|  |  |
| --- | --- |
| **Name of the student** | **Srusti bagewadi** |
| **Roll No** | **370** |
| **USN** | **01fe21bec028** |
| **Div** | **C division** |

**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: Variable: A variable in C programming is a user-defined or a user-readable custom name assigned to a memory location. Variables hold a value that can be modified and reused many times during the program execution.  Example: int roll\_no =36;  Data Type: A data type in C programming specifies the type of data that a variable can hold. It defines the size and layout of the variable's memory, as well as the range of values that it can take.  Example: int roll\_no; |
| **2. Explain the concept of data types in C programming. Discuss the different types of data types available in C.** |
| A: Data types in C programming define the type of data that a variable can hold. They determine the operations that can be performed on the data and the memory required to store it. Common data types in C include:  int: Integer data type to store whole numbers.  float: Floating-point data type to store decimal numbers.  char: Character data type to store single characters.  double: Double-precision floating-point data type.  void: Represents an absence of type or an incomplete type.  Arrays, Structures, Pointers: Composite data types that can hold multiple elements or references. |
| **3. How are variables declared and initialized in C programming? Provide examples of variable declarations with different data types.** |
| A: Variables are declared by specifying the data type followed by the variable name. They can be initialized (assigned an initial value) at the time of declaration or later.  Examples:  int roll\_no; // Declaration  roll\_no = 36; // Initialization  float average = 50.6; // Declaration and initialization |
| **4. Discuss the scope and lifetime of variables in C programming. What are global and local variables?** |
| A: Scope is the variable region in which it can be used. Beyond that area, you cannot use a variable. The local and global are two scopes for C variables. The local scope is limited to the code or function in which the variable is declared. Global scope is the entire program.  Lifetime is the time for which a variable can hold its memory. The lifetime of a variable is static and automatic. The static lifetime variable remains active till the end of the program. An automatic lifetime variable or global variable activates when they are called else they vanish when the function executes.  Global Variables: Declared outside of any function, accessible globally.  Local Variables: Declared within a function or block, accessible only within that function or block. |
| **5. Explain the concept of type casting in C programming. When is type casting necessary, and how is it performed?** |
| A: Type casting in C programming involves converting a value from one data type to another. It is necessary when performing operations where operands have different data types.  Type casting can be performed explicitly by placing the desired data type in parentheses before the value to be cast.  Example:  float average;  int total = 100;  int count = 5;  average = (float)total / count;  Type casting can also happen implicitly during certain operations, such as when assigning a value of one data type to a variable of another data type, but this can lead to loss of precision or truncation. |

|  |
| --- |
| **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 is a concise way to write conditional expressions in C programming. It's often used as a shorthand for simple if-else statements.  Syntax: condition , expression1 : expression2  If the condition evaluates to true, expression1 is executed; otherwise, expression2 is executed.  Example:  int x = 100;  int y = (x > 50) ? 1000 : 2000;  printf("%d\n", y); |
| **2.Discuss the bitwise operators available in C programming. Explain their usage with suitable examples.** |
| A: Bitwise operators manipulate individual bits of operands.  AND (&): Performs a bitwise AND operation.  OR (|): Performs a bitwise OR operation.  XOR (^): Performs a bitwise XOR (exclusive OR) operation.  NOT (~): Performs a bitwise NOT operation (complement).  Left Shift (<<): Shifts the bits of the left operand to the left by the number of positions specified by the right operand.  Right Shift (>>): Shifts the bits of the left operand to the right by the number of positions specified by the right operand.  Example:  unsigned int a = 5; // 0000 0101  unsigned int b = 3; // 0000 0011  unsigned int result;  result = a & b; // result is 0000 0001 (1 in decimal) |
| **3. Explain the difference between the postfix and prefix increment operators (++) in C programming. Provide examples to illustrate** |
| A: Postfix Increment (x++): Increments the value of x after its current value has been used.  Prefix Increment (++x): Increments the value of x before its value is used in an expression.  Example:  int x = 5;  int y, z;  y = x++; // y = 5, x = 6 (x is incremented after assigning its value to y)  z = ++x; // z = 7, x = 7 (x is incremented before assigning its value to z) |
| **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 (&&): Returns true if both operands are true.  Logical OR (||): Returns true if at least one of the operands is true.  They are commonly used in conditional expressions to control the flow of execution based on multiple conditions.  Example:  int x = 5;  int y = 10;  if (x > 0 && y < 15) |
| **5. Discuss the concept of operator precedence and associativity in C programming. Provide examples to demonstrate how they affect expression evaluation.** |
| A: Precedence: Determines the order of evaluation of operators in an expression. Operators with higher precedence are evaluated before those with lower precedence.  Associativity: Defines the order in which operators of the same precedence are evaluated (either left-to-right or right-to-left).  Example:  int result = 2 + 3 \* 4; |

|  |
| --- |
| **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 in C programming is used to execute one code block from multiple alternatives based on the value of an expression.  It provides a more efficient way to handle multiple conditions compared to multiple if-else statements.  The expression is evaluated once, and control jumps to the matching case label. If no match is found, control jumps to the default label (if provided).  Unlike if-else statements, the switch statement evaluates only one expression and then compares it with multiple values.  Example:  int choice = 2;  switch(choice) {  case 1:  printf("Option 1 selected");  break;  case 2:  printf("Option 2 selected");  break;  default:  printf("Invalid choice");  } |
| **2. Explain the concept of nested control structures in C programming. Provide an example demonstrating nested if-else statements.** |
| A: Nested control structures in C programming refer to the situation where one control structure is nested inside another control structure.  This allows for more complex decision-making within a program.  Example of nested if-else statements:  int x = 10;  if (x > 0) {  if (x % 2 == 0) {  printf("Positive and even");  }  else {  printf("Positive and odd");  }  }  else {  printf("Negative");  } |
| **3. Discuss the role of the break and continue statements in loop control in C programming. Provide examples to illustrate their usage.** |
| A: Break: Used to exit the loop immediately, bypassing any remaining iterations.  for (int i = 0; i < 10; i++) {  if (i == 5) {  break; // Exit the loop when i is 5  }  printf("%d ", i);  }  Continue: Skips the remaining code inside the loop for the current iteration and moves to the next iteration.  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.** |
| A: Initialization, Condition, and Increment: The for loop combines the initialization, condition checking, and increment/decrement of the loop variable into a single line, making it more concise and readable.  Example of for loop  for (int i = 0; i < 5; i++) {  printf("%d ", i);  }  int i = 0;  while (i < 5) {  printf("%d ", i);  i++;  }  Scope of Loop Variable: The loop variable in a for loop is typically local to the loop, which prevents accidental modification outside the loop. |
| **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 is a feature of logical expressions where the second operand of logical AND (&&) and logical OR (||) is evaluated only if the result of the expression can be determined by evaluating the first operand.  If the result of the expression can be determined by evaluating the first operand:  For logical AND (&&), if the first operand evaluates to false, the overall expression is false, so the second operand is not evaluated.  For logical OR (||), if the first operand evaluates to true, the overall expression is true, so the second operand is not evaluated.  Example:  int x = 5, y = 10;  if (x > 0 && y > 5) {  // Both conditions are evaluated only if x > 0 is true  } |

|  |
| --- |
| **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 in C programming is a declaration of a function that specifies its name, return type, and parameters (if any), but does not include the function body.  Syntax: return\_type function\_name(parameters);  Function prototypes are necessary because C requires functions to be declared before they are used. This allows the compiler to verify that the function is called correctly with the appropriate arguments and return type.  Example:  int add(int, int);  int add(int a, int b) {  return a + b;  } |
| **2. Explain the difference between call by value and call by reference in C programming. Provide examples to illustrate both concepts.** |
| A: Call by Value: In call by value, the value of the actual parameter is copied into the formal parameter of the function. Changes made to the formal parameter inside the function do not affect the actual parameter.  void changeValue(int x) {  x = 10;  }  int main() {  int value = 5;  changeValue(value);  return 0;  }  Call by Reference: In call by reference, the address of the actual parameter is passed to the formal parameter, allowing the function to modify the value at that address directly.  void changeValue(int \*x) {  \*x = 10;  }  int main() {  int value = 5;  changeValue(&value);  // 'value' is now 10 after the function call  return 0;  } |
| **3. Discuss the concept of recursion in C programming. Provide an example of a recursive function and explain how it works.** |
| A: Recursion in C programming is the process of a function calling itself directly or indirectly.  Example of a recursive function to calculate factorial:  int factorial(int n) {  if (n == 0 || n == 1)  return 1;  else  return n \* factorial(n - 1);  } |
| **4. What is the significance of the return statement in C programming? How are values returned from functions?** |
| A: The return statement in C programming is used to exit a function and return a value (if the function has a return type other than void).  It also passes a value back to the caller of the function.  Example:  int add(int a, int b) {  return a + b;  } |
| **5. Describe the role of function parameters and arguments in C programming. How are function arguments passed to parameters?** |
| Function parameters are variables declared in the function definition that receive values when the function is called.  Function arguments are the actual values passed to the function when it is called.  Arguments are passed to parameters either by value (copy of the value is passed) or by reference (address of the value is passed).  Example:  void printValues(int x, int y) {  printf("x: %d, y: %d\n", x, y);  }  int main() {  int a = 5, b = 10;  printValues(a, b);  return 0;  } |

|  |
| --- |
| **Arrays:** |
| **1. Explain the concept of arrays in C programming. How are arrays declared and initialized?** |
| A: Arrays in C are a collection of elements of the same data type stored in contiguous memory locations. They provide a way to store multiple values of the same type under a single variable name.  Array declaration and initialization:  Declaration: type array\_name[size];  int numbers[5];  Initialization: type array\_name[size] = {value1, value2, ...};  int scores[3] = {90, 85, 88}; |
| **2. Discuss the difference between a one-dimensional array and a multi-dimensional array in C programming. Provide examples of both.** |
| A: One-dimensional array: A simple array that stores elements in a linear sequence.  int arr[5];  Multi-dimensional array: An array of arrays, where each element can be accessed using multiple indices.  int matrix[3][3]; |
| **3. Describe the process of accessing array elements in C programming. How are array indices used to access elements?** |
| A: Array elements are accessed using indices. The index of the first element in C arrays is 0.  Individual elements are accessed using square brackets [] with the index inside.  Example:  int arr[5] = {10, 20, 30, 40, 50};  int element = arr[2]; |
| **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: In C programming, strings are represented as arrays of characters terminated by a null character '\0'.  The null character signifies the end of the string, indicating where the string's contents end.  It is used by string functions to determine the length of the string.  Example:  char str[6] = "Hello"; // 'str' contains "Hello\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 in C allows the creation of arrays whose size is determined at runtime.  Functions like malloc(), calloc(), and realloc() are used to allocate memory dynamically.  Memory allocated dynamically must be deallocated using the free() function to avoid memory leaks.  Example:  int \*arr;  arr = (int \*)malloc(5 \* sizeof(int));  if (arr == NULL) {  exit(1);  }  free(arr); |

|  |
| --- |
| **Pointers:** |
| **1. Describe the purpose and usage of pointers in C programming. How are pointers declared and initialized?** |
| A: Pointers in C programming are variables that store memory addresses, enabling direct manipulation of memory.  They are widely used for dynamic memory allocation, accessing array elements, and building complex data structures like linked lists and trees.  Pointers are declared by specifying the data type they point to, followed by an asterisk (\*).  Example of pointer declaration and initialization:  int \*ptr; // Declaration of a pointer to an integer  int num = 10;  ptr = &num; // Initialization: ptr points to the address of 'num' |
| **2. Explain the concept of pointer arithmetic in C programming. Provide examples to illustrate addition and subtraction operations on pointers.** |
| A: Pointer arithmetic in C allows performing arithmetic operations on pointers.  Addition and subtraction operations move the pointer by a certain number of elements based on the pointer's data type size.  Example:  int arr[5] = {1, 2, 3, 4, 5};  int \*ptr = arr;  ptr++;  ptr--; |
| **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: In this approach, a copy of the argument's value is passed to the function. Changes made to the parameter inside the function do not affect the original value.  Pass by Reference: Pointers are used as function parameters, allowing the function to modify the original value through its memory address.  Example:  void increment(int x) {  x++;  }  void incrementByRef(int \*x) {  (\*x)++; // Changes made to '\*x' affect the original value  } |
| **4. Describe the concept of NULL pointers in C programming. How are NULL pointers used and checked for in programs?** |
| A: NULL pointers in C are pointers that do not point to any memory location.  They are often used to indicate an invalid or uninitialized pointer.  They are checked using an equality comparison with the NULL macro (NULL), defined in <stdlib.h>.  Example:  int \*ptr = NULL;  if (ptr == NULL) {  printf("Pointer is NULL\n");  } |
| **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 play a crucial role in dynamic memory allocation, allowing programs to request memory from the heap at runtime.  Functions like malloc(), calloc(), and realloc() return pointers to the allocated memory.  Memory allocated dynamically should be deallocated using the free() function to prevent memory leaks.  Example:  int \*arr;  arr = (int \*)malloc(5 \* sizeof(int)); // Allocate memory for an array of 5 integers  if (arr == NULL) {  exit(1);  }  free(arr); |

|  |
| --- |
| **Strings:** |
| **1. Discuss the concept of strings in C programming. How are strings represented and manipulated in C?** |
| A: Representation and Manipulation:  In C programming, a string is represented as an array of characters terminated by a null character \0. Strings can be manipulated using  various functions provided by the standard library or by manually iterating through the characters of the array. |
| **2. Explain the difference between character arrays and string literals in C programming. Provide examples to illustrate both concepts.** |
| A: Character Arrays: These are arrays of characters declared by the user to store strings. They can be modified because they are mutable.  char str[20];  String Literals: These are constant arrays of characters implicitly created by enclosing text within double quotes. They cannot be modified because they are stored in read-only memory.  char \*str = "Hello"; |
| **3. Describe common string manipulation functions available in the C standard library. Provide examples of functions like strlen, strcpy, strcat, and strcmp.** |
| A: strlen: Returns the length of a string.  #include <string.h>  size\_t strlen(const char \*str);  strcpy: Copies one string to another.  #include <string.h>  char \*strcpy(char \*dest, const char \*src);  strcat: Concatenates two strings.  #include <string.h>  char \*strcat(char \*dest, const char \*src);  strcmp: Compares two strings lexicographically.  #include <string.h>  int strcmp(const char \*str1, const char \*str2); |

|  |
| --- |
| **4. Discuss the concept of string tokenization in C programming. How are strings split into tokens using delimiter characters?** |
| A: String tokenization is the process of splitting a string into smaller tokens based on a delimiter character or a set of delimiter characters. In C, the strtok() function is commonly used for this purpose.  #include <string.h>  #include <stdio.h>  int main() {  char str[] = "apple,banana,cherry";  char \*token = strtok(str, ",");  while (token != NULL) {  printf("%s\n", token);  token = strtok(NULL, ",");  }  return 0;  } |
| **5. Explain the importance of null-terminated strings in C programming. How does the null character ('\0') signify the end of a string?** |
| A: In C programming, strings are null-terminated, meaning the null character \0 signifies the end of a string. This allows functions to determine the length of a string and prevents buffer overflow errors by providing a clear endpoint.  char str[] = "Hello";  The null character ensures that string manipulation functions know where the string ends. Without it, functions like strlen() wouldn't know when to stop counting characters. |

|  |
| --- |
| **Structures and Unions:** |
| **1. Describe the purpose and usage of structures in C programming. How are structures declared and accessed?** |
| A: Purpose and Usage: Structures in C are used to group together different data types under a single name. They allow you to create complex data types representing real-world entities. Structures facilitate organizing related data items into a single unit, making code more readable and maintainable.  Declaration and Access: Structures are declared using the struct keyword followed by a tag name and a list of members enclosed in curly braces. To access members of a structure, you use the dot (.) operator.  struct Person {  char name[50];  int age;  float salary;  };  struct Person person1;  person1.age = 25; |
| **2. Discuss the concept of structure members in C programming. How are individual members of a structure accessed and modified?** |
| A: Structure members are the individual variables contained within a structure. They are accessed using the dot (.) operator followed by the member name.  struct Person {  char name[50];  int age;  float salary;  };  struct Person person1;  strcpy(person1.name, "John");  person1.age = 25;  person1.salary = 50000.0;  Unions:  Purpose and Usage: Unions, like structures, allow you to group together different data types. However, unlike structures, unions allocate memory that is large enough to hold the largest member. This means that only one member of a union can be used at a time, making unions suitable for cases where you need to represent a single value from a set of possible types.  Declaration and Usage: Unions are declared similarly to structures but use the union keyword. Accessing members of a union is done in the same way as structures.  union Data {  int i;  float f;  char str[20];  };  union Data data;  data.i = 10; |
| **3. Explain the difference between structures and unions in C programming. When would you choose one over the other?** |
| A: You choose structures when you want to group together multiple variables that are relevant simultaneously. Unions are chosen when you want to save memory and are dealing with mutually exclusive data types, and you only need to access one member at a time. |
| **4. Describe the concept of nested structures in C programming. How are structures within structures defined and accessed?** |
| A: Definition and Access: Nested structures are structures defined within another structure. They are accessed using the dot (.) operator multiple times, starting from the outer structure to the inner one.  struct Address {  char city[50];  char state[50];  };  struct Employee {  char name[50];  int empID;  struct Address address;  };  struct Employee emp1;  strcpy(emp1.address.city, "New York"); |
| **5. Discuss the concept of typedef in C programming. How is typedef used to define custom data types, including structures and unions?** |
| A: Purpose and Usage: typedef in C is used to create aliases or custom names for existing data types, including structures and unions. It helps in improving code readability and abstraction by creating custom data type names.  typedef struct {  int day;  int month;  int year;  } Date;  Date today;  today.day = 16;  today.month = 3;  today.year = 2024;  Defining Custom Data Types: typedef can be used with structures and unions to create custom data types without needing to use the struct or union keywords every time you declare a variable of that type.  typedef struct {  char name[50];  int age;  } Person;  Person person1;  typedef union {  int i;  float f;  char c;  } MyUnion;  MyUnion data; |

|  |
| --- |
| **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: Concept:  File handling in C programming involves operations such as opening, reading from, writing to, and closing files. It allows programs to interact with files stored on the computer's file system. This functionality is provided by the standard input/output library (stdio.h) in C.  Opening, Reading, and Writing Files:  Opening Files: Files are opened using the fopen() function, which takes the file name and mode as arguments. Modes include "r" for reading, "w" for writing (creating a new file or overwriting an existing one), and "a" for appending data to an existing file.  FILE \*filePointer;  filePointer = fopen("filename.txt", "r");  Reading from Files: Data is read from files using functions like fscanf() or fgets().  char buffer[100];  fgets(buffer, 100, filePointer);  Writing to Files: Data is written to files using functions like fprintf() or fputs().  fprintf(filePointer, "Hello, world!\n"); |
| **2. Describe the role of file pointers in C programming. How are file pointers used to navigate and manipulate files?** |
| A: Role: File pointers are used to keep track of the current position within a file. They are essential for navigation and manipulation of files. Functions like fseek() and ftell() are used to move the file pointer to a specific position or retrieve its current position, respectively.  fseek(filePointer, 0, SEEK\_SET); // Move file pointer to the beginning of the file |
| **3. Discuss the difference between text files and binary files in C programming. How are they opened and processed differently?** |
| A: Text Files: Text files store data in a human-readable format, with each character represented by its ASCII or Unicode value. They are opened using modes like "r" or "w" and are processed using standard input/output functions.  Binary Files: Binary files store data in a format that is not human-readable, typically consisting of raw binary data. They are opened using modes like "rb" or "wb" to indicate binary mode, and special functions like fread() and fwrite() are used for reading and writing binary data. |
| **4. Explain the purpose of file modes in C programming. Provide examples of different file modes like "r", "w", "a", etc.** |
| A: Purpose: File modes specify the intended operation on a file. Common file modes include:  "r": Opens a file for reading.  "w": Opens a file for writing. If the file already exists, its contents are overwritten. If it doesn't exist, a new file is created.  "a": Opens a file for appending. Data is written to the end of the file.  "rb", "wb", "ab": Binary file modes for reading, writing, and appending. |
| **5. Describe error handling techniques in file operations in C programming. How are errors detected and handled when working with files?** |
| A: Detection: Errors in file operations can be detected by checking the return value of file handling functions. For example, fopen() returns NULL if it fails to open the file.  Handling: Error handling in file operations involves appropriate actions based on the error encountered. This may include displaying an error message, closing the file, or terminating the program.  SFILE \*filePointer;  filePointer = fopen("filename.txt", "r");  if (filePointer == NULL) {  printf("Error opening file.\n");  exit(1);  } |

