|  |  |
| --- | --- |
| **Name of the student** | **SURYANARAYANA KHANNUR** |
| **Roll No** | **266** |
| **USN** | **01FE22BEC097** |
| **Div** | **B** |

**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.** |
| **A:** A variable is a container that stores a value or a set of values in C programming. A data type is used to define the type of data that a variable can hold.  For example,   1. int num – Here int is integer data type whereas num is a variable 2. char type – Here char is character data type whereas type is a variable |
| **2. Explain the concept of data types in C programming. Discuss the different types**  **of data types available in C.** |
| **A:** A data type is used to define the type of data a variable can hold. Data types are characterised by different memory requirements and operations that can be performed on them. The different data types available in C are as follows: -   1. Primary  * int (4 bytes) – to store integers * char (1 byte) – to store characters * float (4 bytes) – to store decimal numbers with single precision * double (8 bytes) – to store decimal numbers with double precision * void – used with function when it does not return any value  1. Derived  * array – collection of similar items stored at continuous memory locations * pointer – variable that can hold the address of another variable  1. User defined  * struct – used to create a date type which is a collection of variables of different data types grouped together under a single name * union – all the members defined share the same memory location i.e. any change brought in one member will be reflected in all other members * enum – used to assign meaningful names to integral constants |
| **3. How are variables declared and initialized in C programming? Provide**  **examples of variable declarations with different data types**. |
| **A:** In C programming, variables are declared and initialised with the following syntax: -  <data type> <variable\_name> = <initial\_value>;  Examples: -   1. Integer variables: int age = 18; 2. Floating point variables: float pi = 3.1415; 3. Character variables: char division = ‘B’; 4. Double variables: double salary = 20000.50; 5. Arrays: int number\_list [5] = {0,1,2,3,4}; 6. Pointers: int \*ptr = NULL;   **4. Discuss the scope and lifetime of variables in C programming. What are global**  **and local variables?** |
| **A:** The scope and lifetime of a variables determines where and for how long a variable is accessible and usable within a program.   * Global variables: They are accessible from any part of the program and exist for the entire duration of the program. * Local variables: They are only accessible within a particular function and have lifetime equal to the duration of the function. |
| **5. Explain the concept of type casting in C programming. When is type casting**  **necessary, and how is it performed?** |
| **A:** Type casting is the process of converting a value from one data type to another. It is necessary when you want to perform operations involving variables of different data types.  Examples: -   * int num1 = 5;   float num2 = num1; //Implicit   * float num1 = 5.61;   int num2 = (int)num1; //Explicit |
|  |
| **Operators** |
| **1. Describe the purpose and usage of the ternary conditional operator (? :) in C**  **programming. Provide an example demonstrating its usage.** |
| **A:** The ternary conditional operator provides a concise way to write conditional expressions and can be used as a shorthand for simple if-else statements. It has the following syntax: -  <condition>? <expression1>: <expression2>;  It means, if condition is true <expression1> is evaluated, else <expression2> is evaluated  Ex: - result = (num>0)? 1: 0; // if num > 0 then result =1 else result=0; |
| **2.** **Discuss the bitwise operators available in C programming. Explain their usage**  **with suitable examples.** |
| **A:** Bitwise operators are used to perform operations at the bit level. They are as follows: -   1. Bitwise AND (&)   int a = 2; //0010  int b = 3; //0011  int c = a & b; //0010   1. Bitwise OR (|)   int a = 2; //0010  int b = 3; //0011  int c = a | b; //0011   1. Bitwise XOR (^)   int a = 2; //0010  int b = 3; //0011  int c = a ^ b; //0001   1. Bitwise NOT (~)   int a = 2; //0010  int b = ~a; //1101   1. Left Shift (<<)   int a = 2; //0010  int b = a << 2; //1000   1. Right Shift (>>)   int a = 2; //0010  int b = a >> 1; //0001 |
| **3.** **Explain the difference between the postfix and prefix increment operators (++)**  **in C programming. Provide examples to illustrate.** |
| **A:** Postfix increment operator (x++): The value of the variable is first used in the expression where the operator is applied and then the variable is incremented.  Prefix increment operator (++x): The variable is first incremented and then the updated value is used in the expression where the operator is applied.  Ex: - int i = 10; //i=10  int j = ++i; //i=11, j=11  int k = i++; //k=11, i=12 |
| **4.** **What is the significance of the logical AND (&&) and logical OR (||) operators**  **in C programming? How are they used in conditional expressions?** |
| **A:** Logical AND OR operators are used to combine multiple conditions in conditional expressions and control the flow of execution accordingly.  Ex: if (x>0 && y>0) and if (x>0 || y<0) |
| **5. Discuss the concept of operator precedence and associativity in C programming.**  **Provide example to demonstrate how they affect expression evaluation.** |
| **A:** In C programming, operator precedence and associativity determine the order in which operators are evaluated in the expression.  Ex: int answer = 1 + 2 \* 4 / 2 //answer = 5  int answer = (1 + 2) \* 4 / 2 //answer = 6 |
|  |
| **Control Structures** |
| **1.** **Describe the purpose and usage of the switch statement in C programming. How**  **does it differ from the if-else statement?** |
| **A:** The switch statement is used to control the flow of execution based on the value of an expression. It provides a convenient way to compare the value of an expression against multiple constant values and to execute accordingly. The switch statement only works with integer, character or enumerated data types whereas the if-else statement can work with any expression that evaluates to a Boolean value. The switch statement can sometimes be more efficient than multiple if-else statements, especially when the number of cases is large, as it uses direct branching. The switch provides better readability and maintainability when dealing with multiple conditions. |
| **2. Explain the concept of nested control structures in C programming. Provide an**  **example demonstrating nested if-else statements.** |
| **A:** Nested control structures refer to the situation where one control structure is placed inside another control structure allowing sequential evaluation of conditions.  Ex: if (num % 2 == 0) {  printf("Number %d is even.\n", num);  if (num > 10) {  printf("Number %d is greater than 10.\n", num);  } else {  printf("Number %d is not greater than 10.\n", num);  }  } else {  printf("Number %d is odd.\n", num);  } |
| **3. Discuss the role of the break and continue statements in loop control in C**  **programming. Provide examples to illustrate their usage.** |
| **A:** The break statement is used to immediately terminate the execution of the innermost loop and transfer control to the statement following the loop. The continue statement is used to skip the current iteration of the loop and proceed with the next iteration.  Ex: #include <stdio.h>  int main () {  int i;  for (i = 1; i <= 10; i++) {  if (i == 7) {  break;  }  if (i%2 == 0)  continue;  printf ("%d ", i);  }  return 0;  } |
| **4.** **What are the advantages of using the for loop over the while loop in C**  **programming? Provide examples comparing the two.** |
| **A:** The for loop is more compact and can provide a clearer expression of loop initialization, condition and update in a single line. This makes the code more readable and concise.  Ex: for (i=0;i<10;i++)  {  statements;  }  is same as  i=0;  while (i<10)  {  statements;  i++;  } |
| **5. Explain the concept of short-circuit evaluation in C programming. How does it**  **affect the evaluation of logical expressions in if statements** |
| **A:** Short circuit evaluation implies that the second operand of the logical expression is only evaluated if the result of the expression cannot be determined solely by evaluating the first operand. This means that in certain situations, the evaluation of the second operand is skipped, resulting in improved efficiency and avoiding potential errors such as division by zero or accessing invalid memory locations. |
|  |
| **Functions** |
| **1.** **Describe the purpose and structure of a function prototype in C programming.**  **Why is it necessary to declare function prototypes?** |
| **A:** A function prototype serves as a declaration of a function before its actual definition or implementation. It provides essential information about the function to the compiler, allowing the compiler to verify the correctness of function calls and perform type checking during compilation. Syntax:  return\_type function\_name (parameter1\_type parameter1\_name, parameter2\_name parameter2, …); |
| **2.** **Explain the difference between call by value and call by reference in C**  **programming. Provide examples to illustrate both concepts.** |
| **A:** In call by value, a copy of the actual parameter’s value is passed to the function parameter. In call by reference, the memory address of the actual parameter is passed to the function parameter.  Ex: Function calls  fun(num); //num is an int type, it is call by value  fun(&num); //num is an int type, it is call by reference |
| **3.** **Discuss the concept of recursion in C programming. Provide an example of a**  **recursive function and explain how it works** |
| **A:** Recursion is an approach to solving problems which a involves a function calling itself.  #include <stdio.h>  // Factorial  int factorial (int n) {  if (n == 0 || n == 1) {  return 1;  } else {  return n \* factorial (n - 1);  }  }  int main () {  int num = 5;  int result = factorial(num);  printf ("Factorial of %d is %d\n", num, result);  return 0;  }  In the example, when factorial(5) is called, it calls factorial(4), then factorial(3), and so on until it reaches factorial(1). At this point, the base case is met, and the recursion stops. |
| **4.** **What is the significance of the return statement in C programming? How are**  **values returned from functions?** |
| **A:** In C programming, the return statement is used to terminate the execution of a function and return a value back to the caller. It hands back the control to the calling function along with the value computed or processed within function. It helps to provide results to other parts of the programs.  Syntax:  return expression; |
| **5.** **Describe the role of function parameters and arguments in C programming.**  **How are function arguments passed to parameters?** |
| **A:** Function parameters are placeholders defined in the function definition, and they specify the type and order of data that the function expects to receive when it is called. Parameters act as variables within the function’s scope. Function arguments are actual values passed to a function when it is called and they represent the data that the function will operate on.  Ex: - void fun (int x, int y) //Here x and y are function parameters and it is function definition  fun (a, b); //Here a and b are integer arguments of the function call |
