");
        return 0;
    }

    for (int i = 0; i < rows; i++)
    {
        arr[i] = (int*)malloc(columns * sizeof(int));
        if (arr[i] == NULL)
        {
            printf("Memory allocation failed for row %d\n", i);
            for (int j = 0; j < i; j++)
                free(arr[j]);
            free(arr);
            return 0;
        }
    }
    return arr;
}
```

O/P:

```
Enter the number of rows and columns: 2 2
2D array elements:
0 1
2 3
Memory has been deallocated
```

7. Pass a Double Pointer to a Function

Write a function void setPointer(int **ptr) that sets the pointer passed to it to point to a dynamically allocated integer.

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

// Function prototype
void setPointer(int **ptr);

int main()
{
    int *p = 0;

    // Call the function
    setPointer(&p);

    if (p != NULL)
    {
        printf("Before update: %d\n", *p);
        *p = 42;
        printf("After update: %d\n", *p);
        free(p);
        printf("Memory deallocated\n");
    }
    return 0;
}
```

```
/*
Name: setPointer()
Return Type: void
Parameter:(data type of each parameter): int**
Short description: it is used to set the pointer
*/

// Function to set the pointer
void setPointer(int **ptr)
{
   if (ptr == 0)
      return;
   *ptr = (int *)malloc(sizeof(int));
   if (*ptr == NULL)
   {
      printf("Memory allocation failed\n");
      return;
   }

   // Set the allocated integer to a default value
   **ptr = 1;
   printf("Memory allocated for the integer and initialized to %d\n", **ptr);
}
```

O/P:

```
        Memory allocated for the integer and initialized to 1
        Before update: 1
```

```
After update: 42
Memory deallocated
```

8. Dynamic Array of Strings

Create a function void allocateStringArray(char ***arr, int n) that dynamically allocates memory for an array of n strings using a double pointer.

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

// Function prototype
void allocateStringArray(char ***arr, int n);

int main()
{
   char **arr;
   int n = 5;

   // Call the function to allocate memory for 5 strings
   allocateStringArray(&arr, n);

   // Ex: Assigning strings to allocated memory
   strcpy(arr[0], "Hi");
   strcpy(arr[1], "this");
   strcpy(arr[2], "is");
   strcpy(arr[3], "Nanditha");
```

```
    strcpy(arr[4], "M");

    for (int i = 0; i < n; i++)
        printf("arr[%d] = %s\n", i, arr[i]);

    for (int i = 0; i < n; i++)
        free(arr[i]);
    free(arr);
    return 0;
}

/*
Name: allocateStringArray()
Return Type: void
Parameter:(data type of each parameter): char*** and int
Short description: it is used to dynamically allocate memory for an array of n
strings
*/

// Function to dynamically allocate memory for an array of n strings
void allocateStringArray(char ***arr, int n)
{
    *arr = (char **)malloc(n * sizeof(char *));
    if (*arr == NULL)
    {
        printf("Memory allocation failed\n");
        return;
    }
    for (int i = 0; i < n; i++)
    {
        (*arr)[i] = (char *)malloc(100 * sizeof(char));
        if ((*arr)[i] == NULL)
        {
            printf("Memory allocation failed for string %d\n", i);
            return;
        }
    }
    printf("Memory allocated successfully for %d strings\n", n);
}
```

O/P:

```
        Memory allocated successfully for 5 strings
        arr[0] = Hi
        arr[1] = this
        arr[2] = is
        arr[3] = Nanditha
        arr[4] = M
```

9. String Array Manipulation Using Double Pointer

Implement a function void modifyStringArray(char **arr, int n) that modifies each string in an array of strings using a double pointer.

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

```
#include <stdlib.h>

// Function prototype
void modifyStringArray(char **arr, int n);

int main()
{
    int n = 2;
    char *arr[] = {  strdup("Good"), strdup("day") };

    printf("Before modification:\n");
    for (int i = 0; i < n; i++)
        printf("%s\n", arr[i]);

