|  |  |
| --- | --- |
| Program No: 01 | Date: 23/07/2025 |
| Program Title : Write programs to demonstrate the use of storage classes (local variable, global variable, static variable, register variable) in C. | |
| /\*Program 1 USE DIFFERENT STORAGE CLASSES (LOCAL,GLOBAL,STATIC,REGISTER)  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 23/07/2025  \*/  #include<stdio.h>  int a=10,b=10; //global var  void display() {  int c=10;  static int d=10; //static var  printf("local under increment: %d\t static under increment:%d\n",c++,d++);  }  int main(){  register int e=11; //register var  int b=5; //local var  printf("Global variable: %d\nLocal variable: %d\n",a,b);  printf("Register variable: %d\n",e);  printf("Static variable: \n");  display();  display();  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 02 | Date: 23/07/2025 |
| Program Title : Use a menu-driven program to insert, search, delete and sort elements in an array using functions (use global variables). | |
| /\*PROGRAM-2 A MENU FOR ARRAY OPERATIONS(INSERT,DELETE,DISPLAY,SEARCH,SORT) USING GLOBAL VARIABLE  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 23/07/2025  \*/  #include<stdio.h>  int stack[5]; //declaring stack  int top = -1; //declaring variable positionof top element  int insert(int e) { //Function to insert element into stack  if (top + 1 == 5) {  printf("Error: Stack is Full");  }  else {  stack[++top] = e;  }  return top;  }  int erase() { //function to delete top element  if (top == -1){  printf("Error: Stack is empty");  }  else {  printf("\n %d",stack[top--]);  }  return top;  }  void search(int b, int a[5], int top){ //function to search elements  int isfound = 0, i;  for (i = 0; i <= top; i++) {  if (b == a[i]) {  isfound = 1;  printf("Element found at [%d] position. \n", i);  }  }  if (isfound == 0)  printf("element not found");  }    void display() { //function to display the elements in stack  if (top == -1)  printf("Empty Stack");  else {  int i;  for (i = 0; i <= top; i++){  printf("%d \t",stack[i]);  }  printf("\n");  }  }  void sort(){ //function to sort the stack  int i, j, temp;  for (i = 0; i < 5; i++) {  for (j = 0; j < 5; j++) {  if(stack[i] < stack[j]) {  temp = stack[i];  stack[i] = stack[j];  stack[j] = temp;  }  }  }  }  int menu() { //function for menu  int ch;  printf("\n INSERT-1 \n DELETE-2 \n DISPLAY-3 \n SEARCH-4 \n SORT-5 \n EXIT -6 \n Enter your choice: ");  scanf("%d", &ch);  return ch;  }  void processStack() { //working of menu  int ch, b;  for (ch = menu(); ch != 6; ch = menu()) {  switch(ch){  case 1: //insert  printf("Enter the value to insert: ");  scanf("%d", &ch);  insert(ch);  break;  case 2: //delete  erase();  break;  case 3: //display  display();  break;  case 4: //search  printf("Enter the value to search: ");  scanf("%d",&b);  search(b, stack, top);  break;  case 5: // sort  sort();  break;  default: //any other options  printf("Error: Wrong Choice");  break;  }  }  }  int main() {  processStack();  return 0;  } | |
| Output | |
| Program No: 03 | Date: 23/07/2025 |
| Program Title : Use a menu-driven program to insert, search, delete and sort elements in an array using functions (use only local variables). | |
| /\*PROGRAM-3 A MENU FOR ARRAY OPERATIONS(INSERT,DELETE,DISPLAY,SEARCH,SORT) USING LOCAL VARIABLE  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 23/07/2025  \*/  #include <stdio.h>  int insert(int a[5], int pos, int e) { //function to insert an element  if (pos + 1 == 5) {  printf("Error: Array is Full\n");  }  else {  a[++pos] = e;  }  return pos;  }  int erase(int a[5], int pos) { //function to delete top element  if (pos == -1) {  printf("Error: Array is Empty\n");  }  else {  printf("Deleted element: %d\n", a[pos--]);  }  return pos;  }  void display(int a[5], int pos){ //function to display entire array  if (pos == -1) {  printf("Error: Array is Empty\n");  }  else {  int i;  for (i = 0; i <= pos; i++) {  printf("%d\t", a[i]);  }  printf("\n");  }  }  void search(int b, int a[5], int pos){ //function to search for an element and display its index  int isfound = 0, i;  for (i = 0; i <= pos; i++) {  if(b==a[i]) {  isfound=1;  printf("Element found at [%d] position. \n",i);  }  }  if (isfound == 0)  printf("element not found");  }  void sort(int a[5]){ //function to sort the stack  int i, j, temp;  for (i = 0; i < 5; i++) {  for (j = 0; j < 5; j++) {  if(a[i] < a[j]) {  temp = a[i];  a[i] = a[j];  a[j] = temp;  }  }  }  }  int menu() { //function to create menu interface  int ch;  printf("\nInsert - 1\nDelete - 2\nDisplay - 3\nSearch - 4\nSort - 5\nExit - 6\nEnter your choice: ");  scanf("%d", &ch);  return ch;  }  void processArray() { //working of menu  int a[5], pos = -1, b;  int ch, value;  for (ch = menu(); ch != 6; ch = menu()) {  switch (ch) {  case 1:  printf("Enter value to insert: ");  scanf("%d", &value);  pos = insert(a, pos, value);  break;  case 2:  pos = erase(a, pos);  break;  case 3:  display(a, pos);  break;  case 4:  printf("Enter the element to search: ");  scanf("%d",&b);  search(b,a,pos);  break;  case 5:  sort(a);  break;  default:  printf("Error: Wrong Choice.\n");  }  }  }  int main() {  processArray();  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 04 | Date: 23/07/2025 |
| Program Title : Search for all the occurrences of an element in an integer array (positions). | |
| /\*PROGRAM-4 ARRAY SEARCH  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 23/07/2025  \*/  #include<stdio.h>  void search(int b,int a[5]){ //function for search function  int isfound = 0, i;  for (i = 0; i < 5; i++){  if (b == a[i]) {  isfound = 1;  printf("Element found at [%d] position. \n",i); //displays index  }  }  if (isfound = 0)  printf("element not found");  }  int main() {  int a[5], ch, i;  for (i = 0; i < 5; i++) {  printf("Enter the [%d] element : ",i);  scanf("%d",&a[i]);  }  for (i = 0; i < 5; i++) {  printf("%d \t",a[i]);  }  printf("\n");  printf("Enter the element to search: ");  scanf("%d", &ch);  search(ch,a);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 05 | Date: 23/07/2025 |
| Program Title : Sort the array elements in ascending order (minimum three functions: read, disp and sort). | |
| /\*PROGRAM-5 SORT ARRAY IN ASC WITH ALTEAST 3 FUNCTIONS - READ DISP SORT  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 23/07/2025  \*/  #include<stdio.h>  void read(int n,int arr[n]){ //function to insert elements in array  int i;  for (i = 0; i < n; i++) {  printf("Enter the value for %d :",i);  scanf("%d", &arr[i]);  }  }  void disp(int n,int arr[n]) { //function to display all elemenyts in array  int i;  for (i = 0; i < n; i++) {  printf("%d\t", arr[i]);  }  }  void sort(int n,int arr[n]) { //function for sorting the elements in array in ascending order  int i, j, temp;  for (i = 0; i < n; i++) {  for (j = 0; j < n; j++) {  if (arr[i] < arr[j]) {  temp = arr[i];  arr[i] = arr[j];  arr[j] = temp;  }  }  }  }  int menu() { //funtion for menu interface  int ch;  printf("\nREAD-1\nSORT-2\nDISPLAY-3\nEXIT-4\nENTER YOUR CHOICE: ");  scanf("%d", &ch);  return ch;  }  void processArray(int n,int arr[n]) { //working of menu  int ch;  for (ch = menu(); ch != 4; ch = menu()) {  switch(ch) {  case 1:  read(n,arr);  break;  case 2:  sort(n,arr);  break;  case 3:  disp(n,arr);  break;  default:  printf("Errror: Wrong Choice\n");  break;  }  }  }  int main() {  int n;  printf("Enter limit of Array:");  scanf("%d",&n);  int arr[n];  processArray(n,arr);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 06 | Date: 23/07/2025 |
| Program Title : Display the array elements in the same order using a recursive function. | |
| /\*PROGRAM-6 DISPLAY ARRAY USING RECURSIVE FUNCTION  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 25/07/2025  \*/  #include<stdio.h>  int j = 0, arr[10];  void display() { //display function as a recursive function  if (j < 10) {  printf("%d\t",arr[j]);  j++;  display();  }  If (j == 10)  j = 0;  }  int main() {  int i=0;  for (i = 0; i < 10; i++) {  printf("Enter the value for %d:", i);  scanf("%d", &arr[i]);  }  display();  return 0;  } | |
| Output | |
| Program No: 07 | Date: 23/07/2025 |
| Program Title : Display array elements in reverse order using a recursive function. | |
| /\*PROGRAM-7 DISPLAY ARRAY INN REVERSE USING RECURSIVE FUNCTION  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 26/07/2025  \*/  #include<stdio.h>  int j = 10, arr[10];  void display() { //function to display elements in reverse using recursive function  if (j > 0) {  j--;  printf("%d\t",arr[j]);  display();  if (j == 0)  j = 10;  }  }  int main() {  int i = 0;  for (i = 0; i < 10; i++) {  printf("Enter the value for %d:", i);  scanf("%d", &arr[i]);  }  display();  return 0;  } | |
| Output | |
| Program No: 08 | Date: 23/07/2025 |
| Program Title : Write a program to Perform the addition of two matrix and Subtraction of one matrix from another. | |
| /\*PROGRAM-8 MATRIX ADDITION AND SUBTRACTION  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 25/07/2025  \*/  #include<stdio.h>  int a[10][10], b[10][10], m, n;  void insert(int e[10][10],int m, int n){ //function to insert values into the matrix  int i, j;  for (i = 0; i < m; i++){  for (j = 0; j < n; j++){  printf("Enter the value of [%d] [%d] \: ", i, j);  scanf("%d", &e[i][j]);  }  }  }  void print(int p[10][10],int m, int n){ //function to print a matrix  int i, j;  for (i = 0; i < m; i++) {  for (j = 0; j < n; j++) {  printf("%d\t",p[i][j]);  }  printf("\n");  }  }  void add() { //funtion to add two matrices  int sum[10][10], i, j;  for (i = 0; i < m; i++) {  for (j = 0; j < n; j++) {  sum[i][j]=a[i][j]+b[i][j];  }  }  print(sum,m,n);  }  void diffn() { //function to subtract a matrix from another  int dif[10][10], i, j;  for (i = 0; i < m; i++) {  for (j = 0; j < n; j++) {  dif[i][j]=a[i][j]-b[i][j];  }  }  print(dif,m,n);  }  int main() {  printf("Enter the dimensions of the Matrix: ");  scanf("%d%d",&m,&n);  printf("Enter the First matrix :\n");  insert(a,m,n);  printf("Enter the Second matrix : \n");  insert(b,m,n);  printf("The first matrix is : \n");  print(a,m,n);  printf("The Second matrix is : \n");  print(b,m,n);  printf("The sum of matrices is : \n");  add(m,n);  printf("The difference of matrices is : \n");  diffn(m,n);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 09 | Date: 23/07/2025 |
| Program Title : Write a program to perform multiplication of two matrix. | |
| /\*PROGRAM-9 MATRIX MULTIPLICATION  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 25/07/2025  \*/  #include<stdio.h>  int a[10][10],b[10][10], m, n, p;  void insert(int e[10][10], int m, int n) { //function to insert values in the matrix  int i, j;  for (i = 0; i < m; i++) {  for(j=0;j<n;j++){  printf("Enter the value of [%d] [%d]: ", i, j);  scanf("%d",&e[i][j]);  }  }  }  void print(int p[10][10], int m, int n) { //funtion to display the matrix  int i, j;  for (i = 0; i < m; i++) {  for (j = 0; j < n; j++) {  printf("%d\t",p[i][j]);  }  printf("\n");  }  }  void multi() { //function to multipy two matrices  int prod[10][10], i, j, k;  for ( i = 0; i < m; i++) { //declaring intial value of elements of product to be zero  for (j = 0; j < p; j++) {  prod[i][j]=0;  }  }  for (i = 0; i < m; i++) {  for (j = 0; j < p; j++) {  for (k = 0; k < n; k++) {  prod[i][j] += a[i][k] \* b[k][j];  }  }  }  print(prod,m,p);  }  int main() {  printf("Enter the dimensions of first Matrix: ");  scanf("%d%d",&m,&n);  printf("Enter the First matrix :\n");  insert(a,m,n);  printf("Enter the number of columns for the Second Matrix: ");  scanf("%d", &p);  printf("Enter the Second matrix : \n");  insert(b,n,p);  printf("The first matrix is : \n");  print(a,m,n);  printf("The Second matrix is : \n");  print(b,n,p);  printf("The product of matrices is : \n");  multi();  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 10 | Date: 23/07/2025 |
