|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Programming in C** | **Semester:** | **II** |
| **Date of Performance:** | **07 / 02 / 2025** | **DIV/ Batch No:** | **C4-1** |
| **Student Name:** | **Dhruv Pankhania** | **Roll No:** | **16010124216** |

**Experiment No: 5**

**Title: Strings and string handling functions**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| Write a program in C to demonstrate use of strings and string handling functions. |

|  |
| --- |
| **COs to be achieved:** |
| **CO3: Apply the concepts of arrays and strings.** |

|  |
| --- |
| **Theory:** |
| In C programming, a string is an array of characters terminated by a null character ('\0'). Strings are represented using character arrays. To handle strings effectively, C provides a set of built-in functions in the <string.h> library.  Key functions for string:   * strlen(): Returns the length of a string (excluding the null-terminator). * strcpy(): Copies a string from source to destination. * strncpy(): Copies up to n characters from source to destination. * strcat(): Appends one string to the end of another. * strncat(): Appends up to n characters from source to destination. * strcmp(): Compares two strings lexicographically. * strncmp(): Compares the first n characters of two strings. * strchr(): Searches for the first occurrence of a character in a string. * strrchr(): Searches for the last occurrence of a character in a string. * strstr(): Searches for the first occurrence of a substring in a string. * strtok(): Tokenizes a string into substrings based on delimiters. * sprintf(): Formats and stores a string into a character array. * sscanf(): Reads formatted input from a string and stores it in variables. * strdup(): Duplicates a string by allocating memory and copying it. * strspn(): Returns the length of the initial segment of a string containing only characters from a set. * strcspn(): Returns the length of the initial segment of a string excluding characters from a set. * strpbrk(): Searches for the first occurrence of any character from a set in a string. * strtok\_r(): A reentrant version of strtok() for thread-safe tokenization. * memcpy(): Copies memory from source to destination. * memset(): Sets a block of memory to a specified value. |

|  |
| --- |
| **Problem Statements:** |
| 1. Write a program that takes a string as input and counts the number of vowels and consonants in the string without using the inbuilt library function. Ignore spaces and punctuation. 2. Write a program to manage student records. The program will handle the following operations using the string functions provided:  * Input the student's name and grade (two strings). * Display the length of both the student's name and grade. * Copy the student's name into a new string and display it. * Concatenate a fixed string (e.g., " - Excellent Student") to the student's name and display the result. * Compare two students' names lexicographically and display which student has the lexicographically greater name. * Search for a substring in the student's name (e.g., "John" in "Johnny") and display the position of the first occurrence. * Search for a character in the grade string (e.g., 'A') and display the position of the first occurrence. * Tokenize the student's grade if it contains multiple components (e.g., "A B C") and display each component. |
|  |

|  |
| --- |
| **Code :** |
| 1.  #include<stdio.h>  *int* main(){  *int* vowel = 0, consonants = 0, specialChar = 0;  *char* str[100];    printf("Enter a string: ");    gets(str);  *int* i;    for(i = 0; str[i]!='\0'; i++){      if(str[i] == 'a' || str[i] == 'e' || str[i] == 'i' || str[i] == 'o' || str[i] == 'u' || str[i] == 'A' || str[i] == 'E' || str[i] == 'I' || str[i] == 'O' || str[i] == 'U'){        vowel++;      }      else{        if(((*int*)str[i]>65 && (*int*)str[i]<90) || ((*int*)str[i]>96 && (*int*)str[i]<123)){          consonants++;        }        else{          if(str[i]==' '){}          else{            specialChar++;          }      }}  }  printf("The Number of Vowels in the String is: %d", vowel);  printf("\nThe Number of Consonants in the String is: %d", consonants);  printf("\nThe Number of Special Characters in the String is: %d", specialChar);    return 0;  }  2. #include<stdio.h>  #include<string.h>  *int* main(){  *char* name1[100], grade[100], nameCpy[100], name2[100], subString[100];    printf("Enter the Name of the Student: ");    gets(name1);    printf("Enter the Grade of the Student: ");    gets(grade);    printf("The Length of the Student`s Name is: %d\nThe Length of the Student`s Grade is: %d", strlen(name1), strlen(grade));    strcpy(nameCpy, name1);    printf("\nStudent`s Name copied into another String: %s", nameCpy);    strcat(nameCpy, " - Excellent Student");    printf("\n%s", nameCpy);    printf("\nEnter Second Name to be Compared with the First Name: ");    gets(name2);  *int* result = strcmp(name2, name1);    if(result>0){          printf("%s is lexicographically greater than %s\n", name1, name2);        } else if (result < 0) {          printf("%s is lexicographically greater than %s\n", name2, name1);        } else {          printf("Both names are the same.\n");        }    printf("Enter a Substring to be searched in the first name: ");  gets(subString);  subString[strcspn(subString, "\n")] = 0;  *char* \*pos1 = strstr(name1, subString);  if(pos1){    printf("The first occurance of %s is at positon %d\n", subString, (pos1-name1)+1);  } else{    printf("Substring not Found in the Name.\n");  }  printf("Enter a Substring to be searched in the grade: ");  gets(subString);  subString[strcspn(subString, "\n")] = 0;  *char* \*pos2 = strstr(grade, subString);  if(pos2){    printf("The first occurance of %s is at positon %d\n", subString, (pos2-grade)+1);  } else{    printf("Substring not Found in the Name.\n");  }  *char* \*token = strtok(grade, " ");  while(token != NULL){    printf("%s\n", token);    token = strtok(NULL, " ");  }  return 0;  } |

|  |
| --- |
| **Output:** |
| 1.   2. |

|  |
| --- |
| **Post Lab Subjective/Objective type Questions:** |
| 1. In C, what will happen if you pass an uninitialized string or a string without a null terminator to any of the string handling functions (e.g., strcpy(), strlen(), strcmp())?   **Ans.** If an uninitialized string or a string without a null terminator is passed, it may lead to undefined behavior in functions like strcpy(), strlen(), and strcmp(), potentially causing a crash. Since string functions depend on the null terminator to determine where to stop their operations, they will either fail or produce errors if the terminator is missing. This can result in issues such as buffer overflow, segmentation faults, and out-of-bounds memory access.   1. In C, how does memory allocation for strings work? What are the potential risks associated with string manipulation in C, and how can buffer overflow issues be prevented?   **Ans.** In C, strings are stored as character arrays and can be allocated using fixed-size arrays (stack memory), dynamic allocation (malloc(), free(), heap memory), or string literals (read-only memory). Improper string handling can lead to buffer overflow, segmentation faults, memory leaks, and undefined behavior. Buffer overflow occurs when data exceeds allocated space, potentially corrupting memory. To prevent this, use strncpy() instead of strcpy(), fgets() instead of gets(), and always allocate sufficient memory with malloc(), ensuring proper deallocation with free(). |

|  |
| --- |
| **Conclusion:** |
| We explored the **<string.h>** library in C, which provides functions for handling and manipulating strings efficiently. Since C strings are **character arrays** terminated by **\0**, manual operations can be complex. We learned about functions like **strcpy()** for copying, **strcmp()** for comparing, and **strncat()** for concatenation. These functions simplify string processing, improve efficiency, and help prevent errors like buffer overflows. |

|  |
| --- |
| **Signature of faculty in-charge with Date:** |