|  |
| **Arrays** |
| **1.** **Explain the concept of arrays in C programming. How are arrays declared and initialized?** |
| **A:** An array is a data structure that collection of data of the same type in contiguous memory locations.  The syntax for declaring an array is as follows: -  data\_type array\_name[array\_size];  Example: - Initializing an array  int numbers [5] = {0,1,2,3,4}; |
| **2.** **Discuss the difference between a one-dimensional array and a multi-dimensional array in C programming. Provide examples of both.** |
| **A:** A one-dimensional array is a collection of elements of the same data type arranged in contiguous memory location whereas a multi-dimensional array is an array of arrays where each element of the array holds another array.  Ex: - int array[5] = {1,2,3,4,5}; //One dimensional array  int matrix[3][3] = {{1,2,3},{4,5,6},{7,8,9}}; //Multi-dimensional array |
| **3.** **Describe the process of accessing array elements in C programming. How are array indices used to access elements?** |
| **A:** Array indices are zero based meaning that the first element of the array has an index 0 .  Array elements can be accessed as follows: -  int numbers[5] = {1,2,3,4,5}; //Initialising an array  int x = numbers[1]; //Second element of array i.e 2 is assigned to x |
| **4. What is the significance of the null character ('\0') in C strings? How is it used**  **to determine the end of a string?** |
| **A:** Null character is used to terminate C strings and marks their end in the memory. For example : -  char str[ ] = “Hi”;  The above array str[ ] contains characters ‘H’, ‘i’, ‘\0’. |
| **5. Explain the concept of dynamic memory allocation for arrays in C programming. How are dynamic arrays allocated and deallocated?** |
| **A:** Dynamic memory allocation allows you to allocate memory for data structures such as arrays at runtime, rather that at compile time. This flexibility is particularly useful when you don’t know the size of the array beforehand.  Allocating an array dynamically  int \*array = (int\*)malloc(n\*sizeof(int)); //n is number of elements  Deallocating an array  free(array); |
|  |
| **Pointers** |
| **1.** **Describe the purpose and usage of pointers in C programming. How are pointers**  **declared and initialized?** |
| **A:** Pointers are variables that store memory addresses. They are used for dynamic memory allocation, passing arguments by reference and implementing data structures.  Declaring and initialising a pointer  int x = 5;  int \*ptr = &x; |
| **2.** **Explain the concept of pointer arithmetic in C programming. Provide examples**  **to illustrate addition and subtraction operations on pointers.** |
| **A:** Pointer arithmetic refers to performing arithmetic operations on pointers to manipulate memory addresses.  Ex:int numbers[ ]={1,2,3,4,5};  int \*ptr=numbers //where numbers[ ] is an array  ptr=ptr+2; //points to element at index 2  ptr=ptr-2; //points to element at index 0 |
| **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.** |
| **A:** Pass by value  #include <stdio.h>  void increment(int num) {  num++; // Increment the parameter 'num'  printf("Inside function: %d\n", num); // Output: Inside function: incremented value  }  int main() {  int x = 5;  printf("Before function call: %d\n", x); // Output: Before function call: 5  increment(x); // Call the function with 'x'  printf("After function call: %d\n", x); // Output: After function call: 5  return 0;  }  Pass by reference  #include <stdio.h>  void incrementByRef(int \*ptr) {  (\*ptr)++; // Increment the value pointed to by 'ptr'  printf("Inside function: %d\n", \*ptr); // Output: Inside function: incremented value  }  int main() {  int x = 5;  printf("Before function call: %d\n", x); // Output: Before function call: 5  incrementByRef(&x); // Call the function with the address of 'x'  printf("After function call: %d\n", x); // Output: After function call: incremented value  return 0;  } |
|  |
| **4. Describe the concept of NULL pointers in C programming. How are NULL**  **pointers used and checked for in programs?** |
| **A:** A NULL pointer is a pointer that does not point to any valid memory address.  Example: - int \*ptr = NULL;  Checking of NULL pointers  if (ptr == NULL)  printf (“Pointer is NULL”); |
| **5.** **Explain the role of pointers in dynamic memory allocation in C programming.**  **How are pointers used to allocate and deallocate memory dynamically?** |
| **A:** Pointers are used to dynamically allocate memory using malloc(),calloc and realloc() functions  and are used to deallocate allocated memory using free().  Ex: - int \*ptr = (int \*)malloc(5 \* sizeof(int)); //Allocating memory  free(ptr); //Deallocating memory |
|  |
| **Strings** |
| **1.** **Discuss the concept of strings in C programming. How are strings represented**  **and manipulated in C?** |
| **A:** String is a character array terminated by a NULL character. For example : -  char str[ ] = “Hi”;  The above array str[ ] contains characters ‘H’, ‘i’, ‘\0’.  Strings can be manipulated using string manipulation functions located in <string.h> library using functions such as strlen(),strcpy(),strcat(). |
| **2.** **Explain the difference between character arrays and string literals in C**  **programming. Provide examples to illustrate both concepts.** |
| **A:**  The contents of character arrays can be modified and each character can be accessed (i.e. they are stored in contiguous memory locations) whereas string literals are enclosed in double quotes and their contents cannot be modified once initialised  Ex: char str[ ] = “Hello”; //Character array  char \*str = “Hello” //String literal |
| **3.** **Describe common string manipulation functions available in the C standard**  **library. Provide examples of functions like strlen, strcpy, strcat, and strcmp.** |
| **A:** The string manipulation functions are as follows: -   1. strlen() – Length of a string   char str[]=”Hello”;  int length=strlen(str);   1. strcpy – copying a string   strcpy(destination,source);   1. strcat – Concatenating strings   strcat(str1,str2);   1. strcmp – Comparing length of strings   strcmp(str1,str2); |
| **4.** **Discuss the concept of string tokenization in C programming. How are strings**  **split into tokens using delimiter characters?** |
| **A:** String tokenisation is the process of breaking down a string into smaller parts called tokens based on specified delimiter characters. This technique is used to parse input strings,process text data or tokenise user input.Strings can be tokeised using the strtok() function in <string.h> library.  Prototype: char\* strtok(char \*str, const char\* delimiters); |
|  |
| **5.** **Explain the importance of null-terminated strings in C programming. How does**  **the null character ('\0') signify the end of a string?** |
| **A:** When the C compiler encounters the null character, it signifies the end of the string It helps programmers to work with strings without need for explicitly specifying the length of string in programs. |
|  |
| **Structures and Unions** |
| **1.** **Describe the purpose and usage of structures in C programming. How are**  **structures declared and accessed?** |
| **A:** Structures are used to group together variables with different data types.Example of declaration of a structure is as follows:  struct student //Declaration of structure tag  {  char name[30];  int class;  char division;  int roll\_number;  };  Struct student student1 = {“James”,9,’B’,765}; //Declaration of structures variable  We can access name by specifying student1.name, and similarly other data types can be accessed. |
| **2.** **Discuss the concept of structure members in C programming. How are**  **individual members of a structure accessed and modified?** |
| **A:** Structure variables are individual members of the structure.  For :- struct Structure  {  DataType member1;  DataType member2;  };  For above structure if a structure variable Structure1 is declared then each member can be accessed and modified using Structure1.member1 and so on. |
| **3. Explain the difference between structures and unions in C programming. When**  **would you choose one over the other** |
| **A:** Each member of a structure occupies its own separate memory space within the structure whereas for a union all members share the same memory space. Unions would be preferrable when one member needs to be accessed at any given time whereas structures should be used when multiple pieces of data are used simulataneously. |
| **4.** **Describe the concept of nested structures in C programming. How are structures**  **within structures defined and accessed?** |
| **A:** Nested structures are defined by including one structures definition inside another structure definition. For examples:  struct StructureA  {  DataType member1;  struct StructureB Structure2;  }Structure1;  Struct StructureB  {  Datatype member2;  DataType member3;  }  The member 3 can be accessed as Structure1.Structure2.member3 |
| **5.** **Discuss the concept of typedef in C programming. How is typedef used to define**  **custom data types, including structures and unions?** |
| **A:**typedef is a keyword used to create synonyms and aliases for structures,unions and data types.  Syntax  Typedef existingtype newtypename; |
|  |
| **File Handling** |
| **1.** **Explain the concept of file handling in C programming. How are files opened,**  **read from, and written to using standard file handling functions?** |
| **A:** Opening Files  Ex:- FILE\* file\_ptr=fopen(“input”,”r”);  Reading from files  Ex: fscanf(file\_ptr,”%d”,&num);    Writing to files  Ex: fprintf(file\_ptr,”Hello World”);  Closing files  Ex:fclose(file\_ptr); |
| **2.** **Describe the role of file pointers in C programming. How are file pointers used**  **to navigate and manipulate files?** |
| **A:** File pointers are used to navigate and manipulate files by maintaining the current position within the file and facilitating data transfer between the program and the file on the disk.They provide a way to interact with the files and allow different operations to be performed on it. |
| **3.** **Discuss the difference between text files and binary files in C programming.**  **How are they opened and processed differently?** |
| **A:** Text files store data that is readable by a human usually encoded in Unicode. Binary files store data as a sequence of bytes which are not human readable. Both the types of files are opened using same set of file handling functions but in different modes.(wb for writing a binary file and w for writing a text file). |
| **4. Explain the purpose of file modes in C programming. Provide examples of**  **different file modes like "r", "w", "a", etc.** |
| **A:** r – read mode  FILE \*fileptr=fopen(“input.txt”,”r”);  w – write mode  FILE \*fileptr=fopen(“output.txt”,”w”);  a – append mode  FILE \*fileptr=fopen(“example.txt”,”a”); |
| **5.** **Describe error handling techniques in file operations in C programming. How**  **are errors detected and handled when working with files?** |
| **A:** File related functions in C return special values to indicate errors.Also the errno variable can be used for error handling.Proper cleanup of resources like closing functions is essential to prevent resource leaks and ensure system stability.For example: - fclose() can be used to close files on encountering errors. |