| Program Title : Write a program to find the transpose of a matrix. | |
| /\*PROGRAM-10 MATRIX TRANSPOSE  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 25/07/2025  \*/  #include<stdio.h>  int a[10][10], t[10][10];  void print(int e[10][10], int m, int n) { //function to print the matrix.  int i, j;  for (i = 0; i < m; i++) {  for (j = 0; j < n; j++) {  printf("%d\t", e[i][j]);  }  printf("\n");  }  }  void transpose(int m, int n) { //function to find transpose of matrix  int i,j;  for (i = 0; i < m; i++) {  for(j = 0; j < n; j++) {  t[j][i] = a[i][j];  }  }  printf("The Transpose of the matrix is : \n");  print(t,n,m);  }  int main() {  int m,n;  printf("Enter the dimensions of the Matrix:");  scanf("%d%d", &m, &n);  int i, j;  for (i = 0; i < m; i++) {  for (j = 0; j < n; j++) {  printf("Enter the value of [%d] [%d]: ", i, j);  scanf("%d",&a[i][j]);  }  }  printf("The matrix is : \n");  print(a,m,n);  transpose(m,n);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 11 | Date: 23/07/2025 |
| Program Title : Write a program to find the Determinant of a matrix (2x2 and 3x3). | |
| /\* PROGRAM-11 DETERMINANT OF A MATRIX  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 26/07/2025  \*/  #include <stdio.h>  int main() {  int size, a[3][3], i, j;  float det;  printf("Enter the size of the square matrix (2 or 3): ");  scanf("%d",&size);  if (size != 2 && size != 3) {  printf("Only 2x2 and 3x3 matrices are supported.\n");  return 1;  }  printf("Enter the elements of the matrix:\n");  for (i = 0; i < size; i++) {  for (j = 0; j < size; j++) {  printf("a[%d][%d]: ", i, j);  scanf("%d", &a[i][j]);  }  }  printf("The matrix is:\n");  for (i = 0;I < size; i++) {  for (j = 0; j < size; j++) {  printf("%d\t", a[i][j]);  }  printf("\n");  }  if (size == 2) {  det=a[0][0] \* a[1][1] - a[0][1] \* a[1][0]; // For 2x2: |A| = ad - bc  } else if (size==3) {  det = a[0][0] \* (a[1][1] \* a[2][2] - a[1][2] \* a[2][1]) //expansion of formula - a[0][1] \* (a[1][0] \* a[2][2] - a[1][2] \* a[2][0])  + a[0][2] \* (a[1][0] \* a[2][1] - a[1][1] \* a[2][0]);  }  printf("Determinant of the matrix = %.2f\n", det);  return 0;  } | |
| Output | |
| Program No: 12 | Date: 23/07/2025 |
| Program Title : Implement stack operations using arrays. | |
| /\*PROGRAM-12 STACK OPERATIONS USING ARRAY  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 26/07/2025  \*/  #include <stdio.h>  int push(int stack[5], int top, int e) { //function to push elements onto stack  if (top + 1 == 5) {  printf("Error: Stack is Full\n");  } else {  stack[++top] = e;  printf("Pushed\n");  }  return top;  }  int pop(int stack[5], int top){ //function to pop the top element from stack  if (top == -1){  printf("Error: Stack is Empty\n");  } else {  printf("Popped element: %d\n", stack[top--]);  }  return top;  }  void peek(int stack[5], int top) { //function to peek the top element in stack  if (top == -1) {  printf("Stack is Empty\n");  }else{  printf("Top element: %d \n",stack[top]);  }  }  int menu() { //function for menu interface  int ch;  printf("\nPush - 1\nPop - 2\nPeek - 3\nExit - 4\nEnter your choice: ");  scanf("%d", &ch);  return ch;  }  void processStack() { //working of menu  int stack[5], top = -1;  int ch, value;  for (ch = menu(); ch != 4; ch = menu()) {  switch (ch) {  case 1:  printf("Enter value to insert: ");  scanf("%d", &value);  top = push(stack, top, value);  break;  case 2:  top = pop(stack, top);  break;  case 3:  peek(stack, top);  break;  default:  printf("Error: Wrong Choice.\n");  }  }  }  int main() {  processStack();  return 0;  } | |
| Output | |
| Program No: 13 | Date: 23/07/2025 |
| Program Title : Read a String and Just print it in the reverse order. | |
| /\* PROGRAM-13 STRING REVERSAL  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 27/07/2025  \*/  #include<stdio.h>  void str\_rev(char a[20]){ //program to reverse a string  int i=0;  while (a[i] != '\0') {  i++;  }  for ( ; i >= 0; i--) {  printf("%c", a[i]);  }  }  int main() {  char a[20];  printf("Enter a String: ");  gets(a);  str\_rev(a);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 14 | Date: 23/07/2025 |
| Program Title : Read a String and Reverse the string in the same array itself. | |
| /\* PROGRAM-14 STRING REVERSAL IN THE SAME ARRRAY  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 27/07/2025  \*/  #include <stdio.h>  #include <string.h>  void reverse(char str[20]) { //function to reverse the string  int i = 0, j;  char temp;  j = strlen(str) - 1;  while (i < j) { //swaping end characters  temp = str[i];  str[i] = str[j];  str[j] = temp;  i++;  j--;  }  }  int main() {  char str[20];  printf("Enter a string: ");  fgets(str, sizeof(str), stdin);  printf("String is %s",str);  reverse(str);  printf("Reversed string : %s\n", str);  return 0;  } | |
| Output | |
| Program No: 15 | Date: 23/07/2025 |
| Program Title : Read n Strings and display them in the ascending order. | |
| /\* PROGRAM-15 SORTING N STRINGS  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 28/07/2025  \*/  #include <stdio.h>  #include <string.h>  int main() {  int n, i, j;  char str[20][100], temp[100];  printf("Enter the number of strings: ");  scanf("%d", &n);  getchar(); // to consume the newline after scanf  for (i = 0; i < n; i++) {  printf("Enter string %d: ", i + 1);  fgets(str[i], sizeof(str[i]), stdin);  str[i][strcspn(str[i], "\n")] = '\0'; // remove newline  }  for (i = 0; i < n - 1; i++) { // Sort strings using bubble sort  for (j = i + 1; j < n; j++) {  if (strcmp(str[i], str[j]) > 0) {  strcpy(temp, str[i]);  strcpy(str[i], str[j]);  strcpy(str[j], temp);  }  }  }  printf("\nStrings in ascending order:\n");  for (i = 0; i < n; i++) {  printf("%s\n", str[i]);  }  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 16 | Date: 23/07/2025 |
| Program Title : Reverse a string using Stack. | |
| /\* PROGRAM-16 STRING REVERSAL USING STACK  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 28/07/2025  \*/  #include <stdio.h>  char stack[100];  int top=-1;  void push(char e) { //function to push elements onto stack  if (top + 1 == 100) {  printf("Error: Stack is Full\n");  }  else {  stack[++top] = e;  printf("%c",e);  }  }  void pop(){ //function to top pop element from stack  if (top == -1) {  printf("Error: Stack is Empty\n");  }  else {  printf("%c", stack[top--]);  }  }  void peek() { //function to display the top element of the stack  int i;  if (top == -1) {  printf("Stack is Empty\n");  }  else {  printf("%c \n", stack[top]);  }  }  int menu() {  int ch;  printf("\nPush - 1\nPop - 2\nPeek - 3\nReverse - 4\nExit - 5\nEnter your choice: ");  scanf("%d", &ch);  return ch;  }  void processArray() {  int ch;  char value;  for (ch = menu(); ch != 5; ch = menu()) {  switch (ch) {  case 1:  printf("Enter value to insert: ");  scanf(" %c", &value);  push(value);  break;  case 2:  pop();  break;  case 3:  peek();  break;  case 4:  while (top != -1) { //function to pop all elements  pop();  }  break;  default:  printf("Error: Wrong Choice.\n");  }  }  }  int main() {  processArray();  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 17 | Date: 10/09/2025 |
| Program Title : Convert an expression from infix expression to postfix using stack. | |
| /\*Program 17 INFIX TO POSTFIX CONVERSION USING STACK  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 12/9/2025  \*/  #include <stdio.h>  #include <ctype.h>  #include <string.h>  #define s 20  char stack[s], expr[s], post[s];  int top = -1, j = 0;  void push(char e) { // Function to insert element into stack  stack[++top] = e;  }  char pop() { // Function to delete top element  if (top == -1) {  printf("Error: Stack is empty");  return '\0';  } else {  return stack[top--];  }  }  char peek() { // Function to return top element  if (top == -1) {  return '\0';  } else {  return stack[top];  }  }  int isOperator(char c) { // Function to check if character is operator  return (c == '+' || c == '-' || c == '\*' || c == '/' || c == '^');  }  char chariden(char d) { // Function to identify character type  if (isalnum(d)) {  return 'n';  } else if (isOperator(d)) {  return 'o';  } else if (d == '(' || d == ')') {  return 'p';  } else {  return 'x';  }  }  int precedence(char d) { // Function to return precedence of operators  switch (d) {  case '^': return 3;  case '\*':  case '/': return 2;  case '+':  case '-': return 1;  default: return 0;  }  }  void oper(char d) { // Function to handle operator  int e, f;  e = precedence(d);  f = precedence(peek());  while (top != -1 && isOperator(peek()) && (e <= f)) {  post[j++] = pop();  f = precedence(peek());  }  push(d);  }  void para(char d) { // Function to handle parenthesis  if (d == '(') {  push(d);  } else if (d == ')') {  while (top != -1 && peek() != '(') {  post[j++] = pop();  }  if (peek() == '(')  pop(); // Remove '(' from stack  else  printf("Error: Mismatched parentheses\n");  }  }  void IntoPost(char expr[s]) { // Function to convert infix to postfix  int i;  char c, d;  for (i = 0; expr[i] != '\0'; i++) {  d = expr[i];  c = chariden(d);  switch (c) {  case 'o':  oper(d);  break;  case 'n':  post[j++] = d;  break;  case 'p':  para(d);  break;  default:  break;  }  }  while (top != -1) {  if (peek() == '(') {  printf("Error: Mismatched parentheses\n");  pop();  } else {  post[j++] = pop();  }  }  post[j] = '\0'; // Null terminate postfix string  printf("Postfix Expression: %s\n", post);  }  int main() {  printf("Enter the expression: ");  scanf("%s", expr);  IntoPost(expr);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 18 | Date: 10/09/2025 |
| Program Title : Convert an expression from infix expression to prefix using stack | |
| /\*Program 18 INFIX TO PREFIX CONVERSION USING STACK  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 12/9/2025  \*/  #include <stdio.h>  #include <ctype.h>  #include <string.h>  #define s 20  char stack[s], expr[s], pre[s];  int top = -1, j = 0;  void push(char e) { // Function to insert element into stack  stack[++top] = e;  }  char pop() { // Function to delete top element  if (top == -1) {  printf("Error: Stack is empty");  return '\0';  } else {  return stack[top--];  }  }  char peek() { // Function to return top element  if (top == -1) {  return '\0';  } else {  return stack[top];  }  }  int isOperator(char c) { // Function to check if character is operator  return (c == '+' || c == '-' || c == '\*' || c == '/' || c == '^');  }  char chariden(char d) { // Function to identify character type  if (isalnum(d)) {  return 'n';  } else if (isOperator(d)) {  return 'o';  } else if (d == '(' || d == ')') {  return 'p';  } else {  return 'x';  }  }  int precedence(char d) { // Function to return precedence of operators  switch (d) {  case '^': return 3;  case '\*':  case '/': return 2;  case '+':  case '-': return 1;  default: return 0;  }  }  void oper(char d) { // Function to handle operator  int e, f;  e = precedence(d);  f = precedence(peek());  while (top != -1 && isOperator(peek()) && ((e < f) || (e == f))) {  pre[j++] = pop();  }  push(d);  }  void para(char d) { // Function to handle parenthesis  if (d == '(') {  push(d);  } else if (d == ')') {  while (top != -1 && peek() != '(') {  pre[j++] = pop();  }  if (peek() == '(')  pop(); // Remove '(' from stack  else  printf("Error: Mismatched parentheses\n");  }  }  void reverse(char str[s]) { // Function to reverse the string and swap parenthesis  int i = 0, j;  char temp;  j = strlen(str) - 1;  while (i < j) { // Swapping end characters  temp = str[i];  str[i] = str[j];  str[j] = temp;  i++;  j--;  }  int len = strlen(str);  for (i = 0; i < len; i++) {  if (str[i] == '(')  str[i] = ')';  else if (str[i] == ')')  str[i] = '(';  }  }  void IntoPre(char expr[s]) { // Function to convert infix to prefix  int i;  char c, d;  for (i = 0; expr[i] != '\0'; i++) {  d = expr[i];  c = chariden(d);  switch (c) {  case 'o':  oper(d);  break;  case 'n':  pre[j++] = d;  break;  case 'p':  para(d);  break;  default:  break;  }  }  while (top != -1) {  if (peek() == '(') {  printf("Error: Mismatched parentheses\n");  pop();  } else {  pre[j++] = pop();  }  }  pre[j] = '\0'; // Null terminate prefix string  reverse(pre);  printf("Prefix Expression: %s\n", pre);  }  int main() {  printf("Enter the expression: ");  scanf("%s", expr);  reverse(expr);  IntoPre(expr);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 19 | Date: 10/09/2025 |