**Part- B**

|  |
| --- |
| 1. **Hello world** |
| #include <stdio.h>  void readFromFile(const char\* filename)  {  FILE\* file = fopen(filename, "r");  if (file == NULL)  {  printf("Error opening file: %s\n", filename);  return;  }  char buffer[100]; // Assuming maximum line length is 100 characters  while (fgets(buffer, sizeof(buffer), file) != NULL) {  printf("\n results are in written to %s\n",filename);  }  fclose(file);  }  void writeToFile(const char\* filename, const char\* content)  {  FILE\* file = fopen(filename, "w");  if (file == NULL) {  printf("Error opening file: %s\n", filename);  return;  }  fprintf(file, "%s\n", content);  fclose(file);  }  int main() {  const char message[] = "HELLO WORLD"; // Use const char\* for string literals  writeToFile("output.txt", message);  readFromFile("output.txt");  return 0;  }    **2.Factorial**  #include<stdio.h>  // Function prototypes  int factorial(int);  int readfromfile(char\*, int\*);  void writetofile(char\*, int, int);  int main() {  int n, result;  // Read an integer from the input file  readfromfile("input.txt", &n);  // Calculate the factorial of the input number  result = factorial(n);  // Write the result to the output file  writetofile("output.txt", n, result);  return 0;  }  // Function to calculate factorial of a number  int factorial(int n) {  int i, fact = 1;  // Loop to calculate factorial  for(i = 1; i <= n; i++) {  fact \*= i;  }  return fact;  }  // Function to read an integer from a file  int readfromfile(char\* filename, int \*n) {  FILE \*file = fopen(filename, "r");  // Check if the file opened successfully  if(file == NULL) {  printf("Error opening input file");  return 0;  }  fscanf(file, "%d", n); // Read an integer from the file  // Close the file  fclose(file);  // Return 1 to indicate success  return 1;  }  // Function to write the result to a file  void writetofile(char\* filename, int n, int result) {  FILE \*file = fopen(filename, "w");  // Check if the file opened successfully  if(file == NULL) {  printf("Error opening output file");  return;  }  // Write the result to the file  fprintf(file, "Factorial of %d = %d\n", n, result);  // Close the file  fclose(file);  // Print a message indicating successful write  printf("\nFactorial is calculated and written to %s\n", filename);  }  **Output**    **3.Prime numbers**  #include<stdio.h>  // Function prototypes  int prime(int);  void readfromfile(char\* filename, int \*n);  void writetofile(char\* filename, char \*result);  int main() {  int n;  // Read an integer from the input file  readfromfile("input.txt", &n);  // Check if the number is prime  prime(n);  return 0;  }  // Function to check if a number is prime  int prime(int n) {  int i, flag = 1;  // Loop to check for factors of n  for(i = 2; i < n; i++) {  if(n % i == 0) {  flag = 0;  break;  }  }  // Declare a character array to store result  char result[50];  // Write the result to the result string  if(flag == 1) {  sprintf(result, "\nThe given number is a prime number");  } else {  sprintf(result, "\nThe given number is not a prime number");  }  // Write the result to the output file  writetofile("output.txt", result);  }  // Function to read an integer from a file  void readfromfile(char\* filename, int \*n) {  FILE \*file = fopen(filename, "r");  // Check if the file opened successfully  if(file == NULL) {  printf("Error opening input file");  return;  }  // Read an integer from the file  fscanf(file, "%d", n);  // Close the file  fclose(file);  }  // Function to write a string to a file  void writetofile(char\* filename, char \*result) {  FILE \*file = fopen(filename, "w");  // Check if the file opened successfully  if(file == NULL) {  printf("Error opening output file %s", filename);  return;  }  // Write the result to the file  fprintf(file, "%s\n", result);  // Close the file  fclose(file);  // Print a message indicating successful write  printf("\nResults are written to %s\n", filename);  }  Output    **4.Fiboncii Series**  #include<stdio.h>  void fiboncii(int);  void readfromfile(char\*,int\*);  void writetofile(char\*,int);  int main()  {  int n;  readfromfile("input.txt",&n);  fiboncii(n);  return 0;  }  void fiboncii(int n)  {  int i,a=0,b=1,next\_term;  FILE \*file=fopen("output.txt","w");  if(file==NULL)  {  printf("Error opening input file");  return;  }  for(i=0;i<n;i++)  {  fprintf(file,"%d ",a);  next\_term=a+b;  a=b;  b=next\_term;  }  fclose(file);  printf("Fibonacci series for is written to output.txt\n");  }  void readfromfile(char\*filename,int \*n)  {  FILE \*file=fopen(filename,"r");  if(file==NULL)  {  printf("Error opening input file");  return;  }  fscanf(file,"%d",n);  fclose(file);  }  void writetofile(char\*filename,int result)  {  FILE \*file=fopen(filename,"w");  if(file==NULL)  {  printf("error opening input file %s",filename);  return;  }  fprintf(file,"d\n",result);  fclose(file);  printf("\n results are in written to %s\n",filename);  }  **Output** |
| **5.Sum of Digits**  #include<stdio.h>  // Function prototypes  int sumofdigits(int);  void readfromfile(char\*, int\*);  void writetofile(char\*, int);  int main() {  int n;  // Read an integer from the file  readfromfile("input.txt", &n);  // Calculate the sum of digits  int sum = sumofdigits(n);  // Write the result to the output file  writetofile("output.txt", sum);  return 0;  }  // Function to calculate the sum of digits of a number  int sumofdigits(int n) {  int sum = 0, ld;  // Loop to extract digits and calculate sum  while(n != 0) {  ld = n % 10; // Extract the last digit  sum += ld; // Add the last digit to the sum  n = n / 10; // Remove the last digit  }  return sum;  }  // Function to read an integer from a file  void readfromfile(char\* filename, int \*n) {  FILE \*file = fopen(filename, "r");  // Check if the file opened successfully  if(file == NULL) {  printf("Error opening input file");  return;  }  // Read an integer from the file  fscanf(file, "%d", n);  // Close the input file  fclose(file);  }  // Function to write an integer to a file  void writetofile(char\* filename, int result) {  FILE \*file = fopen(filename, "w");  // Check if the file opened successfully  if(file == NULL) {  printf("Error opening output file %s", filename);  return;  }  // Write the result to the file  fprintf(file, "Sum of digits = %d\n", result);  // Close the output file  fclose(file);  // Print a message indicating successful write  printf("\nResult is written to %s\n", filename);  }  Output    6.Reverse  #include<stdio.h>  // Function prototypes  int reverse(int);  void readfromfile(char\*, int\*);  void writetofile(char\*, int);  int main() {  int n;  // Read an integer from file  readfromfile("input.txt", &n);  // Calculate the reverse of the number  int reversed = reverse(n);  // Write the reversed number to the output file  writetofile("output.txt", reversed);  return 0;  }  // Function to reverse a given number  int reverse(int n) {  int ld, rev = 0;  // Loop to reverse the number  while(n != 0) {  ld = n % 10; // Extract the last digit  rev = rev \* 10 + ld; // Append the digit to the reversed number  n = n / 10; // Remove the last digit from the original number  }  // Return the reversed number  return rev;  }  // Function to read an integer from a file  void readfromfile(char\* filename, int \*n) {  FILE \*file = fopen(filename, "r");  // Check if the file opened successfully  if(file == NULL) {  printf("Error opening input file");  return;  }  // Read an integer from the file  fscanf(file, "%d", n);  // Close the input file  fclose(file);  }  // Function to write an integer to a file  void writetofile(char\* filename, int result) {  FILE \*file = fopen(filename, "w");  // Check if the file opened successfully  if(file == NULL) {  printf("Error opening output file %s", filename);  return;  }  // Write the reversed number to the file  fprintf(file, "Reverse number = %d\n", result);  // Close the output file  fclose(file);  // Print message indicating successful write  printf("\nResults are written to %s\n", filename);  }  **Output**    7.Area of Shapes  #include<stdio.h>  // Function prototypes  float rectangle(float, float);  float triangle(float, float);  float circle(float);  void readfromfile(char\*, float\*, float\*, float\*, float\*);  void writetofile(char\*, float, float, float, float, float, float, float);  int main() {  // Declare variables for dimensions and areas  float l, b, h, r, tri, rect, cir;  // Read dimensions from file  readfromfile("input.txt", &l, &b, &h, &r);  // Calculate areas for triangle, rectangle, and circle  tri = triangle(b, h);  rect = rectangle(l, b);  cir = circle(r);  // Write results to output file  writetofile("output.txt", l, b, h, r, tri, rect, cir);  return 0;  }  // Function to calculate area of rectangle  float rectangle(float l, float b) {  return l \* b;  }  // Function to calculate area of triangle  float triangle(float b, float h) {  return (b \* h) / 2;  }  // Function to calculate area of circle  float circle(float r) {  return 3.14 \* r \* r;  }  // Function to read dimensions from file  void readfromfile(char\* filename, float\* l, float\* b, float\* h, float\* r) {  FILE\* file = fopen(filename, "r");  if (file == NULL) {  printf("Error opening input file");  return;  }  // Read dimensions from file  int count = fscanf(file, "%f%f%f%f", l, b, h, r);  if (count != 4) {  printf("Error reading input from file: %s\n", filename);  fclose(file);  return;  }  fclose(file);  }  // Function to write results to file  void writetofile(char\* filename, float l, float b, float h, float r, float tri, float rect, float cir) {  FILE\* file = fopen(filename, "w");  if (file == NULL) {  printf("Error opening output file");  return;  }  // Write dimensions and areas to file  fprintf(file, "Length = %f\n", l);  fprintf(file, "Breadth = %f\n", b);  fprintf(file, "Height = %f\n", h);  fprintf(file, "Radius = %f\n", r);  fprintf(file, "Area of Triangle = %f\n", tri);  fprintf(file, "Area of Rectangle = %f\n", rect);  fprintf(file, "Area of Circle = %f\n", cir);  fclose(file);  printf("\nResults are written to %s\n", filename);  }  Output    8.Calculator  #include<stdio.h>  // Function prototypes  void calculator(float, float);  void readfromfile(char\*, float\*, float\*);  void writetofile(char\*, float);  int main() {  // Declare variables for numbers  float num1, num2;  // Read numbers from file  readfromfile("input.txt", &num1, &num2);  // Perform calculations  calculator(num1, num2);  return 0;  }  // Function to perform calculations based on operator  void calculator(float num1, float num2) {  char opp;  float result;  // Prompt user to enter operator  printf("Enter the operator (+, -, \*, /): ");  scanf(" %c", &opp);  // Perform operation based on operator  switch(opp) {  case '+':  result = num1 + num2;  break;  case '-':  result = num1 - num2;  break;  case '\*':  result = num1 \* num2;  break;  case '/':  if(num2 == 0) {  result = -1; // Indicate division by zero  } else {  result = num1 / num2;  }  break;  default:  result = 0; // Default result if operator is invalid  break;  }  // Write result to file  writetofile("output.txt", result);  }  // Function to read numbers from file  void readfromfile(char\* filename, float\* num1, float\* num2) {  FILE\* file = fopen(filename, "r");  if(file == NULL) {  printf("Error opening input file");  return;  }  fscanf(file, "%f%f", num1, num2); // Read numbers from file  fclose(file);  }  // Function to write result to file  void writetofile(char\* filename, float result) {  FILE\* file = fopen(filename, "w");  if(file == NULL) {  printf("Error opening output file %s", filename);  return;  }  fprintf(file, "%.2f\n", result); // Write result to file  fclose(file);  printf("\nResults are written to %s\n", filename);  }  Output    9.Array Operations  #include<stdio.h>  // Function prototypes  void readfromfile(char\*, int[], int\*);  void writetofile(char\*, int, int, int, float);  void readarray(int[], int, FILE\*);  void printarray(int[], int, FILE\*);  int add(int[], int);  int largest(int[], int);  int smallest(int[], int);  float avg(int[], int);  int main() {  int n, addition, maximum, minimum;  float average;  int nu[100];  // Read data from file  readfromfile("input.txt", nu, &n);  // Calculate sum, largest, smallest, and average  addition = add(nu, n);  maximum = largest(nu, n);  minimum = smallest(nu, n);  average = avg(nu, n);  // Write results to output file  writetofile("output.txt", addition, maximum, minimum, average);  return 0;  }  // Function to read data from file  void readfromfile(char\* filename, int nu[], int \*n) {  FILE \*file = fopen(filename, "r");  if (file == NULL) {  printf("Error opening input file");  return;  }  fscanf(file, "%d", n); // Read the number of elements  for (int i = 0; i < \*n; i++) {  fscanf(file, "%d", &nu[i]); // Read the elements  }  fclose(file);  }  // Function to write results to file  void writetofile(char\* filename, int addition, int maximum, int minimum, float average) {  FILE \*file = fopen(filename, "w");  if (file == NULL) {  printf("Error opening output file");  return;  }  // Write the results to the file  fprintf(file, "Sum of the numbers = %d\n", addition);  fprintf(file, "Largest number = %d\n", maximum);  fprintf(file, "Smallest number = %d\n", minimum);  fprintf(file, "Average of the numbers = %.2f\n", average);  fclose(file);  }  // Function to read array from file  void readarray(int nu[], int n, FILE \*file) {  for (int i = 0; i < n; i++) {  fscanf(file, "%d", &nu[i]);  }  }  // Function to print array values  void printarray(int nu[], int n, FILE \*file) {  for (int i = 0; i < n; i++) {  fprintf(file, "%d ", nu[i]);  }  }  // Function to calculate sum of elements  int add(int nu[], int n) {  int sum = 0;  for (int i = 0; i < n; i++) {  sum += nu[i];  }  return sum;  }  // Function to find the largest element  int largest(int nu[], int n) {  int max = nu[0];  for (int i = 1; i < n; i++) {  if (nu[i] > max) {  max = nu[i];  }  }  return max;  }  // Function to find the smallest element  int smallest(int nu[], int n) {  int min = nu[0];  for (int i = 1; i < n; i++) {  if (nu[i] < min) {  min = nu[i];  }  }  return min;  }  // Function to calculate the average  float avg(int nu[], int n) {  float sum = 0;  for (int i = 0; i < n; i++) {  sum += nu[i];  }  return sum / n;  }  Output    10.Linear search  #include<stdio.h>  int linearsearch(int n,int nu[]);  int readfromfile(char\*,int[]);  void writetofile(char\*,int);  void readarray(int[],int n,FILE\*);  void printarray(int[],int n,FILE\*);  int main()  {  int n,search;  n=readfromfile("input.txt",NULL);  if(n==0)  {  printf("Unable to read data from file\n");  return 1;  }  int nu[n];  if(readfromfile("input.txt",nu)!=n)  {  printf("Unable to read data from file\n");  return 1;  }  search=linearsearch(n,nu);  writetofile("output.txt",search);  return 0;  }  void readarray(int nu[],int n,FILE \*file)  {  int i;  printf("enter the array value:");  for(i=0;i<n;i++)  {  fscanf(file,"%d",&nu[i]);  }  }  void printarray(int nu[],int n,FILE \*file)  {  int i;  printf("\narray value are:");  for(i=0;i<n;i++)  {  fprintf(file,"\n%d",nu[i]);  }  }  int linearsearch(int n,int nu[])  {  int i,key;  printf("\nenter the element to be search");  scanf("%d",&key);  for(i=0;i<n;i++)  {  if(nu[i]==key)  {  return i;  }  }  return -1;  }  void writetofile(char\*filename,int search)  {  FILE \*file=fopen(filename,"w");  if(file==NULL)  {  fprintf(file,"key element is found");  return;  }  if(search!=-1)  {  fprintf(file,"key element is found");  }  else  {  fprintf(file,"key element not found");  }  fclose(file);  printf("\nsearch result is written to %s\n",filename);  }  int readfromfile(char\*filename,int nu[])  {  FILE \*file=fopen(filename,"r");  if(file==NULL)  {  printf("Error opening input file");  return 0;  }  int n;  fscanf(file,"%d",&n);  if(nu==NULL)  {  fclose(file);  return n;  }  int i=0;  while(fscanf(file,"%d",&nu[i])!=EOF)  {  i++;  }  fclose(file);  return i;  }    Output    11.Bubblesort  #include<stdio.h>  // Function prototypes  int readfromfile(char\*, int[]);  void writetofile(char\*, int[], int);  void readarray(int[], int n, FILE\*);  void printarray(int[], int n, FILE\*);  int bubblesort(int[], int);  int main() {  int n, sort;  // Read the number of elements from the file  n = readfromfile("input.txt", NULL);  // Check if reading from file failed  if (n == 0) {  printf("Unable to read data from file\n");  return 1;  }  // Declare an array to store the elements  int nu[n];  // Read the elements from the file into the array  if (readfromfile("input.txt", nu) != n) {  printf("Unable to read data from file\n");  return 1;  }  // Perform bubble sort  sort = bubblesort(nu, n);  // Write sorted array to output file  writetofile("output.txt", nu, n);  return 0;  }  // Function to read array from file  int readfromfile(char\* filename, int nu[]) {  FILE\* file = fopen(filename, "r");  if (file == NULL) {  printf("Error opening input file");  return 0;  }  // Read the number of elements from file  int n;  fscanf(file, "%d", &n);  // If nu is NULL, just return the number of elements  if (nu == NULL) {  fclose(file);  return n;  }  // Read the elements into the array  for (int i = 0; i < n; i++) {  fscanf(file, "%d", &nu[i]);  }  fclose(file);  return n; // Return the number of elements  }  // Function to write array to file  void writetofile(char\* filename, int nu[], int n) {  FILE\* file = fopen(filename, "w");  if (file == NULL) {  printf("Error opening input file");  return;  }  // Write the number of elements followed by the elements themselves  for (int i = 0; i < n; i++) {  fprintf(file, "%d\n", nu[i]);  }  printf("\nBubble sort result is written to %s\n", filename);  fclose(file);  }  // Function to perform bubble sort  int bubblesort(int nu[], int n) {  int i, j, isSorted = 1;  // Outer loop for passes  for (i = 0; i < n - 1; i++) {  // Inner loop for comparisons and swapping  for (j = 0; j < n - 1 - i; j++) {  if (nu[j] > nu[j + 1]) {  // Swap if current element is greater than the next one  int temp = nu[j];  nu[j] = nu[j + 1];  nu[j + 1] = temp;  isSorted = 0; // Set flag to indicate array is not sorted  }  }  // If the array is already sorted, break the loop  if (isSorted) {  break;  }  }  }  Output    12.Selectionsort  #include<stdio.h>  // Function prototypes  int readfromfile(char\*, int[]);  void writetofile(char\*, int[], int);  void readarray(int[], int n, FILE\*);  void printarray(int[], int n, FILE\*);  int selectionsort(int[], int);  int main() {  int n, sort;  // Read the number of elements from the input file  n = readfromfile("input.txt", NULL);  // Check if reading from file failed  if (n == 0) {  printf("Unable to read data from file\n");  return 1;  }  // Declare an array to store the elements  int nu[n];  // Read the elements from the file into the array  if (readfromfile("input.txt", nu) != n) {  printf("Unable to read data from file\n");  return 1;  }  // Perform selection sort  sort = selectionsort(nu, n);  // Write sorted array to output file  writetofile("output.txt", nu, n);  return 0;  }  // Function to read array from file  int readfromfile(char\* filename, int nu[]) {  FILE\* file = fopen(filename, "r");  if (file == NULL) {  printf("Error opening input file");  return 0;  }  int n;  fscanf(file, "%d", &n); // Read the number of elements  if (nu == NULL) {  fclose(file);  return n;  }  // Read the elements into the array  for (int i = 0; i < n; i++) {  fscanf(file, "%d", &nu[i]);  }  fclose(file);  return n; // Return the number of elements read  }  // Function to perform selection sort  int selectionsort(int nu[], int n) {  int i, j, indexofmin;  for (i = 0; i < n; i++) {  indexofmin = i;  for (j = i + 1; j < n; j++) {  if (nu[j] < nu[indexofmin]) {  indexofmin = j;  }  }  // Swap the elements  int temp = nu[i];  nu[i] = nu[indexofmin];  nu[indexofmin] = temp;  }  }  // Function to write array to file  void writetofile(char\* filename, int nu[], int n) {  FILE\* file = fopen(filename, "w");  if (file == NULL) {  printf("Error opening input file");  return;  }  // Write the elements to the file  for (int i = 0; i < n; i++) {  fprintf(file, "%d\n", nu[i]);  }  printf("\nSelection sort result is written to %s\n", filename);  fclose(file);  }  Output    13.Binary search  #include<stdio.h>  // Function prototypes  int binarysearch(int n, int nu[]);  int readfromfile(char\*, int[]);  void writetofile(char\*, int);  void readarray(int[], int n, FILE\*);  void printarray(int[], int n, FILE\*);  int main() {  int n, search;  // Read the number of elements from the input file  n = readfromfile("input.txt", NULL);  // Check if reading from file failed  if (n == 0) {  printf("Unable to read data from file\n");  return 1;  }  // Declare an array to store the elements  int nu[n];  // Read the elements from the file into the array  if (readfromfile("input.txt", nu) != n) {  printf("Unable to read data from file\n");  return 1;  }  // Perform binary search  search = binarysearch(n, nu);  // Write search result to output file  writetofile("output.txt", search);  return 0;  }  // Function to read array from file  int readfromfile(char\* filename, int nu[]) {  FILE\* file = fopen(filename, "r");  if (file == NULL) {  printf("Error opening input file");  return 0;  }  int n;  fscanf(file, "%d", &n); // Read the number of elements  if (nu == NULL) {  fclose(file);  return n;  }  int i = 0;  while (fscanf(file, "%d", &nu[i]) != EOF) { // Read elements until EOF  i++;  }  fclose(file);  return i; // Return the number of elements read  }  // Function to perform binary search  int binarysearch(int n, int nu[]) {  int low = 0, high = n - 1, mid, key;  printf("\nEnter the element to be searched: ");  scanf("%d", &key);  while (low <= high) {  mid = (low + high) / 2;  if (nu[mid] == key) {  return mid; // Return index if key is found  } else if (nu[mid] > key) {  high = mid - 1;  } else {  low = mid + 1;  }  }  return -1; // Return -1 if key is not found  }  // Function to write search result to file  void writetofile(char\* filename, int search) {  FILE\* file = fopen(filename, "w");  if (file == NULL) {  printf("Error opening input file");  return;  }  if (search != -1) {  fprintf(file, "Key element is found at index: %d\n", search); // Write if key is found  } else {  fprintf(file, "Key element not found\n"); // Write if key is not found  }  fclose(file);  printf("\nSearch result is written to %s\n", filename);  }  Output    14.Insertion sort  #include<stdio.h>  // Function prototypes  int readfromfile(char\*,int[]);  void writetofile(char\*,int[],int);  void readarray(int[],int n,FILE\*);  void printarray(int[],int n,FILE\*);  int insertionsort(int[],int);  int main()  {  int n,sort;  // Read the number of elements from the file  n=readfromfile("input.txt",NULL); // Check if reading from file failed  if(n==0)  {  printf("Unable to read data from file\n");  return 1;  }  int nu[n]; // Declare an array to store the elements  if(readfromfile("input.txt",nu)!=n) //// Read the elements from the file into the array  {  printf("Unable to read data from file\n");  return 1;  }  sort=insertionsort(nu,n); // Perform insertion sort  writetofile("output.txt",nu,n); // Write sorted array to output file  return 0;  }  // Function to read array from standard input  void readarray(int nu[],int n,FILE \*file)  {  int i;  printf("enter the array value:");  for(i=0;i<n;i++)  {  fscanf(file,"%d",&nu[i]);  }  }  // Function to print array to standard input  void printarray(int nu[],int n,FILE \*file)  {  int i;  printf("\narray value are:");  for(i=0;i<n;i++)  {  fprintf(file,"\n%d",nu[i]);  }  }  int insertionsort(int nu[],int n)  {  int i,j,key;  for(i=0;i<n;i++)  {  key=nu[i];  j=i-1;  while(j>=0 && nu[j]>key)  {  nu[j+1]=nu[j];  j=j-1;  }  nu[j+1]=key;  }  }  void writetofile(char\*filename,int nu[],int n) // Function to write array to file  {  FILE \*file=fopen(filename,"w");  if(file==NULL)  {  printf("Error opening input file");  return;  }  // Write the number of elements followed by the elements themselves  for(int i=0;i<n;i++)  {  fprintf(file,"%d\n",nu[i]);  }  printf("\ninsertion sort result is written to %s\n",filename);  }  int readfromfile(char\*filename,int nu[])  {  FILE \*file=fopen(filename,"r");  if(file==NULL)  {  printf("Error opening input file");  return 0;  }  int n;  fscanf(file,"%d",&n);  if(nu==NULL) // If nu is NULL, just return the number of elements  {  fclose(file);  return n;  }  int i=0;  for(int i=0;i<n;i++)  {  fscanf(file,"%d",&nu[i]);  }  fclose(file);  return n;  }  Output    15.Palidrome Check  #include<stdio.h>  // Function prototypes  int palindrome(int);  void readfromfile(char\*, int\*);  void writetofile(char\*, char\*);  int main() {  int n;  // Read an integer from file  readfromfile("input.txt", &n);  // Check if the number is a palindrome  palindrome(n);  return 0;  }  // Function to check if a number is palindrome  int palindrome(int n) {  int ld, i, rev = 0, palin = n;  // Reverse the number  while(n != 0) {  ld = n % 10;  rev = rev \* 10 + ld;  n = n / 10;  }  char result[50];  // Check if the original number is equal to its reverse  if(rev == palin) {  sprintf(result, "\n%d is a palindrome number", palin);  } else {  sprintf(result, "\n%d is not a palindrome number", palin);  }  // Write the result to file  writetofile("output.txt", result);  }  // Function to read an integer from file  void readfromfile(char\* filename, int \*n) {  FILE \*file = fopen(filename, "r");  // Check if file opened successfully  if(file == NULL) {  printf("Error opening input file");  return;  }  // Read integer from file  fscanf(file, "%d", n);  // Close the input file  fclose(file);  }  // Function to write a string to file  void writetofile(char\* filename, char \*result) {  FILE \*file = fopen(filename, "w");  // Check if file opened successfully  if(file == NULL) {  printf("Error opening output file %s", filename);  return;  }  // Write string to file  fprintf(file, "%s\n", result);  // Close the output file  fclose(file);  // Print message indicating successful write  printf("\nResults are written to %s\n", filename);  }  Output    16. Addition of matrix  #include <stdio.h>  #define N 4  void add(int A[][N], int B[][N], int C[][N]);  void readMatrixFromFile(const char \*filename, int matrix[][N]);  void writeMatrixToFile(const char \*filename, int matrix[][N]);  int main()  {  int A[N][N], B[N][N], C[N][N]; // Matrices A, B, and C  // Read matrices A and B from file  readMatrixFromFile("input.txt", A);  readMatrixFromFile("input2.txt", B);  // Calculate sum of matrices A and B  add(A, B, C);  // Write the result matrix C to a file  writeMatrixToFile("output.txt", C);  printf("Result matrix has been written to output.txt'\n");  return 0;  }  void add(int A[][N], int B[][N], int C[][N])  {  for (int i = 0; i < N; i++)  for (int j = 0; j < N; j++)  C[i][j] = A[i][j] + B[i][j];  }  void readMatrixFromFile(const char \*filename, int matrix[][N])  {  FILE \*file = fopen(filename, "r");  if (file == NULL)  {  printf("Error opening file '%s' for reading.\n", filename);  return;  }  for (int i = 0; i < N; i++)  for (int j = 0; j < N; j++)  fscanf(file, "%d", &matrix[i][j]);  fclose(file);  }  void writeMatrixToFile(const char \*filename, int matrix[][N])  {  FILE \*file = fopen(filename, "w");  if (file == NULL)  {  printf("Error opening file '%s' for writing.\n", filename);  return;  }  for (int i = 0; i < N; i++)  {  for (int j = 0; j < N; j++)  fprintf(file, "%d ", matrix[i][j]);  fprintf(file, "\n");  }  fclose(file);  }  OUTPUT  C:\Users\Lenovo\AppData\Local\Packages\Microsoft.Windows.Photos_8wekyb3d8bbwe\TempState\ShareServiceTempFolder\Screenshot (41).jpeg  C:\Users\Lenovo\Desktop\8356c081-abf9-4465-a5ce-86cb969c3cf6.jpgC:\Users\Lenovo\AppData\Local\Packages\Microsoft.Windows.Photos_8wekyb3d8bbwe\TempState\ShareServiceTempFolder\Screenshot (43).jpeg  17. Subtraction of matrix  #include <stdio.h>  #define N 4  void add(int A[][N], int B[][N], int C[][N]);  void readMatrixFromFile(const char \*filename, int matrix[][N]);  void writeMatrixToFile(const char \*filename, int matrix[][N]);  int main()  {  int A[N][N], B[N][N], C[N][N]; // Matrices A, B, and C  // Read matrices A and B from file  readMatrixFromFile("input.txt", A);  readMatrixFromFile("input2.txt", B);  // Calculate sum of matrices A and B  add(A, B, C);  // Write the result matrix C to a file  writeMatrixToFile("output.txt", C);  printf("Result matrix has been written to output.txt'\n");  return 0;  }  void add(int A[][N], int B[][N], int C[][N])  {  for (int i = 0; i < N; i++)  for (int j = 0; j < N; j++)  C[i][j] = A[i][j] - B[i][j];  }  void readMatrixFromFile(const char \*filename, int matrix[][N])  {  FILE \*file = fopen(filename, "r");  if (file == NULL)  {  printf("Error opening file '%s' for reading.\n", filename);  return;  }  for (int i = 0; i < N; i++)  for (int j = 0; j < N; j++)  fscanf(file, "%d", &matrix[i][j]);  fclose(file);  }  void writeMatrixToFile(const char \*filename, int matrix[][N])  {  FILE \*file = fopen(filename, "w");  if (file == NULL)  {  printf("Error opening file '%s' for writing.\n", filename);  return;  }  for (int i = 0; i < N; i++)  {  for (int j = 0; j < N; j++)  fprintf(file, "%d ", matrix[i][j]);  fprintf(file, "\n");  }  fclose(file);  }  OUTPUT  C:\Users\Lenovo\AppData\Local\Packages\Microsoft.Windows.Photos_8wekyb3d8bbwe\TempState\ShareServiceTempFolder\Screenshot (44).jpeg  18.Multiplication of matrix  #include <stdio.h>  #include <stdlib.h>  #define R1 2 // number of rows in Matrix-1  #define C1 2 // number of columns in Matrix-1  #define R2 2 // number of rows in Matrix-2  #define C2 2 // number of columns in Matrix-2  void mulMat(int mat1[][C1], int mat2[][C2], int result[][C2]);  void readMatrixFromFile(const char \*filename, int rows, int cols, int matrix[][cols]);  void writeMatrixToFile(const char \*filename, int rows, int cols, int matrix[][cols]);  int main() {  int mat1[R1][C1], mat2[R2][C2], result[R1][C2];  // Read matrices from files  readMatrixFromFile("input.txt", R1, C1, mat1);  readMatrixFromFile("input2.txt", R2, C2, mat2);  // Perform matrix multiplication  mulMat(mat1, mat2, result);  // Write the resulting matrix to a file  writeMatrixToFile("output.txt", R1, C2, result);  printf("Matrix multiplication result has been written to 'output.txt'\n");  return 0;  }  void mulMat(int mat1[][C1], int mat2[][C2], int result[][C2]) {  printf("Multiplication of given two matrices is:\n");  for (int i = 0; i < R1; i++) {  for (int j = 0; j < C2; j++) {  result[i][j] = 0;  for (int k = 0; k < R2; k++) {  result[i][j] += mat1[i][k] \* mat2[k][j];  }  printf("%d\t", result[i][j]);  }  printf("\n");  }  }  void readMatrixFromFile(const char \*filename, int rows, int cols, int matrix[][cols]) {  FILE \*file = fopen(filename, "r");  if (file == NULL) {  printf("Error opening file '%s' for reading.\n", filename);  exit(EXIT\_FAILURE);  }  for (int i = 0; i < rows; i++)  for (int j = 0; j < cols; j++)  fscanf(file, "%d", &matrix[i][j]);  fclose(file);  }  void writeMatrixToFile(const char \*filename, int rows, int cols, int matrix[][cols]) {  FILE \*file = fopen(filename, "w");  if (file == NULL) {  printf("Error opening file '%s' for writing.\n", filename);  exit(EXIT\_FAILURE);  }  for (int i = 0; i < rows; i++) {  for (int j = 0; j < cols; j++)  fprintf(file, "%d ", matrix[i][j]);  fprintf(file, "\n");  }  fclose(file);  }  OUTPUT  C:\Users\Lenovo\Downloads\WhatsApp Image 2024-03-17 at 17.19.02 (2).jpegC:\Users\Lenovo\Downloads\WhatsApp Image 2024-03-17 at 17.19.02 (1).jpeg  C:\Users\Lenovo\Downloads\WhatsApp Image 2024-03-17 at 17.19.02.jpeg  19.Transpose of a matrix  #include <stdio.h>  #define N 4  void transpose(int A[][N], int B[][N]);  void readMatrixFromFile(const char \*filename, int matrix[][N]);  void writeMatrixToFile(const char \*filename, int matrix[][N]);  int main() {  int A[N][N], B[N][N]; // Matrices A and B  // Read matrix A from file  readMatrixFromFile("input.txt", A);  // Calculate transpose of matrix A  transpose(A, B);  // Write the resulting matrix B to a file  writeMatrixToFile("output.txt", B);  printf("Transpose of matrix A has been written to 'output.txt'\n");  return 0;  }  void transpose(int A[][N], int B[][N]) {  for (int i = 0; i < N; i++)  for (int j = 0; j < N; j++)  B[i][j] = A[j][i];  }  void readMatrixFromFile(const char \*filename, int matrix[][N]) {  FILE \*file = fopen(filename, "r");  if (file == NULL) {  printf("Error opening file '%s' for reading.\n", filename);  return;  }  for (int i = 0; i < N; i++)  for (int j = 0; j < N; j++)  fscanf(file, "%d", &matrix[i][j]);  fclose(file);  }  void writeMatrixToFile(const char \*filename, int matrix[][N]) {  FILE \*file = fopen(filename, "w");  if (file == NULL) {  printf("Error opening file '%s' for writing.\n", filename);  return;  }  for (int i = 0; i < N; i++) {  for (int j = 0; j < N; j++)  fprintf(file, "%d ", matrix[i][j]);  fprintf(file, "\n");  }  fclose(file);  }  OUTPUT  C:\Users\Lenovo\Downloads\WhatsApp Image 2024-03-17 at 17.26.24.jpeg  20. String concatenation  #include <stdio.h>  #include <stdlib.h>  #define MAX\_LENGTH 100  int main() {  FILE \*file1, \*file2;  char string1[MAX\_LENGTH], string2[MAX\_LENGTH];  // Open input files  file1 = fopen("input.txt", "r");  file2 = fopen("input2.txt", "r");  // Check if files opened successfully  if (file1 == NULL || file2 == NULL) {  printf("Error opening input files.\n");  return 1;  }  // Read strings from files  fgets(string1, MAX\_LENGTH, file1);  fgets(string2, MAX\_LENGTH, file2);  // Close input files  fclose(file1);  fclose(file2);  // Concatenate strings  strcat(string1, string2);  // Print concatenated string  printf("Concatenated string: %s\n", string1);  return 0;  }  OUTPUT  C:\Users\Lenovo\Downloads\WhatsApp Image 2024-03-17 at 17.14.19.jpeg  21.String copy  #include <stdio.h>  #include <stdlib.h>  #define MAX\_LENGTH 100  int main() {  FILE \*sourceFile, \*destinationFile;  char sourceString[MAX\_LENGTH], destinationString[MAX\_LENGTH];  // Open the source file for reading  sourceFile = fopen("input.txt", "r");  if (sourceFile == NULL) {  printf("Error opening source file.\n");  return 1;  }  // Read the string from the source file  fgets(sourceString, MAX\_LENGTH, sourceFile);  // Close the source file  fclose(sourceFile);  // Open the destination file for writing  destinationFile = fopen("output.txt", "w");  if (destinationFile == NULL) {  printf("Error opening destination file.\n");  return 1;  }  // Write the string to the destination file  fputs(sourceString, destinationFile);  // Close the destination file  fclose(destinationFile);  // Open the destination file for reading  destinationFile = fopen("output.txt", "r");  if (destinationFile == NULL) {  printf("Error opening destination file for reading.\n");  return 1;  }  // Read the string from the destination file  fgets(destinationString, MAX\_LENGTH, destinationFile);  // Close the destination file  fclose(destinationFile);  // Print the copied string  printf("Copied string: %s\n", destinationString);  return 0;  }  OUTPUT  C:\Users\Lenovo\Downloads\WhatsApp Image 2024-03-17 at 17.14.39.jpeg  22.String compare  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #define MAX\_LENGTH 100  int main() {  FILE \*file1, \*file2, \*resultFile;  char string1[MAX\_LENGTH], string2[MAX\_LENGTH];  // Open the first file for reading  file1 = fopen("input.txt", "r");  if (file1 == NULL) {  printf("Error opening input.txt.\n");  return 1;  }  // Read the first string from the first file  fgets(string1, MAX\_LENGTH, file1);  // Close the first file  fclose(file1);  // Open the second file for reading  file2 = fopen("input2.txt", "r");  if (file2 == NULL) {  printf("Error opening input2.txt.\n");  return 1;  }  // Read the second string from the second file  fgets(string2, MAX\_LENGTH, file2);  // Close the second file  fclose(file2);  // Compare the strings  int compareResult = strcmp(string1, string2);  // Open the result file for writing  resultFile = fopen("output.txt", "w");  if (resultFile == NULL) {  printf("Error opening output.txt for writing.\n");  return 1;  }  // Write the comparison result to the result file  if (compareResult == 0) {  fputs("Strings are equal.", resultFile);  } else {  fputs("Strings are not equal.", resultFile);  }  // Close the result file  fclose(resultFile);  return 0;  }  OUTPUT  C:\Users\Lenovo\Downloads\WhatsApp Image 2024-03-17 at 17.14.50.jpeg |

