**Name: Kannika Kulkarni**

**Division: F**

**Roll No: 147**

**Course: Data Structure Applications**

**Semester: 4th**

**ASSIGNMENT: 1**

**Part- A**

|  |
| --- |
| **Variables and Data Types** |
| 1.What is the difference between a variable and a data type in C programming? Provide examples to illustrate.  Ans. Data type: Specifies the kind of information (characters, integers, etc.) that a variable can store.  A variable is a designated storage place that, depending on the kind of data, has a particular value (e.g., an integer to record age). |
| 2.What is the difference between a variable and a data type in C programming? Provide examples to illustrate.  Ans. Size: Allotted memory, such as an int, which is normally 4 bytes.   Value range: Permitted values (integer numbers within a given range, for example, int).   Operations: Allowed activities (mathematical operations for numeric types, for example).   Categories:   Fundamental:   int :whole numbers (like -5 and 10).   float: Decimal numbers (such as 3.14 and -1.25).   char: A single character, such as "a" or "Z." |
| 3. How are variables declared and initialized in C programming? Provide examples of variable declarations with different data types.  Ans. Declaration:  Specifies the data type and creates a named space in memory.  Syntax: data type variable name;  Initialization :  Assigns an initial value during declaration.  Syntax: data type variable name = value;  Examples:  int age; (integer variable)  float pi = 3.14; (float variable with initial value)  char initial = 'A'; (character variable) |
| 4. Discuss the scope and lifetime of variables in C programming. What are global and local variables?  Ans. Scope defines the section of the code that contains a variable's accessibility.   There are two primary types:   Local: Excluding a block (code enclosed in curly brackets {}).   Global: Available at all times during the program.   Duration:  Relates to the duration of memory allotted to the variable.   Local variables: Only present when a block is being executed.   Global variables: Last till the end of the program.     Local: Used to avoid name conflicts within particular functions.   Global: Due to possible unforeseen negative effects throughout the program, it should only be utilized seldom. |
| 5. Explain the concept of type casting in C programming. When is type casting necessary, and how is it performed?  Ans. Casting a value explicitly from one data type to another is known as type casting.   Required at:   transferring values between incompatible types (such as float and integer).   using routines designed to handle particular data types.   Cast operator (data type) was used before the value.   age = 25 int; (float)age = float pi; |
| **OPERATORS:** |
| 1.Describe the purpose and usage of the ternary conditional operator (?:) in C programming. Provide an example demonstrating its usage.  Ans. When evaluating a condition in C, the ternary conditional operator?: returns one of two values according to whether the condition is true or false. It is frequently employed as a condensed version of an if-else statement.  Eg:int x = 10;  int y = (x > 5) ? 100 : 200  printf("Value of y: %d\n", y); |
| 2.Discuss the bitwise operators available in C programming. Explain their usage with suitable examples.  Ans.  i]Bitwise operators :These operators work with specific bits that are contained in a variable. The bitwise operators that C offers are as follows:    ii]Bitwise AND (&):  Syntax: operand1 & operand2  Performs a bitwise AND operation on corresponding bits of operand1 and operand2.  Example: int result = 5 & 3;    iii]Bitwise OR (|):  Syntax: operand1 | operand2  Performs a bitwise OR operation on corresponding bits of operand1 and operand2.  Example: int result = 5 | 3;    iv]Bitwise XOR (^):  Syntax: operand1 ^ operand2  Performs a bitwise XOR (exclusive OR) operation on corresponding bits of operand1 and operand2.  Example: int result = 5 ^ 3;    V]Bitwise NOT (~):  Syntax: ~operand  Performs a bitwise NOT (complement) operation, which flips all the bits of operand.  Example: int result = ~5; |
| 3. Explain the difference between the postfix and prefix increment operators (++) in C programming. Provide examples to illustrate.  Ans.  i]Operator for Postfix Increment (x++):   The variable's current value is returned first in the postfix increment operator, and the value is subsequently increased.   For instance:  int x = 5;  int y = x++;  printf("x: %d, y: %d\n", x, y);    ii]Operator for Prefix Increment (++x):   The variable's value is increased first via the prefix increment operator, and the increased value is then returned.   For instance:  int x = 5;  int y = ++x;  printf("x: %d, y: %d\n", x, y); |
| 4. What is the significance of the logical AND (&&) and logical OR (||) operators in C programming? How are they used in conditional expressions?  Ans.  i] Logical AND (&&):  The logical AND operator returns true (1) if both of its operands are true, otherwise it returns false (0).  int x = 5;  int y = 10;  if (x > 0 && y > 0)      printf("Both x and y are positive\n");    ii]Logical OR (||):  The logical OR operator returns true (1) if at least one of its operands is true, otherwise it returns false (0).  int x = 5;  int y = 10;  if (x == 0 || y == 0)      printf("Either x or y is zero\n"); |