| Program Title : Evaluate an infix expression using stack | |
| /\*Program 19 EVALUATION OF INFIX EXPRESSION  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 15/9/2025  \*/  #include <stdio.h>  #include <ctype.h>  int precedence(char op) { // function to return precedence of operators  switch (op) {  case '/':  case '\*': return 1;  case '+':  case '-': return 0;  }  return -1;  }  int apply(int a, int b, char sign) { // function to apply the operator on two operands  switch (sign) {  case '+': return a + b;  case '-': return a - b;  case '\*': return a \* b;  case '/': return a / b;  }  return 0;  }  int evaluate(char a[100]) { // function to evaluate infix expression  int i = 0, vstack[100], vtop = -1, otop = -1;  char opstack[100], sign;  int v1, v2;  int number = 0; // variable to store the parsed number  while (a[i] != '\0') {  if (isdigit(a[i])) {  number = 0;  while (isdigit(a[i])) {  number = number \* 10 + (a[i] - '0');  i++;  }  vstack[++vtop] = number;  continue; // Go to the next character in the outer loop  } else if (a[i] == '+' || a[i] == '-' || a[i] == '\*' || a[i] == '/') {  while (otop != -1 && precedence(opstack[otop]) >= precedence(a[i])) {  v2 = vstack[vtop--];  v1 = vstack[vtop--];  sign = opstack[otop--];  vstack[++vtop] = apply(v1, v2, sign);  }  opstack[++otop] = a[i];  }  i++;  }  while (otop != -1) {  v2 = vstack[vtop--];  v1 = vstack[vtop--];  sign = opstack[otop--];  vstack[++vtop] = apply(v1, v2, sign);  }  return vstack[0];  }  int main() {  char a[100];  int result;  printf("enter infix expression: ");  scanf("%s", a);  result = evaluate(a);  printf("RESULT: %d", result);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 20 | Date: 10/09/2025 |
| Program Title : Evaluate an expression using stack by converting it into postfix before evaluating. | |
| /\*Program 20 EVALUATING INFIX TO POSTFIX EXPRESSION  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 15/9/2025  \*/  #include <stdio.h>  #include <ctype.h>  #include <string.h>  #define s 20  char stack[s], expr[s], post[s];  int top = -1, j = 0;  void push(char e) { stack[++top] = e; } // Function to insert element into stack  char pop() { // Function to delete top element  if (top == -1) {  printf("Error: Stack is empty");  return '\0';  } else {  return stack[top--];  }  }  char peek() { // Function to return top element  if (top == -1) {  return '\0';  } else {  return stack[top];  }  }  int isOperator(char c) { return (c == '+' || c == '-' || c == '\*' || c == '/' || c == '^'); } // Function to check if character is operator  char chariden(char d) { // Function to identify character type  if (isalnum(d)) {  return 'n';  } else if (isOperator(d)) {  return 'o';  } else if (d == '(' || d == ')') {  return 'p';  } else {  return 'x';  }  }  int precedence(char d) { // Function to return precedence of operators  switch (d) {  case '^': return 3;  case '\*':  case '/': return 2;  case '+':  case '-': return 1;  default: return 0;  }  }  void oper(char d) { // Function to handle operator  int e, f;  e = precedence(d);  f = precedence(peek());  while (top != -1 && isOperator(peek()) && (e <= f)) {  post[j++] = pop();  f = precedence(peek());  }  push(d);  }  void para(char d) { // Function to handle parenthesis  if (d == '(') {  push(d);  } else if (d == ')') {  while (top != -1 && peek() != '(') {  post[j++] = pop();  }  if (peek() == '(')  pop(); // Remove '(' from stack  else  printf("Error: Mismatched parentheses\n");  }  }  void IntoPost(char expr[s]) { // Function to convert infix to postfix  int i;  char c, d;  for (i = 0; expr[i] != '\0'; i++) {  d = expr[i];  c = chariden(d);  switch (c) {  case 'o':  oper(d);  break;  case 'n':  post[j++] = d;  break;  case 'p':  para(d);  break;  default:  break;  }  }  while (top != -1) {  if (peek() == '(') {  printf("Error: Mismatched parentheses\n");  pop();  } else {  post[j++] = pop();  }  }  post[j] = '\0'; // Null terminate postfix string  printf("postfix Expression: %s\n", post);  }  int evaluatePostfix(char post[s]) { // function to evaluate postfix expression  int i = 0, vstack[100], vtop = -1;  int v1, v2;  while (post[i] != '\0') {  if (isdigit(post[i])) {  vstack[++vtop] = post[i] - '0'; // Convert char to int  } else if (isOperator(post[i])) {  v2 = vstack[vtop--];  v1 = vstack[vtop--];  vstack[++vtop] = apply(v1, v2, post[i]);  }  i++;  }  return vstack[vtop];  }  int main() {  printf("Enter the infix expression: ");  scanf("%s", expr);  printf("The infix expression:\n%s\n", expr);  IntoPost(expr);  printf("The postfix expression:\n%s\n", post);  printf("The evaluated result is: %d\n", evaluatePostfix(post));  return 0;  }  int main() {  printf("Enter the infix expression: ");  scanf("%s", expr);  printf("The infix expression:\n%s\n", expr);  IntoPost(expr);  printf("The postfix expression:\n%s\n", post);  printf("The evaluated result is: %d\n", evaluatePostfix(post));  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 21 | Date: 10/09/2025 |
| Program Title : A letter means push and an asterisk means pop in the following sequence. Give the sequence of values returned by the pop operations when this sequence of operations is performed on an initially empty LIFO stack. | |
| /\*Program 21 STACK ASTERISK POPPING  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 17/9/2025  \*/  #include<stdio.h>  #define size 100  char expr[size];  int top=-1;  void push(char a) { //function to insert element into stack  if (top + 1 == size)  printf("Stack is Full");  else {  expr[++top] = a;  }  }  char pop() { //function to delete top element  if (top == -1) {  printf("Stack is Empty\n");  return '\0';  } else {  printf("%c ",expr[top--]);  return expr[top];  }  }  char peek() { //function to return top element  if (top == -1){  printf("Stack is Empty");  return '\0';  }  else{  printf("Top element: %c",expr[top]);  return expr[top];  }  }  void check(char e){ //function to check and pop asterisks  if (e == '\*' ){  pop();  }  else  push(e);  }  int main(){  int i;  printf("Enter the String:");  scanf("%s",expr);  for (i = 0;expr[i] != '\0'; i++){ //traversing through the string  check(expr[i]);  }  while (top != -1){ //popping remaining elements  pop();  }  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 22 | Date: 12/09/2025 |
| Program Title : Read and display a sparse matrix. | |
| /\*Program 22 SPARSE MATRIX  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 20/09/2025  \*/  #include <stdio.h>  // Function to insert elements into matrix  void insert(int x, int y, int mat[x][y]) {  int i, j;  printf("Enter elements of the matrix:\n");  for (i = 0; i < x; i++) {  for (j = 0; j < y; j++) {  scanf("%d", &mat[i][j]);  }  }  }  // Function to display matrix  void display(int x, int y, int mat[x][y]) {  int i, j;  for (i = 0; i < x; i++) {  for (j = 0; j < y; j++) {  printf("%d ", mat[i][j]);  }  printf("\n");  }  }  // Function to convert matrix to sparse representation  int toSparse(int x, int y, int mat[x][y], int s[x\*y][3]) {  int i, j, k = 1;  s[0][0] = x;  s[0][1] = y;  for (i = 0; i < x; i++) {  for (j = 0; j < y; j++) {  if (mat[i][j] != 0) {  s[k][0] = i;  s[k][1] = j;  s[k][2] = mat[i][j];  k++;  }  }  }  s[0][2] = k - 1;  return k; // Return number of non-zero elements  }  // Function to display sparse matrix  void printsparse(int k, int s[k][3]) {  int i;  printf("Sparse Matrix Representation (row, column, value):\n");  for (i = 0; i < k; i++) {  printf("%d %d %d\n", s[i][0], s[i][1], s[i][2]);  }  }  int main() {  int x, y;  printf("Enter dimensions of matrix (rows columns): ");  scanf("%d %d", &x, &y);  int mat[x][y];  insert(x, y, mat);  printf("Matrix:\n");  display(x, y, mat);  int sparse[x \* y][3];  int k = toSparse(x, y, mat, sparse);  printf("Sparse Representation of Matrix:\n");  printsparse(k, sparse);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 23 | Date: 12/09/2025 |
| Program Title : Write a program to add two sparse matrix. | |
| /\*Program 23 SPARSE MATRIX ADDITION  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 20/09/2025  \*/  #include <stdio.h>  // Function to insert elements into matrix  void insert(int x, int y, int mat[x][y]) {  int i, j;  printf("Enter elements of the matrix:\n");  for (i = 0; i < x; i++) {  for (j = 0; j < y; j++) {  scanf("%d", &mat[i][j]);  }  }  }  // Function to convert matrix to sparse representation  int toSparse(int x, int y, int mat[x][y], int s[x \* y + 1][3]) {  int i, j, k = 1;  s[0][0] = x;  s[0][1] = y;  for (i = 0; i < x; i++) {  for (j = 0; j < y; j++) {  if (mat[i][j] != 0) {  s[k][0] = i;  s[k][1] = j;  s[k][2] = mat[i][j];  k++;  }  }  }  s[0][2] = k - 1;  return k; // Return number of non-zero elements  }  // Function to display sparse matrix  void printsparse(int k, int s[k][3]) {  int i;  printf("Sparse Matrix Representation (row, column, value):\n");  for (i = 0; i < k; i++) {  printf("%d %d %d\n", s[i][0], s[i][1], s[i][2]);  }  }  // Function to add two sparse matrices  void addSparse(int s1\_rows, int sp1[s1\_rows][3], int s2\_rows, int sp2[s2\_rows][3], int result[s1\_rows + s2\_rows][3], int \*res\_rows) {  // Both sparse1 and sparse2 have header at index 0  int i = 1, j = 1, r = 1;  // Set header  result[0][0] = sp1[0][0];  result[0][1] = sp1[0][1];  // Add non-zero elements  while (i < s1\_rows && j < s2\_rows) {  if (sp1[i][0] < sp2[j][0] || (sp1[i][0] == sp2[j][0] && sp1[i][1] < sp2[j][1])) {  result[r][0] = sp1[i][0];  result[r][1] = sp1[i][1];  result[r][2] = sp1[i][2];  i++;  r++;  } else if (sp1[i][0] > sp2[j][0] || (sp1[i][0] == sp2[j][0] && sp1[i][1] > sp2[j][1])) {  result[r][0] = sp2[j][0];  result[r][1] = sp2[j][1];  result[r][2] = sp2[j][2];  j++;  r++;  } else {  result[r][0] = sp1[i][0];  result[r][1] = sp1[i][1];  result[r][2] = sp1[i][2] + sp2[j][2];  i++;  j++;  r++;  }  }  while (i < s1\_rows) {  result[r][0] = sp1[i][0];  result[r][1] = sp1[i][1];  result[r][2] = sp1[i][2];  i++;  r++;  }  while (j < s2\_rows) {  result[r][0] = sp2[j][0];  result[r][1] = sp2[j][1];  result[r][2] = sp2[j][2];  j++;  r++;  }  result[0][2] = r - 1; // Update non-zero count in header  \*res\_rows = r; // Total rows in result sparse  printf("Addition of Sparse Matrices Result:\n");  printsparse(\*res\_rows, result);  }  int main() {  int x1, y1, x2, y2;  printf("Enter dimensions of first matrix (rows columns): ");  scanf("%d %d", &x1, &y1);  int mat1[x1][y1];  insert(x1, y1, mat1);  printf("Enter dimensions of second matrix (rows columns): ");  scanf("%d %d", &x2, &y2);  if (x1 != x2 || y1 != y2) {  printf("Error: Matrices must have the same dimensions for addition.\n");  return 1;  }  int mat2[x2][y2];  insert(x2, y2, mat2);  int sp1[x1 \* y1 + 1][3], sp2[x2 \* y2 + 1][3];  int k1 = toSparse(x1, y1, mat1, sp1);  int k2 = toSparse(x2, y2, mat2, sp2);  printf("Sparse Representation of First Matrix:\n");  printsparse(k1, sp1);  printf("Sparse Representation of Second Matrix:\n");  printsparse(k2, sp2);  int result[k1 + k2][3], k3;  addSparse(k1, sp1, k2, sp2, result, &k3);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 24 | Date: 12/09/2025 |