**Part C**

|  |
| --- |
| **1.Title of the program** |
| Code  1. Implementation of stacks using arrays  #include <limits.h>  #include <stdio.h>  #include <stdlib.h>  // A structure to represent a stack  struct Stack {  int top;  unsigned capacity;  int\* array;  };  // function to create a stack of given capacity. It initializes size of  // stack as 0  struct Stack\* createStack(unsigned capacity)  {  struct Stack\* stack = (struct Stack\*)malloc(sizeof(struct Stack));  stack->capacity = capacity;  stack->top = -1;  stack->array = (int\*)malloc(stack->capacity \* sizeof(int));  return stack;  }  // Stack is full when top is equal to the last index  int isFull(struct Stack\* stack)  {  return stack->top == stack->capacity - 1;  }  // Stack is empty when top is equal to -1  int isEmpty(struct Stack\* stack)  {  return stack->top == -1;  }  // Function to add an item to stack. It increases top by 1  void push(struct Stack\* stack, int item)  {  if (isFull(stack))  return;  stack->array[++stack->top] = item;  printf("%d pushed to stack\n", item);  }  // Function to remove an item from stack. It decreases top by 1  int pop(struct Stack\* stack)  {  if (isEmpty(stack))  return INT\_MIN;  return stack->array[stack->top--];  }  // Function to return the top from stack without removing it  int peek(struct Stack\* stack)  {  if (isEmpty(stack))  return INT\_MIN;  return stack->array[stack->top];  }  // Driver program to test above functions  int main()  {  FILE \*input\_file = fopen("input.txt", "r");  FILE \*output\_file = fopen("output.txt", "w");  if (input\_file == NULL || output\_file == NULL) {  printf("Error opening files.\n");  return 1;  }  struct Stack\* stack = createStack(100);  int item;  while (fscanf(input\_file, "%d", &item) == 1) {  push(stack, item);  }    fprintf(output\_file, "Popped elements from stack:\n");  while (!isEmpty(stack)) {  int popped\_item = pop(stack);  fprintf(output\_file, "%d\n", popped\_item);  }  fclose(input\_file);  fclose(output\_file);  return 0;  }  Output  C:\Users\Lenovo\Downloads\WhatsApp Image 2024-03-17 at 13.55.24.jpeg  C:\Users\Lenovo\Downloads\WhatsApp Image 2024-03-17 at 13.55.26.jpeg  2.Stacks using linked lists  //stackslinkedlist  #include <limits.h>  #include <stdio.h>  #include <stdlib.h>  #define MAX\_LINE\_LENGTH 1000  // A structure to represent a stack node  struct StackNode {  int data;  struct StackNode\* next;  };  // Function to create a new stack node  struct StackNode\* newNode(int data)  {  struct StackNode\* stackNode =  (struct StackNode\*)malloc(sizeof(struct StackNode));  stackNode->data = data;  stackNode->next = NULL;  return stackNode;  }  // Check if stack is empty  int isEmpty(struct StackNode\* root)  {  return !root;  }  // Function to push an item to the stack  void push(struct StackNode\*\* root, int data)  {  struct StackNode\* stackNode = newNode(data);  stackNode->next = \*root;  \*root = stackNode;  printf("%d pushed to stack\n", data);  }  // Function to pop an item from stack  int pop(struct StackNode\*\* root)  {  if (isEmpty(\*root))  return INT\_MIN;  struct StackNode\* temp = \*root;  \*root = (\*root)->next;  int popped = temp->data;  free(temp);  return popped;  }  // Function to return the top element of stack  int peek(struct StackNode\* root)  {  if (isEmpty(root))  return INT\_MIN;  return root->data;  }  int main()  {  // Open the input file in read mode  FILE \*input\_file = fopen("input.txt", "r");  if (input\_file == NULL) {  printf("Error opening input file.\n");  return 1;  }  // Open the output file in write mode  FILE \*output\_file = fopen("output.txt", "w");  if (output\_file == NULL) {  printf("Error opening output file.\n");  fclose(input\_file);  return 1;  }  struct StackNode\* root = NULL;  int item;  while (fscanf(input\_file, "%d", &item) == 1) {  push(&root, item);  }  fprintf(output\_file, "Popped elements from stack:\n");  while (!isEmpty(root)) {  int popped\_item = pop(&root);  fprintf(output\_file, "%d\n", popped\_item);  }  fclose(input\_file);  fclose(output\_file);  return 0;  }  Output    C:\Users\Lenovo\Downloads\WhatsApp Image 2024-03-17 at 13.55.24.jpeg  3.Conversion of infix to postfix expression  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #define MAX\_EXPRESSION\_LENGTH 1000  // Function to return precedence of operators  int prec(char c) {  if (c == '^')  return 3;  else if (c == '/' || c == '\*')  return 2;  else if (c == '+' || c == '-')  return 1;  else  return -1;  }  // Function to return associativity of operators  char associativity(char c) {  if (c == '^')  return 'R';  return 'L'; // Default to left-associative  }  // The main function to convert infix expression to postfix expression  void infixToPostfix(char s[], FILE \*output\_file) {  char result[MAX\_EXPRESSION\_LENGTH];  int resultIndex = 0;  int len = strlen(s);  char stack[MAX\_EXPRESSION\_LENGTH];  int stackIndex = -1;  for (int i = 0; i < len; i++) {  char c = s[i];  // If the scanned character is an operand, add it to the output string.  if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c <= '9')) {  result[resultIndex++] = c;  }  // If the scanned character is an ‘(‘, push it to the stack.  else if (c == '(') {  stack[++stackIndex] = c;  }  // If the scanned character is an ‘)’, pop and add to the output string from the stack  // until an ‘(‘ is encountered.  else if (c == ')') {  while (stackIndex >= 0 && stack[stackIndex] != '(') {  result[resultIndex++] = stack[stackIndex--];  }  stackIndex--; // Pop '('  }  // If an operator is scanned  else {  while (stackIndex >= 0 && (prec(s[i]) < prec(stack[stackIndex]) ||  prec(s[i]) == prec(stack[stackIndex]) &&  associativity(s[i]) == 'L')) {  result[resultIndex++] = stack[stackIndex--];  }  stack[++stackIndex] = c;  }  }  // Pop all the remaining elements from the stack  while (stackIndex >= 0) {  result[resultIndex++] = stack[stackIndex--];  }  result[resultIndex] = '\0';  fprintf(output\_file, "%s\n", result);  }  int main() {  FILE \*input\_file = fopen("input.txt", "r");  FILE \*output\_file = fopen("output.txt", "w");  if (input\_file == NULL || output\_file == NULL) {  printf("Error opening files.\n");  return 1;  }  char exp[MAX\_EXPRESSION\_LENGTH];  while (fgets(exp, MAX\_EXPRESSION\_LENGTH, input\_file) != NULL) {  infixToPostfix(exp, output\_file);  }  fclose(input\_file);  fclose(output\_file);  printf("Conversion done successfully.\n");  return 0;  }  Output  C:\Users\Lenovo\Downloads\WhatsApp Image 2024-03-17 at 13.55.27.jpeg  C:\Users\Lenovo\Downloads\WhatsApp Image 2024-03-17 at 13.55.28 (1).jpeg  4.Check for a balanced parenthesis  #include <stdbool.h>  #include <stdio.h>  #define MAX\_EXPRESSION\_LENGTH 1000  // Function to check if parentheses are balanced  bool isBalanced(char exp[]) {  // Initialising Variables  bool flag = true;  int count = 0;  // Traversing the Expression  for (int i = 0; exp[i] != '\0'; i++) {  if (exp[i] == '(') {  count++;  } else {  // It is a closing parenthesis  count--;  }  if (count < 0) {  // This means there are more closing parenthesis  // than opening ones  flag = false;  break;  }  }  // If count is not zero,  // It means there are more opening parenthesis  if (count != 0) {  flag = false;  }  return flag;  }  int main() {  FILE \*input\_file = fopen("input.txt", "r");  FILE \*output\_file = fopen("output.txt", "w");  if (input\_file == NULL || output\_file == NULL) {  printf("Error opening files.\n");  return 1;  }  char exp[MAX\_EXPRESSION\_LENGTH];  while (fgets(exp, MAX\_EXPRESSION\_LENGTH, input\_file) != NULL) {  if (isBalanced(exp)) {  fprintf(output\_file, "%s is Balanced\n", exp);  } else {  fprintf(output\_file, "%s is Not Balanced\n", exp);  }  }  fclose(input\_file);  fclose(output\_file);  printf("Balance check done successfully.\n");  return 0;  }  Output  C:\Users\Lenovo\Downloads\WhatsApp Image 2024-03-17 at 13.55.29.jpeg  C:\Users\Lenovo\Downloads\WhatsApp Image 2024-03-17 at 13.55.30.jpeg  5. Queues using arrays  #include <limits.h>  #include <stdio.h>  #include <stdlib.h>  // A structure to represent a queue  struct Queue {  int front, rear, size;  unsigned capacity;  int\* array;  };  // Function to create a queue of given capacity. It initializes size of queue as 0  struct Queue\* createQueue(unsigned capacity) {  struct Queue\* queue = (struct Queue\*)malloc(sizeof(struct Queue));  queue->capacity = capacity;  queue->front = queue->size = 0;  // This is important, see the enqueue  queue->rear = capacity - 1;  queue->array = (int\*)malloc(queue->capacity \* sizeof(int));  return queue;  }  // Queue is full when size becomes equal to the capacity  int isFull(struct Queue\* queue) {  return (queue->size == queue->capacity);  }  // Queue is empty when size is 0  int isEmpty(struct Queue\* queue) {  return (queue->size == 0);  }  // Function to add an item to the queue. It changes rear and size  void enqueue(struct Queue\* queue, int item) {  if (isFull(queue))  return;  queue->rear = (queue->rear + 1) % queue->capacity;  queue->array[queue->rear] = item;  queue->size = queue->size + 1;  printf("%d enqueued to queue\n", item);  }  // Function to remove an item from queue. It changes front and size  int dequeue(struct Queue\* queue) {  if (isEmpty(queue))  return INT\_MIN;  int item = queue->array[queue->front];  queue->front = (queue->front + 1) % queue->capacity;  queue->size = queue->size - 1;  return item;  }  // Function to get front of queue  int front(struct Queue\* queue) {  if (isEmpty(queue))  return INT\_MIN;  return queue->array[queue->front];  }  // Function to get rear of queue  int rear(struct Queue\* queue) {  if (isEmpty(queue))  return INT\_MIN;  return queue->array[queue->rear];  }  int main() {  FILE \*input\_file = fopen("input.txt", "r");  FILE \*output\_file = fopen("output.txt", "w");  if (input\_file == NULL || output\_file == NULL) {  printf("Error opening files.\n");  return 1;  }  struct Queue\* queue = createQueue(1000);  int item;  while (fscanf(input\_file, "%d", &item) == 1) {  enqueue(queue, item);  }  fprintf(output\_file, "Front item is %d\n", front(queue));  fprintf(output\_file, "Rear item is %d\n", rear(queue));  fclose(input\_file);  fclose(output\_file);  return 0;  }  Output  C:\Users\Lenovo\Downloads\WhatsApp Image 2024-03-17 at 14.00.11.jpeg  C:\Users\Lenovo\Downloads\WhatsApp Image 2024-03-17 at 14.00.11 (1).jpeg  6.Queues using linked list  #include <stdio.h>  #include <stdlib.h>  // A linked list (LL) node to store a queue entry  struct QNode {  int key;  struct QNode\* next;  };  // The queue, front stores the front node of LL and rear stores the last node of LL  struct Queue {  struct QNode \*front, \*rear;  };  // A utility function to create a new linked list node.  struct QNode\* newNode(int k) {  struct QNode\* temp = (struct QNode\*)malloc(sizeof(struct QNode));  temp->key = k;  temp->next = NULL;  return temp;  }  // A utility function to create an empty queue  struct Queue\* createQueue() {  struct Queue\* q = (struct Queue\*)malloc(sizeof(struct Queue));  q->front = q->rear = NULL;  return q;  }  // The function to add a key k to q  void enQueue(struct Queue\* q, int k) {  // Create a new LL node  struct QNode\* temp = newNode(k);  // If queue is empty, then new node is front and rear both  if (q->rear == NULL) {  q->front = q->rear = temp;  return;  }  // Add the new node at the end of queue and change rear  q->rear->next = temp;  q->rear = temp;  }  // Function to remove a key from given queue q  void deQueue(struct Queue\* q) {  // If queue is empty, return NULL.  if (q->front == NULL)  return;  // Store previous front and move front one node ahead  struct QNode\* temp = q->front;  q->front = q->front->next;  // If front becomes NULL, then change rear also as NULL  if (q->front == NULL)  q->rear = NULL;  free(temp);  }  int main() {  FILE \*input\_file = fopen("input.txt", "r");  FILE \*output\_file = fopen("output.txt", "w");  if (input\_file == NULL || output\_file == NULL) {  printf("Error opening files.\n");  return 1;  }  struct Queue\* q = createQueue();  int item;  while (fscanf(input\_file, "%d", &item) == 1) {  enQueue(q, item);  }  fprintf(output\_file, "Queue Front: %d\n", (q->front != NULL ? q->front->key : -1));  fprintf(output\_file, "Queue Rear: %d\n", (q->rear != NULL ? q->rear->key : -1));  fclose(input\_file);  fclose(output\_file);  // Free memory allocated to the queue  while (q->front != NULL) {  deQueue(q);  }  free(q);  return 0;  }  Output  C:\Users\Lenovo\Downloads\WhatsApp Image 2024-03-17 at 14.39.05.jpeg  C:\Users\Lenovo\Downloads\WhatsApp Image 2024-03-17 at 14.39.05 (1).jpeg  7. Palindrome string  #include<stdio.h>  #include<stdlib.h>  #include<ctype.h>  #include<string.h>  #define SIZE 50  struct stack {  char el[SIZE];  int top;  };  typedef struct stack STACK;  void push(STACK \*s, char ch) {  if (s->top == SIZE - 1)  printf("Stack overflow\n");  else  s->el[++(s->top)] = ch;  }  char pop(STACK \*s) {  if (s->top == -1) {  printf("pop: STACK Underflow\n");  return '\0';  } else {  return s->el[(s->top)--];  }  }  void reversestring(STACK \*s, FILE \*input\_file, FILE \*output\_file) {  char inputstr[SIZE], outputstr[SIZE];  fscanf(input\_file, "%s", inputstr);  int i = 0;  while (inputstr[i] != '\0') {  push(s, inputstr[i]);  i++;  }  i = 0;  while (s->top != -1) {  outputstr[i] = pop(s);  i++;  }  outputstr[i] = '\0';  if (strcmp(inputstr, outputstr) == 0)  fprintf(output\_file, "%s is a palindrome string\n", inputstr);  else  fprintf(output\_file, "%s is not a palindrome string\n", inputstr);  }  int main() {  FILE \*input\_file = fopen("input.txt", "r");  FILE \*output\_file = fopen("output.txt", "w");  if (input\_file == NULL || output\_file == NULL) {  printf("Error opening files.\n");  return 1;  }  STACK st, \*s;  s = &st;  s->top = -1;  reversestring(s, input\_file, output\_file);  fclose(input\_file);  fclose(output\_file);  return 0;  }  Output  C:\Users\Lenovo\Downloads\Screenshot (38).png  8. Circular Queue  //circular queue  #include <stdio.h>  #include <stdlib.h>  #define MAX 5  int queue[MAX];  int front = -1, rear = -1;  void insert(int);  int delete();  void display();  void saveToFile();  void readFromFile();  int main() {  readFromFile(); // Load data from file, if any  int choice, data;  while(1) {  printf("\n1. Insert\n");  printf("2. Delete\n");  printf("3. Display\n");  printf("4. Exit\n");  printf("Enter your choice: ");  scanf("%d", &choice);  switch(choice) {  case 1:  printf("Enter data to insert: ");  scanf("%d", &data);  insert(data);  break;  case 2:  data = delete();  if (data != -1)  printf("Deleted element is %d\n", data);  break;  case 3:  display();  break;  case 4:  saveToFile(); // Save data to file before exiting  exit(0);  default:  printf("Invalid choice!\n");  }  }  return 0;  }  void insert(int data) {  if ((front == 0 && rear == MAX - 1) || (front == rear + 1)) {  printf("Queue Overflow!\n");  return;  }  if (front == -1)  front = 0;  rear = (rear + 1) % MAX;  queue[rear] = data;  printf("%d inserted into the queue.\n", data);  }  int delete() {  if (front == -1) {  printf("Queue Underflow!\n");  return -1;  }  int data = queue[front];  printf("Element deleted is %d\n", data);  if (front == rear)  front = rear = -1;  else  front = (front + 1) % MAX;  return data;  }  void display() {  if (front == -1) {  printf("Queue is empty!\n");  return;  }  printf("Queue elements are: ");  int i;  for (i = front; i != rear; i = (i + 1) % MAX)  printf("%d ", queue[i]);  printf("%d\n", queue[i]);  }  void saveToFile() {  FILE \*file = fopen("output.txt", "w");  if (file == NULL) {  printf("Error opening file for writing!\n");  return;  }  for (int i = front; i != rear; i = (i + 1) % MAX)  fprintf(file, "%d\n", queue[i]);  fprintf(file, "%d\n", queue[rear]);  fclose(file);  }  void readFromFile() {  FILE \*file = fopen("input.txt", "r");  if (file == NULL) {  printf("No previous data found!\n");  return;  }  int data;  while (fscanf(file, "%d", &data) != EOF)  insert(data);  fclose(file);  }  C:\Users\Lenovo\Downloads\Screenshot (39) (1).png |

9. singly circular insert position

#include <stdio.h>

#include <stdlib.h>

// Structure for a node in the linked list

struct node {

int data;

struct node \*link;

};

typedef struct node \*NODE;

// Function prototypes

NODE create\_node();

NODE display(NODE head);

int count(NODE head);

NODE insert\_position(NODE head);

void writeToFile(NODE head);

NODE readFromFile();

int main() {

NODE head = readFromFile(); // Read data from file if it exists

int choice;

while (1) {

printf("Enter your choice:\n");

printf("1. Display\n");

printf("2. Insert at Position\n");

printf("3. Count Nodes\n");

printf("4. Exit\n");

scanf("%d", &choice);

switch (choice) {

case 1:

display(head);

break;

case 2:

head = insert\_position(head);

writeToFile(head); // Write updated list to file

break;

case 3:

printf("Number of nodes: %d\n", count(head));

break;

case 4:

writeToFile(head); // Write final list to file before exiting

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid input!\n");

break;

}

}

return 0;

}

// Function to create a new node

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if (n1 == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

// Function to display the linked list

NODE display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n");

return head;

}

NODE temp = head;

while (temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

return head;

}

// Function to count the number of nodes in the linked list

int count(NODE head) {

int cnt = 0;

NODE temp = head;

while (temp != NULL) {

cnt++;

temp = temp->link;

}

return cnt;

}

NODE insert\_position(NODE head)

{

NODE temp, p, cur;

int pos = 1, cnt = 1;

printf("\nEnter the position: ");

scanf("%d", &pos);

temp = create\_node();

if (head == NULL && pos ==1)

{

temp->link=temp;

head=temp;

}

/\* else if (head == NULL && pos == 1)

{

return temp;

}\*/

else if (pos == 1)

{

cur=head;

while(cur->link!=head)

{

cur=cur->link;

}

cur->link=temp;

temp->link=head;

head=temp;

return head;

}

else

{

p = NULL;

cur = head;

do

{

p = cur;

cur = cur->link;

cnt++;

}while(cur!=head && pos!=cnt);

if (cnt == pos)

{

p->link = temp;

temp->link = cur;

}

else if (pos > cnt + 1)

{

printf("Invalid position\n");

}

}

return head;

}

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully.\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

}

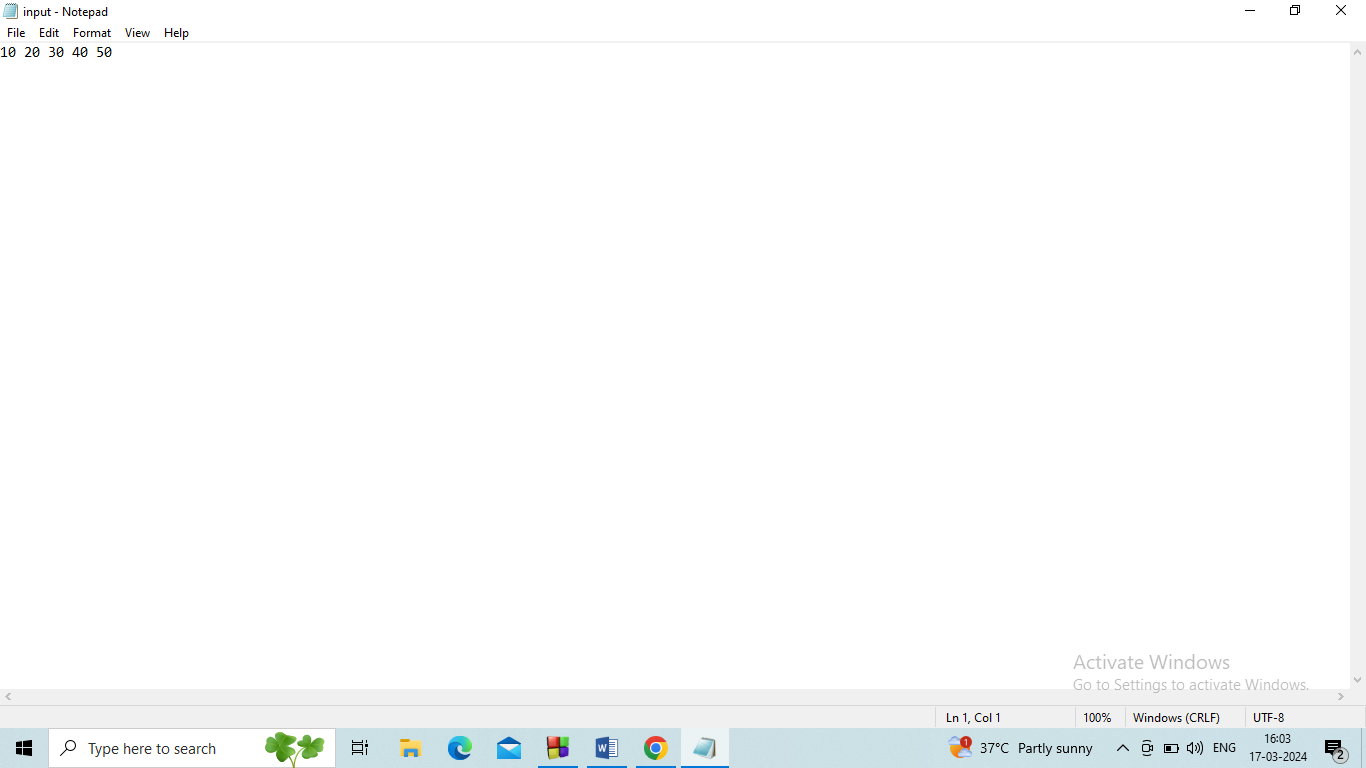
}

fclose(file);

printf("Data read from file successfully.\n");

  return head;