**Part- B**

|  |
| --- |
| 1. **Hello World** |
| **Code**  //HELLO WORLD  #include <stdio.h>  #include <stdlib.h>  int main()  {  FILE \*file\_out = fopen("output.txt","w");  if (file\_out == NULL)  {  printf("File could not be opened\n");  exit(-1);  }  fprintf(file\_out,"Hello,World!");  fprintf(stdout,"Data successfully written to file\n");  fclose(file\_out);  return 0;  } |
| **Output** |

|  |
| --- |
| 1. **Factorial** |
| **Code**  //FACTORIAL OF THE NUMBER  #include <stdio.h>  #include <stdlib.h>  int main()  {  FILE \*file\_in = fopen("input.txt","r");  FILE \*file\_out = fopen("output.txt","w");  if (file\_in == NULL || file\_out == NULL)  {  printf("File could not be opened\n");  exit(-1);  }  int num,fact=1;  fscanf(file\_in,"%d",&num);  for (int i=1;i<=num;i++)  {  fact\*=i;  }  fprintf(file\_out,"%d!=%d",num,fact);  fprintf(stdout,"Data successfully written to file\n");  fclose(file\_in);  fclose(file\_out);  return 0;  } |
| **Output** |

|  |
| --- |
| **3. Prime Numbers** |
| **Code**  //PRIME NUMBER  #include <stdio.h>  #include<stdlib.h>  int prime(int i);  int main()  {  FILE \*file\_in = fopen("input.txt","r");  FILE \*file\_out = fopen("output.txt","w");  if (file\_in == NULL || file\_out == NULL)  {  printf("File could not be opened\n");  exit(-1);  }  int num;  int Isprime;  fscanf(file\_in,"%d",&num);  Isprime=prime(num);  if (Isprime==1)  fprintf(file\_out,"The number %d is prime",num);  else  fprintf(file\_out,"The number %d is not prime",num);  fprintf(stdout,"Data successfully written to file\n");  fclose(file\_in);  fclose(file\_out);  return 0;  }  int prime(int i)  {  int isprime=1;  int j;  for (j=2;j<i;j++)  {  if (i%j==0)  {  isprime=0;  break;  }  }  return isprime;  } |
| **Output** |

|  |
| --- |
| **4. Fibonacci Series** |
| **Code**  //FIBONACCI SERIES  #include <stdio.h>  #include <stdlib.h>  int main()  {  FILE \*file\_in = fopen("input.txt","r");  FILE \*file\_out = fopen("output.txt","w");  if (file\_in == NULL || file\_out == NULL)  {  printf("File could not be opened\n");  exit(-1);  }  int a=0,b=1,c=0,n;  fscanf(file\_in,"%d",&n);  for (int i=n;i>0;i--)  {  fprintf(file\_out,"%d ",a);  c=a+b;  a=b;  b=c;  }  fprintf(stdout,"Data successfully written to file\n");  fclose(file\_in);  fclose(file\_out);  return 0;  } |
| **Output** |

|  |
| --- |
| **5. Sum of Digits** |
| **Code**  //SUM OF DIGITS OF A NUMBER  #include <stdio.h>  #include <stdlib.h>  int main()  {  FILE \*file\_in = fopen("input.txt","r");  FILE \*file\_out = fopen("output.txt","w");  if (file\_in == NULL || file\_out == NULL)  {  printf("File could not be opened\n");  exit(-1);  }  int num,sum=0;  fscanf(file\_in,"%d",&num);  while (num!=0)  {  sum+=num%10;  num=num/10;  }  fprintf(file\_out,"Sum of digits is %d",sum);  fprintf(stdout,"Data successfully written to file\n");  fclose(file\_in);  fclose(file\_out);  return 0;  } |
| **Output** |

|  |
| --- |
| **6. Reverse the digits of a number** |
| **Code**  //REVERSING A NUMBER  #include <stdio.h>  #include <math.h>  #include <stdlib.h>  int main()  {  FILE \*file\_in = fopen("input.txt","r");  FILE \*file\_out = fopen("output.txt","w");  if (file\_in == NULL || file\_out == NULL)  {  printf("File could not be opened\n");  exit(-1);  }  int num,n,count=0;  fscanf(file\_in,"%d",&num);  count=(int)(log10(num))+1;  n=num;  num=0;  while (n!=0)  {  count--;  num+=(n%10)\*pow(10,count);  n/=10;  }  fprintf(file\_out,"The reversed number is %d",num);  fprintf(stdout,"Data successfully written to file\n");  fclose(file\_in);  fclose(file\_out);  return 0;  } |
| **Output** |

|  |
| --- |
| **7.a) Palindrome Check (Number)** |
| **Code**  //PALINDROME  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  int main()  {  FILE \*file\_in = fopen("input.txt","r");  FILE \*file\_out = fopen("output.txt","w");  if (file\_in == NULL || file\_out == NULL)  {  printf("File could not be opened\n");  exit(-1);  }  int num,n,N,count=0;  fscanf(file\_in,"%d",&num);  count=(int)(log10(num))+1;  n=num;  N=num;  num=0;  while (n!=0)  {  count--;  num+=(n%10)\*pow(10,count);  n/=10;  }  if (N==num)  fprintf(file\_out,"The number is a palindrome");  else  fprintf(file\_out,"The number is not a palindrome");  fprintf(stdout,"Data successfully written to file\n");  fclose(file\_in);  fclose(file\_out);  return 0;  } |
| **Output** |

|  |
| --- |
| **7.b) Palindrome Check (String)** |
| **Code**  //PALINDROME  #include <stdio.h>  #include <stdlib.h>  #define SIZE 100  int main()  {  FILE \*file\_in = fopen("input.txt","r");  FILE \*file\_out = fopen("output.txt","w");  if (file\_in == NULL || file\_out == NULL)  {  printf("File could not be opened\n");  exit(-1);  }  char A[SIZE];  fgets(A,SIZE,file\_in);  int len=0,flag=1;  for (int i=0;A[i]!='\0';i++)  {  len++;  }  len--;  for (int i=0;i<len;i++,len--)  {  if (A[i]!=A[len])  {  flag=0;  break;  }  }  if (flag==1)  {  fprintf(file\_out,"It is a palindrome");  }  else  {  fprintf(file\_out,"It is not a palindrome");  }  fprintf(stdout,"Data successfully written to file\n");  fclose(file\_in);  fclose(file\_out);  return 0;  } |
| **Output** |

|  |
| --- |
| **8. Area of shapes** |
| **Code**  //AREA OF SHAPES  #define PI 3.14  #include <stdio.h>  #include <stdlib.h>  int main()  {  FILE \*file\_in = fopen("input.txt","r");  FILE \*file\_out = fopen("output.txt","w");  if (file\_in == NULL || file\_out == NULL)  {  printf("File could not be opened\n");  exit(-1);  }  float length,breadth,base,height,radius;  fscanf(file\_in,"%f%f",&length,&breadth);  fprintf(file\_out,"Area of rectangle is %f\n\n",length\*breadth);  fscanf(file\_in,"%f%f",&base,&height);  fprintf(file\_out,"Area of triangle is %f\n\n",0.5\*base\*height);  fscanf(file\_in,"%f",&radius);  fprintf(file\_out,"Area of circle is %f",PI\*radius\*radius);  fprintf(stdout,"Data successfully written to file\n");  fclose(file\_in);  fclose(file\_out);  return 0;  } |
| **Output** |

|  |
| --- |
| **9. Simple Calculator** |
| **Code**  //SIMPLE CALCULATOR  #include <stdio.h>  #include <stdlib.h>  int main()  {  FILE \*file\_in = fopen("input.txt","r");  FILE \*file\_out = fopen("output.txt","w");  if (file\_in == NULL || file\_out == NULL)  {  printf("File could not be opened\n");  exit(-1);  }  float num1,num2;  int choice;  fscanf(file\_in,"%f%f",&num1,&num2);  printf("CALCULATOR\n");  printf("1.Addition\n2.Subtraction\n3.Multiplication\n4.Division\n5.Exit\n");  for (;;)  {  fscanf(file\_in,"%d",&choice);  switch (choice)  {  case 1:fprintf(file\_out,"%f\n",num1+num2); break;  case 2:fprintf(file\_out,"%f\n",num1-num2); break;  case 3:fprintf(file\_out,"%f\n",num1\*num2); break;  case 4:fprintf(file\_out,"%f\n",num1/num2); break;  case 5:fprintf(stdout,"Data successfully written to file\n");  fclose(file\_in);  fclose(file\_out);  exit(0); break;  }  }  } |
| **Output** |