| Program Title : Write a program to multiply two sparse matrix. | |
| /\*Program 24 SPARSE MATRIX MULTIPLICATION  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 20/09/2025  \*/  #include <stdio.h>  // Function to insert elements into matrix  void insert(int x, int y, int mat[x][y]) {  int i, j;  printf("Enter elements of the matrix:\n");  for (i = 0; i < x; i++) {  for (j = 0; j < y; j++) {  scanf("%d", &mat[i][j]);  }  }  }  // Function to convert matrix to sparse representation  int toSparse(int x, int y, int mat[x][y], int s[x \* y + 1][3]) {  int i, j, k = 1;  s[0][0] = x;  s[0][1] = y;  for (i = 0; i < x; i++) {  for (j = 0; j < y; j++) {  if (mat[i][j] != 0) {  s[k][0] = i;  s[k][1] = j;  s[k][2] = mat[i][j];  k++;  }  }  }  s[0][2] = k - 1;  return k; // Return number of rows in sparse (including header)  }  // Function to display sparse matrix  void printsparse(int k, int s[k][3]) {  int i;  printf("Sparse Matrix Representation (row, column, value):\n");  for (i = 0; i < k; i++) {  printf("%d %d %d\n", s[i][0], s[i][1], s[i][2]);  }  }  // Function to multiply two sparse matrices (new format)  void multiplySparse(int s1\_rows, int sp1[s1\_rows][3], int s2\_rows, int sp2[s2\_rows][3], int result[sp1[0][0] \* sp2[0][1] + 1][3], int \*res\_rows) {  int i, j, r = 1;  int x1 = sp1[0][0], y1 = sp1[0][1], x2 = sp2[0][0], y2 = sp2[0][1];  // Set header for result  result[0][0] = x1;  result[0][1] = y2;  // Matrix multiplication  for (i = 0; i < x1; i++) {  for (j = 0; j < y2; j++) {  int sum = 0;  int a, b;  for (a = 1; a < s1\_rows; a++) {  if (sp1[a][0] == i) {  for (b = 1; b < s2\_rows; b++) {  if (sp2[b][0] == sp1[a][1] && sp2[b][1] == j) {  sum += sp1[a][2] \* sp2[b][2];  }  }  }  }  if (sum != 0) {  result[r][0] = i;  result[r][1] = j;  result[r][2] = sum;  r++;  }  }  }  result[0][2] = r - 1; // non-zero count  \*res\_rows = r;  printf("Multiplication of Sparse Matrices Result:\n");  printsparse(\*res\_rows, result);  }  int main() {  int x1, y1, x2, y2;  printf("Enter dimensions of first matrix (rows columns): ");  scanf("%d %d", &x1, &y1);  int mat1[x1][y1];  insert(x1, y1, mat1);  printf("Enter dimensions of second matrix (rows columns): ");  scanf("%d %d", &x2, &y2);  if (x2 != y1) {  printf("Error: Matrices must have the same dimensions for multiplication.\n");  return 1;  }  int mat2[x2][y2];  insert(x2, y2, mat2);  int sp1[x1 \* y1 + 1][3], sp2[x2 \* y2 + 1][3];  int k1 = toSparse(x1, y1, mat1, sp1);  int k2 = toSparse(x2, y2, mat2, sp2);  printf("Sparse Representation of First Matrix:\n");  printsparse(k1, sp1);  printf("Sparse Representation of Second Matrix:\n");  printsparse(k2, sp2);  int result[x1 \* y2 + 1][3], k3;  multiplySparse(k1, sp1, k2, sp2, result, &k3);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 25 | Date: 08/08/2025 |
| Program Title : Read a polynomial and display. | |
| /\*Program 25 READING POLYNOMIAL  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 08/08/2025  \*/  #include<stdio.h>  #define size 10  void read(int poly[],int deg) { // Function to read polynomial  int i;  for(i = deg; i >= 0; i--) {  printf("Enter the coefficient of x^ %d : ", i);  scanf("%d", &poly[i]);  }  }  void print(int poly[], int deg) { // Function to print polynomial  int i;  for(i = deg; i >= 0; i--) {  if(poly[i] == 0)  continue;  if(i != deg){  if(poly[i] > 0 ) {  printf(" + ");  } else if(poly[i] < 0) {  printf(" - ");  }  } else if(poly[i] < 0) {  printf("-");  }  int coeff = poly[i] < 0 ? -poly[i] : poly[i];  if(i == 0){  printf("%d", coeff);  }else if(i == 1) {  printf("%dx", coeff);  }else {  printf("%dx^%d", coeff, i);  }  }  }  int main() {  int poly[size], deg;  printf("Enter the degree of polynomial: ");  scanf("%d", &deg);  read(poly, deg);  print(poly, deg);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 26 | Date: 08/08/2025 |
| Program Title : Add two polynomials. | |
| /\*Program 26 ADDING POLYNOMIAL  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 08/08/2025  \*/  #include<stdio.h>  #define size 10  void read(int poly[],int deg) { // Function to read polynomial  int i;  for(i = deg; i >= 0; i--) {  printf("Enter the coefficient of x^ %d : ", i);  scanf("%d", &poly[i]);  }  }  void print(int poly[], int deg) { // Function to print polynomial  int i;  for(i = deg; i >= 0; i--) {  if(poly[i] == 0)  continue;  if(i != deg){  if(poly[i] > 0 ) {  printf(" + ");  } else if(poly[i] < 0) {  printf(" - ");  }  } else if(poly[i] < 0) {  printf("-");  }  int coeff = poly[i] < 0 ? -poly[i] : poly[i];  if(i == 0){  printf("%d", coeff);  }else if(i == 1) {  printf("%dx", coeff);  }else {  printf("%dx^%d", coeff, i);  }  }  }  // Function to add two polynomials  void sum(int poly1[], int poly2[], int sum[], int deg1, int deg2){  int i, deg, a, b;  deg = deg1 > deg2 ? deg1 : deg2;  // Zero-initialize arrays up to deg  for(i = 0; i <= deg; i++) {  a = (i <= deg1) ? poly1[i] : 0;  b = (i <= deg2) ? poly2[i] : 0;  sum[i] = a + b;  }  printf("\nSum of polynomial 1 and polynomial 2 is: \n");  print(sum, deg);  }  int main() {  int poly1[size], poly2[size], s[size], deg1, deg2;  printf("Enter the degree of polynomial 1: ");  scanf("%d", &deg1);  read(poly1, deg1);  printf("Enter the degree of polynomial 2: ");  scanf("%d", &deg2);  read(poly2, deg2);  sum(poly1, poly2, s, deg1, deg2);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 27 | Date: 08/08/2025 |
| Program Title : Subtract two polynomials. | |
| /\*Program 27 SUBTRACTING POLYNOMIAL  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 08/08/2025  \*/  #include<stdio.h>  #define size 10  void read(int poly[],int deg) { // Function to read polynomial  int i;  for(i = deg; i >= 0; i--) {  printf("Enter the coefficient of x^ %d : ", i);  scanf("%d", &poly[i]);  }  }  void print(int poly[], int deg) { // Function to print polynomial  int i;  for(i = deg; i >= 0; i--) {  if(poly[i] == 0)  continue;  if(i != deg){  if(poly[i] > 0 ) {  printf(" + ");  } else if(poly[i] < 0) {  printf(" - ");  }  } else if(poly[i] < 0) {  printf("-");  }  int coeff = poly[i] < 0 ? -poly[i] : poly[i];  if(i == 0){  printf("%d", coeff);  }else if(i == 1) {  printf("%dx", coeff);  }else {  printf("%dx^%d", coeff, i);  }  }  }  // Function to subtract two polynomials  void diff(int poly1[],int poly2[],int diff[],int deg1,int deg2){  int i, deg, a, b;  deg = deg1 > deg2 ? deg1 : deg2;  for(i = 0; i <= deg; i++) {  a = (i <= deg1) ? poly1[i] : 0;  b = (i <= deg2) ? poly2[i] : 0;  diff[i] = a - b;  }  printf("\nDifference of polynomial 1 and polynomial 2 is: \n");  print(diff, deg);  }  int main(){  int poly1[size],poly2[size],d[size],deg1,deg2;  int i;  for(i = 0; i < size; i++) {  poly1[i] = 0;  poly2[i] = 0;  d[i] = 0;  }  printf("Enter the degree of polynomial 1: ");  scanf("%d", &deg1);  read(poly1, deg1);  printf("Enter the degree of polynomial 2: ");  scanf("%d", &deg2);  read(poly2, deg2);  diff(poly1, poly2, d, deg1, deg2);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 28 | Date: 08/08/2025 |
| Program Title : Multiply two polynomials. | |
| /\*Program 28 MULTIPLYING POLYNOMIAL  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 13/08/2025  \*/  #include<stdio.h>  #define size 10  void read(int poly[],int deg) { // Function to read polynomial  int i;  for(i = deg; i >= 0; i--) {  printf("Enter the coefficient of x^ %d : ", i);  scanf("%d", &poly[i]);  }  }  void print(int poly[], int deg) { // Function to print polynomial  int i;  for(i = deg; i >= 0; i--) {  if(poly[i] == 0)  continue;  if(i != deg){  if(poly[i] > 0 ) {  printf(" + ");  } else if(poly[i] < 0) {  printf(" - ");  }  } else if(poly[i] < 0) {  printf("-");  }  int coeff = poly[i] < 0 ? -poly[i] : poly[i];  if(i == 0){  printf("%d", coeff);  }else if(i == 1) {  printf("%dx", coeff);  }else {  printf("%dx^%d", coeff, i);  }  }  }  // Function to multiply two polynomials  void multiply(int poly1[], int poly2[], int prod[], int deg1, int deg2) {  int i, j;  // Initialize product array to zero  for(i = 0; i <= deg1 + deg2; i++) {  prod[i] = 0;  }  // Multiply polynomials  for(i = 0; i <= deg1; i++) {  for(j = 0; j <= deg2; j++) {  prod[i + j] += poly1[i] \* poly2[j];  }  }  printf("\nProduct of polynomial 1 and polynomial 2 is: \n");  print(prod, deg1 + deg2);  }  int main() {  int poly1[size], poly2[size], p[size \* 2], deg1, deg2;  int i;  for(i = 0; i < size; i++) {  poly1[i] = 0;  poly2[i] = 0;  p[i] = 0;  }  printf("Enter the degree of polynomial 1: ");  scanf("%d", &deg1);  read(poly1, deg1);  printf("Enter the degree of polynomial 2: ");  scanf("%d", &deg2);  read(poly2, deg2);  printf("\nPolynomial 1 is: \n");  print(poly1, deg1);  printf("\nPolynomial 2 is: \n");  print(poly2, deg2);  multiply(poly1, poly2, p, deg1, deg2);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 29 | Date: 08/08/2025 |
| Program Title : Demonstrate queue using array. | |
| /\*Program 29 QUEUE  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 09/08/2025  \*/  #include<stdio.h>  int q[5],front=-1,rear=-1;  void enqueue(int v) {  if (rear == 4) {  printf("Error:Queue is Full.\n");  } else {  if (front == -1)  front = 0;  q[++rear] = v;  printf("Enqueued\n");  }  }  void dequeue() {  if (front == -1 || front > rear) {  printf("Error: Empty Queue.\n");  } else {  printf("Disqueued Element: %d\n", q[front]);  front++;  if (rear < front)  front = rear = -1;  }  }  void display() {  if (front == -1 || rear == -1 || front > rear) {  printf("Error: Empty Queue.\n");  } else {  int i;  for (i = front; i <= rear; i++) {  printf("%d \t", q[i]);  }  printf("\n");  }  }  int menu() { //function for menu  int ch;  printf("\n ENQUEUE-1 \n DEQUEUE-2 \n DISPLAY-3 \n EXIT -4 \n Enter your choice : ");  scanf("%d", &ch);  return ch;  }  void processQueue() { //working of menu  int ch, b;  for (ch = menu(); ch != 4; ch = menu()) {  switch (ch) {  case 1:  printf("Enter the value to insert : ");  scanf("%d", &b);  enqueue(b);  break;  case 2:  dequeue();  break;  case 3:  display();  break;  default:  printf("Error: Wrong Choice");  break;  }  }  }  int main() {  processQueue();  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 30 | Date: 08/08/2025 |
| Program Title : Program to demonstrate circular queue using array. | |
| /\*Program 30 CIRCULAR QUEUE  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 09/08/2025  \*/  #include<stdio.h>  int q[5],front=-1,rear=-1;  void enqueue(int v) {  if ((rear+1)%5 == front) {  printf("Error:Queue is Full.\n");  } else {  if (front == -1)  front = 0;  rear = (rear+1)%5;  q[rear] = v;  printf("Enqueued\n");  }  }  void dequeue() {  if (front == -1) {  printf("Error: Empty Queue.\n");  } else {  printf("Disqueued Element: %d\n", q[front]);  if (front == rear) {  front = rear = -1;  } else {  front = (front+1)%5;  }  }  }  void display() {  if (front == -1) {  printf("Error: Empty Queue.\n");  } else {  int i;  for (i = front;; i = (i+1)%5) {  printf("%d \t", q[i]);  if (i == rear)  break;  }  printf("\n");  }  }  int menu() { //function for menu  int ch;  printf("\n ENQUEUE-1 \n DEQUEUE-2 \n DISPLAY-3 \n EXIT -4 \n Enter your choice : ");  scanf("%d", &ch);  return ch;  }  void processQueue() { //working of menu  int ch, b;  for (ch = menu(); ch != 4; ch = menu()) {  switch (ch) {  case 1:  printf("Enter the value to insert : ");  scanf("%d", &b);  enqueue(b);  break;  case 2:  dequeue();  break;  case 3:  display();  break;  default:  printf("Error: Wrong Choice");  break;  }  }  }  int main() {  processQueue();  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 31 | Date: 08/08/2025 |
| Program Title : Implement heapsort. | |