}



display(head);

break;

case 2:

head = insert\_position(head);

writeToFile(head); // Write updated list to file

break;

case 3:

printf("Number of nodes: %d\n", count(head));

break;

case 4:

writeToFile(head); // Write final list to file before exiting

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid input!\n");

break;

}

}

return 0;

}

// Function to create a new node

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if (n1 == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

// Function to display the linked list

NODE display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n");

return head;

}

NODE temp = head;

while (temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

return head;

}

int count(NODE head) {

int cnt = 0;

NODE temp = head;

while (temp != NULL) {

cnt++;

temp = temp->link;

}

return cnt;

}

NODE insert\_position(NODE head)

return cnt;

}

NODE insert\_position(NODE head)

NODE temp, p, cur;

int pos = 1, cnt = 1;

printf("\nEnter the position: ");

scanf("%d", &pos);

temp = create\_node();

if (head == NULL && pos ==1)

{

temp->link=temp;

head=temp;

}

/\* else if (head == NULL && pos == 1)

{

return temp;

}\*/

else if (pos == 1)

{

cur=head;

while(cur->link!=head)

{

cur=cur->link;

}

cur->link=temp;

temp->link=head;

head=temp;

return head;

}

else

{

p = NULL;

cur = head;

do

{

p = cur;

cur = cur->link;

cnt++;

}while(cur!=head && pos!=cnt);

if (cnt == pos)

{

p->link = temp;

temp->link = cur;

}

else if (pos > cnt + 1)

{

printf("Invalid position\n");

}

}

return head;

}

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully.\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

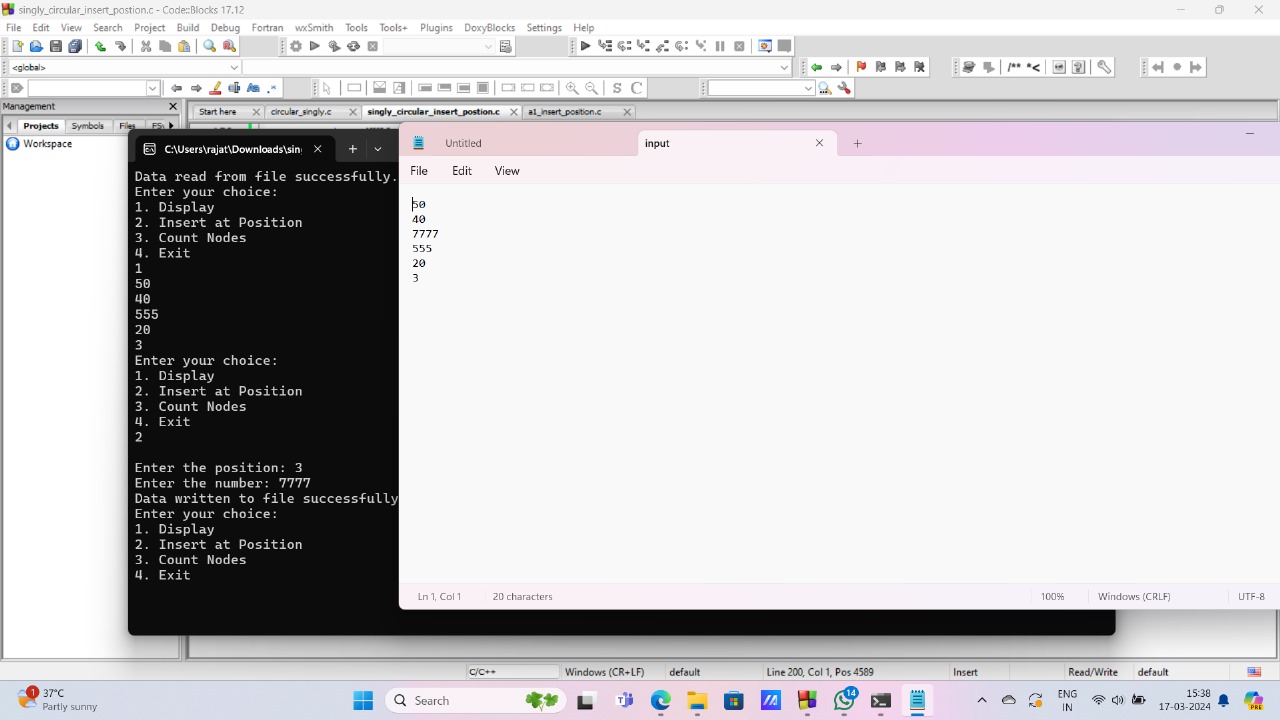
}

}

fclose(file);

printf("Data read from file successfully.\n");

  return head;

} 

10. singly circular delete position

#include<stdio.h>

#include<stdlib.h>

// Structure for a node in the linked list

struct node {

int data;

struct node \*link;

};

typedef struct node \*NODE;

// Function prototypes

NODE create\_node();

NODE insert\_front(NODE head);

NODE delete\_position(NODE head);

NODE display(NODE head);

void writeToFile(NODE head);

NODE readFromFile();

int count(NODE head);

int main() {

NODE head = readFromFile(); // Read data from file if it exists

int choice;

while(1) {

printf("Enter your choice:\n");

printf("1. Insert Front\n");

printf("2. Display\n");

printf("3. Delete Position\n");

printf("4. Count\n");

printf("5. Exit\n");

scanf("%d", &choice);

switch(choice) {

case 1:

head = insert\_front(head);

writeToFile(head); // Write updated list to file

break;

case 2:

display(head);

break;

case 3:

head = delete\_position(head);

writeToFile(head); // Write updated list to file

break;

case 4:

printf("Count = %d\n", count(head));

break;

case 5:

printf("THANK YOU\n");

writeToFile(head); // Write final list to file before exiting

exit(0);

default:

printf("INVALID INPUT\n");

break;

}

}

}

NODE create\_node() {

NODE n1 = (NODE)malloc(sizeof(struct node));

if(n1 == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

// Function to insert a node at the front of the linked list

NODE insert\_front(NODE head) {

NODE temp = create\_node();

if(head == NULL) {

head = temp;

} else {

temp->link = head;

head = temp;

}

return head;

}

// Function to display the linked list

NODE display(NODE head) {

NODE temp = head;

if(temp == NULL) {

printf("THE LIST IS EMPTY\n");

} else {

while(temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

}

printf("\n");

return head;

}

// Function to delete a node at a specified position in the linked list

NODE delete\_position(NODE head) {

NODE cur, temp;

int pos, cnt = 0;

printf("\nEnter the position: ");

scanf("%d", &pos);

if (head == NULL) {

printf("List is empty.\n");

return head;

}

cur = head;

do {

cnt++;

cur = cur->link;

} while (cur != head);

if (pos <= 0 || pos > cnt) {

printf("Invalid position.\n");

return head;

}

if (pos == 1) {

temp = head;

while(cur->link != head) {

cur = cur->link;

}

cur->link = temp->link;

head = temp->link;

free(temp);

} else {

cur = head;

for (int i = 1; i < pos - 1; i++) {

cur = cur->link;

}

temp = cur->link;

cur->link = temp->link;

free(temp);

}

printf("Node at position %d deleted successfully.\n", pos);

return head;

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

}

}

fclose(file);

printf("Data read from file successfully\n");

return head;

}

// Function to count the number of nodes in the linked list

int count(NODE head) {

int cnt = 0;

NODE temp = head;

while (temp != NULL) {

cnt++;

temp = temp->link;

}

   return cnt;

}

if(n1 == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

// Function to insert a node at the front of the linked list

NODE insert\_front(NODE head) {

NODE temp = create\_node();

if(head == NULL) {

head = temp;

} else {

temp->link = head;

head = temp;

}

return head;

}

// Function to display the linked list

NODE display(NODE head) {

NODE temp = head;

if(temp == NULL) {

printf("THE LIST IS EMPTY\n");

} else {

while(temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

}

printf("\n");

return head;

}

NODE delete\_position(NODE head) {

NODE cur, temp;

int pos, cnt = 0;

printf("\nEnter the position: ");

scanf("%d", &pos);

if (head == NULL) {

printf("List is empty.\n");

return head;

}

cur = head;

do {

cnt++;

cur = cur->link;

} while (cur != head);

if (pos <= 0 || pos > cnt) {

printf("Invalid position.\n");

return head;

}

if (pos == 1) {

temp = head;

while(cur->link != head) {

cur = cur->link;

}

cur->link = temp->link;

head = temp->link;

free(temp);

} else {

while(cur->link != head) {

cur = cur->link;

}

cur->link = temp->link;

head = temp->link;

free(temp);

} else {

cur = head;

for (int i = 1; i < pos - 1; i++) {

cur = cur->link;

}

temp = cur->link;

cur->link = temp->link;

free(temp);

}

printf("Node at position %d deleted successfully.\n", pos);

return head;

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

} [Grab your reader’s attention with a great quote from the document or use this space to emphasize a key point. To place this text box anywhere on the page, just drag it.]

}

fclose(file);

printf("Data read from file successfully\n");

return head;

}

// Function to count the number of nodes in the linked list

int count(NODE head) {

11. singly circular insert end

#include <stdio.h>

#include <stdlib.h>

// Structure for a node in the linked list

struct node {

int data;

struct node \*link;

};

typedef struct node \*NODE;

// Function prototypes

NODE create\_node();

NODE insert\_end(NODE head);

void display(NODE head);

void writeToFile(NODE head);

NODE readFromFile();

int main() {

NODE head = readFromFile(); // Read data from file if it exists

int choice;

while (1) {

printf("Enter your choice:\n");

printf("1. Insert End\n");

printf("2. Display\n");

printf("3. Exit\n");

scanf("%d", &choice);

switch (choice) {

case 1:

head = insert\_end(head);

writeToFile(head); // Write updated list to file

break;

case 2:

display(head);

break;

case 3:

writeToFile(head); // Write final list to file before exiting

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid input!\n");

break;

}

}

return 0;

}

// Function to create a new node

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if (n1 == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

NODE insert\_end(NODE head) {

NODE temp, p;

temp = create\_node();

if (temp == NULL) {

// Handle the case where temp is NULL differently

return head;

}

if (head == NULL) {

head = temp;

head->link = head; // Make the node point to itself

} else {

p = head;

while (p->link != head) {

p = p->link;

}

p->link = temp;

temp->link = head;

}

return head;

}

void display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n");

return;

}

NODE temp = head;

do {

printf("%d\n", temp->data);

temp = temp->link;

} while (temp != head);

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

do {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

} while (temp != head);

fclose(file);

printf("Data written to file successfully.\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

head->link = head; // Make the node point to itself

} else {

NODE p = head;

while (p->link != head) {

p = p->link;

}

p->link = newNode;

newNode->link = head;

}

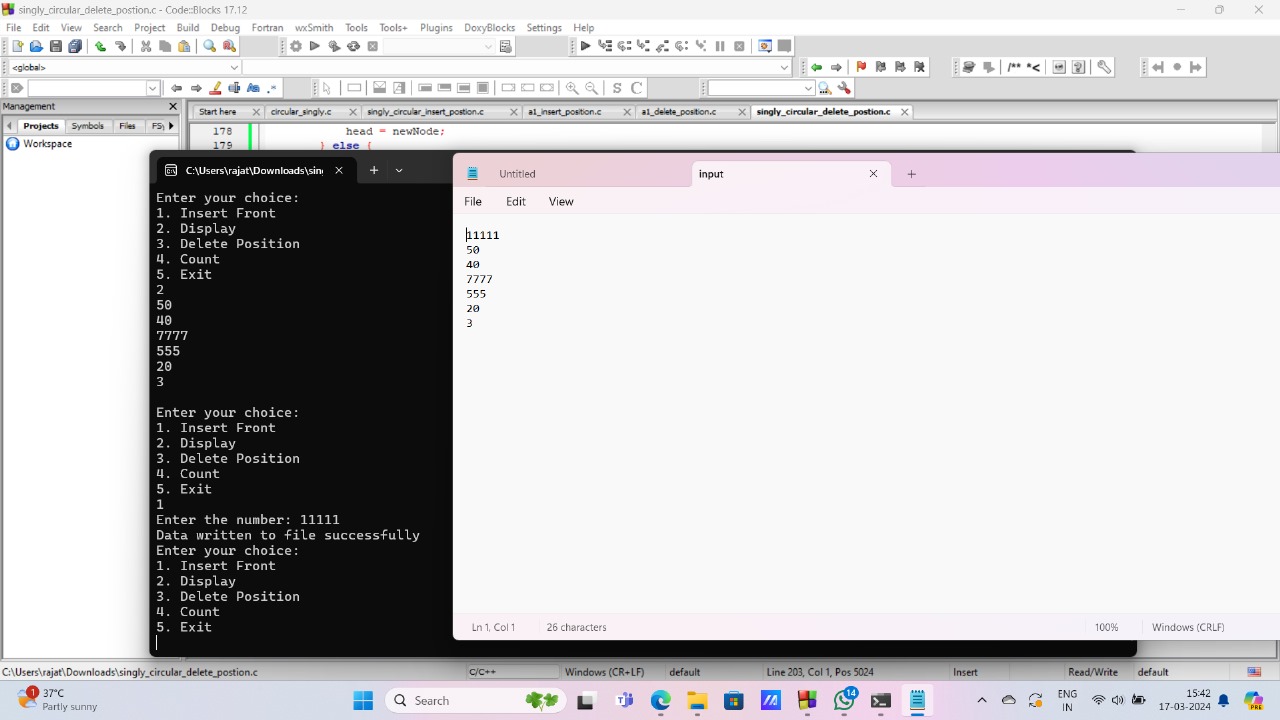
}

fclose(file);

printf("Data read from file successfully.\n");

  return head;

}



break;

case 3:

writeToFile(head); // Write final list to file before exiting

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid input!\n");

break;

}

}

return 0;

}

// Function to create a new node

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if (n1 == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

NODE insert\_end(NODE head) {

NODE temp, p;

temp = create\_node();

if (temp == NULL) {

// Handle the case where temp is NULL differently

return head;

}

if (head == NULL) {

head = temp;

head->link = head; // Make the node point to itself

} else {

p = head;

while (p->link != head) {

p = p->link;

}

p->link = temp;

temp->link = head;

}

return head;

}

void display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n");

return;

}

NODE temp = head;

do {

printf("%d\n", temp->data);

temp = temp->link;

} while (temp != head);

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

do {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

} while (temp != head);

fclose(file);

printf("Data written to file successfully.\n");

}

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

do {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

} while (temp != head);

fclose(file);

printf("Data written to file successfully.\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

head->link = head; // Make the node point to itself

} else {

NODE p = head;

while (p->link != head) {

p = p->link;

}

p->link = newNode;

newNode->link = head;

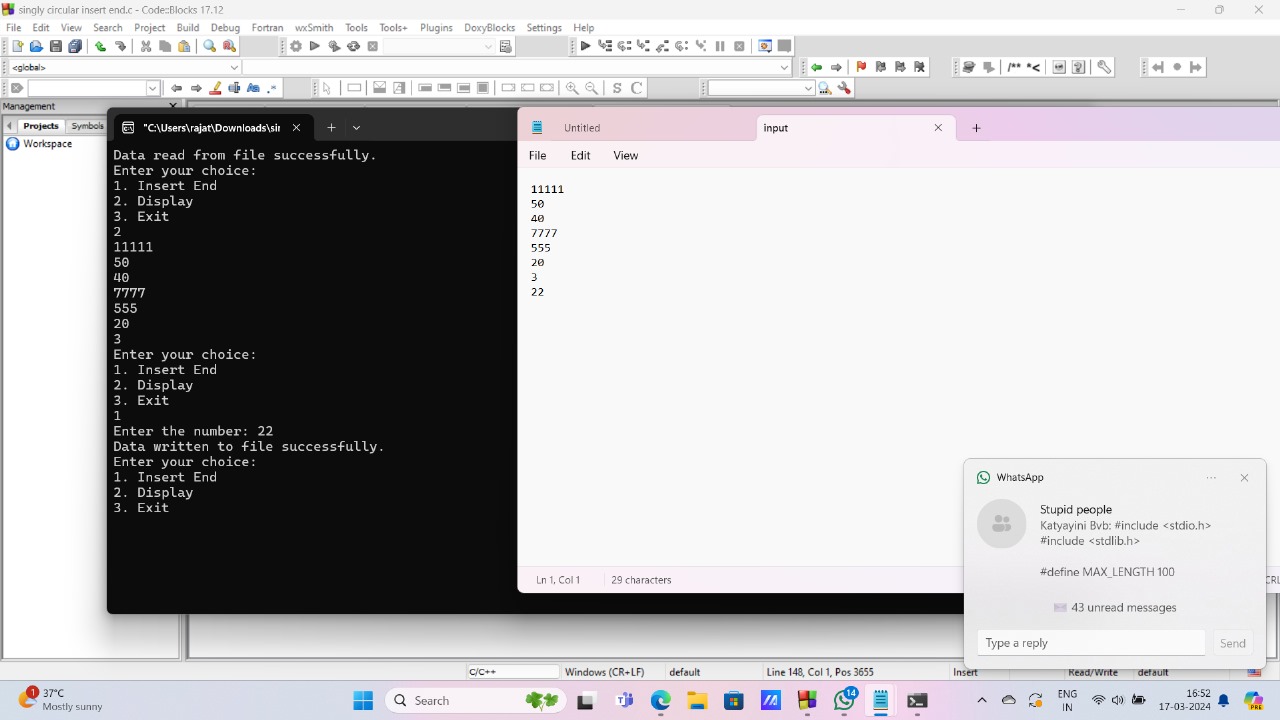
}

}

fclose(file);

printf("Data read from file successfully.\n");

  return head;

} 

12. doubly circular insert front

#include <stdio.h>

#include <stdlib.h>

// Define a structure for a node in the linked list

struct node {

int data;

struct node \*link;

};

// Define a pointer type for the node structure

typedef struct node \*NODE;

// Function prototypes

NODE create\_node();

NODE insert\_front(NODE head);

void display(NODE head);

void writeToFile(NODE head);

NODE readFromFile();

int main() {

// Read the linked list from the file (if exists)

NODE head = readFromFile();

if (head == NULL) {

printf("Failed to read from file. Starting with an empty list.\n");

}

int choice;

// Menu-driven program

while (1) {

printf("Enter your choice:\n");

printf("1. Insert Front\n");

printf("2. Display\n");

printf("3. Exit\n");

scanf("%d", &choice);

switch (choice) {

case 1:

head = insert\_front(head); // Insert element at front

writeToFile(head); // Write linked list to file

break;

case 2:

display(head); // Display the linked list

break;

case 3:

writeToFile(head); // Write linked list to file before exiting

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid input!\n");

break;

}

}

return 0;

}

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node)); // Allocate memory for the node

if (n1 == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data); // Read data for the node

n1->link = NULL; // Set the link to NULL

return n1;

}

NODE insert\_front(NODE head) {

NODE temp = create\_node(); // Create a new node

temp->link = head; // Set the link of the new node to the current head

head = temp; // Update head to point to the new node

return head;

}

void display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n"); // If list is empty, print a message

return;

}

NODE temp = head;

while (temp != NULL) {

printf("%d\n", temp->data); // Print data of each node

temp = temp->link; // Move to the next node

}

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w"); // Open the file for writing

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data); // Write data of each node to file

temp = temp->link; // Move to the next node

}

fclose(file); // Close the file

printf("Data written to file successfully.\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r"); // Open the file for reading

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node)); // Allocate memory for a new node

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

newNode->data = value; // Set data of the new node

newNode->link = NULL; // Set link to NULL

if (head == NULL) {

head = newNode; // If list is empty, new node becomes head

} else {

NODE temp = head;

while (temp->link != NULL) {

temp = temp->link; // Traverse till the end of the list

}

temp->link = newNode; // Set the link of the last node to the new node

}

}

fclose(file); // Close the file

printf("Data read from file successfully.\n");

  return head;

}

printf("Memory allocation failed.\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data); // Read data for the node

n1->link = NULL; // Set the link to NULL

return n1;

}

NODE insert\_front(NODE head) {

NODE temp = create\_node(); // Create a new node

temp->link = head; // Set the link of the new node to the current head

head = temp; // Update head to point to the new node

return head;

}

void display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n"); // If list is empty, print a message

return;

}

NODE temp = head;

while (temp != NULL) {

printf("%d\n", temp->data); // Print data of each node

temp = temp->link; // Move to the next node

}

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w"); // Open the file for writing

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link; // Move to the next node

}

fclose(file); // Close the file

printf("Data written to file successfully.\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r"); // Open the file for reading

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node)); // Allocate memory for a new node

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

newNode->data = value; // Set data of the new node

newNode->link = NULL; // Set link to NULL

if (head == NULL) {

head = newNode; // If list is empty, new node becomes head

} else {

NODE temp = head;

while (temp->link != NULL) {

temp = temp->link; // Traverse till the end of the list

}

temp->link = newNode; // Set the link of the last node to the new node

}

}

fclose(file); // Close the file

13. doubly circular insert end

#include <stdio.h>

#include <stdlib.h>

// Structure for a node in the linked list

struct node {

int data;

struct node \*link;

};

typedef struct node \*NODE;

// Function prototypes

NODE create\_node();

NODE insert\_end(NODE head);

void display(NODE head);

void writeToFile(NODE head);

NODE readFromFile();

int main() {

NODE head = readFromFile(); // Read data from file if it exists

int choice;

while (1) {

printf("Enter your choice:\n");

printf("1. Insert End\n");

printf("2. Display\n");

printf("3. Exit\n");

scanf("%d", &choice);

switch (choice) {

case 1:

if (head == NULL) {

head = newNode; // If list is empty, new node becomes head

} else {

NODE temp = head;

while (temp->link != NULL) {

temp = temp->link; // Traverse till the end of the list

}

temp->link = newNode; // Set the link of the last node to the new node

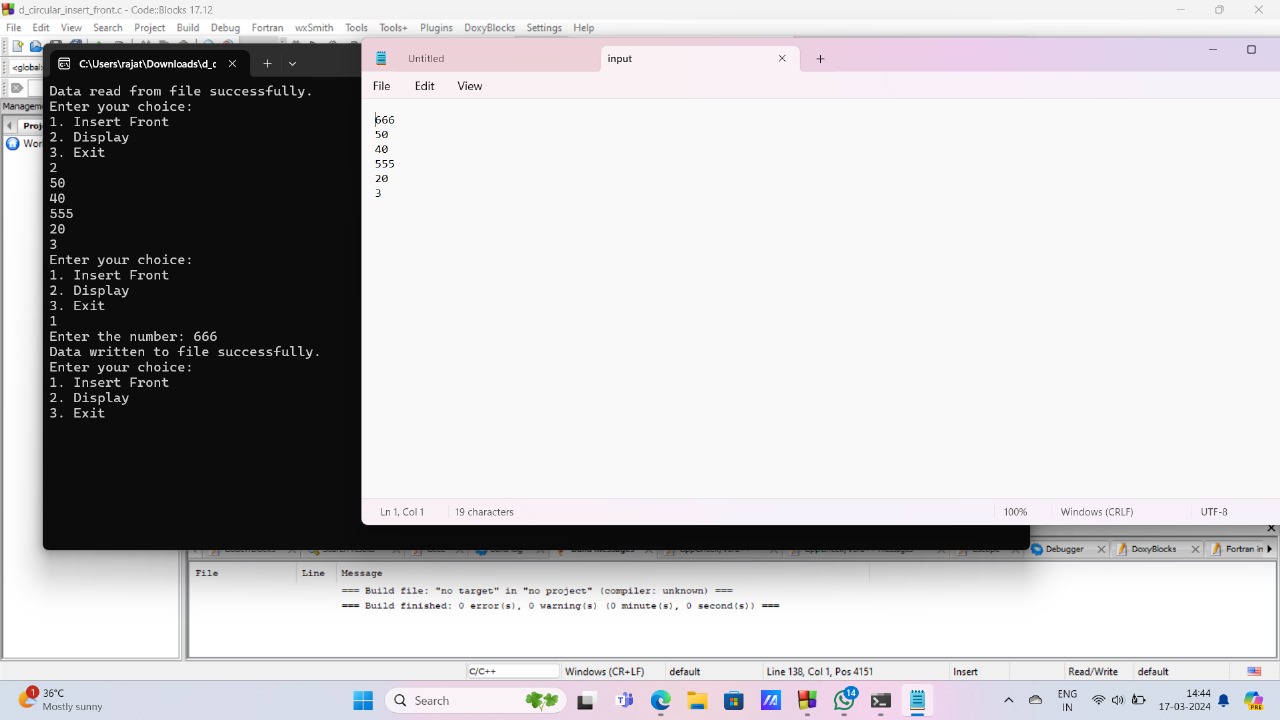
}

}

fclose(file); // Close the file

printf("Data read from file successfully.\n");

  return head;

} 

switch (choice) {

case 1:

head = insert\_end(head);

writeToFile(head); // Write updated list to file

break;

case 2:

display(head);

break;

case 3:

writeToFile(head); // Write final list to file before exiting

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid input!\n");

break;

}

}

return 0;

}

// Function to create a new node

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if (n1 == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

NODE insert\_end(NODE head) {

NODE temp = create\_node(); // Create a new node

if (head == NULL) {

head = temp;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link; // Traverse till the end of the list

}

p->link = temp; // Set the link of the last node to the new node

}

return head;

}

void display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully.\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully.\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != N NULL) {

p = p->link;

}

p->link = newNode;