| 5. Discuss the concept of operator precedence and associativity in C programming. Provide examples to demonstrate how they affect expression evaluation  Ans.  i]Operator Order of Priority:   The order in which operators are evaluated in an expression is specified by operator precedence. Higher precedence operators are evaluated before lower precedence operators.   For instance, multiplication (\*) is evaluated as 2 + (3 \* 4) since it has a greater precedence than addition (+). This yields a value of 14.   To force evaluation in a specific sequence and override the default precedence, use parentheses ().   For instance: int result = 2 + 3 \* 4;  Ii]Associativity of Operators:   The sequence in which operators with the same precedence are evaluated is specified by operator associativity. It can be associative from right to left or left to right, or left to right or left to left.   For instance, because the addition operator (+) is associative from left to right, 2 + 3 + 4 is evaluated as (2 + 3) + 4, which yields 9.   The assignment (=) operator is associative from right to left, though, thus a = b = 5 is evaluated as a = (b = 5), which assigns the value 5 to both a and b.   For instance: int result1 = 2 + 3 + 4;  int a, b;  a = b = 5; |
| **CONTROL STRUCTURES:** |
| 1.Describe the purpose and usage of the switch statement in C programming. How does it differ from the if-else statement?  Ans The switch statement is used to regulate the flow of execution according to a variable or expression's value. It offers a different approach to writing several if-else statements that examine the same variable's value.  When several conditional branches are created based on the value of a single variable, the switch statement makes the code clearer and more concise.   It offers a more effective substitute for several if-else statements, particularly in situations when there are a lot of potential values.    Switch statement differs from if else in manner:  The switch statement compares the value of a single expression against multiple constant values, while the if-else statement evaluates different expressions or conditions independently. |
| 2. Explain the concept of nested control structures in C programming. Provide an example demonstrating nested if-else statements.  Ans: The term "nested control structures" describes the use of control structures (like if-else, switch, while, for, etc.) inside other control structures. This makes it possible to develop looping and decision-making structures that are more intricate.  Eg:  {      int x = 10;      int y = 20;      if (x > 0) {          if (y > 0) {              printf("Both x and y are positive.\n");          } else {              printf("x is positive, but y is non-positive.\n");          }      } else {          printf("x is non-positive.\n");      }  } |
| 3. Discuss the role of the break and continue statements in loop control in C programming. Provide examples to illustrate their usage.  Ans: ] Statement of Break:   An early break in a loop's execution can be achieved with the break statement.   A loop is instantly broken when a break statement is met, and control is moved to the sentence that comes after the loop.  Eg:  for (int i = 0; i < 10; i++) {      if (i == 5) {          break; // Exit the loop when i equals 5      }      printf("%d ", i);  }  ii]Statement Continued:   To move on to the following iteration of a loop and skip the remainder of the current iteration, use the continue statement.   Control returns to the loop's condition or increment/decrement statement upon encountering a continue statement, skipping the remaining statements in the loop body.  Eg:  for (int i = 0; i < 10; i++) {      if (i % 2 == 0) {          continue; // Skip even numbers      }      printf("%d ", i);  } |
| 4.What are the advantages of using the for loop over the while loop in C programming? Provide examples comparing the two.  Ans: One Location for Initialization, Condition, and Increment:    The for loop makes the loop structure more clear-cut and understandable by allowing you to declare the increment/decrement, the loop condition, and the loop variables all on one line.   In particular, for straightforward loops with distinct setup, condition, and increment/decrement logic, this might result in clearer, more understandable code.   More Suitable for Looping Through Ranges:     When iterating over a range of data, like iterating over an array or completing a predetermined amount of iterations, the for loop is frequently utilized.   Because of its design, it works well in situations where there is a need for the loop control variable to be initialized, tested against a condition, and updated in a predictable way.  Eg:      // Example using for loop      printf("Using for loop:\n");      for (int i = 0; i < 5; i++) {          printf("%d ", i);      }      printf("\n");        // Example using while loop      printf("Using while loop:\n");      int j = 0;      while (j < 5) {          printf("%d ", j);          j++;      } |