| /\*Program 31 HEAPSORT  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 10/08/2025  \*/  #include <stdio.h>  #define MAX 100  int heap[MAX], size = 0;  void swap(int i, int j) //function to swap two elements  {  int temp = heap[i];  heap[i] = heap[j];  heap[j] = temp;  }  void heapify(int i, int n) //function to maintain heap property  {  int largest = i;  int left = 2 \* i + 1;  int right = 2 \* i + 2;  if (left < n && heap[left] > heap[largest])  largest = left;  if (right < n && heap[right] > heap[largest])  largest = right;  if (largest != i) {  swap(i, largest);  heapify(largest, n);  }  }  void heapSort() //function to perform heap sort  {  int i;  for (i = size / 2 - 1; i >= 0; i--)  heapify(i, size);  for (i = size - 1; i > 0; i--) {  swap(0, i);  heapify(0, i);  }  }  void display() //function to display the array  {  int i;  for (i = 0; i < size; i++)  printf("%d ", heap[i]);  printf("\n");  }  int main()  {  printf("Enter number of elements: ");  scanf("%d", &size);  printf("Enter %d elements:\n", size);  int i;  for (i = 0; i < size; i++)  scanf("%d", &heap[i]);  printf("Before Heap Sort :\n");  display();  heapSort();  printf("After Heap Sort:\n");  display();  return 0;  } | |
| Output | |
| Program No: 32 | Date: 08/08/2025 |
| Program Title : Program to demonstrate a priority queue using an array. | |
| /\*Program 32 PRIORITY QUEUE  @ALBIN MAMMEN MATHEW  Roll No: 08  Date: 10/08/2025  \*/  #include <stdio.h>  #define SIZE 5 // Maximum size of the queue  int values[SIZE]; // Array for values  int priorities[SIZE]; // Array for priorities  int count = 0; // Number of elements in the queue  // Function to insert an element with priority  void enqueue(int value, int priority) {  if (count == SIZE) {  printf("Queue is Full! (Overflow)\n");  return;  }  values[count] = value;  priorities[count] = priority;  count++;  printf("Inserted value %d with priority %d\n", value, priority);  }  // Function to find index of highest priority element  int findHighestPriorityIndex() {  int highest = 0;  int i;  for (i = 1; i < count; i++) {  if (priorities[i] > priorities[highest]) {  highest = i;  }  // If priorities are equal, the one inserted earlier stays first  }  return highest;  }  // Function to delete the highest priority element  void dequeue() {  if (count == 0) {  printf("Queue is Empty! (Underflow)\n");  return;  }  int index = findHighestPriorityIndex();  printf("Deleted value %d with priority %d\n", values[index], priorities[index]);  // Shift remaining elements  int i;  for (i = index; i < count - 1; i++) {  values[i] = values[i + 1];  priorities[i] = priorities[i + 1];  }  count--;  }  // Function to display all elements in the queue  void display() {  if (count == 0) {  printf("Queue is Empty!\n");  return;  }  printf("Priority Queue elements:\n");  int i;  for (i = 0; i < count; i++) {  printf("Value: %d | Priority: %d\n", values[i], priorities[i]);  }  }  int main() {  int choice, value, priority;  printf("Priority Queue Implementation using Array\n");  while (1) {  printf("\nMenu:\n");  printf("1. Enqueue (Insert)\n");  printf("2. Dequeue (Delete Highest Priority)\n");  printf("3. Display\n");  printf("4. Exit\n");  printf("Enter your choice: ");  scanf("%d", &choice);  switch (choice) {  case 1:  printf("Enter value: ");  scanf("%d", &value);  printf("Enter priority (higher number = higher priority): ");  scanf("%d", &priority);  enqueue(value, priority);  break;  case 2:  dequeue();  break;  case 3:  display();  break;  case 4:  printf("Exiting program...\n");  return 0;  default:  printf("Invalid choice! Try again.\n");  }  }  return 0;  } | |
| Output | |
| Program No: 33 | Date: 19/09/2025 |
| Program Title : Define a structure for dates with dd/mm/yyyy. Provide functions for reading, displaying and comparing two dates are equal or not. | |
| /\*PROGRAM 33 DATE USING STRUCTURE  @ALBIN MAMMEN MATHEW  Roll No: 08  19/09/2025  \*/  #include<stdio.h>  struct date { //Structure to represent the date  int day, month, year;  };  typedef struct date date;  void read(date \*d) { //Function to insert the date  printf("\nEnter the date (dd-mm-yyy) : ");  scanf("%d-%d-%d", &d->day, &d->month, &d->year);  }  void display(date d){ //Function to display the date  printf("Given Date is %d/%d/%d \n",d.day,d.month,d.year);  }  void comparedate(date d, date e) { //Function to compare the dates  if (d.day == e.day && d.month == e.month && d.year == e.year)  printf("Dates are same.\n");  else  printf("The dates are not same.\n");  }  int main () {  date d1, d2;  read(&d1);  display(d1);  read(&d2);  display(d2);  comparedate(d1,d2);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 34 | Date: 19/09/2025 |
| Program Title : Define a structure for employees with eno,ename, esal and dno. Read  n employees information and provide functions for the following:  a.  Searching an employee by no  b.  Sorting the employees by                i. Name  ii. Salary  c.   Deleting an employee | |
| /\*PROGRAM 34 EMPLOYEE MANAGEMENT USING STRUCTURE  @ALBIN MAMMEN MATHEW  Roll No: 08  19/09/2025  \*/  #include <stdio.h>  #define max 100  struct employees { //Structure to represent employee details  int eno, dno;  char ename[20];  float esal;  };  typedef struct employees emp;  emp emplist[max]; //array to store employee details  int count = 0; //Count of employees in the data  void insert(){ //function to insert employee details  printf("Enter the employee no :");  scanf("%d",&emplist[count].eno);  printf("Enter the employee name :");  scanf("%s",emplist[count].ename);  printf("Enter the employee salary :");  scanf("%f",&emplist[count].esal);  printf("Enter the department no :");  scanf("%d",&emplist[count].dno);  printf("Data inserted\n");  count++;  }  void delete(){ //function to delete an employee  int del, i, found = 0;  printf("\nEnter employee number to delete: ");  scanf("%d", &del);  for (i = 0; i < count; i++) {  if (emplist[i].eno == del) {  found = 1;  break;  }  }  if (found) {  for (; i < count - 1; i++) {  emplist[i] = emplist[i + 1];  }  count--;  printf("Employee deleted.\n");  } else {  printf("Employee not found.\n");  }  }  void display(){ //function to display the details of all employees  int i;  if (count == 0) {  printf("\nNo employees to display.\n");  return;  }  printf("\n%-10s %-20s %-10s %-10s\n", "EmpNo", "Name", "Salary", "DeptNo");  printf("------------------------------------------------------\n");  for (i = 0; i < count; i++) {  printf("%-10d %-20s %-10.2f %-10d\n",  emplist[i].eno, emplist[i].ename, emplist[i].esal, emplist[i].dno);  }  }  void search(){ //function to search for a employee using Employee number  int no, i, found = 0;  printf("\nEnter employee number to search: ");  scanf("%d", &no);  for (i = 0; i < count; i++) {  if (emplist[i].eno == no) {  found = 1;  break;  }  }  if (found) {  printf("%-10d %-20s %-10.2f %-10d\n", emplist[i].eno, emplist[i].ename, emplist[i].esal, emplist[i].dno);  } else {  printf("Employee not found.\n");  }  }  void sort(){ //function to sort employees by either name or salary  printf("1. Sort by Name\n2. Sort by Salary\n");  int ch, i, j;  scanf("%d",&ch);  if (ch == 1){  emp temp;  for(i = 0; i < count; i++) {  for (j = i + 1; j < count; j++) {  if (strcmp(emplist[i].ename, emplist[j].ename) > 0) {  temp = emplist[i];  emplist[i] = emplist[j];  emplist[j] = temp;  }  }  }  }  else if (ch == 2) {  emp temp;  for(i = 0; i < count; i++) {  for (j = i + 1; j < count; j++) {  if (emplist[i].esal > emplist[j].esal) {  temp = emplist[i];  emplist[i] = emplist[j];  emplist[j] = temp;  }  }  }  }  printf("Employees sorted.\n");  display();  }  int menu(){  int ch;  printf("INSERT - 1\nDELETE - 2\nDISPLAY - 3\nSEARCH - 4\nSORT - 5\nEXIT - 6\n");  printf("Enter your choice: ");  scanf("%d",&ch);  return ch;  }  void processArray(){  int ch, i, a, size;  for(ch = menu(); ch != 6; ch = menu()){  switch (ch) {  case 1:  printf("Number of employees to be inserted: ");  scanf("%d",&size);  for(i = 0;i < size; i++){  insert();  }  break;  case 2 :  delete();  break;  case 3 :  display();  break;  case 4 :  search();  break;  case 5 :  sort();  break;  default :  printf("Invalid choice.\n");  break;  }  }  }  int main(){  printf("EMPLOYEE DETAILS \n");  processArray();  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 35 | Date: 26/09/2025 |
| Program Title : Read a polynomial and display it using structure array | |
| /\* PROGRAM 35: READ AND DISPLAY POLYNOMIAL USING STRUCTURE  @ALBIN MAMMEN MATHEW  Roll No: 08  15/10/2025  \*/  #include <stdio.h>  #include <stdlib.h>  // Structure to represent a term in the polynomial  typedef struct poly {  int coeff; // Coefficient of the term  int exponen; // Exponent of the term  } term;  term a[10]; // Array to store polynomial terms  int n; // Number of terms in the polynomial  // Function to read polynomial terms from user  void read() {  int i;  printf("\nEnter the number of terms: ");  scanf("%d", &n);  printf("Enter the polynomial terms:\n");  for (i = 0; i < n; i++) {  printf("Term %d:\n", i + 1);  printf("\tCoefficient: ");  scanf("%d", &a[i].coeff);  printf("\tExponent: ");  scanf("%d", &a[i].exponen);  }  }  // Function to print the polynomial in readable format  void print() {  int c, e;  printf("\nPolynomial: ");  int i;  for (i = 0; i < n; i++) {  c = a[i].coeff; // Get coefficient  e = a[i].exponen; // Get exponent  if (c == 0) // Skip terms with zero coefficient  continue;  int abs\_c = abs(c); // Absolute value of coefficient  // Print sign for terms except the first  if (i != 0) {  if (c > 0)  printf("+");  else  printf("-");  } else {  if (c < 0)  printf("-");  }  // Print term based on exponent value  if (e == 0) {  printf("%d", abs\_c); // Constant term  } else if (e == 1) {  if (abs\_c == 1)  printf("x"); // x term with coefficient 1  else  printf("%dx", abs\_c); // x term with coefficient not 1  } else {  if (abs\_c == 1)  printf("x^%d", e); // x^e term with coefficient 1  else  printf("%dx^%d", abs\_c, e); // x^e term with coefficient not 1  }  }  printf("\n");  }  int main() {  read(); // Read polynomial from user  print(); // Print the polynomial  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 36 | Date: 26/09/2025 |
| Program Title : Add two polynomials using structure array. | |
| /\* PROGRAM 36: POLYNOMIAL ADDITION USING STRUCTURE ARRAY  @ALBIN MAMMEN MATHEW  Roll No: 08  15/10/2025  \*/  #include <stdio.h>  #include <stdlib.h>  typedef struct {  int coeff;  int expo;  } poly;  poly a[10], b[10], add[10];  int n1, n2, nr;  // Function to display a polynomial  void display(poly p[], int \*n) {  int i;  for (i = 0; i < \*n; i++) {  int c = p[i].coeff;  int e = p[i].expo;  if (c == 0)  continue; // Skip zero coefficient terms  // Sign printing  if (i != 0) {  if (c > 0)  printf("+");  else  printf("-");  } else {  if (c < 0)  printf("-");  }  int ab = abs(c); // Absolute value for printing  // Coefficient & exponent printing  if (e == 0)  printf("%d", ab);  else if (e == 1) {  if (ab == 1)  printf("x");  else  printf("%dx", ab);  } else {  if (ab == 1)  printf("x^%d", e);  else  printf("%dx^%d", ab, e);  }  }  printf("\n");  }  // Function to read the polynomials  void read() {  printf("Enter the number of terms of first polynomial: ");  scanf("%d", &n1);  int i;  for (i = 0; i < n1; i++) {  printf("Term %d (coeff exponent): ", i + 1);  scanf("%d %d", &a[i].coeff, &a[i].expo);  }  printf("First Polynomial: ");  display(a, &n1);  printf("\nEnter the number of terms of second polynomial: ");  scanf("%d", &n2);  for (i = 0; i < n2; i++) {  printf("Term %d (coeff exponent): ", i + 1);  scanf("%d %d", &b[i].coeff, &b[i].expo);  }  printf("Second Polynomial: ");  display(b, &n2);  }  // Function to add two polynomials  void addpoly() {  int i = 0, j = 0, k = 0;  while (i < n1 && j < n2) {  if (a[i].expo > b[j].expo)  add[k++] = a[i++];  else if (a[i].expo < b[j].expo)  add[k++] = b[j++];  else {  add[k].coeff = a[i].coeff + b[j].coeff;  add[k].expo = a[i].expo;  k++;  i++;  j++;  }  }  // Copy remaining terms  while (i < n1)  add[k++] = a[i++];  while (j < n2)  add[k++] = b[j++];  nr = k;  printf("\nResultant Polynomial after Addition: ");  display(add, &nr);  }  int main() {  read();  addpoly();  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 37 | Date: 26/09/2025 |