|  |
| --- |
| **10. Array operations (Maximum, minimum, sum and average)** |
| **Code**  //ARRAY OPERATIONS  #include <stdio.h>  #include <stdlib.h>  #define SIZE 100  int maximum(int\* ,int );  int minimum(int\* ,int );  int sum(int \*,int );  float average(int \*,int );  int main()  {  FILE \*file\_in = fopen("input.txt","r");  FILE \*file\_out = fopen("output.txt","w");  if (file\_in == NULL || file\_out == NULL)  {  printf("File could not be opened\n");  exit(-1);  }  int arr[SIZE];  int N;  fscanf(file\_in,"%d",&N);  for (int i=0;i<N;i++)  {  fscanf(file\_in,"%d",&arr[i]);  }  fprintf(file\_out,"The largest number is %d\n",maximum(arr,N));  fprintf(file\_out,"The smallest number is %d\n",minimum(arr,N));  fprintf(file\_out,"The sum of the array is %d\n",sum(arr,N));  fprintf(file\_out,"The average of the array is %f\n",average(arr,N));  fprintf(stdout,"Data successfully written to file\n");  fclose(file\_in);  fclose(file\_out);  return 0;  }  int maximum(int \*arr,int N)  {  int max=arr[0];  for (int i=0;i<N;i++)  {  if (arr[i]>max)  max=arr[i];  }  return max;  }  int minimum(int \*arr,int N)  {  int min=arr[0];  for (int i=0;i<N;i++)  {  if (arr[i]<min)  min=arr[i];  }  return min;  }  int sum(int \*arr,int N)  {  int Sum=0;  for (int i=0;i<N;i++)  {  Sum+=arr[i];  }  return Sum;  }  float average(int \*arr,int N)  {  float Average=0;  for (int i=0;i<N;i++)  {  Average+=arr[i];  }  return Average/N;  } |
| **Output** |

|  |
| --- |
| **11.a) String operations (Copying)** |
| **Code**  //COPY STRING  #include <stdio.h>  #include <stdlib.h>  #define SIZE 100  int main()  {  FILE \*file\_in = fopen("input.txt","r");  FILE \*file\_out = fopen("output.txt","w");  if (file\_in == NULL || file\_out == NULL)  {  printf("File could not be opened\n");  exit(-1);  }  char A[SIZE],B[SIZE];  int i;  fgets(A,SIZE,file\_in);  for (i=0;A[i]!='\0';i++)  {  B[i]=A[i];  }  B[i]='\0';  fprintf(file\_out,"String B:%s",B);  fprintf(stdout,"Data successfully written to file\n");  fclose(file\_in);  fclose(file\_out);  return 0;  } |
| **Output** |

|  |
| --- |
| **11.b) String operations (Concatenating strings)** |
| **Code**  //CONCATENATING STRING  #include <stdio.h>  #define SIZE 100  int main()  {  char A[SIZE],B[SIZE];  int i,j,k;  printf("Enter string A:");  gets(A);  printf("Enter string B:");  gets(B);  for (i=0;A[i]!='\0';i++);  for (j=i,k=0;B[i]!='\0';j++,k++)  {  A[j]=B[k];  }  A[j]='\0';  printf("Concatenated string:%s",A);  return 0;  } |
| **Output** |

|  |
| --- |
| **11.c) String operations (Comparing strings)** |
| **Code**  //COMPARING STRINGS  #include <stdio.h>  #define SIZE 100  int main()  {  char A[SIZE];  char B[SIZE];  int lenA,lenB;  int i,j;  printf("Enter the string A:");  gets(A);  printf("Enter the string B:");  gets(B);  for (i=0;A[i]!='\0';i++);  for (j=0;B[j]!='\0';j++);  lenA=i;  lenB=j;  if (lenA>lenB)  printf("String A is longer than string B");  else if (lenA<lenB)  printf("String A is shorter than string B");  else  printf("Both strings have same length");  return 0;  } |
| **Output** |

|  |
| --- |
| **12. Linear Search** |
| **Code**  //LINEAR SEARCH  #include <stdio.h>  #define SIZE 100  int Search(int\* ,int ,int );  int main()  {  int arr[SIZE];  int result,N,num;  printf("Enter the length of the array:");  scanf("%d",&N);  printf("Enter the array:");  for (int i=0;i<N;i++)  {  scanf("%d",&arr[i]);  }  printf("Enter the number to search for:");  scanf("%d",&num);  result=Search(arr,N,num);  (result==-1)?printf("Number is not found\n"):printf("Number is found at index %d",result);  return 0;  }  int Search(int \*arr,int N,int num)  {  for (int i=0;i<N;i++)  {  if (arr[i]==num)  return i;  }  return -1;  } |
| **Output** |

|  |
| --- |
| **13. Binary Search** |
| **Code**  // BINARY SEARCH  #include <stdio.h>  int binarySearch(int arr[],int l, int r, int x)  {  while (l <= r) {  int m = l + (r - l) / 2;  if (arr[m] == x)  return m;  if (arr[m] < x)  l = m + 1;  else  r = m - 1;  }  return -1;  }  int main()  {  int arr[] = {1,2,3,4,5};  int n = sizeof(arr) / sizeof(arr[0]);  int x = 5;  int result = binarySearch(arr, 0, n - 1, x);  (result == -1) ? printf("Element is not present"  " in array")  : printf("Element is present at "  "index %d",  result);  return 0;  } |
| **Output** |

|  |
| --- |
| **14. Bubble sort** |
| **Code**  //BUBBLE SORT  #include <stdio.h>  #define SIZE 100  int main()  {  int arr[SIZE];  int n,temp;  printf("Enter the size of the array:");  scanf("%d",&n);  printf("Enter the array:");  for (int i=0;i<n;i++)  {  scanf("%d",&arr[i]);  }  for (int i=0;i<n;i++)  {  for (int j=i+1;j<n;j++)  {  if (arr[i]>arr[j])  {  temp=arr[i];  arr[i]=arr[j];  arr[j]=temp;  }  }  }  for (int i=0;i<n;i++)  {  printf("%d ",arr[i]);  }  return 0;  } |
| **Output** |

|  |
| --- |
| **15. Selection sort** |
| **Code**  #include <stdio.h>  void selection(int arr[], int n)  {  int i, j, small;  for (i = 0; i < n-1; i++)  {  small = i;  for (j = i+1; j < n; j++)  if (arr[j] < arr[small])  small = j;  int temp = arr[small];  arr[small] = arr[i];  arr[i] = temp;  }  }  void printArr(int a[], int n)  {  int i;  for (i = 0; i < n; i++)  printf("%d ", a[i]);  }  int main()  {  int a[] = {7,9,8,5,3,19};  int n = sizeof(a) / sizeof(a[0]);  printf("Before sorting array elements are - \n");  printArr(a, n);  selection(a, n);  printf("\nAfter sorting array elements are - \n");  printArr(a, n);  return 0;  } |
| **Output** |

|  |
| --- |
| **16. Insertion Sort** |
| **Code**  #include <stdio.h>  void insert(int a[], int n)  {  int i, j, temp;  for (i = 1; i < n; i++) {  temp = a[i];  j = i - 1;  while(j>=0 && temp <= a[j])  {  a[j+1] = a[j];  j = j-1;  }  a[j+1] = temp;  }  }  void printArr(int a[], int n)  {  int i;  for (i = 0; i < n; i++)  printf("%d ", a[i]);  }  int main()  {  int a[] = {8,90,76,54,85,74};  int n = sizeof(a) / sizeof(a[0]);  printf("Before sorting array elements are - \n");  printArr(a, n);  insert(a, n);  printf("\nAfter sorting array elements are - \n");  printArr(a, n);  return 0;  } |
| **Output** |