| 5. Explain the concept of short-circuit evaluation in C programming. How does it affect the evaluation of logical expressions in if statements?  Ans: The idea of short-circuit evaluation states that a logical expression's evaluation should end as soon as its result can be ascertained, without considering every component of the expression. This implies that the remaining portions of the expression are not examined if the result of the expression can be ascertained by evaluating only a piece of it.  Eg:  If the left operand of && is evaluated to false, the entire expression is false, and the right operand is not evaluated because the result is already determined. This is because for the entire expression to be true, both operands must be true.  Ex: if (x > 0 && y > 0) |
| **ARRAYS:** |
| 1.Explain the concept of arrays in C programming. How are arrays declared and initialized?  Ans. Declaration: To declare an array, we specify the data type of the elements it will hold and the size of the array. It follows the syntax: data type array name [array size];  Initialization: Arrays can be initialized during declaration or later in the program. Initialization assigns initial values to the elements of the array. There are two ways to initialize arrays:  int numbers[5] = {1, 2, 3, 4, 5} |
| 2.Discuss the difference between a one-dimensional array and a multi dimensional array in C programming. Provide examples of both.  Ans. One-Dimensional Array (1D): A one-dimensional array is a collection of elements stored in a contiguous memory block.  It's like a list of items where each item is accessed using a single index.  Declared with a single pair of square brackets [].  Declaration and initialization of a 1D array  int numbers[5] = {1, 2, 3, 4, 5}  char vowels[5] = {'a', 'e', 'i', 'o', 'u'}  Multi-Dimensional Array: A multi-dimensional array is an array of arrays, where each element of the array is itself an array.It's like a table or grid, where data is arranged in rows and columns.Declared with multiple pairs of square brackets [][], indicating the dimensions.  Declaration and initialization of a 2D array (3x3)  int matrix[3][3] = {1, 2, 3},{4, 5, 6},{7, 8, 9}  3.Describe the process of accessing array elements in C programming. How are array indices used to access elements?  Ans. Arrays are often accessed using loops, such as for or while, to iterate through each element sequentially.  Normally we use for Loop  for (int i = 0; i < 5; i++)  printf("%d ", numbers[i]); |
| 4. What is the significance of the null character ('\0') in C strings? How is it used to determine the end of a string?  Ans. Strings are represented as arrays of characters terminated by a null character '\0'.The null character marks the end of the string, indicating where the string's contents conclude.It's essential for string manipulation functions to determine the string's length and to prevent buffer overflows when working with strings. |
| 5. Explain the concept of dynamic memory allocation for arrays in C programming. How are dynamic arrays allocated and deallocated?  Ans. Dynamic memory allocation in C programming refers to the process of allocating memory during program execution rather than at compile time. This allows for flexibility in memory usage, especially when the size of data structures like arrays cannot be determined beforehand.  Allocation using malloc(), calloc(), realoc()  Deallocation using free() |
| **POINTERS:** |
| 1. Describe the purpose and usage of pointers in C programming. How are pointers declared and initialized?  Ans. Pointers in C programming are variables that store memory addresses. They provide a way to indirectly access the memory location of a variable, allowing for dynamic memory allocation, efficient manipulation of data structures, and interaction with hardware.  Dynamic Memory Allocation  Efficient data structure. |
| 2. Explain the concept of pointer arithmetic in C programming. Provide examples to illustrate addition and subtraction operations on pointers.  Ans. Pointer arithmetic in C programming allows you to perform arithmetic operations directly on pointers. This is particularly useful when working with arrays, as it provides a convenient way to navigate through elements in memory.  Addition Operation  Subtraction Operation  Increment and Decrement Operation |
| 3. Discuss the difference between pass by value and pass by reference in function arguments using pointers in C programming. Provide examples to illustrate both approaches.  Pass by value: In pass by value, a copy of the variable's value is passed to the function. Any changes made to the parameter within the function do not affect the original variable.  #include <stdio.h>  void increment(int num)  num++;  int main()  {  int num = 10;  increment(num);  printf("Original value of num: %d\n", num);  return 0;  }  Pass by Reference using Pointers: In pass by reference, the memory address of the variable is passed to the function using a pointer. Any changes made to the parameter within the function affect the original variable.  #include <stdio.h>  void incrementByRef(int \*ptr)  (\*ptr)++;  int main()  {  int num = 10;  incrementByRef(&num);  printf("Modified value of num: %d\n", num);  return 0;  } |