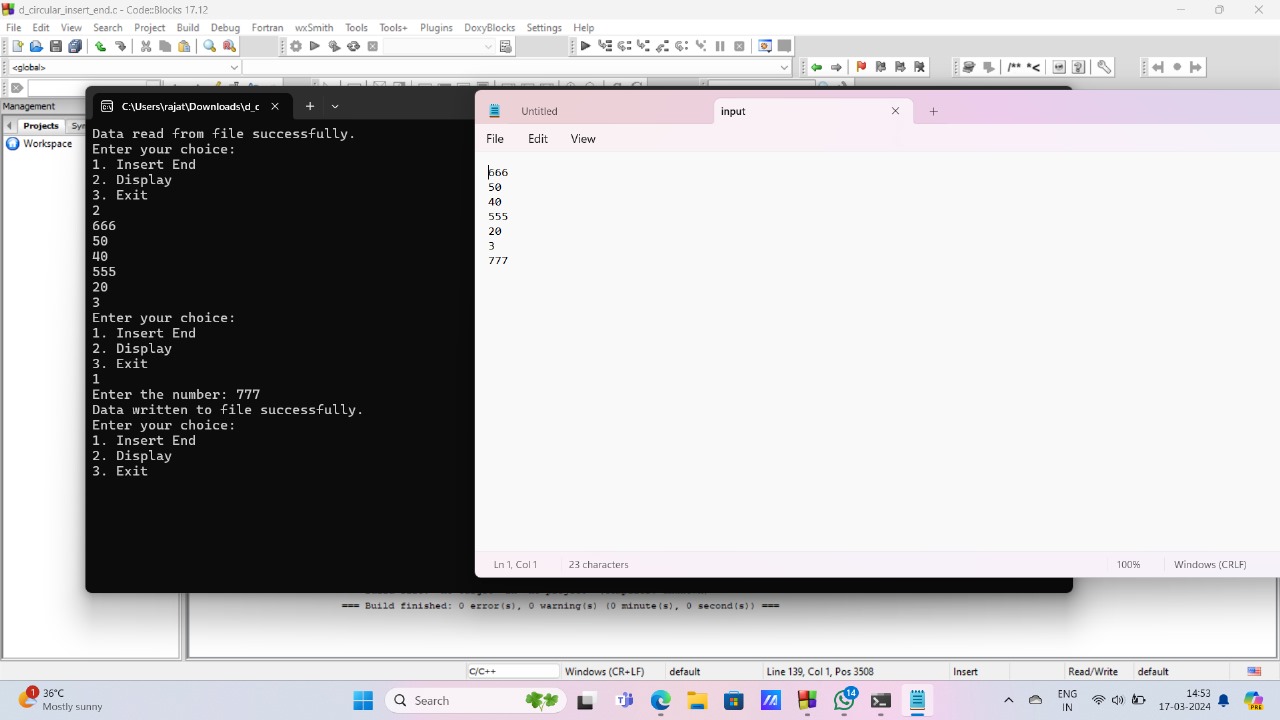
}

}

fclose(file);

printf("Data read from file successfully.\n");

  return head;

} 

14. doubly circular insert position

#include <stdio.h>

#include <stdlib.h>

// Structure for a node in the linked list

struct node {

int data;

struct node \*plink; // Previous link

struct node \*nlink; // Next link

};

typedef struct node \*NODE;

// Function prototypes

NODE create\_node();

NODE display(NODE head);

int count(NODE head);

NODE insert\_position(NODE head);

void writeToFile(NODE head);

NODE readFromFile();

int main() {

NODE head = readFromFile(); // Read data from file if it exists

int choice;

while (1) {

printf("Enter your choice:\n");

printf("1. Display\n");

printf("2. Insert at Position\n");

printf("3. Count Nodes\n");

printf("4. Exit\n");

scanf("%d", &choice);

switch (choice) {

case 1:

head = display(head);

break;

case 2:

head = insert\_position(head);

writeToFile(head); // Write updated list to file

break;

case 3:

printf("Number of nodes: %d\n", count(head));

break;

case 4:

writeToFile(head); // Write final list to file before exiting

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid input!\n");

break;

}

} return 0;

}

// Function to create a new node

NODE create\_node() {

NODE n1 = (NODE)malloc(sizeof(struct node));

if (n1 == NULL) {

printf("Memory allocation failed.\n");

exit(1);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->plink = NULL;

n1->nlink = NULL;

return n1;

}

// Function to display the linked list

NODE display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n");

return head;

}

NODE temp = head;

while (temp != NULL) {

printf("%d\n", temp->data);

temp = temp->nlink;

}

return head;

}

// Function to count the number of nodes in the linked list

int count(NODE head) {

int cnt = 0;

NODE temp = head;

while (temp != NULL) {

cnt++;

temp = temp->nlink;

}

return cnt;

}

NODE insert\_position(NODE head) {

NODE temp, p = head;

int pos, i = 1;

printf("Enter the position: ");

scanf("%d", &pos);

temp = create\_node();

if (head == NULL && pos == 1) {

head = temp;

} else if (pos == 1) {

while (p->nlink != head) {

p = p->nlink;

}

p->nlink = temp;

temp->plink = p;

temp->nlink = head;

head->plink = temp;

head = temp;

} else {

while (i < pos - 1 && p != NULL) {

p = p->nlink;

i++;

}

if (p == NULL) {

printf("Invalid position.\n");

return head;

}

NODE cur = p->nlink;

p->nlink = temp;

temp->plink = p;

if (cur != NULL) {

temp->nlink = cur;

cur->plink = temp;

}

}

return head;

}

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->nlink;

}

fclose(file);

printf("Data written to file successfully.\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(1);

}

newNode->data = value;

newNode->plink = NULL;

newNode->nlink = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->nlink != NULL) {

p = p->nlink;

}

p->nlink = newNode;

newNode->plink = p;

}

}

fclose(file);

printf("Data read from file successfully.\n");

  return head;

}

printf("Memory allocation failed.\n");

exit(1);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->plink = NULL;

n1->nlink = NULL;

return n1;

}

// Function to display the linked list

NODE display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n");

return head;

}

NODE temp = head;

while (temp != NULL) {

printf("%d\n", temp->data);

temp = temp->nlink;

}

return head;

}

// Function to count the number of nodes in the linked list

int count(NODE head) {

int cnt = 0;

NODE temp = head;

while (temp != NULL) {

cnt++;

temp = temp->nlink;

}

return cnt;

}

NODE insert\_position(NODE head) {

NODE temp, p = head;

int pos, i = 1;

printf("Enter the position: ");

scanf("%d", &pos);

temp = create\_node();

if (head == NULL && pos == 1) {

head = temp;

} else if (pos == 1) {

while (p->nlink != head) {

p = p->nlink;

}

p->nlink = temp;

temp->plink = p;

temp->nlink = head;

head->plink = temp;

head = temp;

} else {

while (i < pos - 1 && p != NULL) {

p = p->nlink;

i++;

}

if (p == NULL) {

printf("Invalid position.\n");

return head;

}

NODE cur = p->nlink;

p->nlink = temp;

temp->plink = p;

if (cur != NULL) {

temp->nlink = cur;

cur->plink = temp;

}

}

return head;

}

}

NODE cur = p->nlink;

p->nlink = temp;

temp->plink = p;

if (cur != NULL) {

temp->nlink = cur;

cur->plink = temp;

}

}

return head;

}

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->nlink;

}

fclose(file);

printf("Data written to file successfully.\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)m malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(1);

}

newNode->data = value;

newNode->plink = NULL;

newNode->nlink = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->nlink != NULL) {

p = p->nlink;

}

p->nlink = newNode;

newNode->plink = p;

}

}

fclose(file);

printf("Data read from file successfully.\n");

  return head;

}

15.doubly circular delete position

#include<stdio.h>

#include<stdlib.h>

// Structure for a node in the linked list

struct node {

int data;

struct node \*link;

};

typedef struct node \*NODE;

// Function prototypes

NODE create\_node();

NODE insert\_front(NODE head);

NODE delete\_position(NODE head);

NODE display(NODE head);

void writeToFile(NODE head);

NODE readFromFile();

int count(NODE head);

int main() {

NODE head = readFromFile(); // Read data from file if it exists

int choice;

while(1) {

printf("Enter your choice:\n");

printf("1. Insert Front\n");

printf("2. Display\n");

printf("3. Delete Position\n");

printf("4. Count\n");

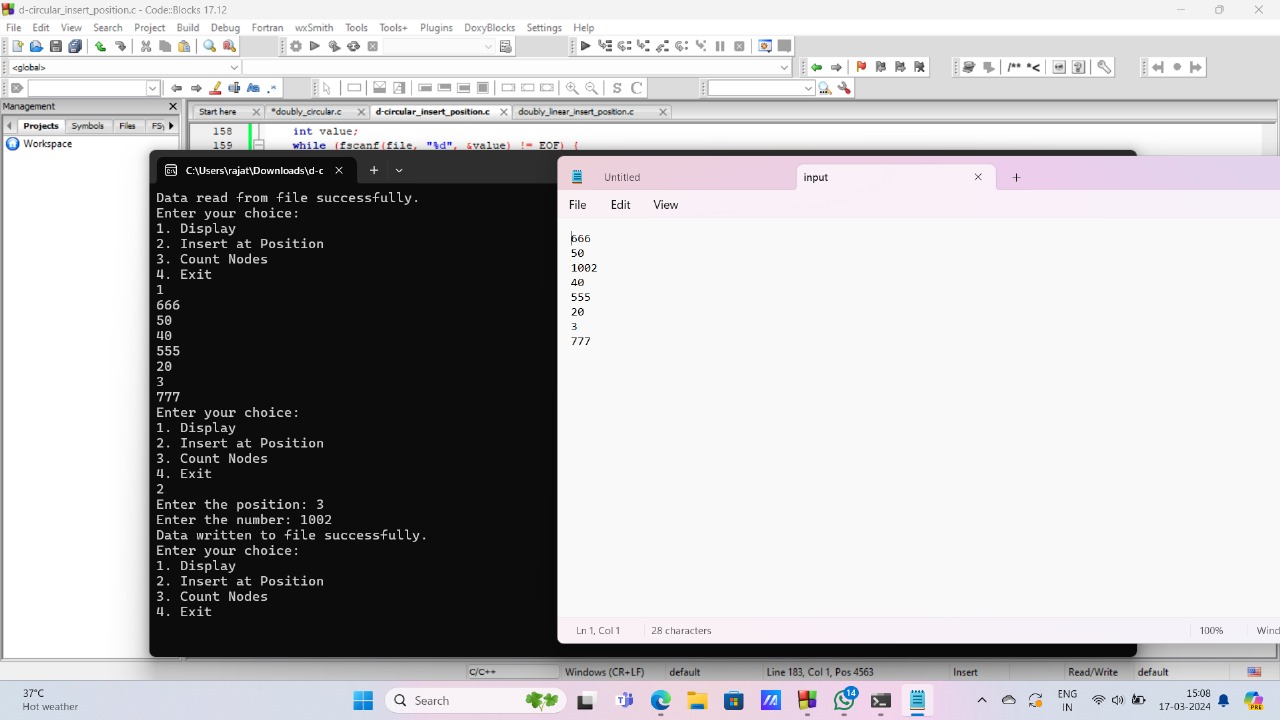
printf("5. Exit\n");

scanf("%d", &choice);

switch(choice) {

case 1:

head = insert\_front(head);



head = insert\_front(head);

writeToFile(head); // Write updated list to file

break;

case 2:

display(head);

break;

case 3:

head = delete\_position(head);

writeToFile(head); // Write updated list to file

break;

case 4:

printf("Count = %d\n", count(head));

break;

case 5:

printf("THANK YOU\n");

writeToFile(head); // Write final list to file before exiting

exit(0);

default:

printf("INVALID INPUT\n");

break;

}

}

}

// Function to create a new node

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if(n1 == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

NODE insert\_front(NODE head) {

NODE temp;

temp = create\_node();

if(head == NULL) {

head = temp;

} else {

temp->link = head;

head = temp;

}

return head;

}

// Function to display the linked list

NODE display(NODE head) {

NODE temp;

temp = head;

if(temp == NULL) {

printf("THE LIST IS EMPTY\n");

} else {

while(temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

}

printf("\n");

return head;

}

// Function to delete a node at a specified position in the linked list

NODE delete\_position(NODE head) {

NODE temp = head, prev = NULL;

int pos, count = 0;

if (head == NULL) {

printf("List is empty. Deletion not possible.\n");

return head;

}

printf("Enter the position to delete: ");

scanf("%d", &pos);

NODE delete\_position(NODE head) {

NODE temp = head, prev = NULL;

int pos, count = 0;

if (head == NULL) {

printf("List is empty. Deletion not possible.\n");

return head;

}

printf("Enter the position to delete: ");

scanf("%d", &pos);

if (pos < 1) {

printf("Invalid position.\n");

return head;

}

if (pos == 1) {

while (temp->link != head) {

temp = temp->link;

}

temp->link = head->link;

free(head);

head = temp->link;

} else {

while (temp->link != head && count < pos - 1) {

prev = temp;

temp = temp->link;

count++;

}

if (count != pos - 1) {

printf("Position out of range.\n");

return head;

} prev->link = temp->link;

free(temp);

}

printf("Node deleted successfully.\n");

return head;

}

// Function to count the number of nodes in the linked list

int count(NODE head) {

int cnt = 0;

NODE temp = head;

while (temp != NULL) {

cnt++;

temp = temp->link;

}

return cnt;

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully\n");

}

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

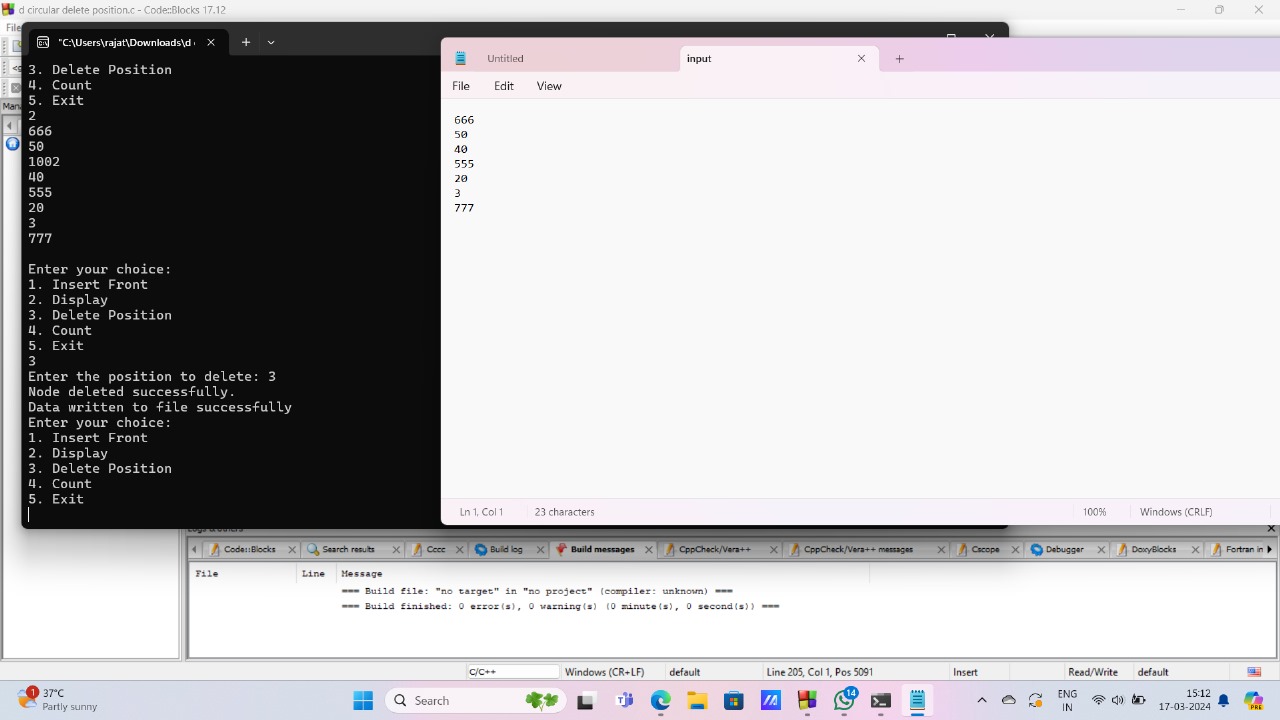
}

}

fclose(file);

printf("Data read from file success fully\n");

  return head;

} 

16. doubly circular delete front

#include <stdio.h>

#include <stdlib.h>

// Structure for a node in the linked list

struct node {

int data;

struct node \*link;

};

typedef struct node \*NODE;

// Function prototypes

NODE create\_node();

NODE insert\_front(NODE head);

NODE delete\_front(NODE head);

NODE display(NODE head);

void writeToFile(NODE head);

NODE readFromFile();

int main() {

NODE head = readFromFile(); // Read data from file if it exists

int choice;

while(1) {

printf("Enter your choice:\n");

printf("1. Delete Front\n");

printf("2. Display\n");

printf("3. Insert Front\n");

printf("4. Exit\n");

scanf("%d", &choice);

switch(choice) {

case 1:

head = delete\_front(head);

writeToFile(head); // Write updated list to file

break;

case 2:

display(head);

break;

case 3:

head = insert\_front(head);

writeToFile(head); // Write updated list to file

break;

case 4:

printf("THANK YOU\n");

writeToFile(head); // Write final list to file before exiting

exit(0);

default:

printf("INVALID INPUT\n");

break;

}

}

}

// Function to create a new node

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if(n1 == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

// Function to delete the front node of the linked list

NODE delete\_front(NODE head) {

if (head == NULL) {

printf("List is empty. Deletion not possible.\n");

return head;

}

NODE temp = head;

while (temp->link != head) {

temp = temp->link;

}

temp->link = head->link;

NODE deleted\_node = head;

head = head->link;

free(deleted\_node);

return head;

}

// Function to display the linked list

NODE display(NODE head) {

if (head == NULL) {

printf("THE LIST IS EMPTY\n");

return head;

}

NODE temp = head;

do {

printf("%d\n", temp->data);

temp = temp->link;

} while (temp != head);

printf("\n");

return head;

}

// Function to insert a node at the front of the linked list

NODE insert\_front(NODE head) {

NODE temp = create\_node();

if (head == NULL) {

head = temp;

head->link = head; // Circular linked list, pointing to itself

} else {

NODE last = head;

while (last->link != head) {

last = last->link;

}

last->link = temp;

temp->link = head;

head = temp;

}

return head;

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing\n");

return;

}

NODE temp = head;

do {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

} while (temp != head);

fclose(file);

printf("Data written to file successfully\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

head->link = head; // Circular linked list, pointing to itself

} else {

NODE last = head;

while (last->link != head) {

last = last->link;

}

last->link = newNode;

newNode->link = head;

}

}

fclose(file);

printf("Data read from file successfully\n");

  return head;

}

exit(1);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

// Function to delete the front node of the linked list

NODE delete\_front(NODE head) {

if (head == NULL) {

printf("List is empty. Deletion not possible.\n");

return head;

}

NODE temp = head;

while (temp->link != head) {

temp = temp->link;

}

temp->link = head->link;

NODE deleted\_node = head;

head = head->link;

free(deleted\_node);

return head;

}

// Function to display the linked list

NODE display(NODE head) {

if (head == NULL) {

printf("THE LIST IS EMPTY\n");

return head;

}

NODE temp = head;

do {

printf("%d\n", temp->data);

temp = temp->link;

} while (temp != head);

printf("\n");

return head;

}

// Function to insert a node at the front of the linked list

NODE insert\_front(NODE head) {

NODE temp = create\_node();

if (head == NULL) {

head = temp;

head->link = head; // Circular linked list, pointing to itself

} else {

NODE last = head;

while (last->link != head) {

last = last->link;

}

last->link = temp;

temp->link = head;

head = temp;

}

return head;

}

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing\n");

return;

}

NODE temp = head;

do {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

} while (temp != head);

fclose(file);

printf("Data written to file successfully\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

head->link = head; // Circular linked list, pointing to itself

} else {

NODE last = head;

while (last->link != head) {

last = last->link;

}

last->link = newNode;

newNode->link = head;

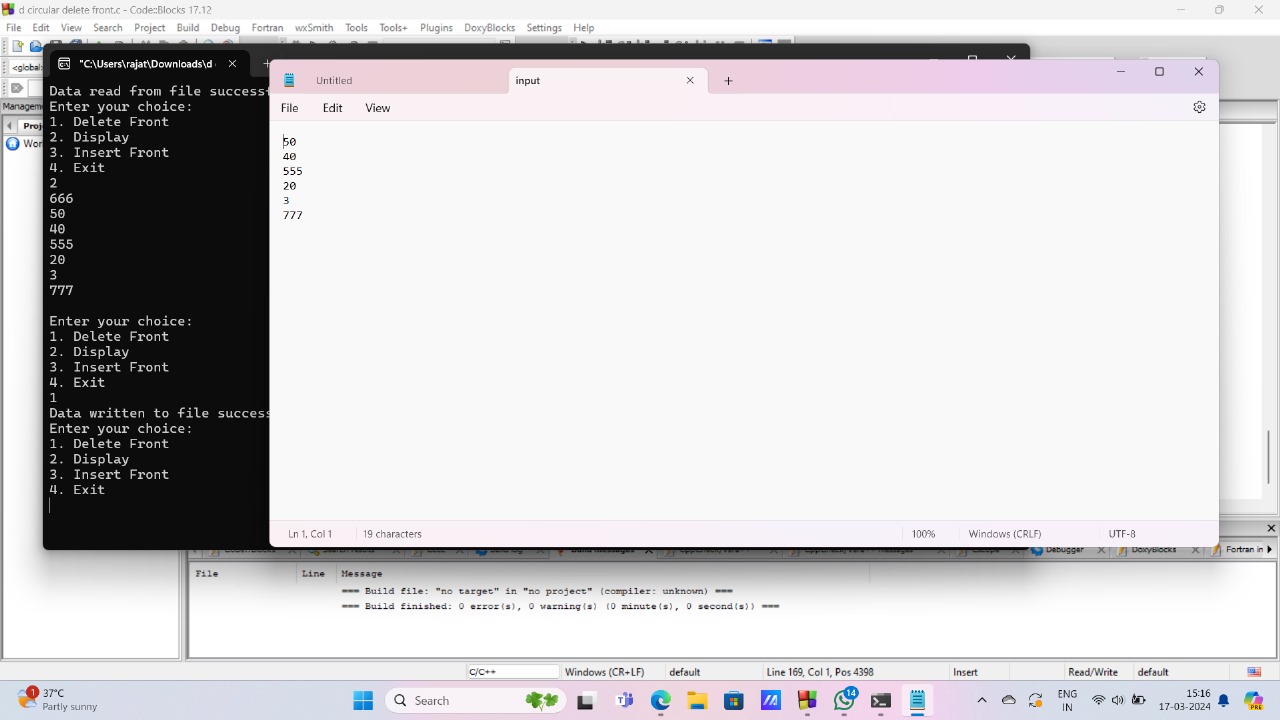
}

}

fclose(file);

printf("Data read from file successfully\n");

  return head;

} 

17. doubly circular delete end

#include<stdio.h>

#include<stdlib.h>

// Structure for a node in the linked list

struct node {

int data;

struct node \*link;

};

typedef struct node \*NODE;

// Function prototypes

NODE create\_node();

NODE insert\_front(NODE head);

NODE delete\_end(NODE head);

void display(NODE head);

void writeToFile(NODE head);

NODE readFromFile();

int main() {

NODE head = readFromFile(); // Read data from file if it exists

int choice;

while(1) {

printf("Enter your choice:\n");

printf("1. Insert Front\n");

printf("2. Display\n");

printf("3. Delete End\n");

printf("4. Exit\n");

scanf("%d", &choice);

switch(choice) {

case 1:

head = insert\_front(head);

writeToFile(head); // Write updated list to file

break;

case 2:

display(head);

break;

case 3:

head = delete\_end(head);

writeToFile(head); // Write updated list to file

break;

case 4:

printf("THANK YOU\n");

writeToFile(head); // Write final list to file before exiting

exit(0);

default:

printf("INVALID INPUT\n");

break;

}

}

}

// Function to create a new node

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if(n1 == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

// Function to insert a node at the front of the linked list

NODE insert\_front(NODE head) {

NODE temp;

temp = create\_node();

if(head == NULL) {

head = temp;

} else {

temp->link = head;

head = temp;

}

return head;

}

// Function to display the linked list

void display(NODE head) {

NODE temp;

temp = head;

if(temp == NULL) {

printf("THE LIST IS EMPTY\n");

} else {

while(temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

}

printf("\n");

}

NODE delete\_end(NODE head)

{

NODE temp=head, p = NULL;

if(temp == NULL) {

printf("List is empty\n");

return head;

}

while(temp->link != NULL) {

p = temp;

temp = temp->link;

}

if(p == NULL) {

// Only one node in the list

free(temp);

return NULL;

} else {

p->link = NULL;

free(temp);

return head;

}

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

}

}

fclose(file);

printf("Data read from file successfully\n");

  return head;

}

exit(1);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

// Function to insert a node at the front of the linked list

NODE insert\_front(NODE head) {

NODE temp;

temp = create\_node();

if(head == NULL) {

head = temp;

} else {

temp->link = head;

head = temp;

}

return head;

}

// Function to display the linked list

void display(NODE head) {

NODE temp;

temp = head;

if(temp == NULL) {

printf("THE LIST IS EMPTY\n");

} else {

while(temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

}

printf("\n");

}

NODE delete\_end(NODE head)

{

NODE temp=head, p = NULL;

if(temp == NULL) {

printf("List is empty\n");

return head;

}

while(temp->link != NULL) {

p = temp;

temp = temp->link;

}

if(p == NULL) {

// Only one node in the list

free(temp);

return NULL;

} else {

p->link = NULL;

free(temp);

return head;

}

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing\n");

return;

}

NODE temp = head;

while (temp != NULL) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

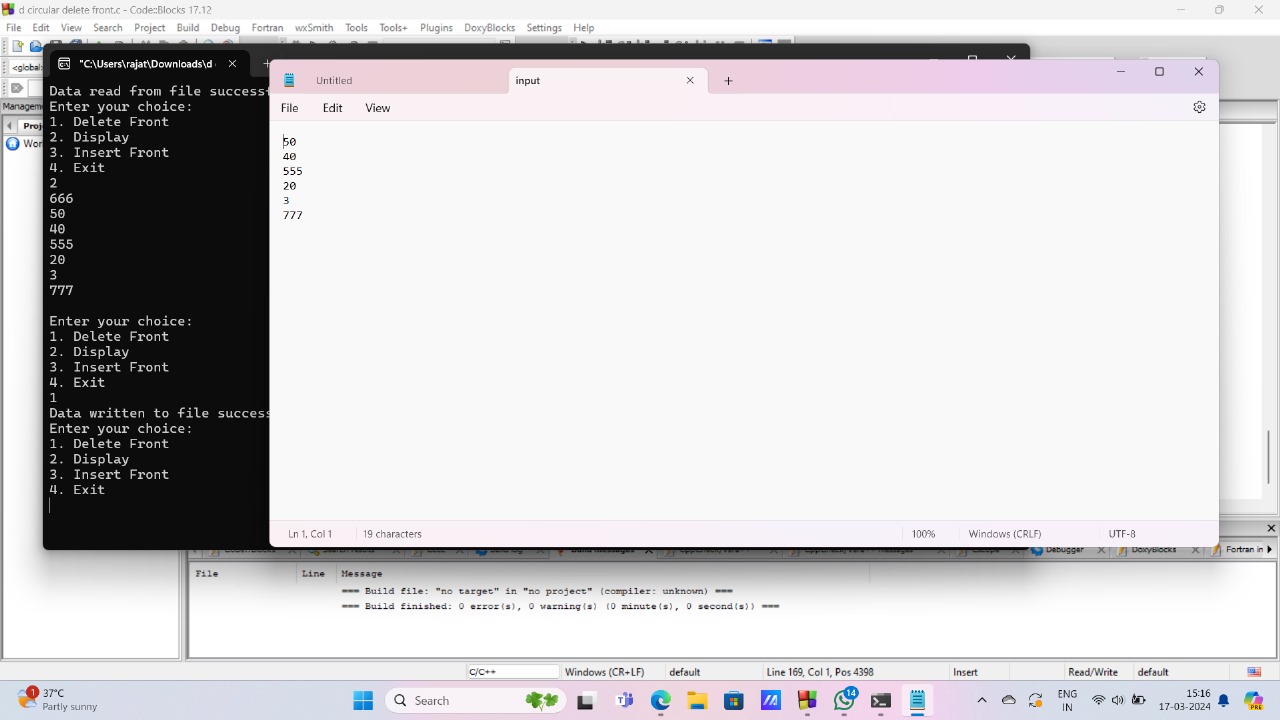
}

}

fclose(file);

printf("Data read from file successfully\n");

  return head;