|  |
| --- |
| **17. Matrix operations (addition, subtraction, multiplication and transpose)** |
| **Code**  //MATRIX OPERATIONS  #include <stdio.h>  #include <stdlib.h>  #define SIZE 4  int main()  {  int m,n1,n2,p;  int mat1[SIZE][SIZE],mat2[SIZE][SIZE],mat3[SIZE][SIZE];  printf("Matrix 1:\n");  printf("Number of rows:");  scanf("%d",&m);  printf("Number of columns:");  scanf("%d",&n1);  printf("\n");  printf("Matrix 2:\n");  printf("Number of rows:");  scanf("%d",&n2);  printf("Number of columns:");  scanf("%d",&p);  printf("Multiplication of matrices\n");  if (n1!=n2)  {  printf("\nMatrix multiplication is not possible");  exit(0);  }  printf("\n\n");  printf("Enter matrix 1:\n");  for (int i=0;i<m;i++)  {  for (int j=0;j<n1;j++)  {  scanf("%d",&mat1[i][j]);  }  }  printf("\n");  printf("Enter matrix 2:\n");  for (int i=0;i<n2;i++)  {  for (int j=0;j<p;j++)  {  scanf("%d",&mat2[i][j]);  }  }  int n=n1;  for (int i=0;i<m;i++)  {  for (int j=0;j<p;j++)  {  mat3[i][j]=0;  for (int k=0;k<n;k++)  {  mat3[i][j]+=mat1[i][k]\*mat2[k][j];  }  }  }  printf("\n\nThe resultant matrix is:\n");  for (int i=0;i<m;i++)  {  for (int j=0;j<p;j++)  {  printf("%d ",mat3[i][j]);  }  printf("\n");  }  printf("\n\nAddition of matrices\n");  if (m==n2 && n1==p)  {  for (int i=0;i<m;i++)  {  for (int j=0;j<p;j++)  {  mat3[i][j]=mat1[i][j]+mat2[i][j];  }  }  }  else  {  printf("Matrix addition is not possible\n");  exit(0);  }  for (int i=0;i<m;i++)  {  for (int j=0;j<p;j++)  {  printf("%d ",mat3[i][j]);  }  printf("\n");  }  printf("\n\nSubtraction of matrices\n");  if (m==n2 && n1==p)  {  for (int i=0;i<m;i++)  {  for (int j=0;j<p;j++)  {  mat3[i][j]=mat1[i][j]-mat2[i][j];  }  }  }  else  {  printf("Matrix subtraction is not possible\n");  exit(0);  }  for (int i=0;i<m;i++)  {  for (int j=0;j<p;j++)  {  printf("%d ",mat3[i][j]);  }  printf("\n");  }  printf("\n\nMatrix Transpose\n");  for (int i=0;i<n1;i++)  {  for (int j=0;j<m;j++)  {  mat1[i][j]=mat1[j][i];  }  printf("\n");  }  for (int i=0;i<n1;i++)  {  for (int j=0;j<m;j++)  {  printf("%d ",mat1[i][j]);  }  printf("\n");  }  } |
| **Output** |

|  |
| --- |
| **18.a) Singly Linked List**  **1) Insert End**  **2) Insert Front**  **3) Delete End**  **4) Delete Front**  **5) Insert Position**  **6) Delete Position**  **7) Counting number of nodes**  **8) Reversing the list**  **9) Display list**  **10) Searching node** |
| **Code**  //SINGLY LINKED LIST  #include <stdio.h>  #include <stdlib.h>  struct node  {  int data;  struct node \*next;  };  typedef struct node \*NODE;  FILE \*file\_in;  FILE \*file\_out;  NODE CreateNode();  void DisplayList(NODE );  NODE InsertEnd(NODE );  NODE InsertFront(NODE );  NODE DeleteEnd(NODE );  NODE DeleteFront(NODE );  NODE InsertPosition(NODE ,int );  NODE DeletePosition(NODE, int );  int CountNodes(NODE );  NODE ReverseList(NODE );  void SearchNode(NODE ,int );  int main()  {  file\_in = fopen("input.txt","r");  file\_out = fopen("output.txt","w");  if (file\_in == NULL || file\_out == NULL)  {  printf("File could not be opened\n");  exit(-1);  }  NODE head=NULL;  int choice,pos,count,toSearch;  printf("SINGLY LINKED LIST\n");  printf("\n1.Insert End\n2.Insert Front\n3.Display List\n4.Delete End\n5.Delete Front\n6.Insert at Position\n7.Delete at Position\n8.Count number of nodes\n9.Reverse the list\n10.Search Node\n11.End\n");  for (;;)  {  fscanf(file\_in,"%d",&choice);  switch (choice)  {  case 1:head=InsertEnd(head); break;  case 2:head=InsertFront(head); break;  case 3:DisplayList(head); break;  case 4:head=DeleteEnd(head); break;  case 5:head=DeleteFront(head); break;  case 6:fscanf(file\_in,"%d",&pos);  head=InsertPosition(head,pos); break;  case 7:fscanf(file\_in,"%d",&pos);  head=DeletePosition(head,pos); break;  case 8:count=CountNodes(head);  fprintf(file\_out,"\nThe number of nodes is %d\n",count); break;  case 9:head=ReverseList(head); break;  case 10:fscanf(file\_in,"%d",&toSearch);  SearchNode(head,toSearch); break;  case 11:fprintf(stdout,"\nData successfully written to file\n");  fclose(file\_in);  fclose(file\_out);  exit(0); break;  }  }  }  NODE CreateNode()  {  NODE New,cur;  New=(NODE)malloc(sizeof(struct node));  if (New==NULL)  {  fprintf(file\_out,"Memory not allocated");  exit(1);  }  fscanf(file\_in,"%d",&New->data);  New->next=NULL;  return New;  }  void DisplayList(NODE head)  {  NODE cur;  cur=head;  if (head==NULL)  {  fprintf(file\_out,"\nEmpty List\n");  return;  }  fprintf(file\_out,"\n");  while (cur!=NULL)  {  fprintf(file\_out,"%d ",cur->data);  cur=cur->next;  }  fprintf(file\_out,"\n");  }  NODE InsertEnd(NODE head)  {  NODE New,cur;  New=CreateNode();  if (head==NULL)  {  head=New;  }  else  {  cur=head;  while (cur->next!=NULL)  {  cur=cur->next;  }  cur->next=New;  }  New->next=NULL;  return head;  }  NODE InsertFront(NODE head)  {  NODE New;  New=CreateNode();  if (head==NULL)  {  New->next=NULL;  }  else  {  New->next=head;  }  head=New;  return head;  }  NODE DeleteEnd(NODE head)  {  NODE cur,prev;  if (head==NULL)  {  fprintf(file\_out,"\nEmpty List\n");  }  else if (head->next==NULL)  {  free(head);  return NULL;  }  else  {  cur=head;  prev=NULL;  while (cur->next!=NULL)  {  prev=cur;  cur=cur->next;  }  fprintf(file\_out,"\nDeleted node is %d\n",cur->data);  prev->next=NULL;  free(cur);  }  return head;  }  NODE DeleteFront(NODE head)  {  NODE cur;  if (head==NULL)  {  fprintf(file\_out,"\nEmpty List\n");  }  else if (head->next==NULL)  {  free(head);  return NULL;  }  else  {  cur=head;  head=head->next;  fprintf(file\_out,"\nDeleted node is %d\n",cur->data);  free(cur);  }  return head;  }  NODE InsertPosition(NODE head,int pos)  {  NODE cur,prev,New;  int count=0,temp=0;  New=CreateNode();  if (pos<=0)  {  fprintf(file\_out,"\nInvalid position\n");  free(New);  return head;  }  cur=head;  while (cur!=NULL)  {  cur=cur->next;  count++;  }  if (pos==1)  {  if (head!=NULL)  New->next=head;  else  New->next=NULL;  head=New;  return head;  }  if (pos<=count+1)  {  prev=NULL;  cur=head;  while (cur!=NULL && temp!=pos-1)  {  prev=cur;  cur=cur->next;  temp++;  }  prev->next=New;  New->next=cur;  return head;  }  if (pos>count+1)  {  fprintf(file\_out,"\nInvalid position\n");  free(New);  return head;  }  }  NODE DeletePosition(NODE head,int pos)  {  NODE prev,cur,New;  int count=0,temp=0;  cur=head;  while (cur!=NULL)  {  cur=cur->next;  count++;  }  if (head==NULL)  {  fprintf(file\_out,"\nList is empty\n");  return head;  }  if (pos<=0)  {  fprintf(file\_out,"\nInvalid position\n");  return head;  }  if (pos==1)  {  cur=head;  head=head->next;  free(cur);  return head;  }  if (pos<=count)  {  prev=NULL;  cur=head;  while (cur!=NULL && temp!=pos-1)  {  prev=cur;  cur=cur->next;  temp++;  }  prev->next=cur->next;  free(cur);  return head;  }  if (pos>count)  {  fprintf(file\_out,"\nInvalid Position\n");  return head;  }  }  int CountNodes(NODE head)  {  int count=0;  NODE cur=head;  while (cur!=NULL)  {  cur=cur->next;  count++;  }  return count;  }  NODE ReverseList(NODE head)  {  NODE prev,cur,temp;  if (head==NULL)  {  fprintf(file\_out,"\nList is empty\n");  return head;  }  prev=NULL;  cur=head;  temp=NULL;  while (cur!=NULL)  {  prev=cur;  cur=cur->next;  prev->next=temp;  temp=prev;  }  head=temp;  return head;  }  void SearchNode(NODE head,int toSearch)  {  NODE cur;  int flag=0;  int position=1;  cur=head;  while (cur!=NULL)  {  if (cur->data==toSearch)  {  flag=1;  break;  }  position++;  cur=cur->next;  }  if (flag==1)  {  fprintf(file\_out,"\nElement %d found at position %d\n",toSearch,position);  }  else  {  fprintf(file\_out,"\n%d not found in the list\n",toSearch);  }  } |
| Output |