    // Call the function
    modifyStringArray(arr, n);

    printf("\nAfter modification:\n");
    for (int i = 0; i < n; i++)
        printf("%s\n", arr[i]);
    return 0;
}

/*
Name: modifyStringArray()
Return Type: void
Parameter:(data type of each parameter): char** and int
```

```
Short description: it is used to modify each string in the array
*/

// Function to modify each string in the array
void modifyStringArray(char **arr, int n)
{
   for (int i = 0; i < n; i++)
   {
      int len = strlen(arr[i]);
      char *new = (char *)malloc(len + 10);
      if (new == NULL)
      {
         perror("Memory allocation failed");
         exit(1);
      }
      strcpy(new, arr[i]);
      strcat(new, " modified string array");

      free(arr[i]);
      arr[i] = new;
   }
}
```

O/P:

```
        Before modification:
        Good
        day
```

```
After modification:
Good modified string array
day modified string array
```

Function Pointers

1. Basic Function Pointer Declaration

Write a program that declares a function pointer for a function int add(int, int) and uses it to call the function and print the result.

```
#include <stdio.h>

// Function prototype
int add(int a, int b);

int main()
{
    // Declare a function pointer
    int (*fptr)(int, int);

    // Assign the address of the `add` function to the pointer
    fptr = &add;

    // Use the function pointer to call the `add` function
    int result = fptr(20, 10);
    printf("Result = %d\n", result);
    return 0;
}

/*
Name: add()
Return Type: int
Parameter:(data type of each parameter): int and int
Short description: it is used to return the sum of two numbers
*/

// Function to return the sum of two numbers
int add(int a, int b)
{
    return a + b;
}
```

O/P:

```
        Result = 30
```

2. Function Pointer as Argument

Implement a function void performOperation(int (*operation)(int, int), int a, int b) that takes a function pointer as an argument and applies it to two integers, printing the result.

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

```
// Function prototypes
int add(int a, int b);
int sub(int a, int b);
void performOperation(int (*operation)(int, int), int a, int b);

int main()
{

    int x = 20, y = 10;

    printf("Addition\n");
    performOperation(add, x, y);

    printf("Subtraction\n");
    performOperation(sub, x, y);

    return 0;
}

/*
Name: add()
Return Type: int
Parameter:(data type of each parameter): int and int
Short description: it is used to return the sum of two numbers
*/

// Function to return the sum of two numbers
```

```
int add(int a, int b)
{
   return a + b;
}

/*
Name: sub()
Return Type: int
Parameter:(data type of each parameter): int and int
Short description: it is used to return the difference of two numbers
*/

int sub(int a, int b)
{
   return a - b;
}

/*
Name: performOperation()
Return Type: void
Parameter:(data type of each parameter): int, int and int
Short description: it is used to perform an operation using a function pointer
*/

// Function to perform an operation using a function pointer
void performOperation(int (*operation)(int, int), int a, int b)
{
   if (operation == NULL)
   {
       printf("Invalid operation\n");
       return;
   }
   int result = operation(a, b);
   printf("Result = %d\n", result);
}
```

O/P:

```
        Addition
        Result = 30
        Subtraction
        Result = 10
```

3. Function Pointer Returning Pointer

Write a program with a function int* max(int *a, int *b) that returns a pointer to the larger of two integers, and use a function pointer to call this function.

```
#include <stdio.h>

// Function prototype
int* max(int *a, int *b);

int main()
{
    int x = 10, y = 30;
    int* (*fptr)(int*, int*);

    fptr = &max;

    // Use the function pointer to call the function
    int *larger = fptr(&x, &y);

    printf("Larger value = %d\n", *larger);

    return 0;
}

/*
Name: max()
Return Type: int*
Parameter:(data type of each parameter): int* and int*
Short description: it is used to return a pointer to the larger of two integers
*/

// Function to return a pointer to the larger of two integers
int* max(int *a, int *b)
{
    if (*a > *b)
        return a;
    else
        return b;
```

```
}
```

O/P:

```
        Larger value = 30
```

4. Function Pointer with Different Functions

Create a program that defines two functions int add(int, int) and int multiply(int, int) and uses a function pointer to dynamically switch between these functions based on user input.