| 4. Describe the concept of NULL pointers in C programming. How are NULL pointers used and checked for in programs?  Ans. NULL pointer is a pointer that does not point to any memory location. It's a special value assigned to pointers to indicate that they are not currently pointing to a valid object or memory address. The concept of NULL pointers is essential for error handling and indicating the absence of a valid reference. |
| 5. Explain the role of pointers in dynamic memory allocation in C programming. How are pointers used to allocate and deallocate memory dynamically?  Ans. Pointers play a crucial role in dynamic memory allocation in C programming. They are used to allocate and deallocate memory dynamically, allowing programs to request memory from the heap at runtime.  Allocation of memory  Deallocation of memory  Reallocation of memory |
| **STRINGS:** |
| 1. Discuss the concept of strings in C programming. How are strings represented and manipulated in C?  Ans. Strings are sequences of characters terminated by a null character '\0'. C does not have a built-in string data type like some other programming languages, so strings are typically represented as arrays of characters.   1. String Representation:Strings are represented as arrays of characters, where each character is stored in a consecutive memory location.The last character of the string is always the null character '\0', which marks the end of the string. 2. String Manipulation:C provides a library of string manipulation functions declared in the <string.h> header, such as strcpy(), strcat(), strlen(), strcmp(), etc.These functions allow you to perform various operations on strings like copying, concatenating, finding the length, and comparing strings. |
| 2. Explain the difference between character arrays and string literals in C programming. Provide examples to illustrate both concepts.  Ans.  1. Character Arrays:Character arrays are arrays of characters that can be explicitly declared and manipulated by the programmer.They are mutable, meaning their contents can be modified after declaration.Character arrays are terminated by a null character '\0' to mark the end of the string.  2. String Literals:  String literals are sequences of characters enclosed in double quotes.They are automatically null-terminated by the compiler, so you don't need to include the null character explicitly.String literals are immutable, meaning their contents cannot be modified after compilation. |
| 3. Describe common string manipulation functions available in the C standard library. Provide examples of functions like strlen, strcpy, strcat, and strcmp.  Ans. The C standard library provides several string manipulation functions declared in the <string.h> header.  strlen(): size\_t strlen(const char \*str): Returns the length of the null-terminated string str.  strcpy(): char \*strcpy(char \*dest, const char \*src): Copies the null-terminated string src to dest.  strcat(): char \*strcat(char \*dest, const char \*src): Concatenates the null-terminated string src to the end of dest.  strcmp(): int strcmp(const char \*str1, const char \*str2): Compares the null-terminated strings str1 and str2. |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Part- B**

|  |
| --- |
| **1.PRINT HELLO WORLD**  #include<stdio.h>  #include<stdlib.h>  int main()  {      FILE \*fptr;      fptr=fopen("Text.txt","w");      fputs("Hello World",fptr);      fclose(fptr);  }  **OUTPUT** |
| **2.FACTORIAL OF THE NUMBER**  #include<stdio.h>  #include<stdlib.h>  #include"C:\Users\shash\OneDrive\Desktop\vscode\.vscode\randG.h"  int main()  {      FILE \*fptr;      int num;      long long int mul=1;      fptr=fopen("text.txt","w");      fprintf(fptr,"%d",rand()%12+1);      fclose(fptr);      fptr=fopen("text.txt","r");      while(fscanf(fptr,"%d",&num)!=EOF);      fclose(fptr);      for(int i=num;i>0;i--)      {          mul=mul\*i;      }      fptr=fopen("text.txt","a");      fputs("\nThe factorial of the number is: ",fptr);      fprintf(fptr,"%lld",mul);      fclose(fptr);  }  **OUTPUT** |
| **3.PRIME NUMBERS** |
| #include<stdio.h>  #include<stdlib.h>  #include<math.h>  #include"C:\Users\shash\OneDrive\Desktop\vscode\.vscode\randG.h"  int prime1(int n)  {      for(int i=2;i<sqrt(n);i++)  {          if(n%i==0)              return 0;      }      return 1;  }  int main()  {      int arr[10],k=0, num;      FILE \*fptr;      randonGenrator(20);      fptr = fopen("text.txt", "r");      if (fptr == NULL)  {          printf("File not available.");          exit(0);      }      while(fscanf(fptr, "%d", &num) != -1)      {          if(prime1(num)){              arr[k]=num;              k++;          }        }      fclose(fptr);      fptr=fopen("text.txt","a");      fprintf(fptr, "\n");      fputs("Prime no.s are : ", fptr);      for(int i=0;i<k;i++)      {          fprintf(fptr, "%d ",arr[i]);      }      fclose(fptr);  return 0;  } |
| OUTPUT |
| **4.FIBONACCI**  #include<stdio.h>  #include"C:\Users\shash\OneDrive\Desktop\vscode\.vscode\randG.h"  int main()  {      int i;      int a=-1;      int b=1;      int c,n;      FILE \*fptr;      randonGenrator(1);      fptr = fopen("text.txt", "r");      fscanf(fptr, "%d", &n);      fclose(fptr);      fptr = fopen("text.txt", "a");      fputs("\nFibonnacci series is : ", fptr);      for(i=1;i<=n;i++)      {          c=a+b;          fprintf(fptr,"%d  ",c);          a=b;          b=c;      }  }  OUTPUT |