| Program Title : Subtract two polynomials using structure array. | |
| /\* PROGRAM 37: POLYNOMIAL SUBTRACTION USING STRUCTURE ARRAY  @ALBIN MAMMEN MATHEW  Roll No: 08  15/10/2025  \*/  #include <stdio.h>  #include <stdlib.h>  typedef struct {  int coeff;  int expo;  } term;  term a[10], b[10], diff[10];  int n1, n2;  // Function to display a polynomial  void display(term p[], int \*n) {  int i;  for (i = 0; i < \*n; i++) {  int c = p[i].coeff;  int e = p[i].expo;  if (c == 0)  continue; // Skip zero coefficient terms  int ab = abs(c);  // Print sign  if (i != 0) {  if (c > 0)  printf("+");  else  printf("-");  } else {  if (c < 0)  printf("-");  }  // Print coefficient and exponent  if (e == 0)  printf("%d", ab);  else if (e == 1) {  if (ab == 1)  printf("x");  else  printf("%dx", ab);  } else {  if (ab == 1)  printf("x^%d", e);  else  printf("%dx^%d", ab, e);  }  }  printf("\n");  }  // Function to read two polynomials  void read() {  printf("Enter the number of terms of first polynomial: ");  scanf("%d", &n1);  int i;  for (i = 0; i < n1; i++) {  printf("Term %d (coeff exponent): ", i + 1);  scanf("%d %d", &a[i].coeff, &a[i].expo);  }  printf("First Polynomial: ");  display(a, &n1);  printf("\nEnter the number of terms of second polynomial: ");  scanf("%d", &n2);  for (i = 0; i < n2; i++) {  printf("Term %d (coeff exponent): ", i + 1);  scanf("%d %d", &b[i].coeff, &b[i].expo);  }  printf("Second Polynomial: ");  display(b, &n2);  }  // Function to perform polynomial subtraction  void subtraction() {  term negative[10];  int i, j, k = 0;  // Negate coefficients of second polynomial  for (i = 0; i < n2; i++) {  negative[i].coeff = -b[i].coeff;  negative[i].expo = b[i].expo;  }  i = 0;  j = 0;  // Subtract like terms (similar to addition)  while (i < n1 && j < n2) {  if (a[i].expo > negative[j].expo)  diff[k++] = a[i++];  else if (a[i].expo < negative[j].expo)  diff[k++] = negative[j++];  else {  diff[k].coeff = a[i].coeff + negative[j].coeff;  diff[k].expo = a[i].expo;  i++;  j++;  k++;  }  }  // Add remaining terms  while (i < n1)  diff[k++] = a[i++];  while (j < n2)  diff[k++] = negative[j++];  printf("\nResultant Polynomial after Subtraction: ");  display(diff, &k);  }  // Main function  int main() {  read();  subtraction();  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 38 | Date: 26/09/2025 |
| Program Title : Multiply two polynomials using structure array. | |
| /\* PROGRAM 38: POLYNOMIAL MULTIPLICATION USING STRUCTURE ARRAY  @ALBIN MAMMEN MATHEW  Roll No: 08  15/10/2025  \*/  #include <stdio.h>  #include <stdlib.h>  typedef struct {  int coeff;  int expo;  } poly;  poly a[10], b[10], prod[20]; // increased size for product terms  int n1, n2, nr;  // Function to display a polynomial  void display(poly p[], int \*n) {  int i;  for (i = 0; i < \*n; i++) {  int c = p[i].coeff;  int e = p[i].expo;  if (c == 0)  continue; // skip zero coeff  // Sign printing  if (i != 0) {  if (c > 0)  printf("+");  else  printf("-");  } else {  if (c < 0)  printf("-");  }  int ab = abs(c);  // Coefficient & exponent printing  if (e == 0) {  printf("%d", ab);  } else if (e == 1) {  if (ab == 1)  printf("x");  else  printf("%dx", ab);  } else {  if (ab == 1)  printf("x^%d", e);  else  printf("%dx^%d", ab, e);  }  }  printf("\n");  }  // Function to read polynomials  void read() {  int i;  printf("Enter the number of terms of first polynomial: ");  scanf("%d", &n1);  for (i = 0; i < n1; i++) {  printf("Term %d (coeff exponent): ", i + 1);  scanf("%d %d", &a[i].coeff, &a[i].expo);  }  printf("First Polynomial: ");  display(a, &n1);  printf("\nEnter the number of terms of second polynomial: ");  scanf("%d", &n2);  for (i = 0; i < n2; i++) {  printf("Term %d (coeff exponent): ", i + 1);  scanf("%d %d", &b[i].coeff, &b[i].expo);  }  printf("Second Polynomial: ");  display(b, &n2);  }  // Function to multiply polynomials  void multiplication() {  int i, j, l, k = 0;  // Multiply each term of a with each term of b  for (i = 0; i < n1; i++) {  for (j = 0; j < n2; j++) {  prod[k].coeff = a[i].coeff \* b[j].coeff;  prod[k].expo = a[i].expo + b[j].expo;  k++;  }  }  nr = k;  // Combine like terms (same exponent)  for (i = 0; i < nr; i++) {  for (j = i + 1; j < nr; j++) {  if (prod[i].expo == prod[j].expo) {  prod[i].coeff += prod[j].coeff;  // shift left  for (l = j; l < nr - 1; l++) {  prod[l] = prod[l + 1];  }  nr--;  j--; // recheck same index  }  }  }  // Sort terms in descending order of exponent  for (i = 0; i < nr - 1; i++) {  for (j = i + 1; j < nr; j++) {  if (prod[i].expo < prod[j].expo) {  poly temp = prod[i];  prod[i] = prod[j];  prod[j] = temp;  }  }  }  printf("\nResultant Polynomial after Multiplication: ");  display(prod, &nr);  }  int main() {  read();  multiplication();  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 39 | Date: 26/09/2025 |
| Program Title : Implement a) malloc , b) calloc and c) free functions. | |
| /\* PROGRAM 39: DYNAMIC MEMORY ALLOCATION FUNCTIONS  @ALBIN MAMMEN MATHEW  Roll No: 08  03/10/2025  \*/  #include <stdio.h>  #include <stdlib.h>  void implement() {  int \*ptr;  int i,n;  printf("\nEnter the number of allocations you need: ");  scanf("%d", &n);  // Allocating memory using malloc  ptr = (int \*)malloc(n \* sizeof(int));  printf("\nEnter the elements:\n");  for (i = 0; i < n; i++) {  scanf("%d", &ptr[i]);  }  printf("\nPrinting the elements (implementation of malloc):\n");  for (i = 0; i < n; i++) {  printf("%d\t", ptr[i]);  }  free(ptr); // Used to free the memory  // Allocating memory using calloc  ptr = (int \*)calloc(n, sizeof(int));  printf("\n\nPrinting initial elements in memory (after allocating using calloc):\n");  for (i = 0; i < n; i++) {  printf("%d\t", ptr[i]);  }  free(ptr); // Free the memory  }  int main() {  implement();  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 40 | Date: 26/09/2025 |
| Program Title : Use malloc to read n integers and find the mean. | |
| /\* PROGRAM 40: MEAN OF n NUMBERS USING MALLOC  @ALBIN MAMMEN MATHEW  Roll No: 08  03/10/2025  \*/  #include <stdio.h>  #include <stdlib.h>  void mean() {  int i,n;  float sum = 0;  float mean = 0;  int \*ptr;  printf("Enter the number of terms: ");  scanf("%d", &n);  // Allocate memory dynamically  ptr = (int \*)malloc(n \* sizeof(int));  printf("\nEnter the numbers:\n");  for (i = 0; i < n; i++) {  scanf("%d", &ptr[i]);  }  // Calculate sum  for (i = 0; i < n; i++) {  sum += ptr[i];  }  // Calculate mean  mean = sum / n;  printf("\nThe mean is: %.2f\n", mean);  // Free the allocated memory  free(ptr);  }  int main() {  mean();  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 41 | Date: 26/09/2025 |
| Program Title : Use calloc to read n numbers and find the mode. | |
| /\* PROGRAM 41: MODE OF n NUMBERS USING CALLOC  @ALBIN MAMMEN MATHEW  Roll No: 08  03/10/2025  \*/  #include <stdio.h>  #include <stdlib.h>  void mode() {  int i,j,n;  int max = 0, mode = 0;  int \*ptr;  printf("Enter the number of terms: ");  scanf("%d", &n);  // Allocate memory dynamically using calloc  ptr = (int \*)calloc(n, sizeof(int));  printf("\nEnter the numbers:\n");  for (i = 0; i < n; i++) {  scanf("%d", &ptr[i]);  }  // Find the mode  for (i = 0; i < n; i++) {  int count = 1;  for (j = i + 1; j < n; j++) {  if (ptr[i] == ptr[j]) {  count++;  }  }  if (count > max) {  max = count;  mode = ptr[i];  }  }  printf("\nMode = %d (appears %d times)\n", mode, max);  // Free the allocated memory  free(ptr);  }  int main() {  mode();  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 42 | Date: 26/09/2025 |
| Program Title : Declare a structure for Books having author\_name and book\_name. Create an array of books using a pointer variable. Provide functions for reading n books and displaying the same using pointers. | |
| /\* PROGRAM 42: BOOK MANAGEMENT - STRUCTURE ARRAY - READ AND DISPLAY USING POINTERS  @ALBIN MAMMEN MATHEW  Roll No: 08  03/10/2025  \*/  #include <stdio.h>  #include <stdlib.h>  struct Book {  char bname[50];  char aname[50];  float price;  };  // Function to read n books  void readBooks(struct Book \*b, int n) {  int i;  for (i = 0; i < n; i++) {  printf("\nEnter details of Book %d\n", i + 1);  printf("Book Name: ");  scanf("%s", (b + i)->bname);  printf("Author Name: ");  scanf("%s", (b + i)->aname);  printf("Price: ");  scanf("%f", &(b + i)->price);  }  }  // Function to display n books  void displayBooks(struct Book \*b, int n) {  int i;  printf("\n--- Book Details ---\n");  for (i = 0; i < n; i++) {  printf("\nBook %d\n", i + 1);  printf("Book Name: %s\n", (b + i)->bname);  printf("Author Name: %s\n", (b + i)->aname);  printf("Price: %.2f\n", (b + i)->price);  }  }  int main() {  struct Book \*b;  int n;  printf("----- Struct Pointer Variable -----\n");  printf("Enter number of books: ");  scanf("%d", &n);  // Dynamically allocate memory for n books  b = (struct Book \*)malloc(n \* sizeof(struct Book));  if (b == NULL) {  printf("Memory allocation failed!\n");  return 0;  }  // Read and display  readBooks(b, n);  displayBooks(b, n);  // Free memory  free(b);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 43 | Date: 26/09/2025 |
| Program Title : Use realloc to implement varchar for any length. | |
| /\* PROGRAM 43: IMPLEMENTING VARCHAR USING REALLOC  @ALBIN MAMMEN MATHEW  Roll No: 08  03/10/2025  \*/  #include <stdio.h>  #include <stdlib.h>  void read() {  char \*ptr;  int size = 10; // default initial size  int len = 0;  getchar(); // consume leftover newline if needed  ptr = (char \*)malloc(size \* sizeof(char));  if (ptr == NULL) {  printf("Memory allocation failed!\n");  exit(1);  }  char ch;  // Read characters until newline  while ((ch = getchar()) != '\n') {  ptr[len++] = ch;  // If memory is full, double the size  if (len == size) {  size \*= 2;  char \*temp = (char \*)realloc(ptr, size \* sizeof(char));  if (temp == NULL) {  printf("Memory reallocation failed!\n");  free(ptr);  exit(1);  }  ptr = temp;  }  }  // Null-terminate the string  ptr[len] = '\0';  printf("You entered: %s\n", ptr);  free(ptr);  }  int main() {  printf("----- Dynamic Input Using Realloc -----\n");  read();  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 44 | Date: 15/09/2025 |
| Program Title : Demonstrate a linked list creation and display. | |
| /\* PROGRAM 44: LINKED LIST CREATION AND DISPLAY  @ALBIN MAMMEN MATHEW  Roll No: 08  18/09/2025  \*/  #include <stdio.h>  #include <stdlib.h>  // Define the node structure  struct Node {  int data;  struct Node \*next;  };  // Function to create a linked list of n nodes  struct Node\* createLinkedList(int n) {  struct Node \*head = NULL, \*temp = NULL, \*newNode = NULL;  int value,i;  for (i = 0; i < n; i++) {  newNode = (struct Node \*)malloc(sizeof(struct Node));  if (newNode == NULL) {  printf("Memory not allocated.\n");  exit(1);  }  printf("Enter value for node %d: ", i + 1);  scanf("%d", &value);  newNode->data = value;  newNode->next = NULL;  if (head == NULL) {  head = newNode;  temp = head;  } else {  temp->next = newNode;  temp = newNode;  }  }  return head;  }  // Function to display linked list  void displayLinkedList(struct Node \*head) {  struct Node \*temp = head;  printf("\nLinked List: ");  while (temp != NULL) {  printf("%d -> ", temp->data);  temp = temp->next;  }  printf("NULL\n");  }  // Function to free memory of linked list  void freeLinkedList(struct Node \*head) {  struct Node \*temp;  while (head != NULL) {  temp = head;  head = head->next;  free(temp);  }  printf("Memory freed successfully.\n");  }  int main() {  int n;  struct Node \*head;  printf("Enter number of nodes: ");  scanf("%d", &n);  head = createLinkedList(n); // Step 1: Create linked list  displayLinkedList(head); // Step 2: Display linked list  freeLinkedList(head); // Step 3: Free memory  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 45 | Date: 15/09/2025 |