} 

18. singly insert front

#include <stdio.h>

#include <stdlib.h>

// Define a structure for a node in the linked list

struct node {

int data;

struct node \*link;

};

// Define a pointer type for the node structure

typedef struct node \*NODE;

// Function prototypes

NODE create\_node();

NODE insert\_front(NODE head);

void display(NODE head);

void writeToFile(NODE head);

NODE readFromFile();

int main() {

// Read the linked list from the file (if exists)

NODE head = readFromFile();

if (head == NULL) {

printf("Failed to read from file. Starting with an empty list.\n");

}

int choice;

// Menu-driven program

while (1) {

printf("Enter your choice:\n");

printf("1. Insert Front\n");

printf("2. Display\n");

printf("3. Exit\n");

scanf("%d", &choice);

switch (choice) {

case 1:

head = insert\_front(head); // Insert element at front

writeToFile(head); // Write linked list to file

break;

case 2:

display(head); // Display the linked list

break;

case 3:

writeToFile(head); // Write linked list to file before exiting

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid input!\n");

break;

}

}return 0;

}

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node)); // Allocate memory for the node

if (n1 == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data); // Read data for the node

n1->link = NULL; // Set the link to NULL

return n1;

}

// Function to insert a node at the front of the linked list

NODE insert\_front(NODE head)

{

NODE temp,p;

temp=create\_node();

if(head==NULL)

{

temp->link=temp;

head=temp;

}

else

{

p=head;

while(p->link!=head)

{

p=p->link;

}

p->link=temp;

temp->link=head;

head=temp;

}

return head;

}

// Function to display the linked list

void display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n"); // If list is empty, print a message

return;

}

NODE temp = head;

while (temp != NULL) {

printf("%d\n", temp->data); // Print data of each node

temp = temp->link; // Move to the next node

}

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w"); // Open the file for writing

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data); // Write data of each node to file

temp = temp->link; // Move to the next node

}

fclose(file); // Close the file

printf("Data written to file successfully.\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r"); // Open the file for reading

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node)); // Allocate memory for a new node

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

newNode->data = value; // Set data of the new node

newNode->link = NULL; // Set link to NULL

if (head == NULL) {

head = newNode; // If list is empty, new node becomes head

} else {

newNode->link = head; // Set link of new node to current head

head = newNode; // Update head to point to the new node

}

}

fclose(file); // Close the file

printf("Data read from file successfully.\n");

  return head;

}

}

printf("Enter the number: ");

scanf("%d", &n1->data); // Read data for the node

n1->link = NULL; // Set the link to NULL

return n1;

}

// Function to insert a node at the front of the linked list

NODE insert\_front(NODE head)

{

NODE temp,p;

temp=create\_node();

if(head==NULL)

{

temp->link=temp;

head=temp;

}

else

{

p=head;

while(p->link!=head)

{

p=p->link;

}

p->link=temp;

temp->link=head;

head=temp;

}

return head;}

void display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n"); // If list is empty, print a message

return;

}

NODE temp = head;

while (temp != NULL) {

printf("%d\n", temp->data); // Print data of each node

temp = temp->link; // Move to the next node

}

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w"); // Open the file for writing

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data); // Write data of each node to file

temp = temp->link; // Move to the next node

}

fclose(file); // Close the file

printf("Data written to file successfully.\n");

}

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r"); // Open the file for reading

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node)); // Allocate memory for a new node

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

newNode->data = value; // Set data of the new node

newNode->link = NULL; // Set link to NULL

if (head == NULL) {

head = newNode; // If list is empty, new node becomes head

} else {

newNode->link = head; // Set link of new node to current head

head = newNode; // Update head to point to the new node

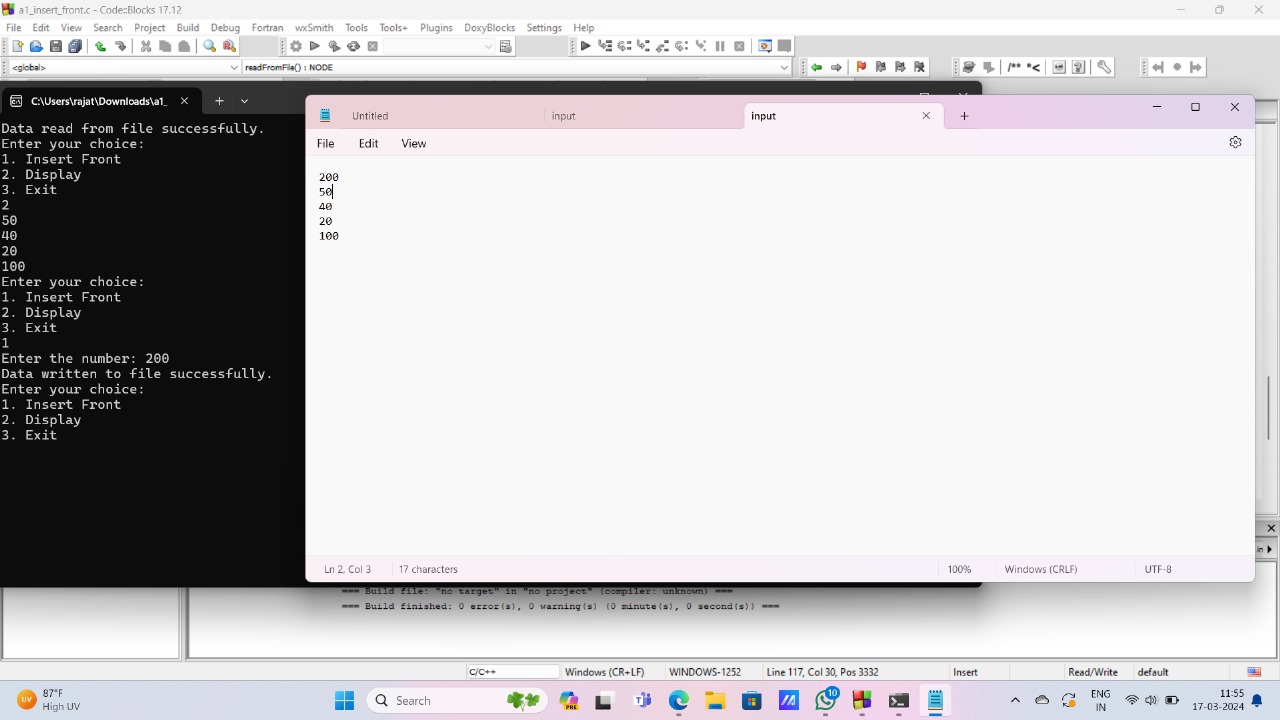
}

}

fclose(file); // Close the file

printf("Data read from file successfully.\n");

  return head;

} 

19. singly insert end

#include <stdio.h>

#include <stdlib.h>

int data; struct node \*link;

};

typedef struct node \*NODE;

NODE create\_node();

NODE insert\_end(NODE head);

NODE display(NODE head);

void writeToFile(NODE head);

NODE readFromFile();

int main() {

NODE head = readFromFile();

int choice;

while (1) {

printf("Enter your choice:\n");

printf("1. Insert End\n");

printf("2. Display\n");

printf("3. Exit\n");

scanf("%d", &choice);

switch (choice) {

case 1:

head = insert\_end(head);

writeToFile(head);

break;

case 2:

display(head);

break;

case 3:

writeToFile(head);

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid input!\n");

break;

}

} return 0;

}

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if (n1 == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;}

NODE insert\_end(NODE head) {

NODE temp, p;

temp = create\_node();

if (head == NULL) {

head = temp;

} else {

p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = temp;

}

return head;

}

NODE display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n");

return head;

}

NODE temp = head;

while (temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

return head;

}

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully.\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

}

}

fclose(file);

printf("Data read from file successfully.\n");

  return head;

}

}return head;

}

NODE display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n");

return head;

}

NODE temp = head;

while (temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

return head;

}

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully.\n");

}

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

}

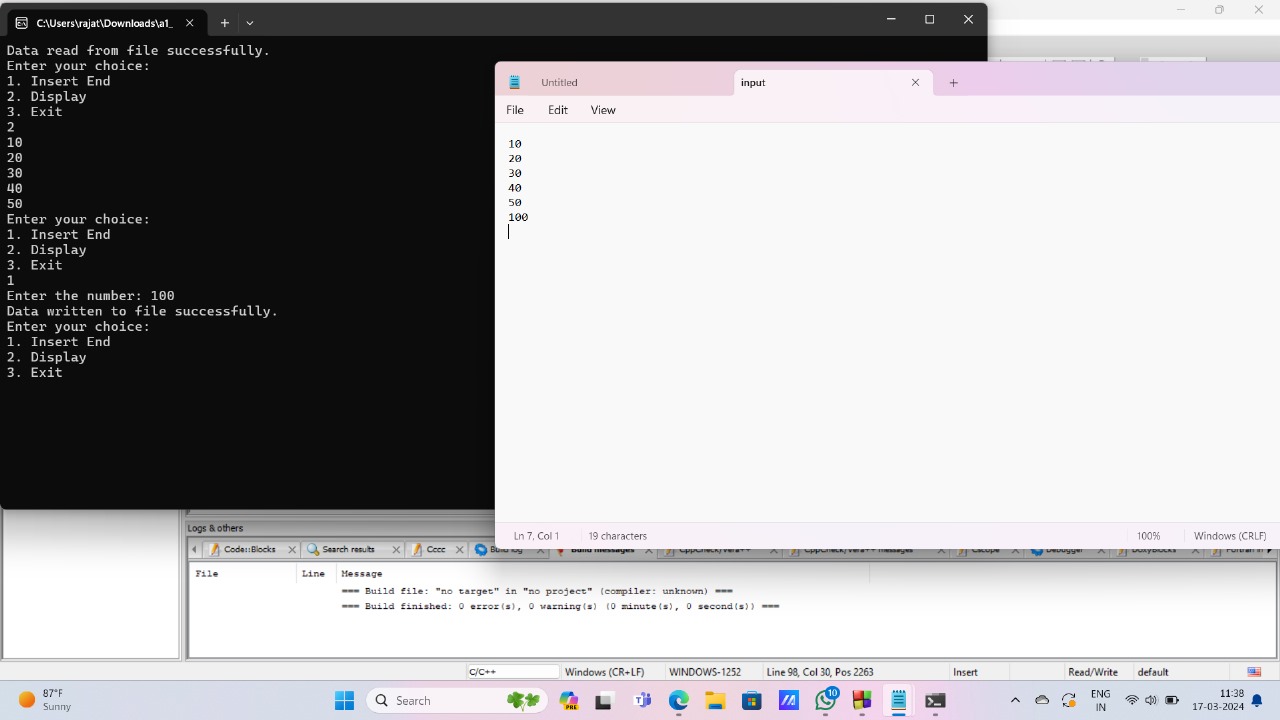
}

fclose(file);

printf("Data read from file successfully.\n");

  return head;

}



20. singly delete end

#include<stdio.h>

#include<stdlib.h>

// Structure for a node in the linked list

struct node {

int data;

struct node \*link;

};

typedef struct node \*NODE;

// Function prototypes

NODE create\_node();

NODE insert\_front(NODE head);

NODE delete\_end(NODE head);

NODE display(NODE head);

void writeToFile(NODE head);

NODE readFromFile();

int main() {

NODE head = readFromFile(); // Read data from file if it exists

int choice;

while(1) {

printf("Enter your choice:\n");

printf("1. Insert Front\n");

printf("2. Display\n");

printf("3. Delete End\n");

printf("4. Exit\n");

scanf("%d", &choice);

switch(choice) {

case 1:

head = insert\_front(head);

writeToFile(head); // Write updated list to file

break;

case 2:

display(head);

break;

case 3:

head = delete\_end(head);

writeToFile(head); // Write updated list to file

break;

case 4:

printf("THANK YOU\n");

writeToFile(head); // Write final list to file before exiting

exit(0);

default:

printf("INVALID INPUT\n");

break;

}

}

}

// Function to create a new node

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if(n1 == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

// Function to insert a node at the front of the linked list

NODE insert\_front(NODE head) {

NODE temp;

temp = create\_node();

if(head == NULL) {

head = temp;

} else {

temp->link = head;

head = temp;

}

return head;

}

// Function to display the linked list

NODE display(NODE head) {

NODE temp;

temp = head;

if(temp == NULL) {

printf("THE LIST IS EMPTY\n");

} else {

while(temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

}

printf("\n");

return head;

}

// Function to delete the node at the end of the linked list

NODE delete\_end(NODE head) {

NODE temp, p;

temp = head;

if(temp == NULL) {

printf("List is empty\n");

} else if(temp->link == NULL) {

head = NULL;

printf("Deleted = %d\n", temp->data);

free(temp);

} else {

while(temp->link != NULL) {

p = temp;

temp = temp->link;

}

p->link = NULL;

printf("Deleted = %d\n", temp->data);

free(temp);

}

return head;

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

}

}

fclose(file);

printf("Data read from file successfully\n");

  return head;

}

break;

case 4:

printf("THANK YOU\n");

writeToFile(head); // Write final list to file before exiting

exit(0);

default:

printf("INVALID INPUT\n");

break;

}

}

}

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if(n1 == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

NODE insert\_front(NODE head) {

NODE temp;

temp = create\_node();

if(head == NULL) {

head = temp;

} else {

temp->link = head;

head = temp; } else {

temp->link = head;

head = temp;

}

return head;

}

NODE display(NODE head) {

NODE temp;

temp = head;

if(temp == NULL) {

printf("THE LIST IS EMPTY\n");

} else {

while(temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

}

printf("\n");

return head;

}

NODE delete\_end(NODE head) {

NODE temp, p;

temp = head;

if(temp == NULL) {

printf("List is empty\n");

} else if(temp->link == NULL) {

head = NULL;

printf("Deleted = %d\n", temp->data);

free(temp);

} else {

while(temp->link != NULL) {

p = temp;

temp = temp->link;

}

p->link = NULL;

printf("Deleted = %d\n", temp->data);

free(temp);

}

return head;

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading\n");

return NULL;

}int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

}

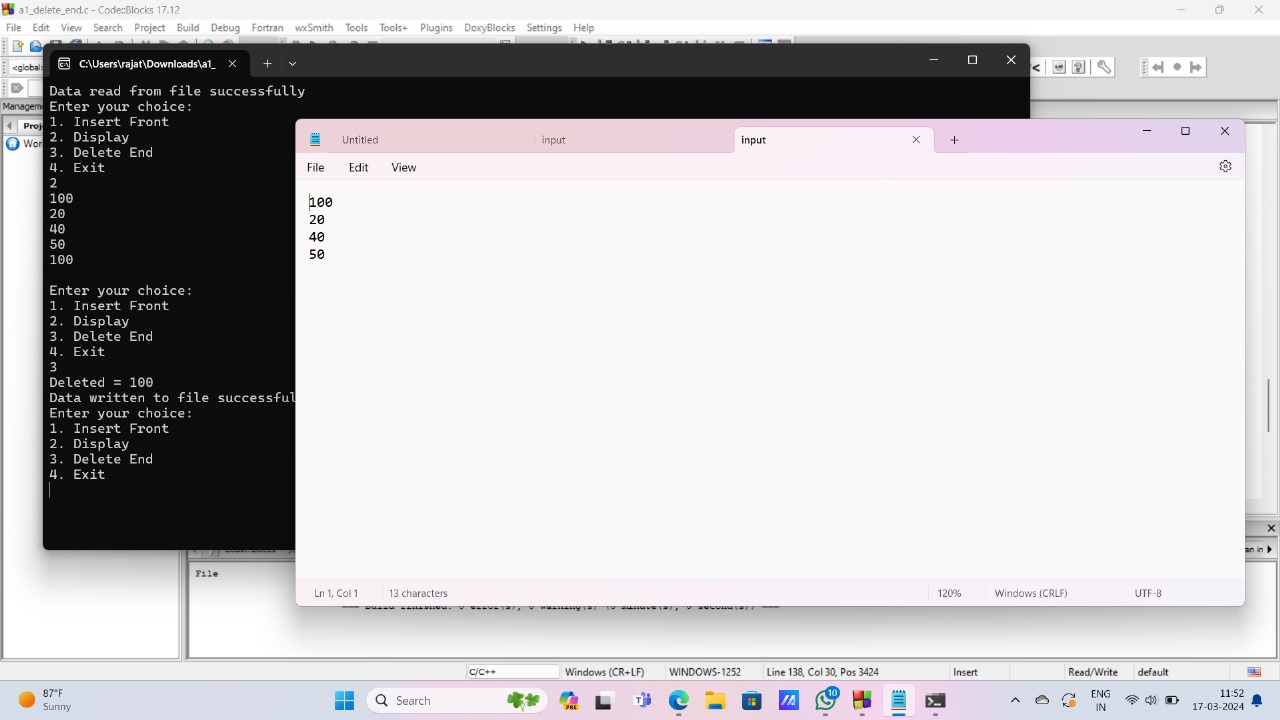
}

fclose(file);

printf("Data read from file successfully\n");

  return head;

}



21.singly delete front

#include<stdio.h>

#include<stdlib.h>

struct node {

int data;

struct node \*link;

};

typedef struct node \*NODE;

NODE create\_node();

NODE insert\_front(NODE head);

NODE delete\_front(NODE head);

NODE display(NODE head);

void writeToFile(NODE head);

NODE readFromFile();

int main() {

NODE head = readFromFile(); // Read data from file if it exists int choice;

while(1) {

printf("Enter your choice:\n");

printf("1. Delete Front\n");

printf("2. Display\n");

printf("3. Insert Front\n");

printf("4. Exit\n");

scanf("%d", &choice);

switch(choice) {

case 1:

head = delete\_front(head);

writeToFile(head); // Write updated list to file

break;

case 2:

display(head);

break;

case 3:

head = insert\_front(head);

writeToFile(head); // Write updated list to file

break;

case 4:

printf("THANK YOU\n");

writeToFile(head); // Write final list to file before exiting

exit(0);

default:

printf("INVALID INPUT\n");

break;

}

}

}

// Function to create a new node

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if(n1 == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

// Function to delete the node at the front of the linked list

NODE delete\_front(NODE head) {

NODE temp;

temp = head;

if(head == NULL) {

printf("List is empty\n");

} else {

head = temp->link;

printf("Deleted data = %d\n", temp->data);

free(temp);

}

return head;

}

// Function to display the linked list

NODE display(NODE head) {

NODE temp;

temp = head;

if(temp == NULL) {

printf("THE LIST IS EMPTY\n");

} else {

while(temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

}

printf("\n");

return head;

}

// Function to insert a node at the front of the linked list

NODE insert\_front(NODE head) {

NODE temp;

temp = create\_node();

if(head == NULL) {

head = temp;

} else {

temp->link = head;

head = temp;

}

return head;

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

}

}

fclose(file);

printf("Data read from file successfully\n");

  return head;

}

case 4:

printf("THANK YOU\n");

writeToFile(head); // Write final list to file before exiting

exit(0);

default:

printf("INVALID INPUT\n");

break;

}

}

}

// Function to create a new node

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if(n1 == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

// Function to delete the node at the front of the linked list

NODE delete\_front(NODE head) {

NODE temp;

temp = head;

if(head == NULL) {

printf("List is empty\n");

} else {

head = temp->link;

printf("Deleted data = %d\n", temp->data);

free(temp);

}

return head;

}

NODE display(NODE head) {

NODE temp;

temp = head;

if(temp == NULL) {

printf("THE LIST IS EMPTY\n");

} else {

while(temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

}

printf("\n");

return head;

}

NODE insert\_front(NODE head) {

NODE temp;

temp = create\_node();

if(head == NULL) {

head = temp;

} else {

temp->link = head;

head = temp;

}

return head;

}

head = temp;

} else {

temp->link = head;

head = temp;

}

return head;

}void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully\n");

}NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) { NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

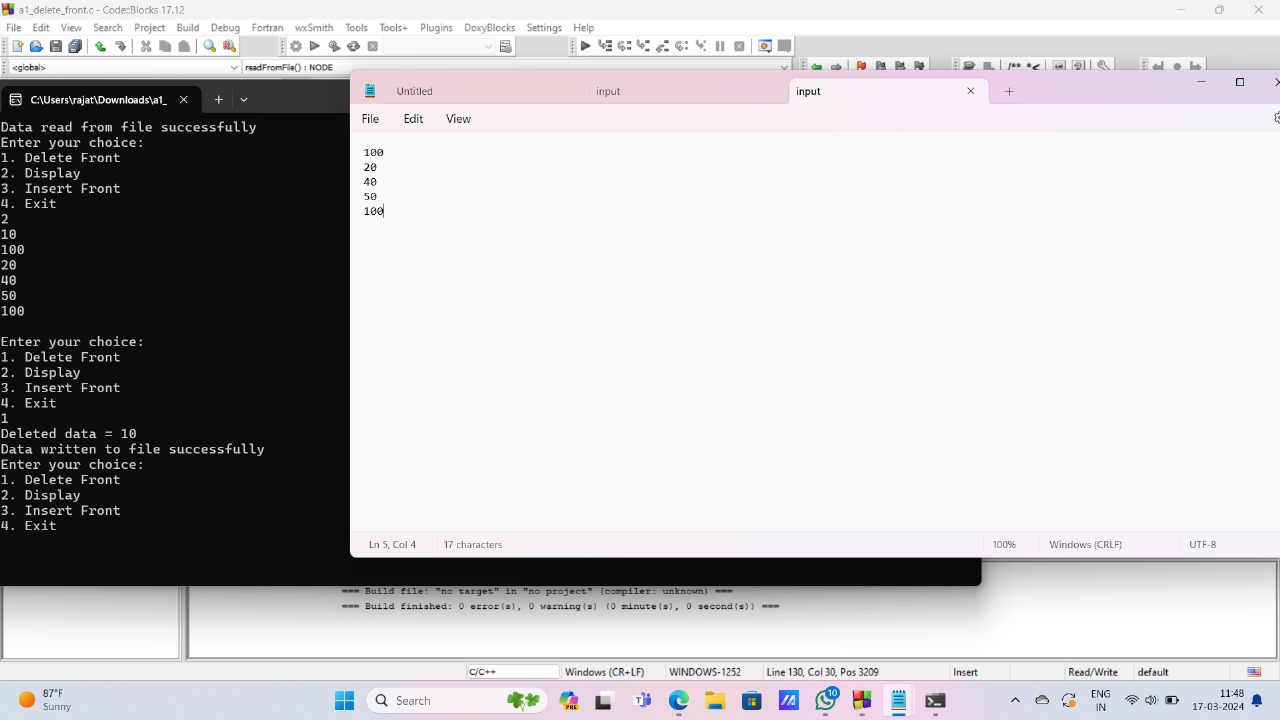
}

}

fclose(file);

printf("Data read from file successfully\n");

  return head;

} 

22.singly delete position

#include<stdio.h>

#include<stdlib.h>

struct node {

int data;

struct node \*link;

};

typedef struct node \*NODE;

NODE create\_node();

NODE insert\_front(NODE head);

NODE delete\_position(NODE head);

NODE display(NODE head);

void writeToFile(NODE head);

NODE readFromFile();

int count(NODE head);

int main() {

NODE head = readFromFile(); // Read data from file if it exists

int choice;

while(1) {

printf("Enter your choice:\n");

printf("1. Insert Front\n");

printf("2. Display\n");

printf("3. Delete Position\n");

printf("4. Count\n");

printf("5. Exit\n");

scanf("%d", &choice);

switch(choice) {

case 1:

head = insert\_front(head);

writeToFile(head); // Write updated list to file

break;

case 2:

display(head);

break;

case 3:

head = delete\_position(head);

writeToFile(head); // Write updated list to file

break;

case 4:

printf("Count = %d\n", count(head));

break;

case 5:

printf("THANK YOU\n");

writeToFile(head); // Write final list to file before exiting

exit(0);

default:

printf("INVALID INPUT\n");

break;

}

}

}

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if(n1 == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

// Function to insert a node at the front of the linked list

NODE insert\_front(NODE head) {

NODE temp;

temp = create\_node();

if(head == NULL) {

head = temp;

} else {

temp->link = head;

head = temp;

}

return head;

}

// Function to display the linked list

NODE display(NODE head) {

NODE temp;

temp = head;

if(temp == NULL) {

printf("THE LIST IS EMPTY\n");

} else {

while(temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

}

printf("\n");

return head;

}

// Function to delete a node at a given position in the linked list

NODE delete\_position(NODE head) {

NODE temp = head, p;

int pos, i = 1;

printf("Enter the position: ");

scanf("%d", &pos);

int cnt = count(head);

if(pos <= 0 || pos > cnt) {

printf("INVALID POSITION\n");

return head;

}

if(temp == NULL) {

printf("List is empty\n");

return head;

}

if(pos == 1) {

head = temp->link;

printf("Deleted = %d\n", temp->data);

free(temp);

return head;

}

while(i < pos - 1) {

temp = temp->link;

i++;

}

p = temp->link;

temp->link = p->link;

printf("Deleted = %d\n", p->data);

free(p);

return head;

}

// Function to count the number of nodes in the linked list

int count(NODE head) {

NODE temp;

temp = head;

int cnt = 0;

while(temp != NULL) {

cnt++;

temp = temp->link;

}

return cnt;

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

}

}

fclose(file);

printf("Data read from file successfully\n");

  return head;

}

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

NODE insert\_front(NODE head) {

NODE temp;

temp = create\_node();

if(head == NULL) {

head = temp;

} else {

temp->link = head;

head = temp;

}

return head;

}

NODE display(NODE head) {

NODE temp;

temp = head;

if(temp == NULL) {

printf("THE LIST IS EMPTY\n");

} else {

while(temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

}

printf("\n");

return head;