| **5. SUM OF DIGITS**  #include<stdio.h>  #include<stdlib.h>  #include"C:\Users\shash\OneDrive\Desktop\vscode\.vscode\randG.h"  int main()  {      randonGenrator(1);      int sum = 0, n, r;      FILE \*fptr = NULL;      fptr = fopen("text.txt", "r");      fscanf(fptr, "%d", &n);      fclose(fptr);      while (n)      {          r = n % 10;          sum = sum + r;          n = n / 10;      }      fptr = fopen("text.txt", "a");      if (fptr == NULL)      {          printf("Unable to open file.\n");          exit(1);      }      fputs("\nThe Sum Of Digit is: ",fptr);      fprintf(fptr, "%d", sum);      fclose(fptr);  }  OUTPUT |
| **6.** **REVERSE THE FILE ELEMENTS**  #include<stdio.h>  #include<stdlib.h>  #include<math.h>  #include"C:\Users\shash\OneDrive\Desktop\vscode\.vscode\randG.h"  int main()  {      int arr[10],k=0, num,n=10;      FILE \*fptr;      randonGenrator(n);      fptr = fopen("text.txt", "r");      if (fptr == NULL)      {          printf("File not available.");          exit(0);      }        while(fscanf(fptr, "%d", &num) != -1)      {          arr[k]=num;          k++;      }      fclose(fptr);      fptr=fopen("text.txt","a");      fputs("\nReverse Order of elements : ", fptr);      for(int i=k-1;i>=0;i--)      {          fprintf(fptr, "%d ",arr[i]);      }      fclose(fptr);  }  OUTPUT |
| **7.PALIMNDROME NO** |
| **8.AREA OF THE SHAPES** |
| **9. SIMPLE CALCULATOR** |
| **10.ARRAY OPERATIONS**  #include<stdio.h>  #include<stdlib.h>  #include<math.h>  #include"C:\Users\shash\OneDrive\Desktop\vscode\.vscode\randG.h"  int main()  {      int arr[100];      FILE \*fptr;      int sum=0, h=0,k=0,num,s=99999;        randonGenrator(20);      fptr = fopen("text.txt", "r");  if (fptr == NULL)      {          printf("File not available.");          exit(0);      }      while(fscanf(fptr, "%d", &num) != -1)      {          arr[k]=num;          k++;      }      fclose(fptr);      for(int i=0;i<k;i++)      {          if(arr[i]>h)              h=arr[i];      }      for(int i=0;i<k;i++)      {          if(arr[i]<s)              s=arr[i];      }      for(int i=0;i<k;i++)          sum=sum+arr[i];      fptr=fopen("text.txt","a");      fputs("\nHighest Element is ",fptr);          fprintf(fptr, "%d ",h);      fputs("\nSmallest Element is ",fptr);          fprintf(fptr, "%d ",s);      fputs("\nSum of the Element is ",fptr);          fprintf(fptr, "%d ",sum);      fputs("\nAverage of the Element is ",fptr);          fprintf(fptr, "%d ",sum/k);  }  OUTPUT |
| **11. STRING OPERATIONS** |
| **12.** **LINEAR SEARCH**  #include <stdio.h>  #include <stdlib.h>  #include "C:\\Users\\shash\\OneDrive\\Desktop\\vscode\\.vscode\\randG.h"  int main()  {      FILE \*fptr;      int num, s, k = 0;      int a[100];      randonGenrator(10);      fptr = fopen("text.txt", "r");      if (fptr == NULL)      {          printf("Unable to open file.\n");          exit(1);      } else      {          while (fscanf(fptr, "%d", &num) != EOF)          {              a[k] = num;              k++;          }      }      fclose(fptr);      fptr=fopen("text.txt","a");      printf("Enter the element to search: ");      scanf("%d", &s);      for (int i = 0; i < 10; i++)      {          if (s == a[i])          {              fputs("\nThe Element is Found.",fptr);              return 0;          }      }      fputs("\nElement not Found.\n",fptr);      fclose(fptr);      return 0;  }  OUTPUT |
| **13. BINARY SEARCH**  #include<stdio.h>  #include<stdlib.h>  #include"C:\Users\shash\OneDrive\Desktop\vscode\.vscode\randG.h"  int binarysearch(int [], int, int);  void bubblesort1(int [], int);  int main() {      int arr[100];      int n = 15, key, num;      int p, k = 0;      FILE \*fptr;      randonGenrator(n);      fptr = fopen("text.txt", "r");      if (fptr == NULL) {          printf("File not available.");          exit(1);      }      while (fscanf(fptr, "%d", &num) != EOF)      {          arr[k] = num;          k++;      }      fclose(fptr);      printf("Enter the key to search: ");      scanf("%d", &key);      bubblesort1(arr, n);      fptr=fopen("text.txt","a");      fputs("\nStored Elements are : ", fptr);      for(int i=0;i<n;i++)      {          fprintf(fptr, "%d ",arr[i]);      }      p = binarysearch(arr, key, n);      if (p == 1)      {          fputs("\nThe key is found\n",fptr);      }      else {          fputs("\nThe key is not found",fptr);      }      fclose(fptr);      return 0;  }  void bubblesort1(int a[], int n) {      int temp;      for (int i = 0; i < n - 1; i++) {          for (int j = 0; j < n - 1 - i; j++)          {              if (a[j] > a[j + 1]) {                  temp = a[j];                  a[j] = a[j + 1];                  a[j + 1] = temp;              }          }      }  }  int binarysearch(int a[], int key, int n) {      int l = 0, u = n - 1;      int mid;      while (l <= u) {          mid = (l + u) / 2;          if (key == a[mid])              return 1;          else if (key < a[mid])              u = mid - 1;          else              l = mid + 1;      }      return -1;  }  OUTPUT  Key =15 |