```
#include <stdio.h>

// Function prototypes
int add(int a, int b);
int mul(int a, int b);

int main()
{
   int (*operation)(int, int);
   int option, x, y;

   printf("Enter the two numbers: ");
   scanf("%d %d", &x, &y);

   printf("Choose an operation:\n1. Addition\n2. Multiplication\n");
   printf("Enter the option: ");
   scanf(" %d", &option);
```

```
    if (option == 1)
        operation = add;
    else if (option == 2)
        operation = mul;
    else
    {
        printf("Invalid option\n");
        return 1;
    }

    // Call the selected function via the function pointer
    int result = operation(x, y);

    if (option == 1)
        printf("The sum of %d and %d is: %d\n", x, y, result);
    else
        printf("The product of %d and %d is: %d\n", x, y, result);

    return 0;
}

/*
Name: add()
Return Type: int
Parameter:(data type of each parameter): int and int
Short description: it is used to return the sum of two numbers
```

```
*/

// Function to return the sum of two numbers
int add(int a, int b)
{
   return a + b;
}

/*
Name: mul()
Return Type: int
Parameter:(data type of each parameter): int and int
Short description: it is used to return the multiplication of two numbers
*/

// Function to return the multiplication of two numbers
int mul(int a, int b)
{
   return a * b;
}
```

O/P:

```
        Enter the two numbers: 5 4
        Choose an operation:
        1. Addition
        2. Multiplication
        Enter the option: 1
```

```
The sum of 5 and 4 is: 9

Enter the two numbers: 2 3
Choose an operation:
1. Addition
2. Multiplication
Enter the option: 2
The product of 2 and 3 is: 6
```

5. Array of Function Pointers

Implement a program that creates an array of function pointers for basic arithmetic operations (addition, subtraction, multiplication, division) and allows the user to select and execute one operation.

```
#include <stdio.h>

// Function prototypes
int add(int a, int b);
int sub(int a, int b);
int mul(int a, int b);
int div(int a, int b);

int main()
{
  int (*operations[])(int, int) = {add, sub, mul, div};

  int option, x, y, result;
```

```
    printf("Enter the two numbers: ");
    scanf("%d %d", &x, &y);

    printf("Choose an operation:\n0. Addition\n1. Subtraction\n2. Multiplication\n3. Division\n");
    printf("Enter the option: ");
    scanf(" %d", &option);

    if (option >= 0 && option <= 3)
    {
        result = operations[option](x, y);
        const char *opNames[] = {"Addition", "Subtraction", "Multiplication", "Division"};
        printf("The result of %s is: %d\n", opNames[option], result);
    }
    else
        printf("Invalid option\n");
    return 0;
}

/*
Name: add()
Return Type: int
Parameter:(data type of each parameter): int and int
Short description: it is used to return the sum of two numbers
*/
```

```
// Function to return the sum of two numbers
int add(int a, int b)
{
   return a + b;
}

/*
Name: sub()
Return Type: int
Parameter:(data type of each parameter): int and int
Short description: it is used to return the difference of two numbers
*/

// Function to return the difference of two numbers
int sub(int a, int b)
{
   return a - b;
}

/*
Name: mul()
Return Type: int
Parameter:(data type of each parameter): int and int
Short description: it is used to return the multiplication of two numbers
*/

// Function to return the multiplication of two numbers
```

```
int mul(int a, int b)
{
   return a * b;
}

/*
Name: div()
Return Type: int
Parameter:(data type of each parameter): int and int
Short description: it is used to return the division of two numbers
*/

// Function to return the division of two numbers
int div(int a, int b)
{
   if (b == 0)
   {
      printf("Division by zero is not allowed\n");
      return 0;
   }
   return a / b;
}
```

O/P:

```
        Enter the two numbers: 10 5
        Choose an operation:
        0. Addition
```

```
1. Subtraction
2. Multiplication
3. Division
Enter the option: 2
The result of Multiplication is: 50
```

6. Using Function Pointers for Sorting

Write a function void sort(int *arr, int size, int (*compare)(int, int)) that uses a function pointer to compare elements, allowing for both ascending and descending order sorting.