}

NODE delete\_position(NODE head) {

NODE temp = head, p;

int pos, i = 1;

printf("Enter the position: ");

scanf("%d", &pos);

int cnt = count(head);

if(pos <= 0 || pos > cnt) {

printf("INVALID POSITION\n");

return head;

}

if(temp == NULL) {

printf("List is empty\n");

return head;

}

if(pos == 1) {

head = temp->link;

printf("Deleted = %d\n", temp->data);

free(temp);

return head;

}

while(i < pos - 1) {

temp = temp->link;

i++;

}

p = temp->link;

temp->link = p->link;

printf("Deleted = %d\n", p->data);

free(p);

return head;

}

// Function to count the number of nodes in the linked list

int count(NODE head) {

NODE temp;

temp = head;

int cnt = 0;

int count(NODE head) {

NODE temp;

temp = head;

int cnt = 0;

while(temp != NULL) {

cnt++;

temp = temp->link;

}

return cnt;

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

}

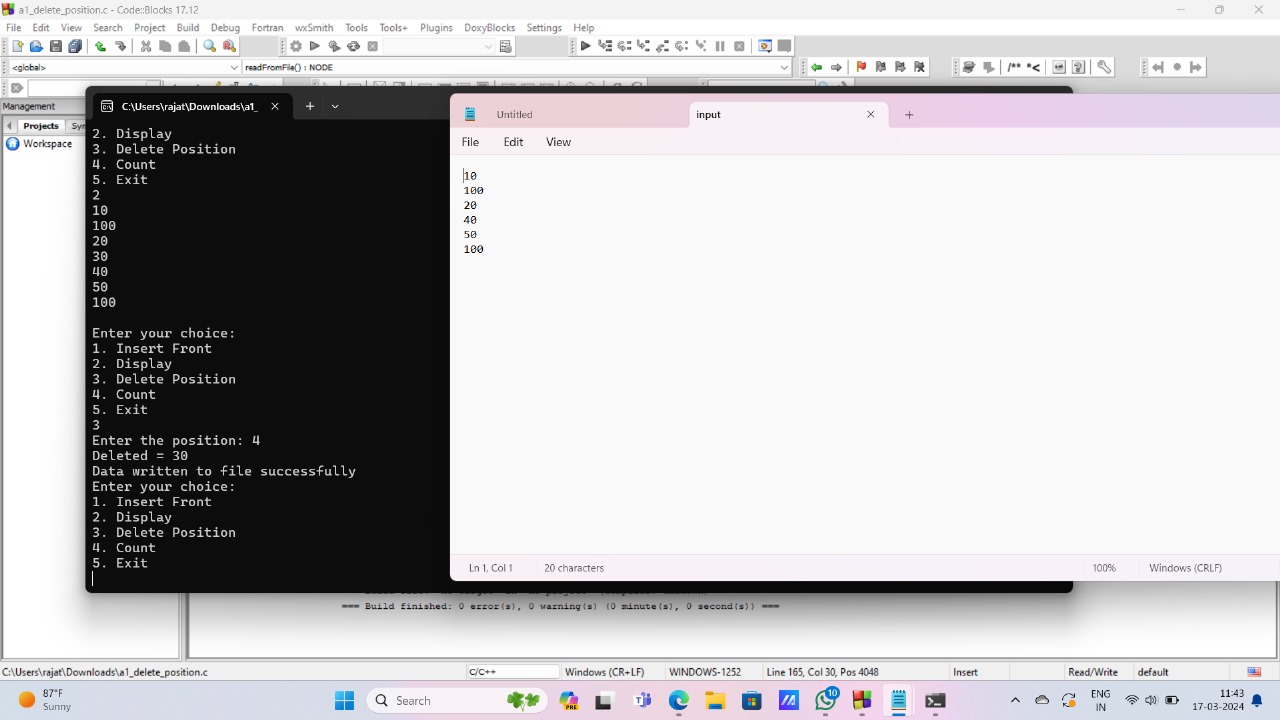
}

fclose(file);

printf("Data read from file successfully\n");

  return head;

}



23.singly insert position

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*link;

};

typedef struct node \*NODE;

NODE create\_node();

NODE display(NODE head);

int count(NODE head);

NODE insert\_position(NODE head);

void writeToFile(NODE head);

NODE readFromFile();

int main() {

NODE head = readFromFile(); // Read data from file if it exists

int choice;

while (1) {

printf("Enter your choice:\n");

printf("1. Display\n");

printf("2. Insert at Position\n");

printf("3. Count Nodes\n");

printf("4. Exit\n");

scanf("%d", &choice);

switch (choice) {

case 1:

display(head);

break;

case 2:

head = insert\_position(head);

writeToFile(head); // Write updated list to file

break;

case 3:

printf("Number of nodes: %d\n", count(head));

break;

case 4:

writeToFile(head); // Write final list to file before exiting

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid input!\n");

break;

}

}

return 0;

}

// Function to create a new node

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if (n1 == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

// Function to display the linked list

NODE display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n");

return head;

}

NODE temp = head;

while (temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

return head;

}

// Function to count the number of nodes in the linked list

int count(NODE head) {

int cnt = 0;

NODE temp = head;

while (temp != NULL) {

cnt++;

temp = temp->link;

}

return cnt;

}

// Function to insert a node at a specified position in the linked list

NODE insert\_position(NODE head) {

int pos, cnt, i = 1;

printf("Enter the position: ");

scanf("%d", &pos);

NODE temp, p;

temp = create\_node();

cnt = count(head);

if (pos <= 0 || pos > cnt + 1) {

printf("INVALID POSITION\n");

free(temp);

return head;

} else if (head == NULL && pos == 1) {

head = temp;

} else if (pos == 1) {

temp->link = head;

head = temp;

} else {

p = head;

while (i < pos - 1 && p != NULL) {

p = p->link;

i++;

}

temp->link = p->link;

p->link = temp;

}

return head;

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully.\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

}

}

fclose(file);

printf("Data read from file successfully.\n");

  return head;

}

printf("Number of nodes: %d\n", count(head));

break;

case 4:

writeToFile(head); // Write final list to file before exiting

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid input!\n");

break;

}

}

return 0;

}

// Function to create a new node

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if (n1 == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

// Function to display the linked list

NODE display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n");

return head;

}

NODE temp = head;

while (temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

return head;

}

// Function to count the number of nodes in the linked list

int count(NODE head) {

int cnt = 0;

NODE temp = head;

while (temp != NULL) {

cnt++;

temp = temp->link;

}

return cnt;

}

// Function to insert a node at a specified position in the linked list

NODE insert\_position(NODE head) {

int pos, cnt, i = 1;

printf("Enter the position: ");

scanf("%d", &pos);

NODE temp, p;

temp = create\_node();

cnt = count(head);

if (pos <= 0 || pos > cnt + 1) {

printf("INVALID POSITION\n");

free(temp);

return head;

} else if (head == NULL && pos == 1) {

head = temp;

} else if (pos == 1) {

NODE temp, p;

temp = create\_node();

cnt = count(head);

if (pos <= 0 || pos > cnt + 1) {

printf("INVALID POSITION\n");

free(temp);

return head;

} else if (head == NULL && pos == 1) {

head = temp;

} else if (pos == 1) {

temp->link = head;

head = temp;

} else {

p = head;

while (i < pos - 1 && p != NULL) {

p = p->link;

i++;

}

temp->link = p->link;

p->link = temp;

}

return head;

}

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully.\n");

}

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

}

}

fclose(file);

printf("Data read from file successfully.\n");

  return head;

}

24.doubly singly insert front

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*link;

};

typedef struct node \*NODE;

NODE create\_node();

NODE insert\_front(NODE head);

void display(NODE head);

void writeToFile(NODE head);

NODE readFromFile();

int main()

NODE head = readFromFile();

if (head == NULL) {

printf("Failed to read from file. Starting with an empty list.\n");

} int choice;

while (1) {

printf("Enter your choice:\n");

printf("1. Insert Front\n");

printf("2. Display\n");

printf("3. Exit\n");

scanf("%d", &choice);

switch (choice) {

case 1:

head = insert\_front(head);

writeToFile(head);

break;

case 2:

display(head);

break;

case 3:

writeToFile(head);

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid input!\n");

break;

}

}

return 0;

}

// Function to create a new node

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node)); // Allocate memory for the node

if (n1 == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data); // Read data for the node

n1->link = NULL; // Set the link to NULL

return n1;

}

// Function to insert a node at the front of the linked list

NODE insert\_front(NODE head) {

NODE temp;

temp = create\_node();

if (head == NULL) {

head = temp;

} else {

temp->link = head;

head = temp;

}

return head;

}

// Function to display the linked list

void display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n"); // If list is empty, print a message

return;

}

NODE temp = head;

while (temp != NULL) {

printf("%d\n", temp->data); // Print data of each node

temp = temp->link; // Move to the next node

}

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w"); // Open the file for writing

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data); // Write data of each node to file

temp = temp->link; // Move to the next node

}

fclose(file); // Close the file

printf("Data written to file successfully.\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r"); // Open the file for reading

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node)); // Allocate memory for a new node

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

newNode->data = value; // Set data of the new node

newNode->link = NULL; // Set link to NULL

if (head == NULL) {

head = newNode; // If list is empty, new node becomes head

} else {

newNode->link = head; // Set link of new node to current head

head = newNode; // Update head to point to the new node

}

}

fclose(file); // Close the file

printf("Data read from file successfully.\n");

  return head;

}

}

p->link = newNode;

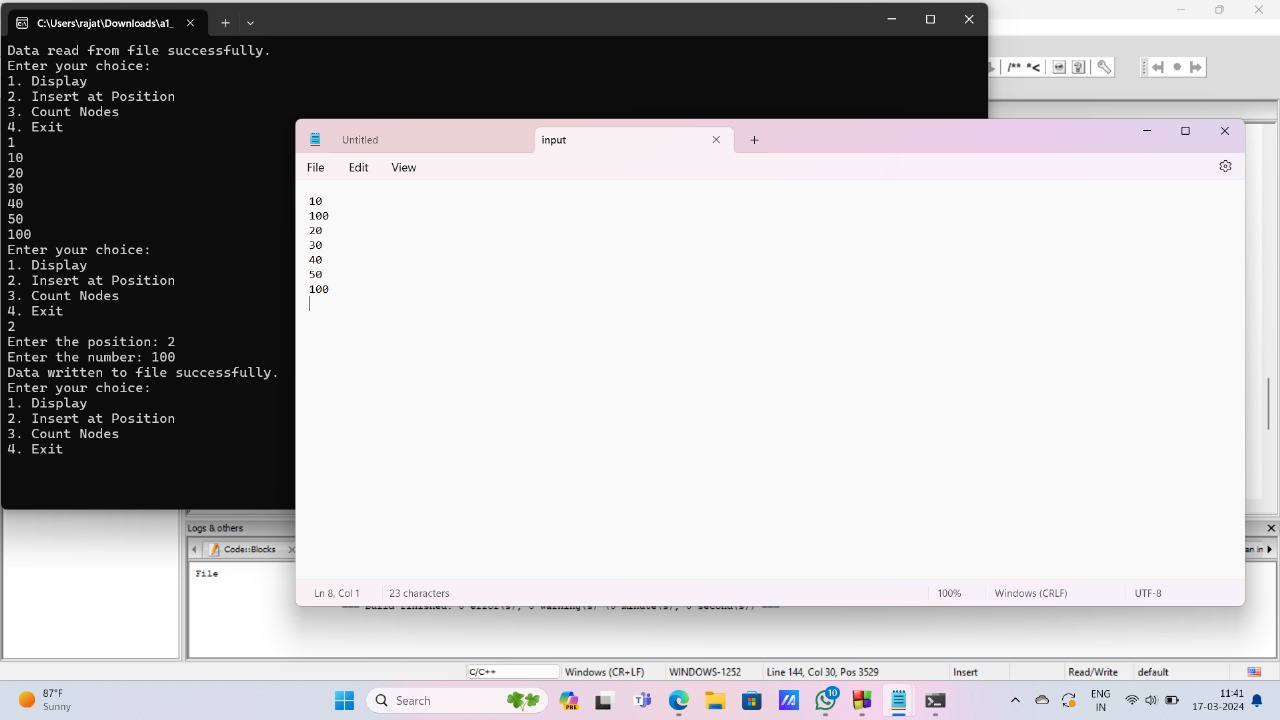
}

}

fclose(file);

printf("Data read from file successfully.\n");

  return head;

} 

case 2:

display(head);

break;

case 3:

writeToFile(head);

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid input!\n");

break;

}

}

return 0;

}

// Function to create a new node

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node)); // Allocate memory for the node

if (n1 == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data); // Read data for the node

n1->link = NULL; // Set the link to NULL

return n1;

}

// Function to insert a node at the front of the linked list

NODE insert\_front(NODE head) {

NODE temp;

temp = create\_node();

if (head == NULL) {

head = temp;

} else {

temp->link = head;

head = temp;

}

return head;

}

// Function to display the linked list

void display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n"); // If list is empty, print a message

return;

}

NODE temp = head;

while (temp != NULL) {

printf("%d\n", temp->data); // Print data of each node

temp = temp->link; // Move to the next node

}

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w"); // Open the file for writing

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data); // Write data of each node to file

temp = temp->link; // Move to the next node

}

fclose(file); // Close the file

printf("Data written to file successfully.\n");

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data); // Write data of each node to file

temp = temp->link; // Move to the next node

}

fclose(file); // Close the file

printf("Data written to file successfully.\n");

}

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r"); // Open the file for reading

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node)); // Allocate memory for a new node

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

newNode->data = value; // Set data of the new node

newNode->link = NULL; // Set link to NULL

if (head == NULL) {

head = newNode; // If list is empty, new node becomes head

} else {

newNode->link = head;

head = newNode; // Update head to point to the new node

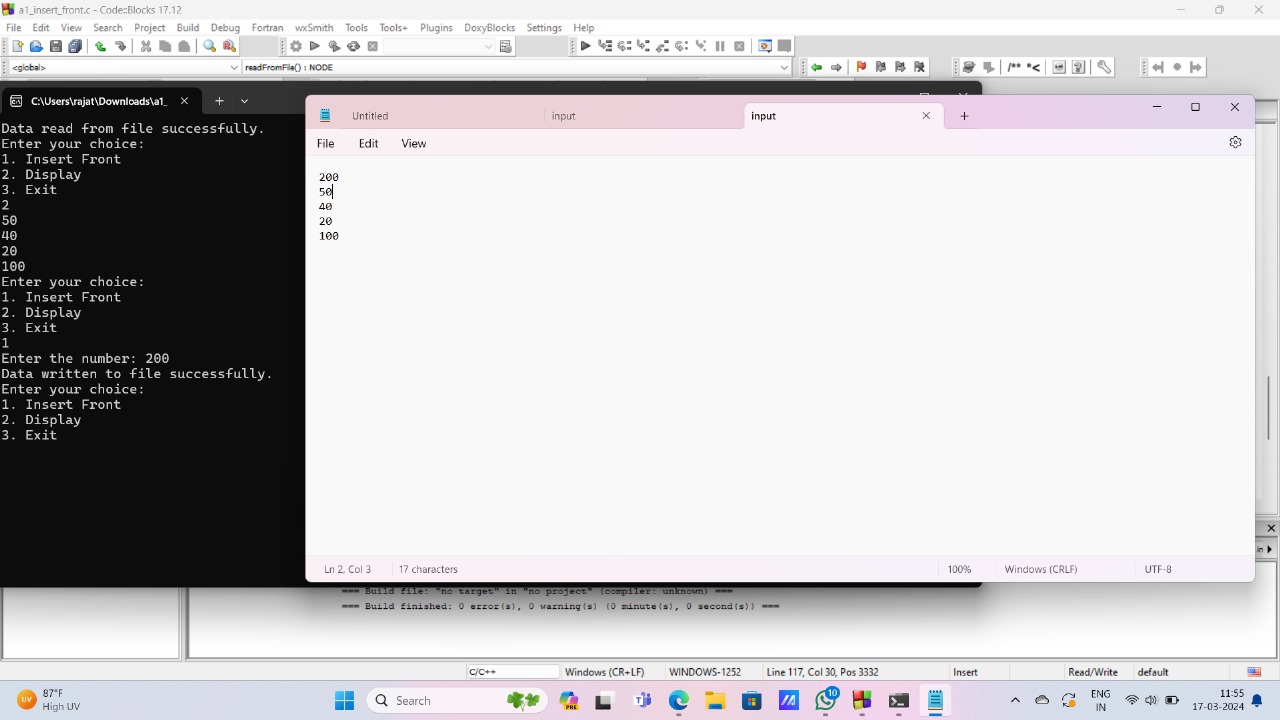
}

}

fclose(file); // Close the file

printf("Data read from file successfully.\n");

  return head;

} 

25.doubly singly insert end

#include <stdio.h>

#include <stdlib.h>

// Structure for a node in the linked list

struct node {

int data;

struct node \*link;

};

typedef struct node \*NODE;

// Function prototypes

NODE create\_node();

NODE insert\_end(NODE head);

void display(NODE head);

void writeToFile(NODE head);

NODE readFromFile();

int main() {

NODE head = readFromFile(); // Read data from file if it exists

int choice;

while (1) {

printf("Enter your choice:\n");

printf("1. Insert End\n");

printf("2. Display\n");

printf("3. Exit\n");

scanf("%d", &choice);

switch (choice) {

case 1:

head = insert\_end(head);

writeToFile(head); // Write updated list to file

break;

case 2:

display(head);

break;

case 3:

writeToFile(head); // Write final list to file before exiting

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid input!\n");

break;

}

}

return 0;

}

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if (n1 == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

// Function to insert a node at the end of the linked list

NODE insert\_end(NODE head) {

NODE temp, p;

temp = create\_node();

if (head == NULL) {

head = temp;

} else {

p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = temp;

}

return head;

}

// Function to display the linked list

void display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully.\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

}

}

fclose(file);

printf("Data read from file successfully.\n");

  return head;

}

n1->link = NULL;

return n1;

}

NODE insert\_end(NODE head) {

NODE temp, p;

temp = create\_node();

if (head == NULL) {

head = temp;

} else {

p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = temp;

}

return head;

}

void display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

}

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully.\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

}

}

fclose(file);

printf("Data read from file successfully.\n");

  return head;

}

26. doubly singly delete end

#include<stdio.h>

#include<stdlib.h>

// Structure for a node in the linked list

struct node {

int data;

struct node \*link;

};

typedef struct node \*NODE;

// Function prototypes

NODE create\_node();

NODE insert\_front(NODE head);

NODE delete\_end(NODE head);

void display(NODE head);

void writeToFile(NODE head);

NODE readFromFile();

int main() {

NODE head = readFromFile(); // Read data from file if it exists

int choice;

while(1) {

printf("Enter your choice:\n");

printf("1. Insert Front\n");

printf("2. Display\n");

printf("3. Delete End\n");

printf("4. Exit\n");

scanf("%d", &choice);

switch(choice) {

case 1:

head = insert\_front(head);

writeToFile(head); // Write updated list to file

break;

case 2:

display(head);

break;

case 3:

head = delete\_end(head);

writeToFile(head); // Write updated list to file

break;

case 4:

printf("THANK YOU\n");

writeToFile(head); // Write final list to file before exiting

exit(0);

default:

printf("INVALID INPUT\n");

break;

}

}

}

// Function to create a new node

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if(n1 == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

// Function to insert a node at the front of the linked list

NODE insert\_front(NODE head) {

NODE temp;

temp = create\_node();

if(head == NULL) {

head = temp;

} else {

temp->link = head;

head = temp;

}

return head;

}

// Function to display the linked list

void display(NODE head) {

NODE temp;

temp = head;

if(temp == NULL) {

printf("THE LIST IS EMPTY\n");

} else {

while(temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

}

printf("\n");

}

NODE delete\_end(NODE head)

{

NODE temp=head, prev=NULL;

if(temp==NULL)

{

printf("List is empty\n");

return head;

}

if(temp->link == NULL) // Only one node in the list

{

free(temp);

return NULL; // Updated head after deletion

}

while(temp->link != NULL)

{

prev = temp;

temp = temp->link;

}

prev->link = NULL;

free(temp);

return head;

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

}

}

fclose(file);

printf("Data read from file successfully\n");

  return head;

}

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

}

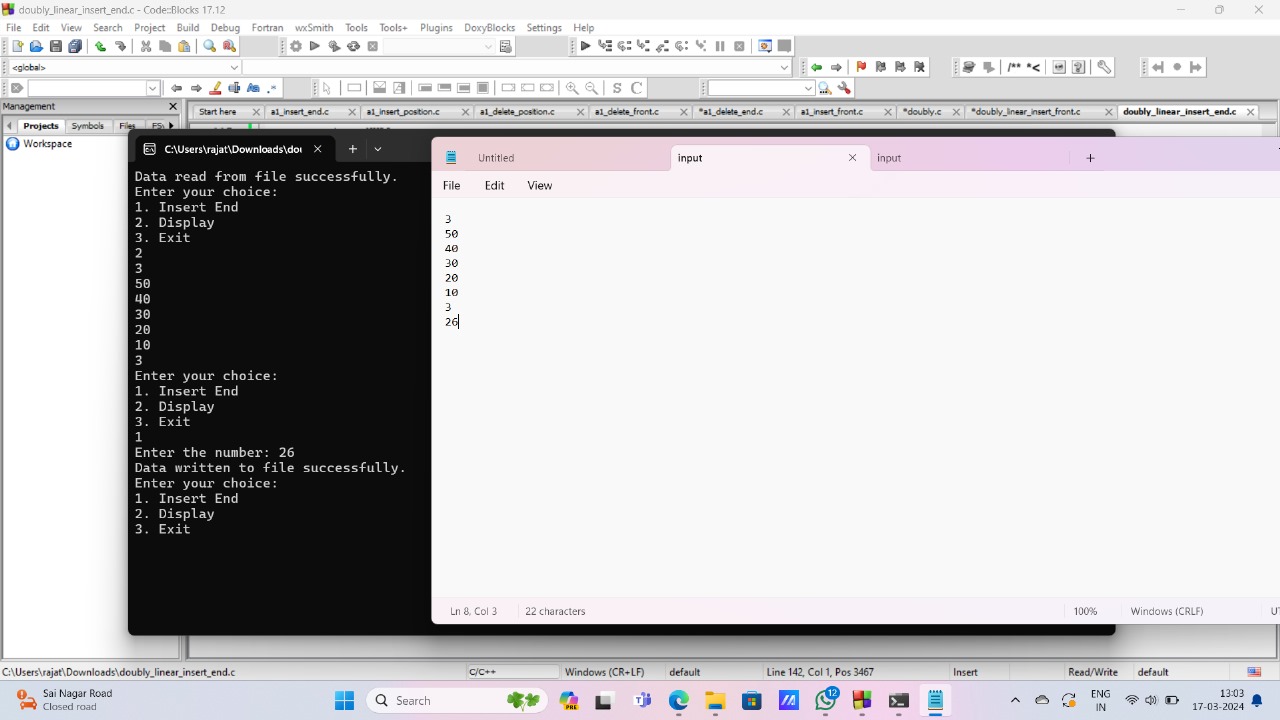
}

fclose(file);

printf("Data read from file successfully.\n");

  return head;

}



NODE readFromFile();

int main() {

NODE head = readFromFile(); // Read data from file if it exists

int choice;

while(1) {

printf("Enter your choice:\n");

printf("1. Insert Front\n");

printf("2. Display\n");

printf("3. Delete End\n");

printf("4. Exit\n");

scanf("%d", &choice);

switch(choice) {

case 1:

head = insert\_front(head);

writeToFile(head); // Write updated list to file

break;

case 2:

display(head);

break;

case 3:

head = delete\_end(head);

writeToFile(head); // Write updated list to file

break;

case 4:

printf("THANK YOU\n");

writeToFile(head); // Write final list to file before exiting

exit(0);

default:

printf("INVALID INPUT\n");

break;

}

}

}

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if(n1 == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

// Function to insert a node at the front of the linked list

NODE insert\_front(NODE head) {

NODE temp;

temp = create\_node();

if(head == NULL) {

head = temp;

} else {

temp->link = head;

head = temp;

}

return head;

}

// Function to display the linked list

void display(NODE head) {

NODE temp;

temp = head;

if(temp == NULL) {

printf("THE LIST IS EMPTY\n");

} else {

while(temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

void display(NODE head) {

NODE temp;

temp = head;

if(temp == NULL) {

printf("THE LIST IS EMPTY\n");

} else {

while(temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

}

printf("\n");

}

NODE delete\_end(NODE head)

{

NODE temp=head, prev=NULL;

if(temp==NULL)

{

printf("List is empty\n");

return head;

}

if(temp->link == NULL) // Only one node in the list

{

free(temp);

return NULL; // Updated head after deletion

}

while(temp->link != NULL)

{

prev = temp;

temp = temp->link;

}

prev->link = NULL;

free(temp);

return head;

}

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

}

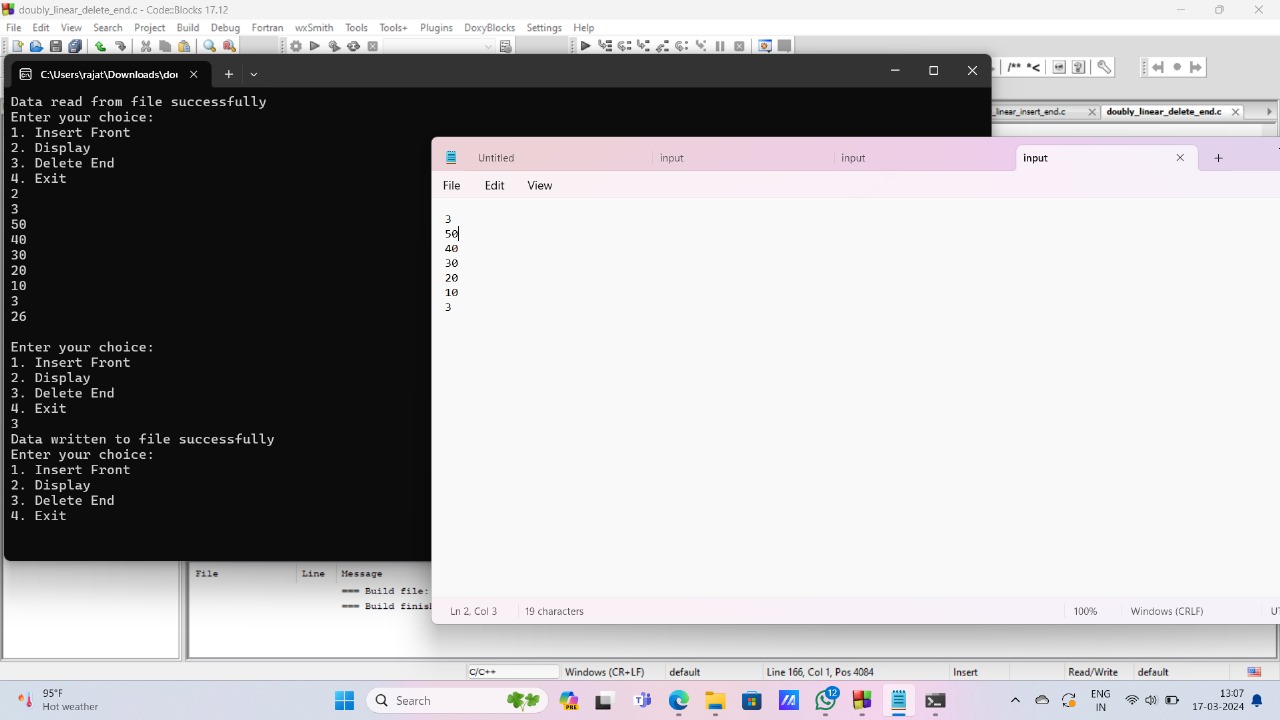
}

fclose(file);

printf("Data read from file successfully\n");

  return head;