| **14.SELECTION SORT**  #include<stdio.h>  #include<stdlib.h>  #include"C:\Users\shash\OneDrive\Desktop\vscode\.vscode\randG.h"  void readarray(int [], int);  void displayarray(int [], int);  void selectionsort(int [], int);  int main()  {      int n=10;      int arr[10],k=0, num;      FILE \*fptr;      randonGenrator(n);      fptr = fopen("text.txt", "r");      if (fptr == NULL)      {          printf("File not available.");          exit(0);      }        while(fscanf(fptr, "%d", &num) != -1)      {          arr[k]=num;          k++;      }      fclose(fptr);      selectionsort(arr, n);      fptr=fopen("text.txt","a");      fputs("\nSorted Array is : ", fptr);      for(int i=0;i<k;i++)      {          fprintf(fptr, "%d ",arr[i]);      }      fclose(fptr);  }  void selectionsort(int a[], int n)  {      int min;      int temp;      for(int i=0;i<n;i++)      {          min=i;          for(int j=i+1;j<n;j++)          {              if(a[j]<a[min])                  min=j;              temp=a[i];              a[i]=a[min];              a[min]=temp;          }      }  }  OUTPUT |
| **15.BUBBLE SORT** |
| #include<stdio.h>  #include<stdlib.h>  #include"C:\Users\shash\OneDrive\Desktop\vscode\.vscode\randG.h"  void bubblesort1(int \*, int);  void bubblesort2(int \*, int);  int main()  {      FILE \*fptr;      int num;      int arr[100];      int k = 0;      char a[100]="Ascending Order: ";      char d[100]="Descending Order: ";      int z=20;      randonGenrator(z);      fptr = fopen("text.txt", "r");      if (fptr == NULL)      {          printf("Unable to open file for reading.\n");          exit(0);      }      //rewind(fptr);      while (fscanf(fptr, "%d", &num) == 1)      {          arr[k] = num;          k++;      }      fclose(fptr);      bubblesort1(arr,z);      fptr=fopen("text.txt","a");      fprintf(fptr, "\n");      fprintf(fptr, "%s", a);      for(int i=0;i<z;i++)      {          fprintf(fptr, "%d ",arr[i]);      }      fclose(fptr);      bubblesort2(arr,z);      fptr=fopen("text.txt","a");      fprintf(fptr, "\n");      fprintf(fptr, "%s", d); //fputs("Ascending Order : ", fptr);      for(int i=0;i<z;i++)      {          fprintf(fptr, "%d ",arr[i]);      }      printf("The Elements are perfectly Sorted n again Stored in File Successfully");      return 0;  }  void bubblesort1(int a[], int n)  {      int temp=0;      for(int i=0;i<n-1;i++)      {          for(int j=0;j<n-1-i;j++)          {              if(a[j]>a[j+1])              {                  temp=a[j];                  a[j]=a[j+1];                  a[j+1]=temp;              }          }      }  }  void bubblesort2(int a[], int n)  {      int temp=0;      for(int i=0;i<n-1;i++)      {          for(int j=0;j<n-1-i;j++)          {              if(a[j]<a[j+1])              {                  temp=a[j];                  a[j]=a[j+1];                  a[j+1]=temp;              }          }      }  }  OUTPUT     |  | | --- | | **17. MATRIX OPERATIONS (ADDITION)** | | #include<stdio.h>  #include<stdlib.h>  #define ROW 10  #define COL 10  void readmatrix(int [][COL], int, int);  void displaymatrix(int [][COL], int, int);  void addmat(int [][COL], int [][COL], int [][COL], int, int);  int main()  {      int a[ROW][COL];      int b[ROW][COL];      int y[ROW][COL];      int r, c;      printf("Enter the order: ");      scanf("%d%d",&r,&c);      printf("Enter the matrix A elements: \n");      readmatrix(a, r, c);      printf("Enter the matrix B elements: \n");      readmatrix(b, r, c);      printf("The matrix A elements are: \n");      displaymatrix(a, r, c);      printf("The matrix B elements are: \n");      displaymatrix(b, r, c);      addmat(a,b,y,r,c);      printf("The Sum of matrix A & B is: \n");      displaymatrix(y, r, c);  }  void addmat(int a[][COL], int b[][COL],int y[][COL], int r,int c)  {      for(int i=0;i<r;i++)      {          for(int j=0;j<c;j++)          {              y[i][j]=a[i][j]+b[i][j];          }      }  }  void displaymatrix(int a[][COL], int r, int c)  {      for(int i=0;i<r;i++)      {          for(int j=0;j<c;j++)          {              printf("%d  ",a[i][j]);          }          printf("\n");      }  }  void readmatrix(int a[][COL], int r, int c)  {      for(int i=0;i<r;i++)      {          for(int j=0;j<c;j++)          {              scanf("%d",&a[i][j]);          }      }  } | | Output | | **MATRIX OPERATIONS(MULTIPLACTION)** | | #include<stdio.h>  #include<stdlib.h>  #define ROW 10  #define COL 10  void readmatrix(int [][COL], int, int);  void displaymatrix(int [][COL], int, int);  void mulmat(int [][COL], int [][COL], int [][COL], int, int, int);  int main()  {      int a[ROW][COL];      int b[ROW][COL];      int y[ROW][COL];      int r1, c1, r2, c2;      printf("Enter the order of matrix A: ");      scanf("%d%d",&r1,&c1);      printf("Enter the matrix A elements: \n");      readmatrix(a, r1, c1);      printf("Enter the order of matrix B: ");      scanf("%d%d",&r2,&c2);      printf("Enter the matrix B elements: \n");      readmatrix(b, r2, c2);      printf("The matrix A elements are: \n");      displaymatrix(a, r1, c1);      printf("The matrix B elements are: \n");      displaymatrix(b, r2, c2);      if(c1!=r2)      {          printf("The Multiplication cannot be performed..!");          exit(0);      }      mulmat(a,b,y,r1,c1,c2);      printf("The Multiplication of matrix A & B is: \n");      displaymatrix(y, r2, c2);  }  void mulmat(int a[][COL], int b[][COL], int y[][COL], int r1, int c1, int c2)  {      for(int i=0;i<r1;i++)      {          for(int j=0;j<c1;j++)          {              y[i][j]=0;              for (int k=0;k<c1;k++)              {                  y[i][j]= y[i][j]+ a[i][k]\*b[k][j];              }          }      }  }  void displaymatrix(int a[][COL], int r, int c)  {      for(int i=0;i<r;i++)      {          for(int j=0;j<c;j++)          {              printf("%d  ",a[i][j]);          }          printf("\n");      }  }  void readmatrix(int a[][COL], int r, int c)  {      for(int i=0;i<r;i++)      {          for(int j=0;j<c;j++)          {              scanf("%d",&a[i][j]);          }      }  }  OUTPUT | |