|  |
| --- |
| **18.b) Doubly Linked List**  **11) Insert End**  **12) Insert Front**  **13) Delete End**  **14) Delete Front**  **15) Insert Position**  **16) Delete Position**  **17) Counting number of nodes**  **18) Reversing the list**  **19) Display list**  **20) Searching node** |
| **Code**  //DOUBLY LINKED LIST  #include <stdio.h>  #include <stdlib.h>  struct node  {  int data;  struct node \*next;  struct node \*prev;  };  typedef struct node \*NODE;  FILE \*file\_in;  FILE \*file\_out;  NODE CreateNode();  NODE InsertEnd(NODE );  NODE InsertFront(NODE );  NODE DeleteEnd(NODE );  NODE DeleteFront(NODE );  NODE InsertPos(NODE ,int );  NODE DeletePos(NODE ,int );  int CountNodes(NODE );  void SearchNode(NODE ,int);  void DisplayForward(NODE );  void DisplayReverse(NODE );  int main()  {  file\_in = fopen("input.txt","r");  file\_out = fopen("output.txt","w");  if (file\_in == NULL || file\_out == NULL)  {  printf("File could not be opened\n");  exit(-1);  }  NODE head=NULL;  int choice,pos,toSearch,count;  printf("\nDOUBLY LINKED LIST\n");  printf("\n1.Insert End\n2.Insert Front\n3.Delete End\n4.Delete Front\n5.Insert at Position\n6.Delete at Position\n7.Count Nodes\n8.Search Nodes\n9.Display Forward\n10.Display Reverse\n11.Exit\n");  for (;;)  {  fscanf(file\_in,"%d",&choice);  switch (choice)  {  case 1:head=InsertEnd(head); break;  case 2:head=InsertFront(head); break;  case 3:head=DeleteEnd(head); break;  case 4:head=DeleteFront(head); break;  case 5:fscanf(file\_in,"%d",&pos);  head=InsertPos(head,pos); break;  case 6:fscanf(file\_in,"%d",&pos);  head=DeletePos(head,pos); break;  case 7:count=CountNodes(head);  fprintf(file\_out,"\nThe number of nodes in the linked list is %d\n",count); break;  case 8:fscanf(file\_in,"%d",&toSearch);  SearchNode(head,toSearch); break;  case 9:DisplayForward(head); break;  case 10:DisplayReverse(head); break;  case 11:fprintf(stdout,"\nData successfully written to file\n");  fclose(file\_in);  fclose(file\_out);  exit(0); break;  }  }  }  NODE CreateNode()  {  NODE New;  New=(NODE)malloc(sizeof(struct node));  if (New==NULL)  {  fprintf(file\_out,"\nMemory not allocated!\n");  exit(1);  }  fscanf(file\_in,"%d",&New->data);  New->next=NULL;  New->prev=NULL;  return New;  }  NODE InsertEnd(NODE head)  {  NODE New,cur;  New=CreateNode();  if (head==NULL)  {  head=New;  New->prev=NULL;  return head;  }  for (cur=head;cur->next!=NULL;cur=cur->next);  cur->next=New;  New->prev=cur;  return head;  }  NODE InsertFront(NODE head)  {  NODE New,cur;  New=CreateNode();  if (head==NULL)  {  head=New;  New->prev=NULL;  New->next=NULL;  return head;  }  New->next=head;  head->prev=New;  New->prev=NULL;  head=New;  return head;  }  NODE DeleteEnd(NODE head)  {  NODE cur,temp;  if (head==NULL)  {  fprintf(file\_out,"\nList is empty\n");  return head;  }  if (head->next==NULL)  {  free(head);  return NULL;  }  for (temp=NULL,cur=head;cur->next!=NULL;temp=cur,cur=cur->next);  free(cur);  temp->next=NULL;  return head;  }  NODE DeleteFront(NODE head)  {  NODE cur;  if (head==NULL)  {  fprintf(file\_out,"\nList is empty\n");  return head;  }  if (head->next==NULL)  {  free(head);  return NULL;  }  cur=head;  head=head->next;  free(cur);  head->prev=NULL;  return head;  }  NODE InsertPos(NODE head,int pos)  {  NODE New,cur,temp;  int count,temp1;  New=CreateNode();  if (pos==1)  {  if (head==NULL)  head=New;  else  {  New->next=head;  head->prev=New;  head=New;  New->prev;  }  return head;  }  if (pos<1)  {  fprintf(file\_out,"\nInvalid position!!\n");  return head;  }  for (cur=head,count=0;cur!=NULL;cur=cur->next,count++);  if (pos<count+1)  {  for (temp=NULL,temp1=0,cur=head;cur!=NULL && temp1!=pos-1;temp=cur,cur=cur->next,temp1++);  temp->next=New;  New->prev=temp;  New->next=cur;  cur->prev=New;  return head;  }  if (pos==count+1)  {  for (cur=head;cur->next!=NULL;cur=cur->next);  cur->next=New;  New->prev=cur;  New->next=NULL;  return head;  }  if (pos>count+1)  {  fprintf(file\_out,"\nInvalid position\n");  return head;  }  }  NODE DeletePos(NODE head,int pos)  {  NODE temp,cur,New;  int count=0,temp1=0;  cur=head;  while (cur!=NULL)  {  cur=cur->next;  count++;  }  if (head==NULL)  {  fprintf(file\_out,"\nList is empty\n");  return head;  }  if (pos<=0)  {  fprintf(file\_out,"\nInvalid position\n");  return head;  }  if (pos==1)  {  if (head->next==NULL)  {  free(head);  return NULL;  }  cur=head;  head=head->next;  head->prev=NULL;  free(cur);  return head;  }  if (pos<count)  {  temp=NULL;  cur=head;  while (cur->next!=NULL && temp1!=pos-1)  {  temp=cur;  cur=cur->next;  temp1++;  }  temp->next=cur->next;  cur->next->prev=temp;  free(cur);  return head;  }  if (pos==count)  {  for (temp=NULL,cur=head;cur->next!=NULL;temp=cur,cur=cur->next);  temp->next=NULL;  free(cur);  return head;  }  if (pos>count)  {  fprintf(file\_out,"\nInvalid Position\n");  return head;  }  }  int CountNodes(NODE head)  {  int count=0;  NODE cur=head;  while (cur!=NULL)  {  cur=cur->next;  count++;  }  return count;  }  void SearchNode(NODE head,int toSearch)  {  NODE cur;  int flag=0;  int position=1;  cur=head;  while (cur!=NULL)  {  if (cur->data==toSearch)  {  flag=1;  break;  }  position++;  cur=cur->next;  }  if (flag==1)  {  fprintf(file\_out,"\nElement %d found at position %d\n",toSearch,position);  }  else  {  fprintf(file\_out,"\n%d not found in the list\n",toSearch);  }  }  void DisplayForward(NODE head)  {  NODE cur;  cur=head;  if (head==NULL)  {  fprintf(file\_out,"\nEmpty List\n");  return;  }  fprintf(file\_out,"\n");  while (cur!=NULL)  {  fprintf(file\_out,"%d ",cur->data);  cur=cur->next;  }  fprintf(file\_out,"\n");  }  void DisplayReverse(NODE head)  {  NODE cur,tail;  cur=head;  if (head==NULL)  {  fprintf(file\_out,"\nEmpty list\n");  return;  }  fprintf(file\_out,"\n");  for (cur=head;cur->next!=NULL;cur=cur->next);  tail=cur;  for (cur=tail;cur!=NULL;cur=cur->prev)  {  fprintf(file\_out,"%d ",cur->data);  }  fprintf(file\_out,"\n");  } |
| Output |