| Program Title : Write a program with functions to insert a new node  a. at the beginning of a Singly Linked List.  b. At the end of the linked list  c. after a specified element in a linked list. | |
| /\* PROGRAM 45: LINKED LIST INSERTION  @ALBIN MAMMEN MATHEW  Roll No: 08  18/09/2025  \*/  #include <stdio.h>  #include <stdlib.h>  // Node structure  struct Node {  int data;  struct Node \*next;  };  // Function to create initial linked list with n nodes  struct Node\* createLinkedList(int n) {  struct Node \*head = NULL, \*temp = NULL, \*newNode = NULL;  int value,i;  for (i = 0; i < n; i++) {  newNode = (struct Node \*)malloc(sizeof(struct Node));  printf("Enter value for node %d: ", i + 1);  scanf("%d", &value);  newNode->data = value;  newNode->next = NULL;  if (head == NULL) {  head = newNode;  temp = head;  } else {  temp->next = newNode;  temp = newNode;  }  }  return head;  }  // Function to insert at beginning  struct Node\* insertAtBeginning(struct Node \*head, int value) {  struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));  newNode->data = value;  newNode->next = head;  head = newNode;  return head;  }  // Function to insert at end  struct Node\* insertAtEnd(struct Node \*head, int value) {  struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));  newNode->data = value;  newNode->next = NULL;  if (head == NULL) {  return newNode;  }  struct Node \*temp = head;  while (temp->next != NULL)  temp = temp->next;  temp->next = newNode;  return head;  }  // Function to insert after a specified element  struct Node\* insertAfterElement(struct Node \*head, int element, int value) {  struct Node \*temp = head;  while (temp != NULL && temp->data != element)  temp = temp->next;  if (temp == NULL) {  printf("Element %d not found in the list.\n", element);  return head;  }  struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));  newNode->data = value;  newNode->next = temp->next;  temp->next = newNode;  return head;  }  // Function to display linked list  void displayLinkedList(struct Node \*head) {  struct Node \*temp = head;  printf("\nLinked List: ");  while (temp != NULL) {  printf("%d -> ", temp->data);  temp = temp->next;  }  printf("NULL\n");  }  // Function to free linked list memory  void freeLinkedList(struct Node \*head) {  struct Node \*temp;  while (head != NULL) {  temp = head;  head = head->next;  free(temp);  }  printf("Memory freed successfully.\n");  }  int main() {  int n, value, element;  struct Node \*head = NULL;  printf("Enter number of nodes to create initially: ");  scanf("%d", &n);  head = createLinkedList(n);  displayLinkedList(head);  // Insert at beginning  printf("\nEnter value to insert at beginning: ");  scanf("%d", &value);  head = insertAtBeginning(head, value);  displayLinkedList(head);  // Insert at end  printf("\nEnter value to insert at end: ");  scanf("%d", &value);  head = insertAtEnd(head, value);  displayLinkedList(head);  // Insert after a specified element  printf("\nEnter element after which to insert: ");  scanf("%d", &element);  printf("Enter value to insert after %d: ", element);  scanf("%d", &value);  head = insertAfterElement(head, element, value);  displayLinkedList(head);  freeLinkedList(head);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 46 | Date: 15/09/2025 |
| Program Title : Write a program with functions to delete a node  a. From the beginning of the linked list  b. From the end of the linked list  c. The node with specified data element. | |
| /\* PROGRAM 46: LINKED LIST DELETION  @ALBIN MAMMEN MATHEW  Roll No: 08  18/09/2025  \*/  #include <stdio.h>  #include <stdlib.h>  // Node structure  struct Node {  int data;  struct Node \*next;  };  // Function to create linked list  struct Node\* createLinkedList(int n) {  struct Node \*head = NULL, \*temp = NULL, \*newNode = NULL;  int value,i;  for (i = 0; i < n; i++) {  newNode = (struct Node \*)malloc(sizeof(struct Node));  printf("Enter value for node %d: ", i + 1);  scanf("%d", &value);  newNode->data = value;  newNode->next = NULL;  if (head == NULL) {  head = newNode;  temp = head;  } else {  temp->next = newNode;  temp = newNode;  }  }  return head;  }  // Function to delete from beginning  struct Node\* deleteFromBeginning(struct Node \*head) {  if (head == NULL) {  printf("List is empty.\n");  return NULL;  }  struct Node \*temp = head;  head = head->next;  free(temp);  return head;  }  // Function to delete from end  struct Node\* deleteFromEnd(struct Node \*head) {  if (head == NULL) {  printf("List is empty.\n");  return NULL;  }  if (head->next == NULL) {  free(head);  return NULL;  }  struct Node \*temp = head;  while (temp->next->next != NULL)  temp = temp->next;  free(temp->next);  temp->next = NULL;  return head;  }  // Function to delete node with specified value  struct Node\* deleteNodeWithValue(struct Node \*head, int value) {  if (head == NULL) {  printf("List is empty.\n");  return NULL;  }  if (head->data == value) {  struct Node \*temp = head;  head = head->next;  free(temp);  return head;  }  struct Node \*temp = head;  while (temp->next != NULL && temp->next->data != value)  temp = temp->next;  if (temp->next == NULL) {  printf("Element %d not found in the list.\n", value);  return head;  }  struct Node \*del = temp->next;  temp->next = del->next;  free(del);  return head;  }  // Function to display linked list  void displayLinkedList(struct Node \*head) {  struct Node \*temp = head;  printf("\nLinked List: ");  while (temp != NULL) {  printf("%d -> ", temp->data);  temp = temp->next;  }  printf("NULL\n");  }  // Function to free linked list  void freeLinkedList(struct Node \*head) {  struct Node \*temp;  while (head != NULL) {  temp = head;  head = head->next;  free(temp);  }  printf("Memory freed successfully.\n");  }  int main() {  int n, value;  struct Node \*head = NULL;  printf("Enter number of nodes to create initially: ");  scanf("%d", &n);  head = createLinkedList(n);  displayLinkedList(head);  // Delete from beginning  printf("\nDeleting node from beginning...\n");  head = deleteFromBeginning(head);  displayLinkedList(head);  // Delete from end  printf("\nDeleting node from end...\n");  head = deleteFromEnd(head);  displayLinkedList(head);  // Delete node with specified value  printf("\nEnter value of node to delete: ");  scanf("%d", &value);  head = deleteNodeWithValue(head, value);  displayLinkedList(head);  freeLinkedList(head);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 47 | Date: 15/09/2025 |
| Program Title : Write a program to create a singly linked list of n nodes and display it in reverse order. | |
| /\* PROGRAM 47: LINKED LIST - REVERSE  @ALBIN MAMMEN MATHEW  Roll No: 08  18/09/2025  \*/  #include <stdio.h>  #include <stdlib.h>  // Node structure  struct Node {  int data;  struct Node \*next;  };  // Function to create linked list  struct Node\* createLinkedList(int n) {  struct Node \*head = NULL, \*temp = NULL, \*newNode = NULL;  int value,i;  for (i = 0; i < n; i++) {  newNode = (struct Node \*)malloc(sizeof(struct Node));  printf("Enter value for node %d: ", i + 1);  scanf("%d", &value);  newNode->data = value;  newNode->next = NULL;  if (head == NULL) {  head = newNode;  temp = head;  } else {  temp->next = newNode;  temp = newNode;  }  }  return head;  }  // Function to display linked list in forward order  void displayLinkedList(struct Node \*head) {  struct Node \*temp = head;  printf("\nLinked List: ");  while (temp != NULL) {  printf("%d -> ", temp->data);  temp = temp->next;  }  printf("NULL\n");  }  // Function to display linked list in reverse order using recursion  void displayReverse(struct Node \*head) {  if (head == NULL)  return;  displayReverse(head->next);  printf("%d -> ", head->data);  }  // Function to free linked list  void freeLinkedList(struct Node \*head) {  struct Node \*temp;  while (head != NULL) {  temp = head;  head = head->next;  free(temp);  }  printf("Memory freed successfully.\n");  }  int main() {  int n;  struct Node \*head = NULL;  printf("Enter number of nodes: ");  scanf("%d", &n);  head = createLinkedList(n);  displayLinkedList(head);  printf("\nLinked List in Reverse Order: ");  displayReverse(head);  printf("NULL\n");    freeLinkedList(head);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 48 | Date: 15/09/2025 |
| Program Title : Sort the elements in a linked list using  a. changing the values (swapping the values)  b. Changing the address (Swapping the address) | |
| /\* PROGRAM 48: LINKED LIST SORTING -> BY VALUES || BY ADDRESS  @ALBIN MAMMEN MATHEW  Roll No: 08  19/09/2025  \*/  #include <stdio.h>  #include <stdlib.h>  // Define the node structure  struct Node {  int data;  struct Node \*next;  };  // Function to create a linked list  struct Node\* createList(int n) {  struct Node \*head = NULL, \*temp, \*newNode;  int val,i;  for (i = 0; i < n; i++) {  newNode = (struct Node\*)malloc(sizeof(struct Node));  printf("Enter value for node %d: ", i + 1);  scanf("%d", &val);  newNode->data = val;  newNode->next = NULL;  if (head == NULL)  head = newNode;  else {  temp = head;  while (temp->next != NULL)  temp = temp->next;  temp->next = newNode;  }  }  return head;  }  // Function to display the linked list  void displayList(struct Node \*head) {  struct Node \*temp = head;  printf("List: ");  while (temp != NULL) {  printf("%d ", temp->data);  temp = temp->next;  }  printf("\n");  }  // a) Sort by swapping values  void sortByValue(struct Node \*head) {  struct Node \*i, \*j;  int temp;  for (i = head; i != NULL; i = i->next) {  for (j = i->next; j != NULL; j = j->next) {  if (i->data > j->data) {  temp = i->data;  i->data = j->data;  j->data = temp;  }  }  }  }  // b) Sort by swapping links (addresses)  struct Node\* sortByAddress(struct Node \*head) {  struct Node \*i, \*j, \*prev, \*temp;  int swapped;  if (head == NULL)  return head;  do {  swapped = 0;  i = head;  prev = NULL;  while (i->next != NULL) {  j = i->next;  if (i->data > j->data) {  swapped = 1;  // Swap the nodes (not just data)  if (prev == NULL) {  i->next = j->next;  j->next = i;  head = j;  prev = j;  } else {  prev->next = j;  i->next = j->next;  j->next = i;  prev = j;  }  } else {  prev = i;  i = i->next;  }  }  } while (swapped);  return head;  }  int main() {  struct Node \*head = NULL;  int n;  printf("Enter number of nodes: ");  scanf("%d", &n);  head = createList(n);  printf("\nOriginal Linked List:\n");  displayList(head);  // Sorting by swapping values  sortByValue(head);  printf("\nLinked List after sorting by values:\n");  displayList(head);  // Sorting by swapping addresses  head = sortByAddress(head);  printf("\nLinked List after sorting by addresses:\n");  displayList(head);  // Free memory  struct Node \*temp;  while (head != NULL) {  temp = head;  head = head->next;  free(temp);  }  printf("\nMemory freed successfully.\n");  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 49 | Date: 26/09/2025 |
| Program Title : Polynomial using linked list - addition and multiplication. | |