}



26. doubly singly delete front

#include <stdio.h>

#include <stdlib.h>

// Structure for a node in the linked list

struct node {

int data;

struct node \*link;

};

typedef struct node \*NODE;

// Function prototypes

NODE create\_node();

NODE insert\_front(NODE head);

NODE delete\_front(NODE head);

NODE display(NODE head);

void writeToFile(NODE head);

NODE readFromFile();

int main() {

NODE head = readFromFile(); // Read data from file if it exists

int choice;

while(1) {

printf("Enter your choice:\n");

printf("1. Delete Front\n");

printf("2. Display\n");

printf("3. Insert Front\n");

printf("4. Exit\n");

scanf("%d", &choice);

switch(choice) {

case 1:

head = delete\_front(head);

writeToFile(head); // Write updated list to file

break;

case 2:

display(head);

break;

case 3:

head = insert\_front(head);

writeToFile(head); // Write updated list to file

break;

case 4:

printf("THANK YOU\n");

writeToFile(head); // Write final list to file before exiting

exit(0);

default:

printf("INVALID INPUT\n");

break;

}

}

}

// Function to create a new node

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if(n1 == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

// Function to delete the node at the front of the linked list

NODE delete\_front(NODE head) {

if (head == NULL) {

printf("List is empty\n");

return head;

}

NODE temp = head;

head = head->link;

free(temp);

return head;

}

// Function to display the linked list

NODE display(NODE head) {

NODE temp = head;

if(temp == NULL) {

printf("THE LIST IS EMPTY\n");

} else {

while(temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

}

printf("\n");

return head;

}

// Function to insert a node at the front of the linked list

NODE insert\_front(NODE head) {

NODE temp = create\_node();

if(head == NULL) {

head = temp;

} else {

temp->link = head;

head = temp;

}

return head;

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

}

}

fclose(file);

printf("Data read from file successfully\n");

  return head;

}

printf("Memory allocation failed\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

// Function to delete the node at the front of the linked list

NODE delete\_front(NODE head) {

if (head == NULL) {

printf("List is empty\n");

return head;

}

NODE temp = head;

head = head->link;

free(temp);

return head;

}

// Function to display the linked list

NODE display(NODE head) {

NODE temp = head;

if(temp == NULL) {

printf("THE LIST IS EMPTY\n");

} else {

while(temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

}

printf("\n");

return head;

}

// Function to insert a node at the front of the linked list

NODE insert\_front(NODE head) {

NODE temp = create\_node();

if(head == NULL) {

head = temp;

} else {

temp->link = head;

head = temp;

}

return head;

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r")

}

fclose(file);

printf("Data written to file successfully\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

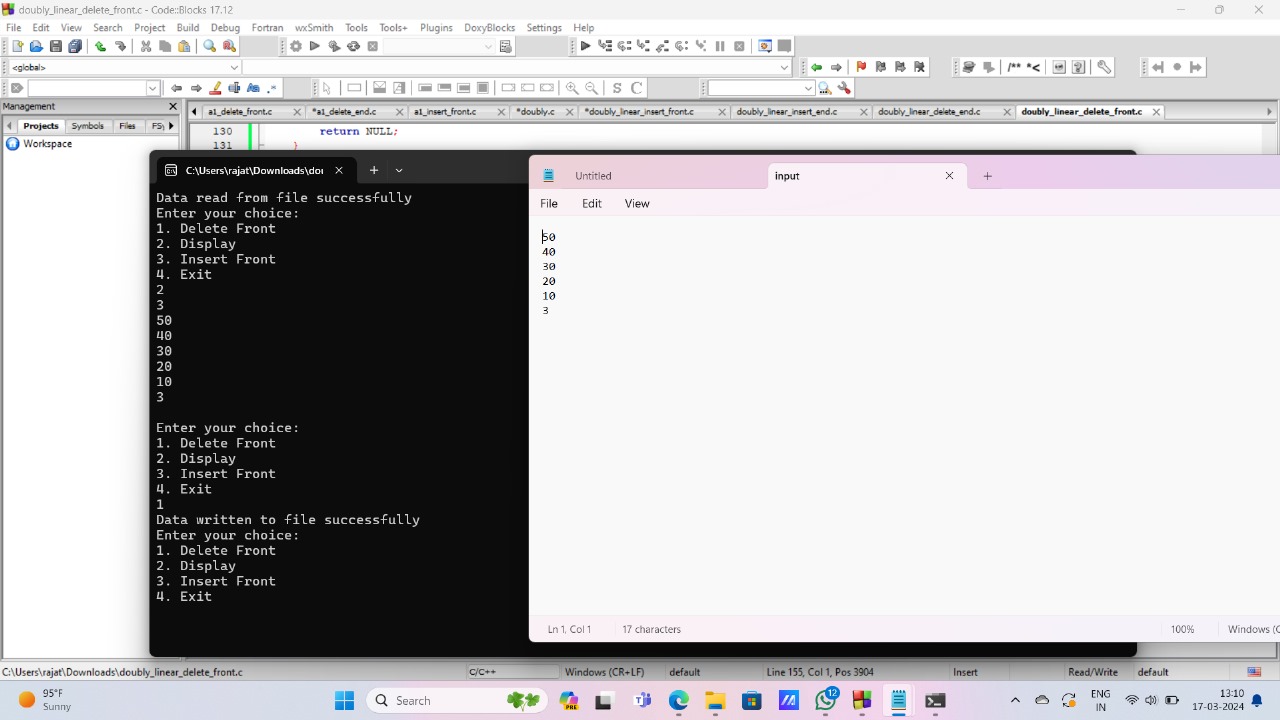
}

}

fclose(file);

printf("Data read from file successfully\n");

  return head;

}

27.doubly insert position

#include <stdio.h>

#include <stdlib.h>

// Structure for a node in the linked list

struct node {

int data;

struct node \*link;

};

typedef struct node \*NODE;

// Function prototypes

NODE create\_node();

NODE display(NODE head);

int count(NODE head);

NODE insert\_position(NODE head);

void writeToFile(NODE head);

NODE readFromFile();

int main() {

NODE head = readFromFile(); // Read data from file if it exists

int choice;

while (1) {

printf("Enter your choice:\n");

printf("1. Display\n");

printf("2. Insert at Position\n");

printf("3. Count Nodes\n");

printf("4. Exit\n");

scanf("%d", &choice);

switch (choice) {

case 1:

display(head);

break;

case 2:

head = insert\_position(head);

writeToFile(head); // Write updated list to file

break;

case 3:

printf("Number of nodes: %d\n", count(head));

break;

case 4:

writeToFile(head); // Write final list to file before exiting

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid input!\n");

break;

}

}

return 0;

}

// Function to create a new node

NODE create\_node() {

NODE n1 = (NODE)malloc(sizeof(struct node));

if (n1 == NULL) {

printf("Memory allocation failed.\n");

exit(1);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

// Function to display the linked list

NODE display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n");

return head;

}

NODE temp = head;

while (temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

return head;

}

// Function to count the number of nodes in the linked list

int count(NODE head) {

int cnt = 0;

NODE temp = head;

while (temp != NULL) {

cnt++;

temp = temp->link;

}

return cnt;

}

// Function to insert a node at a specified position in the linked list

NODE insert\_position(NODE head) {

NODE temp, p, cur;

int pos = 1, i = 1;

printf("Enter the position: ");

scanf("%d", &pos);

temp = create\_node();

if (head == NULL && pos == 1) {

return temp;

} else if (pos == 1) {

temp->link = head;

return temp;

} else {

p = head;

while (i < pos - 1 && p != NULL) {

p = p->link;

i++;

}

if (p == NULL) {

printf("Invalid position\n");

return head;

}

cur = p->link;

p->link = temp;

temp->link = cur;

return head;

}

}

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully.\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(1);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

}

}

fclose(file);

printf("Data read from file successfully.\n");

  return head;

}

NODE n1 = (NODE)malloc(sizeof(struct node));

if (n1 == NULL) {

printf("Memory allocation failed.\n");

exit(1);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

// Function to display the linked list

NODE display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n");

return head;

}

NODE temp = head;

while (temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

return head;

}

int count(NODE head) {

int cnt = 0;

NODE temp = head;

while (temp != NULL) {

cnt++;

temp = temp->link;

}

return cnt;

}

ODE insert\_position(NODE head) {

NODE temp, p, cur;

int pos = 1, i = 1;

printf("Enter the position: ");

scanf("%d", &pos);

temp = create\_node();

if (head == NULL && pos == 1) {

return temp;

} else if (pos == 1) {

temp->link = head;

return temp;

} else {

p = head;

while (i < pos - 1 && p != NULL) {

p = p->link;

i++;

}

if (p == NULL) {

printf("Invalid position\n");

return head;

}

cur = p->link;

p->link = temp;

temp->link = cur;

return head;

}

}

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

return head;

}

}

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully.\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(1);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

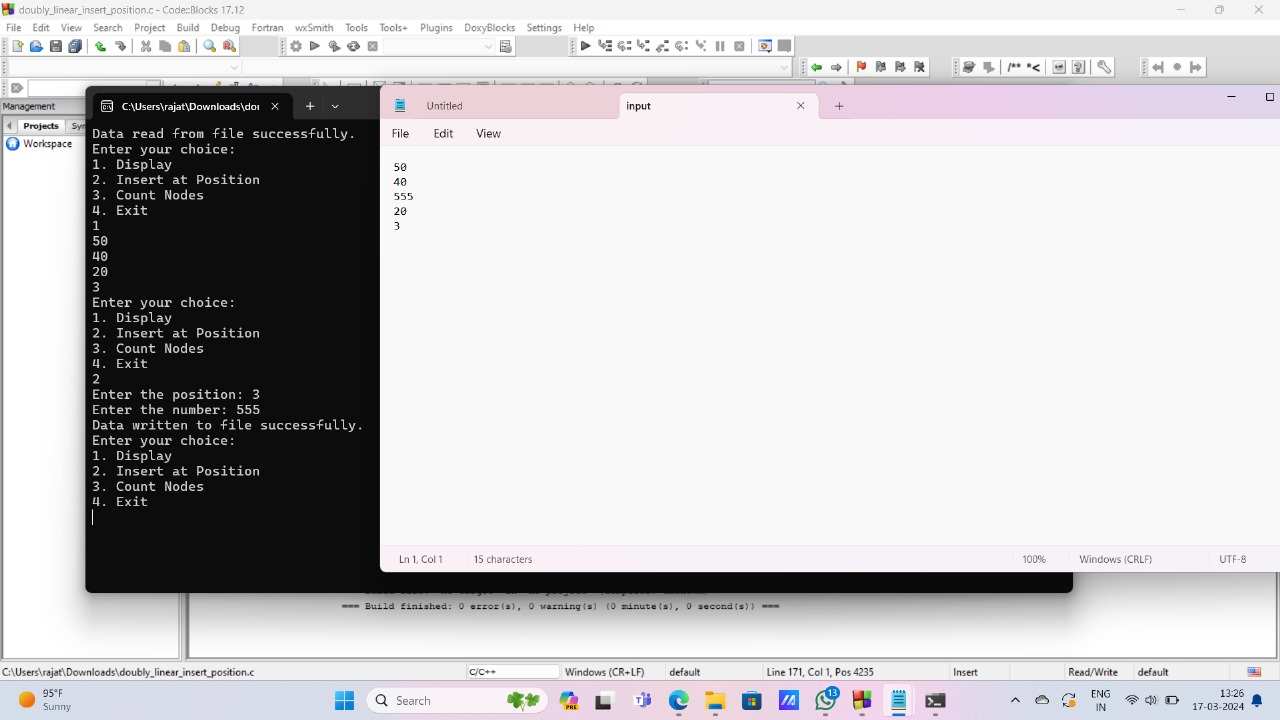
}

}

fclose(file);

printf("Data read from file successfully.\n");

  return head;

}

28.doubly delete position

#include<stdio.h>

#include<stdlib.h>

// Structure for a node in the linked list

struct node {

int data;

struct node \*link;

};

typedef struct node \*NODE;

// Function prototypes

NODE create\_node();

NODE insert\_front(NODE head);

NODE delete\_position(NODE head);

NODE display(NODE head);

void writeToFile(NODE head);

NODE readFromFile();

int count(NODE head);

int main() {

NODE head = readFromFile(); // Read data from file if it exists

int choice;

while(1) {

printf("Enter your choice:\n");

printf("1. Insert Front\n");

printf("2. Display\n");

printf("3. Delete Position\n");

printf("4. Count\n");

printf("5. Exit\n");

scanf("%d", &choice);

switch(choice) {

case 1:

head = insert\_front(head);

writeToFile(head); // Write updated list to file

break;

case 2:

display(head);

break;

case 3:

head = delete\_position(head);

writeToFile(head); // Write updated list to file

break;

case 4:

printf("Count = %d\n", count(head));

break;

case 5:

printf("THANK YOU\n");

writeToFile(head); // Write final list to file before exiting

exit(0);

default:

printf("INVALID INPUT\n");

break;

}

}

}

// Function to create a new node

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if(n1 == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

// Function to insert a node at the front of the linked list

NODE insert\_front(NODE head) {

NODE temp;

temp = create\_node();

if(head == NULL) {

head = temp;

} else {

temp->link = head;

head = temp;

}

return head;

}

// Function to display the linked list

NODE display(NODE head) {

NODE temp;

temp = head;

if(temp == NULL) {

printf("THE LIST IS EMPTY\n");

} else {

while(temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

}

printf("\n");

return head;

}

// Function to delete a node at a specified position in the linked list

NODE delete\_position(NODE head) {

NODE temp = head, p;

int pos, i = 1;

printf("Enter the position: ");

scanf("%d", &pos);

if (temp == NULL && pos == 1) {

printf("List is empty\n");

return head;

} else if (pos == 1) {

head = head->link;

temp->link = NULL;

free(temp);

} else {

temp = head;

while (i < pos - 1) {

temp = temp->link;

i++;

}

if (temp == NULL || temp->link == NULL) {

printf("Invalid position\n");

return head;

}

p = temp->link;

temp->link = p->link;

free(p);

}

return head;

}

// Function to count the number of nodes in the linked list

int count(NODE head) {

int count = 0;

NODE temp = head;

while (temp != NULL) {

count++;

temp = temp->link;

}

return count;

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

}

}

fclose(file);

printf("Data read from file successfully\n");

  return head;

}

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if(n1 == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

// Function to insert a node at the front of the linked list

NODE insert\_front(NODE head) {

NODE temp;

temp = create\_node();

if(head == NULL) {

head = temp;

} else {

temp->link = head;

head = temp;

}

return head;

}

// Function to display the linked list

NODE display(NODE head) {

NODE temp;

temp = head;

if(temp == NULL) {

printf("THE LIST IS EMPTY\n");

} else {

while(temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

}

printf("\n");

return head;

}

// Function to delete a node at a specified position in the linked list

NODE delete\_position(NODE head) {

NODE temp = head, p;

int pos, i = 1;

printf("Enter the position: ");

scanf("%d", &pos);

if (temp == NULL && pos == 1) {

printf("List is empty\n");

return head;

} else if (pos == 1) {

head = head->link;

temp->link = NULL;

free(temp);

} else {

temp = head;

while (i < pos - 1) {

temp = temp->link;

i++;

}

if (temp == NULL || temp->link == NULL) {

printf("Invalid position\n");

return head;

}

p = temp->link;

temp->link = p->link;

free(p);

}

return head;

}

i++;

}

if (temp == NULL || temp->link == NULL) {

printf("Invalid position\n");

return head;

}

p = temp->link;

temp->link = p->link;

free(p);

}

return head;

}

// Function to count the number of nodes in the linked list

int count(NODE head) {

int count = 0;

NODE temp = head;

while (temp != NULL) {

count++;

temp = temp->link;

}

return count;

}

// Function to write the linked list to a file

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing\n");

return;

}

NODE temp = head;

while (temp != NULL) {fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

}

}

fclose(file);

29.singly circular insert end

#include <stdio.h>

#include <stdlib.h>

// Structure for a node in the linked list

struct node {

int data;

struct node \*link;

};

typedef struct node \*NODE;

// Function prototypes

NODE create\_node();

NODE insert\_end(NODE head);

void display(NODE head);

void writeToFile(NODE head);

NODE readFromFile();

int main() {

NODE head = readFromFile(); // Read data from file if it exists

int choice;

while (1) {

printf("Enter your choice:\n");

printf("1. Insert End\n");

printf("2. Display\n");

printf("3. Exit\n");

scanf("%d", &choice);

switch (choice) {

case 1:

head = insert\_end(head);

writeToFile(head); // Write updated list to file

break;

case 2:

display(head);

break;

case 3:

writeToFile(head); // Write final list to file before exiting

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid input!\n");

break;

}

}

return 0;

}

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if (n1 == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

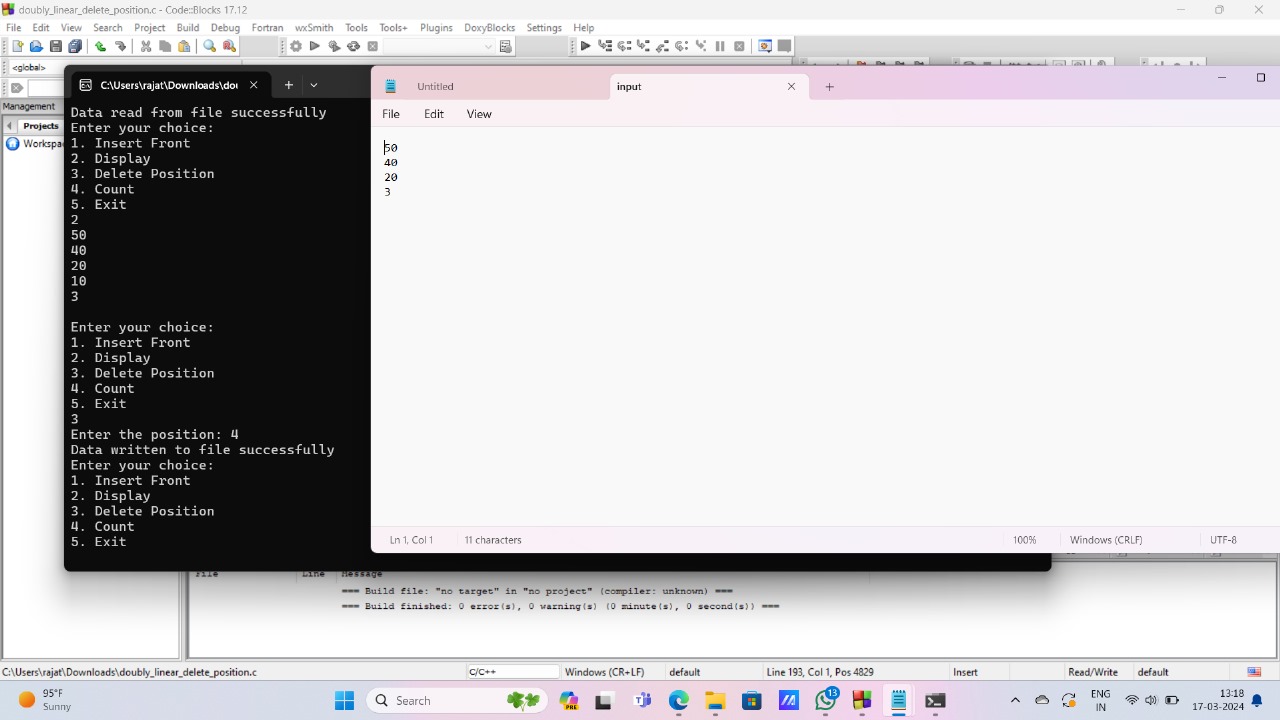
printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

printf("Data read from file successfully\n");

  return head;

}

writeToFile(head); // Write updated list to file

break;

case 2:

display(head);

break;

case 3:

writeToFile(head); // Write final list to file before exiting

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid input!\n");

break;

}

}

return 0;

}

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if (n1 == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

n1->link = NULL;

return n1;

}

NODE insert\_end(NODE head) {

NODE temp, p;

temp = create\_node();

if (head == NULL) {

head = temp;

} else {

p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = temp;

}

return head;

}

void display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

}

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully.\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully.\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link;

}

p->link = newNode;

}

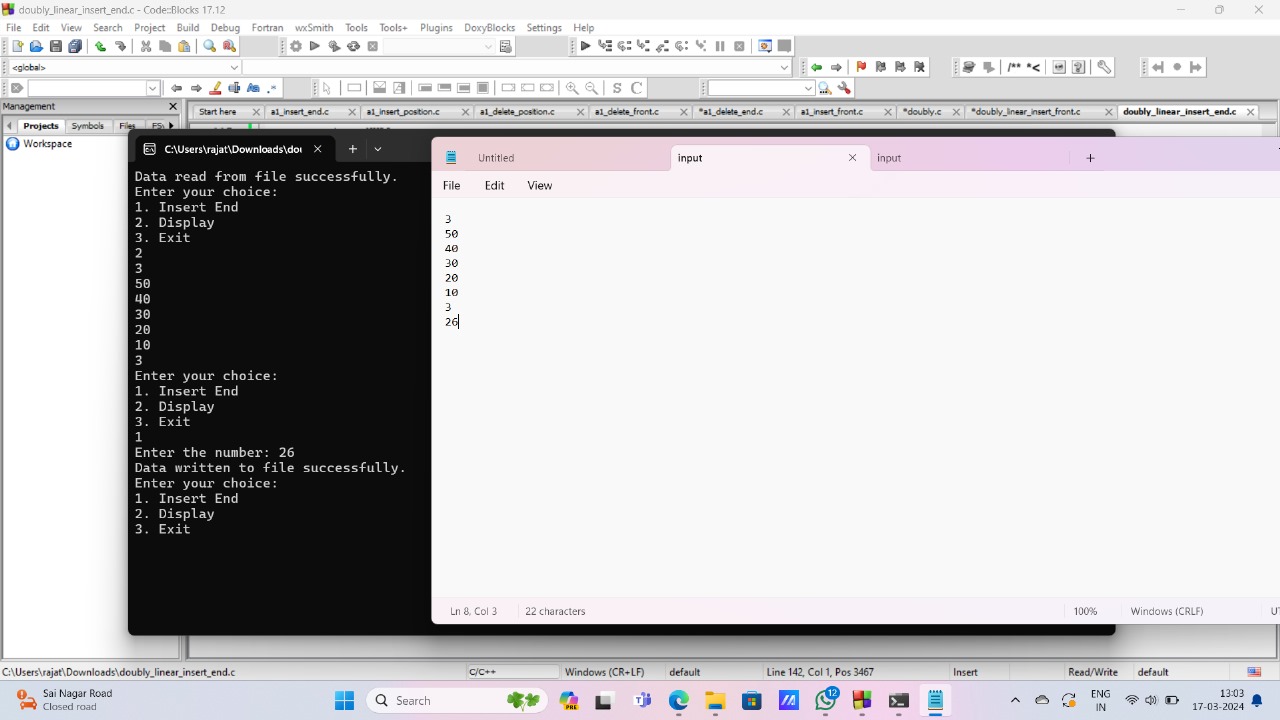
}

fclose(file);

printf("Data read from file successfully.\n");

  return head;

}



30. doubly circular insert end

#include <stdio.h>

#include <stdlib.h>

// Structure for a node in the linked list

struct node {

int data;

struct node \*link;

};

typedef struct node \*NODE;

// Function prototypes

NODE create\_node();

NODE insert\_end(NODE head);

void display(NODE head);

void writeToFile(NODE head);

NODE readFromFile();

int main() {

NODE head = readFromFile(); // Read data from file if it exists

int choice;

while (1) {

printf("Enter your choice:\n");

printf("1. Insert End\n");

printf("2. Display\n");

switch (choice) {

case 1:

head = insert\_end(head);

writeToFile(head); // Write updated list to file

break;

case 2:

display(head);

break;

case 3:

writeToFile(head); // Write final list to file before exiting

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid input!\n");

break;

}

}

return 0;

}

// Function to create a new node

NODE create\_node() {

NODE n1;

n1 = (NODE)malloc(sizeof(struct node));

if (n1 == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

printf("Enter the number: ");

scanf("%d", &n1->data);

n1->link = NULL;

return n1;

}

NODE insert\_end(NODE head) {

NODE temp = create\_node(); // Create a new node

if (head == NULL) {

head = temp;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link; // Traverse till the end of the list

}

p->link = temp; // Set the link of the last node to the new node

}

return head;

}

void display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

}

n1->link = NULL;

return n1;

}

NODE insert\_end(NODE head) {

NODE temp = create\_node(); // Create a new node

if (head == NULL) {

head = temp;

} else {

NODE p = head;

while (p->link != NULL) {

p = p->link; // Traverse till the end of the list

}

p->link = temp; // Set the link of the last node to the new node

}

return head;

}

void display(NODE head) {

if (head == NULL) {

printf("The list is empty.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

printf("%d\n", temp->data);

temp = temp->link;

}

}

void writeToFile(NODE head) {

FILE \*file = fopen("input.txt", "w");

if (file == NULL) {

printf("Failed to open file for writing.\n");

return;

}

NODE temp = head;

while (temp != NULL) {

fprintf(file, "%d\n", temp->data);

temp = temp->link;

}

fclose(file);

printf("Data written to file successfully.\n");

}

// Function to read the linked list from a file

NODE readFromFile() {

NODE head = NULL;

FILE \*file = fopen("input.txt", "r");

if (file == NULL) {

printf("File doesn't exist or couldn't be opened for reading.\n");

return NULL;

}

int value;

while (fscanf(file, "%d", &value) != EOF) {

NODE newNode = (NODE)malloc(sizeof(struct node));

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(0);

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != N NULL) {

p = p->link;

}

p->link = newNode;

}

}

fclose(file);

printf("Data read from file successfully.\n");

  return head;

}

}

newNode->data = value;

newNode->link = NULL;

if (head == NULL) {

head = newNode;

} else {

NODE p = head;

while (p->link != N NULL) {

p = p->link;

}

p->link = newNode;

}

}

fclose(file);

printf("Data read from file successfully.\n");

  return head;

}