|  |
| --- |
| **18.c) Circular Singly Linked List**  **21) Insert End**  **22) Insert Front**  **23) Delete End**  **24) Delete Front**  **25) Insert Position**  **26) Display list** |
| **Code**  //BASIC CIRCULAR LINKED LIST OPERATIONS  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  FILE \*file\_in;  FILE \*file\_out;  struct node  {  int data;  struct node \*next;  };  typedef struct node \*NODE;  NODE getnode();  NODE read\_details();  NODE insert\_end(NODE head);  NODE insert\_front(NODE head);  NODE delete\_end(NODE head);  NODE delete\_front(NODE head);  NODE insert\_pos(NODE head,int pos);  NODE delete\_pos(NODE head,int pos);  void display\_details(NODE head);  int main()  {  file\_in = fopen("input.txt","r");  file\_out = fopen("output.txt","w");  if (file\_in == NULL || file\_out == NULL)  {  printf("File could not be opened\n");  exit(-1);  }  NODE head=NULL;  int choice,pos;  printf("\nCIRCULAR LINKED LIST\n");  printf("\n1.Insert end\n2.Display list\n3.Insert front\n4.Delete end\n5.Delete front\n6.Insert at position\n7.Delete at position\n8.Exit\n");  for (;;)  {  fscanf(file\_in,"%d",&choice);  switch (choice)  {  case 1:head=insert\_end(head); break;  case 2:display\_details(head); break;  case 3:head=insert\_front(head); break;  case 4:head=delete\_end(head); break;  case 5:head=delete\_front(head); break;  case 6:fscanf(file\_in,"%d",&pos);  head=insert\_pos(head,pos); break;  case 7:fscanf(file\_in,"%d",&pos);  head=delete\_pos(head,pos); break;  case 8:fprintf(stdout,"\nData successfully written to file\n");  fclose(file\_in);  fclose(file\_out);  exit(0); break;  }  }  return 0;  }  NODE getnode()  {  NODE New;  New=(NODE)malloc(sizeof(struct node));  if (New==NULL)  {  fprintf(file\_out,"Memory not allocated\n");  exit(0);  }  New->next=New;  return New;  }  NODE read\_details()  {  NODE temp;  temp=getnode();  fscanf(file\_in,"%d",&temp->data);  return temp;  }  NODE insert\_end(NODE head)  {  NODE New,cur;  New=read\_details();  New->next=New;  if (head==NULL)  {  return New;  }  cur=head;  while (cur->next!=head)  {  cur=cur->next;  }  cur->next=New;  New->next=head;  fprintf(file\_out,"\n");  return head;  }  void display\_details(NODE head)  {  NODE cur;  if (head==NULL)  {  fprintf(file\_out,"List is empty");  return;  }  cur=head;  while (cur->next!=head)  {  fprintf(file\_out,"%d ",cur->data);  cur=cur->next;  }  fprintf(file\_out,"%d ",cur->data);  fprintf(file\_out,"\n");  }  NODE insert\_front(NODE head)  {  NODE New,cur;  New=read\_details();  New->next=New;  if (head==NULL)  {  return New;  }  cur=head;  while (cur->next!=head)  {  cur=cur->next;  }  cur->next=New;  New->next=head;  head=New;  fprintf(file\_out,"\n");  return head;  }  NODE delete\_end(NODE head)  {  NODE prev,cur;  if (head==NULL)  {  fprintf(file\_out,"List is empty\n");  return head;  }  if (head->next==head)  {  fprintf(file\_out,"Deleted element:%d\n",head->data);  free(head);  return NULL;  }  prev=NULL;  cur=head;  while (cur->next!=head)  {  prev=cur;  cur=cur->next;  }  prev->next=head;  fprintf(file\_out,"Deleted element:%d\n",cur->data);  free(cur);  return head;  }  NODE delete\_front(NODE head)  {  NODE temp,cur;  if (head==NULL)  {  fprintf(file\_out,"List is empty\n");  return head;  }  if (head->next==head)  {  fprintf(file\_out,"Deleted node:%d\n",head->data);  free(head);  return NULL;  }  cur=head;  while (cur->next!=head)  {  cur=cur->next;  }  temp=head;  head=temp->next;  cur->next=head;  fprintf(file\_out,"Deleted node:%d\n",temp->data);  free(temp);  return head;  }  NODE insert\_pos(NODE head,int pos)  {  NODE New,cur,prev;  for (prev=head;prev->next!=head;prev=prev->next);  if (head==NULL)  {  head=insert\_front(head);  return head;  }  if (pos==1)  {  head=insert\_front(head);  return head;  }  New=read\_details();  if (pos>1)  {  pos--;  for(cur=head;pos!=0;cur=cur->next,prev=prev->next,pos--);  prev->next=New;  New->next=cur;  return head;  }  fprintf(file\_out,"\n");  }  NODE delete\_pos(NODE head,int pos)  {  NODE New,cur,prev;  for (prev=head;prev->next!=head;prev=prev->next);  if (head==NULL)  {  printf("\nList is empty\n");  return head;  }  if (pos==1)  {  head=delete\_front(head);  return head;  }  if (pos>1)  {  pos--;  for(cur=head;pos!=0;cur=cur->next,prev=prev->next,pos--);  prev->next=cur->next;  free(cur);  return head;  }  } |
| Output |
| **18.d) Circular Doubly Linked List**  **27) Insert End**  **28) Insert Front**  **29) Delete End**  **30) Delete Front**  **31) Insert Position**  **32) Delete Position**  **33) Display list** |
| **Code**  //SINGLY LINKED LIST  #include <stdio.h>  #include <stdlib.h>  struct node  {  int data;  struct node \*next;  };  typedef struct node \*NODE;  FILE \*file\_in;  FILE \*file\_out;  NODE CreateNode();  void DisplayList(NODE );  NODE InsertEnd(NODE );  NODE InsertFront(NODE );  NODE DeleteEnd(NODE );  NODE DeleteFront(NODE );  NODE InsertPosition(NODE ,int );  NODE DeletePosition(NODE, int );  int CountNodes(NODE );  NODE ReverseList(NODE );  void SearchNode(NODE ,int );  int main()  {  file\_in = fopen("input.txt","r");  file\_out = fopen("output.txt","w");  if (file\_in == NULL || file\_out == NULL)  {  printf("File could not be opened\n");  exit(-1);  }  NODE head=NULL;  int choice,pos,count,toSearch;  printf("SINGLY LINKED LIST\n");  printf("\n1.Insert End\n2.Insert Front\n3.Display List\n4.Delete End\n5.Delete Front\n6.Insert at Position\n7.Delete at Position\n8.Count number of nodes\n9.Reverse the list\n10.Search Node\n11.End\n");  for (;;)  {  fscanf(file\_in,"%d",&choice);  switch (choice)  {  case 1:head=InsertEnd(head); break;  case 2:head=InsertFront(head); break;  case 3:DisplayList(head); break;  case 4:head=DeleteEnd(head); break;  case 5:head=DeleteFront(head); break;  case 6:fscanf(file\_in,"%d",&pos);  head=InsertPosition(head,pos); break;  case 7:fscanf(file\_in,"%d",&pos);  head=DeletePosition(head,pos); break;  case 8:count=CountNodes(head);  fprintf(file\_out,"\nThe number of nodes is %d\n",count); break;  case 9:head=ReverseList(head); break;  case 10:fscanf(file\_in,"%d",&toSearch);  SearchNode(head,toSearch); break;  case 11:fprintf(stdout,"\nData successfully written to file\n");  fclose(file\_in);  fclose(file\_out);  exit(0); break;  }  }  }  NODE CreateNode()  {  NODE New,cur;  New=(NODE)malloc(sizeof(struct node));  if (New==NULL)  {  fprintf(file\_out,"Memory not allocated");  exit(1);  }  fscanf(file\_in,"%d",&New->data);  New->next=NULL;  return New;  }  void DisplayList(NODE head)  {  NODE cur;  cur=head;  if (head==NULL)  {  fprintf(file\_out,"\nEmpty List\n");  return;  }  fprintf(file\_out,"\n");  while (cur!=NULL)  {  fprintf(file\_out,"%d ",cur->data);  cur=cur->next;  }  fprintf(file\_out,"\n");  }  NODE InsertEnd(NODE head)  {  NODE New,cur;  New=CreateNode();  if (head==NULL)  {  head=New;  }  else  {  cur=head;  while (cur->next!=NULL)  {  cur=cur->next;  }  cur->next=New;  }  New->next=NULL;  return head;  }  NODE InsertFront(NODE head)  {  NODE New;  New=CreateNode();  if (head==NULL)  {  New->next=NULL;  }  else  {  New->next=head;  }  head=New;  return head;  }  NODE DeleteEnd(NODE head)  {  NODE cur,prev;  if (head==NULL)  {  fprintf(file\_out,"\nEmpty List\n");  }  else if (head->next==NULL)  {  free(head);  return NULL;  }  else  {  cur=head;  prev=NULL;  while (cur->next!=NULL)  {  prev=cur;  cur=cur->next;  }  fprintf(file\_out,"\nDeleted node is %d\n",cur->data);  prev->next=NULL;  free(cur);  }  return head;  }  NODE DeleteFront(NODE head)  {  NODE cur;  if (head==NULL)  {  fprintf(file\_out,"\nEmpty List\n");  }  else if (head->next==NULL)  {  free(head);  return NULL;  }  else  {  cur=head;  head=head->next;  fprintf(file\_out,"\nDeleted node is %d\n",cur->data);  free(cur);  }  return head;  }  NODE InsertPosition(NODE head,int pos)  {  NODE cur,prev,New;  int count=0,temp=0;  New=CreateNode();  if (pos<=0)  {  fprintf(file\_out,"\nInvalid position\n");  free(New);  return head;  }  cur=head;  while (cur!=NULL)  {  cur=cur->next;  count++;  }  if (pos==1)  {  if (head!=NULL)  New->next=head;  else  New->next=NULL;  head=New;  return head;  }  if (pos<=count+1)  {  prev=NULL;  cur=head;  while (cur!=NULL && temp!=pos-1)  {  prev=cur;  cur=cur->next;  temp++;  }  prev->next=New;  New->next=cur;  return head;  }  if (pos>count+1)  {  fprintf(file\_out,"\nInvalid position\n");  free(New);  return head;  }  }  NODE DeletePosition(NODE head,int pos)  {  NODE prev,cur,New;  int count=0,temp=0;  cur=head;  while (cur!=NULL)  {  cur=cur->next;  count++;  }  if (head==NULL)  {  fprintf(file\_out,"\nList is empty\n");  return head;  }  if (pos<=0)  {  fprintf(file\_out,"\nInvalid position\n");  return head;  }  if (pos==1)  {  cur=head;  head=head->next;  free(cur);  return head;  }  if (pos<=count)  {  prev=NULL;  cur=head;  while (cur!=NULL && temp!=pos-1)  {  prev=cur;  cur=cur->next;  temp++;  }  prev->next=cur->next;  free(cur);  return head;  }  if (pos>count)  {  fprintf(file\_out,"\nInvalid Position\n");  return head;  }  }  int CountNodes(NODE head)  {  int count=0;  NODE cur=head;  while (cur!=NULL)  {  cur=cur->next;  count++;  }  return count;  }  NODE ReverseList(NODE head)  {  NODE prev,cur,temp;  if (head==NULL)  {  fprintf(file\_out,"\nList is empty\n");  return head;  }  prev=NULL;  cur=head;  temp=NULL;  while (cur!=NULL)  {  prev=cur;  cur=cur->next;  prev->next=temp;  temp=prev;  }  head=temp;  return head;  }  void SearchNode(NODE head,int toSearch)  {  NODE cur;  int flag=0;  int position=1;  cur=head;  while (cur!=NULL)  {  if (cur->data==toSearch)  {  flag=1;  break;  }  position++;  cur=cur->next;  }  if (flag==1)  {  fprintf(file\_out,"\nElement %d found at position %d\n",toSearch,position);  }  else  {  fprintf(file\_out,"\n%d not found in the list\n",toSearch);  }  } |
| Output |