```c
#include <stdio.h>


// Function prototypes
int ascending(int a, int b);
int descending(int a, int b);
void sort(int *arr, int size, int (*compare)(int, int));
void printArray(int *arr, int size);


int main()
{
   int arr[] = {5, 3, 1, 2, 4};
   int size = sizeof(arr) / sizeof(arr[0]);


   printf("Original array:\n");
   printArray(arr, size);
```

```
    printf("Sorting in ascending order:\n");
    sort(arr, size, ascending);
    printArray(arr, size);

    printf("Sorting in descending order:\n");
    sort(arr, size, descending);
    printArray(arr, size);

    return 0;
}

/*
Name: ascending()
Return Type: int
Parameter:(data type of each parameter): int and int
Short description: it is used to return the sorting of elements in ascending order
*/

// Function to return the sorting of elements in ascending order
int ascending(int a, int b)
{
    return a > b;
}

/*
Name: descending()
Return Type: int
```

```
Parameter:(data type of each parameter): int and int
Short description: it is used to return the sorting of elements in descending order
*/

// Function to return the sorting of elements in descending order
int descending(int a, int b)
{
    return a < b;
}

/*
Name: sort()
Return Type: void
Parameter:(data type of each parameter): int*, int and int
Short description: it is used to sort of elements
*/

// Function to sort of elements
void sort(int *arr, int size, int (*compare)(int, int))
{
    for (int i = 0; i < size - 1; i++)
    {
        for (int j = 0; j < size - i - 1; j++)
        {
            if (compare(arr[j], arr[j + 1]))
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

/*
Name: printArray()
Return Type: void
Parameter:(data type of each parameter): int* and int
Short description: it is used to print the array elements
*/

// Function to print the array elements
void printArray(int *arr, int size)
{
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
```

O/P:

```
          Original array:
          5 3 1 2 4
          Sorting in ascending order:
          1 2 3 4 5
```

```
Sorting in descending order:
5 4 3 2 1
```

7. Callback Function

Create a program with a function void execute(int x, int (*callback)(int)) that applies a callback function to an integer and prints the result. Demonstrate with multiple callback functions (e.g., square, cube).

```
#include <stdio.h>


//Function prototypes
int square(int x);
int cube(int x);
void execute(int x, int (*callback)(int));


int main()
{
   int number = 8;


   printf("Execution of square function:\n");
   execute(number, square);


   printf("Execution of cube function:\n");
   execute(number, cube);


   return 0;
}
```

```
/*
Name: square()
Return Type: int
Parameter:(data type of each parameter): int
Short description: it is used to return the square of a number
*/

// Function to return the square of a number
int square(int x)
{
   return x * x;
}

/*
Name: cube()
Return Type: int
Parameter:(data type of each parameter): int
Short description: it is used to return the cube of a number
*/

// Function to return the cube of a number
int cube(int x)
{
   return x * x * x;
}

/*
```

```
Name: execute()

Return Type: void

Parameter:(data type of each parameter): int and int

Short description: it is used to apply the callback to the integer and print the result

*/


//Function to apply the callback to the integer and print the result

void execute(int x, int (*callback)(int))

{

   int result = callback(x);

   printf("Result = %d\n", result);

}
```

O/P:

```
        Execution of square function:

        Result = 64

        Execution of cube function:

        Result = 512
```

8. Menu System Using Function Pointers

Implement a simple menu system where each menu option corresponds to a different function, and a function pointer array is used to call the selected function based on user input.