**Part- C**

|  |
| --- |
| **1.LINKED LIST** |
| #include<stdio.h>  #include<stdlib.h>  struct node  {      int data;      struct node \*next;  };  void addatbeg(struct node \*\*, int);  void displaydata(struct node \*\*);  int deleteatbeg(struct node \*\*);  void addatend(struct node \*\*, int);  int deleteatend(struct node \*\*);  int lenghtoflist(struct node \*);  int searchelement(struct node \*, int );  int frequency(struct node \*, int);  void addatanypos(struct node \*\*, int, int);  void reverselist(struct node \*\*);  void deletestocc(struct node \*\*, int);  void sortlist(struct node \*);  void r\_Duplicate(struct node \*);  void reverseKthNode(struct node \*\*, int );  void deleteAtPos(struct node \*\*, int);  int main()  {      struct node \*head=NULL;      int n, e, f, pos, key;      while(1)      {          printf("\n1.Add at beginning\n");          printf("2.Delete at beginning\n");          printf("3.display list\n");          printf("4.Add at end\n");          printf("5.Delete at end\n");          printf("6.Length of list\n");          printf("7.Search an element\n");          printf("8.Frequency of element:\n");          printf("9.Add node at any given position\n");          printf("10.Reverse List\n");          printf("11.Delete the First occurence\n");          printf("12.Sort\_list \n");          printf("13.Remove Duplicate\n");          printf("14.Reverse Every kth node\n");          printf("15.Delete at position\n");          printf("Enter n: ");          scanf("%d",&n);          switch(n)          {              case 1: printf("Enter the date: ");                      scanf("%d",&e);                      addatbeg(&head, e);                      break;              case 2: if(head==NULL)                          printf("List empty");                      else                      {                          e=deleteatbeg(&head);                      }                      break;              case 3: displaydata(&head);                      break;              case 4: printf("Enter the data: ");                      scanf("%d",&e);                      addatend(&head, e);                      break;              case 5: if(head==NULL)                          printf("List empty");                      else                      {                          e=deleteatend(&head);                      }                      break;              case 6: e=lenghtoflist(head);                      printf("List length=%d",e);;                      break;              case 7: if(head==NULL)                      {                          printf("list empty");                      }                      else                      {                          printf("Enter the key: ");                          scanf("%d",&e);                          f=searchelement(head, e);                          if(f)                              printf("Key is found:");                          else                              printf("Key not found");                      }                      break;              case 8: printf("Enter the key for frequency: ");                      scanf("%d",&e);                      f=frequency(head, e);                      printf("The frequency of %d is %d", e, f);              case 9: printf("Enter the data: ");                      scanf("%d",&e);                      printf("Enter position: ");                      scanf("%d",&pos);                      addatanypos(&head, e, pos);                      break;              case 10:reverselist(&head);                      break;              case 11:printf("Enter the key: ");                      scanf("%d",key);                      deletestocc(&head, key);                      break;              case 12: sortlist(head);                      break;              case 13: r\_Duplicate(head);                      break;              case 14: printf("Enter k: ");                       scanf("%d",&e);                       reverseKthNode(&head, e);                       break;              case 15:printf("Enter pos: ");                      scanf("%d",&e);                      deleteAtPos(&head, e);                      break;              default: exit(0);          }      }  }  void addatbeg(struct node \*\*head, int e)  {      struct node \*p;      p=(struct node\*)malloc(sizeof(struct node));      if(p==NULL)      {          perror(" ");          return;      }      p->data=e;      p->next=\*head;      \*head=p;  }  void displaydata(struct node \*\*head)  {      struct node \*cur=\*head;      if(\*head==NULL)      {          printf("----------List Empty----------\n----------Enter data----------\n");          