|  |
| --- |
| **19. Stack Implementation (using array and linked list)**  **34) Push**  **35) Pop**  **36) Display** |
| **Code**  //ARRAY IMPLEMENTATION OF STACK  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #define SIZE 5  FILE \*file\_in;  FILE \*file\_out;  struct stack  {  int data[SIZE];  int top;  };  typedef struct stack;  int isfull(stack s);  int isempty(stack s);  void push(stack \*ps,int num);  void pop(stack \*ps);  void display(stack s);  int main()  {  file\_in = fopen("input.txt","r");  file\_out = fopen("output.txt","w");  if (file\_in == NULL || file\_out == NULL)  {  printf("File could not be opened\n");  exit(-1);  }  stack s;  int num,choice;  s.top=-1;  printf("\nSTACK\n");  printf("1.Push\n2.Pop\n3.Display\n4.Exit\n");  for (;;)  {  fscanf(file\_in,"%d",&choice);  switch (choice)  {  case 1:  fscanf(file\_in,"%d",&num);  push(&s,num);  break;  case 2:pop(&s);  break;  case 3:display(s);  break;  case 4:fprintf(stdout,"\nData successfully written to file\n");  fclose(file\_in);  fclose(file\_out);  exit(0); break;  }  }  }  int isfull(stack s)  {  if (s.top==SIZE-1)  return 1;  else  return 0;  }  int isempty(stack s)  {  if (s.top==-1)  return 1;  else  return 0;  }  void push(stack \*ps,int num)  {  if (isfull(\*ps))  {  fprintf(file\_out,"Stack is full\n");  return;  }  ps->top++;  ps->data[ps->top]=num;  }  void pop(stack \*ps)  {  int temp;  if (isempty(\*ps))  {  fprintf(file\_out,"Stack is empty\n");  return;  }  temp=ps->data[ps->top];  ps->top--;  fprintf(file\_out,"Deleted data is %d\n",temp);  }  void display(stack s)  {  if (isempty(s))  {  printf("Stack is empty");  return;  }  int i;  for (i=s.top;i>-1;i--)  {  fprintf(file\_out,"%d\n",s.data[i]);  }  }  //LINKED LIST IMPLEMENTATION OF STACK  #include <stdio.h>  #include <stdlib.h>  FILE \*file\_in;  FILE \*file\_out;  struct stack  {  int data;  struct stack \*next;  };  typedef struct stack \*NODE;  NODE getnode();  NODE push(NODE head);  NODE pop(NODE head);  void display(NODE head);  int main()  {  file\_in = fopen("input.txt","r");  file\_out = fopen("output.txt","w");  if (file\_in == NULL || file\_out == NULL)  {  printf("File could not be opened\n");  exit(-1);  }  int choice;  NODE top=NULL;  printf("\nSTACK\n");  printf("1.Push the element\n2.Pop the element\n3.Display the elements\n4.Exit\n");  for (;;)  {  fscanf(file\_in,"%d",&choice);  switch (choice)  {  case 1:top=push(top); break;  case 2:top=pop(top); break;  case 3:display(top); break;  case 4:fprintf(stdout,"\nData successfully written to file\n");  fclose(file\_in);  fclose(file\_out);  exit(0); break;  }  }  }  NODE getnode()  {  NODE New;  New=(NODE)malloc(sizeof(struct stack));  if (New==NULL)  {  fprintf(file\_out,"Memory not allocated\n");  exit(0);  }  fscanf(file\_in,"%d",&New->data);  New->next=NULL;  return New;  }  NODE push(NODE top)  {  NODE New;  New=getnode();  if (top==NULL)  {  return New;  }  New->next=top;  top=New;  return top;  }  NODE pop(NODE top)  {  NODE temp;  if (top==NULL)  {  fprintf(file\_out,"List is empty\n");  return top;  }  temp=top;  top=top->next;  fprintf(file\_out,"Deleted element is %d\n",temp->data);  free(temp);  return top;  }  void display(NODE top)  {  NODE temp;  if (top==NULL)  {  fprintf(file\_out,"List is empty\n");  return;  }  temp=top;  while (temp!=NULL)  {  fprintf(file\_out,"%d\n",temp->data);  temp=temp->next;  }  } |
| Output |
| **20. Queue Implementation (Linear queue)** |
| **Code**  //LINEAR QUEUE  #include <stdio.h>  #include <stdlib.h>  FILE \*file\_in;  FILE \*file\_out;  #define MAX 5  int queue[MAX];  int front=-1;  int rear=-1;  void enqueue(void );  int dequeue(void );  void display(void );  int main()  {  file\_in = fopen("input.txt","r");  file\_out = fopen("output.txt","w");  if (file\_in == NULL || file\_out == NULL)  {  printf("File could not be opened\n");  exit(-1);  }  int choice,element;  printf("\nLINEAR QUEUE\n");  printf("1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");  for (;;)  {  fscanf(file\_in,"%d",&choice);  switch (choice)  {  case 1:enqueue(); break;  case 2:element=dequeue();  if (element!=-1)  fprintf(file\_out,"\n%d is deleted\n",element);  break;  case 3:display(); break;  case 4:fprintf(stdout,"\nData successfully written to file\n");  fclose(file\_in);  fclose(file\_out);  exit(0); break;  }  }  }  void enqueue()  {  int data;  fscanf(file\_in,"%d",&data);  if (rear==MAX-1)  fprintf(file\_out,"Queue Overflow");  else if (front==-1 && rear==-1)  {  front=0;  rear=0;  }  else  rear++;  queue[rear]=data;  }  int dequeue()  {  int data;  if (front==-1 || front>rear)  {  fprintf(file\_out,"Queue Underflow\n");  return -1;  }  else  {  data=queue[front];  front++;  if (front>rear)  front=rear=1;  return data;  }  }  void display()  {  if (front==-1 || front>rear)  printf("\nEmpty queue\n");  else  {  for (int i=front;i<=rear;i++)  fprintf(file\_out,"%d ",queue[i]);  }  fprintf(file\_out,"\n");  } |
| Output |

|  |
| --- |
| **21.Different Queues**  **1) Circular Queue**  **2) Priority Queue** |
| //CIRCULAR QUEUE  #include <stdio.h>  #include <stdlib.h>  FILE \*file\_in;  FILE \*file\_out;  #define SIZE 5  int queue[SIZE];  int front=-1;  int rear=-1;  void enqueue(void );  int dequeue(void );  void display(void );  int main()  {  file\_in = fopen("input.txt","r");  file\_out = fopen("output.txt","w");  if (file\_in == NULL || file\_out == NULL)  {  printf("File could not be opened\n");  exit(-1);  }  int choice,element;  printf("\nCIRCULAR QUEUE\n");  printf("1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");  for (;;)  {  fscanf(file\_in,"%d",&choice);  switch (choice)  {  case 1:enqueue(); break;  case 2:element=dequeue();  if (element!=-1)  fprintf(file\_out,"\n%d is deleted\n",element);  break;  case 3:display(); break;  case 4:fprintf(stdout,"\nData successfully written to file\n");  fclose(file\_in);  fclose(file\_out);  exit(0); break;  }  }  }  void enqueue()  {  int data;  fscanf(file\_in,"%d",&data);  if (front==0 && rear==SIZE-1)  fprintf(file\_out,"Queue Overflow");  else if (front==-1 && rear==-1)  {  front=0;  rear=0;  queue[rear]=data;  }  else if (rear==SIZE-1 && front!=0)  {  rear=0;  queue[rear]=data;  }  else  {  rear++;  queue[rear]=data;  }  }  int dequeue()  {  int data;  if (front==-1 || front>rear)  {  fprintf(file\_out,"Queue Underflow\n");  return -1;  }  data=queue[front];  if (front==rear)  front=rear=-1;  else  {  if (front==SIZE-1)  front=0;  else  front++;  }  return data;  }  void display()  {  if (front==-1 || front>rear)  fprintf(file\_out,"\nEmpty queue\n");  else  {  if (front<rear)  {  for (int i=front;i<=rear;i++)  fprintf(file\_out,"%d ",queue[i]);  }  else  {  for (int i=front;i<SIZE;i++)  fprintf(file\_out,"%d ",queue[i]);  for (int i=0;i<=rear;i++)  fprintf(file\_out,"%d ",queue[i]);  }  }  fprintf(file\_out,"\n");  } |
| Output |