```
#include <stdio.h>


// Function prototypes
```

```
void option1();
void option2();
void option3();
void displayMenu();

int main()
{
    void (*menuFunctions[])(void) = {option1, option2, option3};

    int op;

    while (1)
    {
        displayMenu();
        scanf("%d", &op);

        if (op >= 1 && op <= 3)
        {
            menuFunctions[op - 1]();
            if (op == 3)
                break;
        }
        else
            printf("Invalid option\n");
    }
    return 0;
}
```

```
void option1()
{
    printf("Selected option 1\n");
}

void option2()
{
    printf("Selected option 2\n");
}

void option3()
{
    printf("Selected option 3\n");
}

void displayMenu()
{
    printf("Menu:\n1. Option 1\n2. Option 2\n3. Option 3\n");
    printf("Select an option: ");
}
```

```
O/P:
Menu:
1. Option 1
2. Option 2
3. Option 3
Select an option: 2
```

```
Selected option 2
Menu:
1. Option 1
2. Option 2
3. Option 3
Select an option: 3
Selected option 3
```

9. Dynamic Function Selection

Write a program where the user inputs an operation symbol (+, -, *, /) and the program uses a function pointer to call the corresponding function.

```
#include <stdio.h>

// Function prototypes
int add(int a, int b);
int sub(int a, int b);
int mul(int a, int b);
int div(int a, int b);

int main()
{
    int (*operation)(int, int) = 0;
    char op;
    int a, b;

    // User input
    printf("Enter the two numbers: ");
    scanf("%d %d", &a, &b);

    printf("Enter the operation (+, -, *, /): ");
    scanf(" %c", &op);

    switch (op)
    {
        case '+': operation = add;
                  break;
        case '-': operation = sub;
                  break;
        case '*': operation = mul;
                  break;
        case '/': operation = div;
                  break;
        default: printf("Invalid operation\n");
                 break;
    }

    // Call the function using the function pointer
    int result = operation(a, b);
    printf("The result of %d %c %d is: %d\n", a, op, b, result);
    return 0;
}

/*
```

```
Name: add()
Return Type: int
Parameter:(data type of each parameter): int and int
Short description: it is used to return the sum of two numbers
*/

// Function to return the sum of two numbers
int add(int a, int b)
{
   return a + b;
}

/*
Name: sub()
Return Type: int
Parameter:(data type of each parameter): int and int
Short description: it is used to return the difference of two numbers
*/

// Function to return the difference of two numbers
int sub(int a, int b)
{
   return a - b;
}

/*
Name: mul()
```

```
Return Type: int
Parameter:(data type of each parameter): int and int
Short description: it is used to return the multiplication of two numbers
*/

// Function to return the multiplication of two numbers
int mul(int a, int b)
{
    return a * b;
}

/*
Name: div()
Return Type: int
Parameter:(data type of each parameter): int and int
Short description: it is used to return the division of two numbers
*/

// Function to return the division of two numbers
int div(int a, int b)
{
    if (b == 0)
    {
        printf("Division by zero is not allowed\n");
        return 0;
    }
    return a / b;
}
```

O/P:

```
        Enter the two numbers: 10 5
        Enter the operation (+, -, *, /): /
        The result of 10 / 5 is: 2
```

10. State Machine with Function Pointers

Design a simple state machine where each state is represented by a function, and transitions are handled using function pointers. For example, implement a traffic light system with states like Red, Green, and Yellow.

```
#include <stdio.h>

// Function prototypes
void redState();
void greenState();
void yellowState();

// Function pointer for state transition
void (*currentState)() = NULL;

int main()
{
    currentState = redState;

    for (int i = 0; i < 3; i++)
        currentState();
    return 0;
}

void redState()
{
    printf("Red light (Stop)\n");
    currentState = greenState;
}

void greenState()
{
    printf("Green light (Go)\n");
    currentState = yellowState;
}

void yellowState()
{
    printf("Yellow light (Ready)\n");
    currentState = redState;
}
```

O/P:

```
        Red light (Stop)
        Green light (Go)
        Yellow light (Ready)
```