return;      }      while(cur)      {          printf("%d\t",cur->data);          cur=cur->next;      }  }  int deleteatbeg(struct node \*\*head)  {      struct node \*p=\*head;      \*head=(\*head)->next;      int e=p->data;      free(p);      return e;  }  void addatend(struct node \*\*head, int e)  {      struct node \*p;      struct node \*cur=\*head;      p=(struct node\*)malloc(sizeof(struct node));      if(p==NULL)      {          perror(" ");          return;      }      p->data=e;      p->next=NULL;      if(\*head==NULL)          \*head=p;      else      {          while(cur->next)              cur=cur->next;          cur->next=p;      }  }  int deleteatend(struct node \*\*head)  {      struct node \*cur=\*head;      struct node \*prev=NULL;      int e;      while(cur->next)      {          prev=cur;          cur=cur->next;      }      e=cur->data;      if(prev)          prev->next=NULL;      else          \*head=NULL;      free(cur);      return e;  }  int lenghtoflist(struct node \*head)  {      struct node \*p=head;      int count=0;      while(p)      {          count++;          p=p->next;      }      return count;  }  int searchelement(struct node \*head, int key)  {      struct node \*cur=head;      while(cur)      {          if(key==cur->data)              return 1;          cur=cur->next;      }      return 0;  }  int frequency(struct node \*head, int key)  {      struct node \*cur=head;      int c=0;      while(cur)      {          if(key==cur->data)          {              c++;              cur=cur->next;          }      }      return c;  }  void addatanypos(struct node \*\*head, int e, int pos)  {      struct node \*p;      struct node \*cur=\*head;      struct node \*prev=NULL;      p=(struct node \*)malloc(sizeof(struct node));      if(p==NULL)      {          perror("");          return ;      }      p->data=e;      p->next=NULL;      if(pos<=0 || \*head==NULL)      {          p=\*head;          \*head=p;          return;      }      for(int i=1;i<=pos && cur ;i++)      {          prev=cur;          cur=cur->next;      }      prev->next=p;      p->next=cur;  }  void deleteAtPos(struct node \*\*head, int pos)  {      struct node \*prev=NULL;      struct node \*cur=\*head;      if(pos<=0)      {          \*head=(\*head)->next;          free(cur);          return;      }      else if(pos>lenghtoflist(\*head))      {          while(cur->next)          {              prev=cur;              cur=cur->next;          }           prev->next=NULL;           free(cur);      }      else      {          for(int i=0;i<pos-1;i++)          {              prev=cur;              cur=cur->next;          }          prev->next=cur->next;          free(cur);      }  }  void reverselist(struct node \*\*head)  {      struct node \*p=\*head;      struct node \*q, \*r=NULL;      if(p)          q=p->next;      if(q)          r=q->next;      while(q)      {          q->next=p;          p=q;          q=r;          if(r)              r=r->next;      }      if(\*head)          (\*head)->next=NULL;      \*head=p;  }  void deletestocc(struct node\*\* head, int key)  {      struct node\* cur = \*head;      struct node\* prev = NULL;      if (cur == NULL)          return;      while (cur != NULL && cur->data != key)      {          prev = cur;          cur = cur->next;      }      if (cur == NULL)          return;      if (cur == \*head)      {          \*head = cur->next;          free(cur);          return;      }      prev->next = cur->next;      free(cur);  }  void sortlist(struct node \*head)  {      struct node \*min;      int temp;      struct node \*i, \*j;      for(i=head;i&&i->next;i=i->next)      {          min = i;          for (j=i->next;j!=NULL;j=j->next)          {              if (j->data < min->data)                  min = j;          }          if (min!=i)          {              temp = i->data;              i->data = min->data;              min->data = temp;          }      }  }  void reverseKthNode(struct node \*\*head, int k)  {      if (\*head == NULL || k <= 1)      {          return;      }      int n = lenghtoflist(\*head) / k;      struct node \*cur = \*head;      struct node \*prev = NULL;      for (int i = 0; i < n; i++)      {          struct node \*p = cur;          struct node \*q = NULL;          struct node \*r = NULL;          for (int c = 0; c < k; c++)          {              struct node \*temp = cur->next;              cur->next = q;              q = cur;              cur = temp;          }          if (i == 0)              \*head = q;          else              prev->next = q;          p->next = cur;          prev = p;          r = cur;      }  }  void r\_Duplicate(struct node\* head)  {      if (head == NULL)          return;      struct node\* current = head;      while (current != NULL)      {          struct node\* runner = current;          while (runner->next != NULL)          {              if (runner->next->data == current->data)              {                  struct node\* temp = runner->next;                  runner->next = runner->next->next;                  free(temp);              }              else              {                  runner = runner->next;              }          }          current = current->next;      }  } |
| Output |