| /\* PROGRAM 49: POLYNOMIAL ADDITION AND MULTIPLICATION USING LINKED LIST  @ALBIN MAMMEN MATHEW  Roll No: 08  27/09/2025  \*/  #include <stdio.h>  #include <stdlib.h>  // Structure to represent a polynomial term  struct Node {  int coeff; // coefficient  int power; // power of x  struct Node \*next;  };  // Function to create a new node  struct Node\* createNode(int coeff, int power) {  struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));  newNode->coeff = coeff;  newNode->power = power;  newNode->next = NULL;  return newNode;  }  // Function to insert a node into the polynomial (sorted by power)  struct Node\* insertNode(struct Node\* head, int coeff, int power) {  struct Node\* newNode = createNode(coeff, power);  if (head == NULL || head->power < power) {  newNode->next = head;  return newNode;  }  struct Node\* temp = head;  while (temp->next != NULL && temp->next->power >= power)  temp = temp->next;  if (temp->power == power) {  temp->coeff += coeff; // Combine like terms  free(newNode);  } else {  newNode->next = temp->next;  temp->next = newNode;  }  return head;  }  // Function to display a polynomial  void display(struct Node\* head) {  if (head == NULL) {  printf("0");  return;  }  struct Node\* temp = head;  while (temp != NULL) {  printf("%dx^%d", temp->coeff, temp->power);  if (temp->next != NULL && temp->next->coeff >= 0)  printf(" + ");  temp = temp->next;  }  printf("\n");  }  // Function to add two polynomials  struct Node\* addPoly(struct Node\* p1, struct Node\* p2) {  struct Node\* result = NULL;  while (p1 != NULL || p2 != NULL) {  if (p1 == NULL) {  result = insertNode(result, p2->coeff, p2->power);  p2 = p2->next;  } else if (p2 == NULL) {  result = insertNode(result, p1->coeff, p1->power);  p1 = p1->next;  } else if (p1->power > p2->power) {  result = insertNode(result, p1->coeff, p1->power);  p1 = p1->next;  } else if (p1->power < p2->power) {  result = insertNode(result, p2->coeff, p2->power);  p2 = p2->next;  } else {  result = insertNode(result, p1->coeff + p2->coeff, p1->power);  p1 = p1->next;  p2 = p2->next;  }  }  return result;  }  // Function to multiply two polynomials  struct Node\* multiplyPoly(struct Node\* p1, struct Node\* p2) {  struct Node\* result = NULL;  for (struct Node\* i = p1; i != NULL; i = i->next) {  for (struct Node\* j = p2; j != NULL; j = j->next) {  int coeff = i->coeff \* j->coeff;  int power = i->power + j->power;  result = insertNode(result, coeff, power);  }  }  return result;  }  // Function to free the linked list  void freePoly(struct Node\* head) {  struct Node\* temp;  while (head != NULL) {  temp = head;  head = head->next;  free(temp);  }  }  int main() {  struct Node \*poly1 = NULL, \*poly2 = NULL, \*sum = NULL, \*product = NULL;  int n, coeff, power;  printf("Enter number of terms in 1st polynomial: ");  scanf("%d", &n);  for (int i = 0; i < n; i++) {  printf("Enter coefficient and power: ");  scanf("%d %d", &coeff, &power);  poly1 = insertNode(poly1, coeff, power);  }  printf("Enter number of terms in 2nd polynomial: ");  scanf("%d", &n);  for (int i = 0; i < n; i++) {  printf("Enter coefficient and power: ");  scanf("%d %d", &coeff, &power);  poly2 = insertNode(poly2, coeff, power);  }  printf("\nFirst Polynomial: ");  display(poly1);  printf("Second Polynomial: ");  display(poly2);  // Addition  sum = addPoly(poly1, poly2);  printf("\nSum = ");  display(sum);  // Multiplication  product = multiplyPoly(poly1, poly2);  printf("Product = ");  display(product);  // Free memory  freePoly(poly1);  freePoly(poly2);  freePoly(sum);  freePoly(product);  printf("\nMemory freed successfully.\n");  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 50 | Date: 26/09/2025 |
| Program Title : Linked list using names - insert, delete, display, sort, reverse, count. | |
| /\*PROGRAM 50 LINKED LIST FUNCTIONS - INSERT DELETE DISPLAY SORT REVERSE COUNT  @ALBIN MAMMEN MATHEW  Roll No: 08  26/09/2025  \*/  #include<stdio.h>  #include<stdlib.h>  #include<string.h>  // Structure for a node  struct node{  char name[50];  struct node \*next;  };  typedef struct node node;  node \*head = NULL;  //Function to insert a name at the end  void insert(char a[]) {  node \*t;  if(head == NULL){  head = (node \*)malloc(sizeof(node));  strcpy(head->name, a);  head->next = NULL;  }  else {  t = head;  while (t->next != NULL) {  t = t->next;  }  t->next = (node \*)malloc(sizeof(node));  strcpy(t->next->name, a);  t->next->next = NULL;  }  }  // Function to delete a name  void delete(char a[]) {  node \*t;  if (head == NULL){  printf("EMPTY LINKED LIST\n");  }  else if (strcmp(head->name, a) == 0){  head = head->next;  } else {  t = head;  while (t->next != NULL && strcmp(t->next->name, a) != 0) {  t = t->next;  }  if (t->next == NULL){  printf("Element not found");  }  else {  t->next = t->next->next;  }  }  }  // Function to display all names  void display(){  node \*t;  t = head;  if (t == NULL) {  printf("EMPTY LINKED LIST\n");  return;  }  while (t != NULL ) {  printf("%s \t ", t->name);  t = t->next;  }  printf("\n");  }  // Function to sort names alphabetically  void sort(){  node \*t1, \*t2;  char temp[50];  if (head == NULL){  printf("EMPTY LINKED LIST");  return;  }  else {  for (t1 = head; t1 != NULL; t1 = t1->next) {  for (t2 = t1->next; t2 != NULL; t2 = t2->next) {  if (strcmp(t1->name, t2->name) > 0) {  strcpy(temp, t1->name);  strcpy(t1->name, t2->name);  strcpy(t2->name, temp);  }  }  }  }  }  // Function to reverse the linked list  void reverse(){  node \*prev, \*curr, \*next;  prev = NULL;  curr = head;  if (head == NULL){  printf("EMPTY LINKED LIST");  return;  }  else {  while (curr != NULL) {  next = curr -> next;  curr -> next = prev;  prev = curr;  curr = next;  }  head = prev;  }  }  // Function to count nodes  void count(){  node \*t;  int c = 0;  t = head;  if (t == NULL){  printf("EMPTY LINKED LIST");  return;  }  else {  while (t != NULL ) {  c++;  t = t -> next;  }  }  printf("Number of elements in the linked list: %d", c);  }  int menu() {  int ch;  printf("\nINSERT-1\nDELETE-2\nDISPLAY-3\nSORT-4\nREVERSE-5\nCOUNT-6\nEXIT-7\n");  printf("Enter Your Choice:");  scanf("%d",&ch);  return ch;  }  void processmenu(){  int ch;  char name[50];  for (ch = menu(); ch != 7; ch = menu()) {  switch(ch) {  case 1 :  printf("Enter the name to insert: ");  scanf("%s", name);  insert(name);  break;  case 2 :  printf("Enter the name to delete: ");  scanf("%s", name);  delete(name);  break;  case 3 :  display();  break;  case 4 :  sort();  break;  case 5 :  reverse();  break;  case 6 :  count();  break;  default :  printf("Error: Unknown Option Selected");  }  }  }  int main(){  processmenu();  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 51 | Date: 17/09/2025 |
| Program Title : Perform the respective operations on Linked Stack. | |
| /\*PROGRAM 51 OPERATIONS ON LINKED STACK  @ALBIN MAMMEN MATHEW  Roll No: 08  27/09/2025  \*/  #include <stdio.h>  #include <stdlib.h>  struct Node {  int data;  struct Node \*next;  };  struct Node \*top = NULL;  // Function to push element onto the stack  void push(int value) {  struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));  newNode->data = value;  newNode->next = top;  top = newNode;  printf("Pushed %d onto stack.\n", value);  }  // Function to pop element from the stack  void pop() {  if (top == NULL)  printf("Stack Underflow!\n");  else {  struct Node \*temp = top;  printf("Popped %d from stack.\n", top->data);  top = top->next;  free(temp);  }  }  // Function to display stack elements  void display() {  if (top == NULL)  printf("Stack is empty.\n");  else {  struct Node \*temp = top;  printf("Stack elements: ");  while (temp != NULL) {  printf("%d ", temp->data);  temp = temp->next;  }  printf("\n");  }  }  // Function to display menu and get user choice  int menu() {  int choice;  printf("\n--- Linked Stack Operations ---\n");  printf("1. Push\n");  printf("2. Pop\n");  printf("3. Display\n");  printf("4. Exit\n");  printf("Enter your choice: ");  scanf("%d", &choice);  return choice;  }  // Function to process stack operations  void processStack() {  int choice, value;  while (1) {  choice = menu();  switch (choice) {  case 1:  printf("Enter value to push: ");  scanf("%d", &value);  push(value);  break;  case 2:  pop();  break;  case 3:  display();  break;  case 4:  printf("Exiting program.\n");  exit(0);  default:  printf("Invalid choice! Try again.\n");  }  }  }  int main() {  processStack();  return 0;  } | |
| Output | |
| Program No: 52 | Date: 17/09/2025 |
| Program Title : Perform the respective operations on Linked Queue. | |
| /\*PROGRAM 52 OPERATIONS ON LINKED QUEUE  @ALBIN MAMMEN MATHEW  Roll No: 08  27/09/2025  \*/  #include <stdio.h>  #include <stdlib.h>  struct Node {  int data;  struct Node \*next;  };  struct Node \*front = NULL;  struct Node \*rear = NULL;  // Function to insert (enqueue) an element  void enqueue(int value) {  struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));  newNode->data = value;  newNode->next = NULL;  if (rear == NULL) {  front = rear = newNode;  } else {  rear->next = newNode;  rear = newNode;  }  printf("%d inserted into queue.\n", value);  }  // Function to delete (dequeue) an element  void dequeue() {  if (front == NULL) {  printf("Queue Underflow!\n");  return;  }  struct Node \*temp = front;  printf("Deleted element: %d\n", front->data);  front = front->next;  if (front == NULL)  rear = NULL;  free(temp);  }  // Function to display queue elements  void display() {  if (front == NULL) {  printf("Queue is empty.\n");  return;  }  struct Node \*temp = front;  printf("Queue elements: ");  while (temp != NULL) {  printf("%d ", temp->data);  temp = temp->next;  }  printf("\n");  }  // Function to display menu and return choice  int menu() {  int choice;  printf("\n--- Linked Queue Operations ---\n");  printf("1. Enqueue\n");  printf("2. Dequeue\n");  printf("3. Display\n");  printf("4. Exit\n");  printf("Enter your choice: ");  scanf("%d", &choice);  return choice;  }  // Function to process queue operations  void processQueue() {  int choice, value;  while (1) {  choice = menu();  switch (choice) {  case 1:  printf("Enter value to insert: ");  scanf("%d", &value);  enqueue(value);  break;  case 2:  dequeue();  break;  case 3:  display();  break;  case 4:  printf("Exiting program.\n");  exit(0);  default:  printf("Invalid choice! Try again.\n");  }  }  }  int main() {  processQueue();  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 53 | Date: 26/09/2025 |
| Program Title : Perform the respective operations on Circular Linked List. | |
| /\*PROGRAM 53 OPERATIONS ON CIRCULAR LINKED LIST  @ALBIN MAMMEN MATHEW  Roll No: 08  30/09/2025  \*/  #include <stdio.h>  #include <stdlib.h>  struct Node {  int data;  struct Node \*next;  };  struct Node \*last = NULL;  // Function to insert at the end  void insertEnd(int value) {  struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));  newNode->data = value;  if (last == NULL) {  newNode->next = newNode;  last = newNode;  } else {  newNode->next = last->next;  last->next = newNode;  last = newNode;  }  printf("%d inserted at the end.\n", value);  }  // Function to insert at the beginning  void insertBegin(int value) {  struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));  newNode->data = value;  if (last == NULL) {  newNode->next = newNode;  last = newNode;  } else {  newNode->next = last->next;  last->next = newNode;  }  printf("%d inserted at the beginning.\n", value);  }  // Function to delete a node  void deleteNode(int value) {  if (last == NULL) {  printf("List is empty!\n");  return;  }  struct Node \*temp = last->next;  struct Node \*prev = last;  // Single node case  if (last == last->next && last->data == value) {  free(last);  last = NULL;  printf("%d deleted.\n", value);  return;  }  // Traverse to find node  do {  if (temp->data == value) {  prev->next = temp->next;  if (temp == last)  last = prev;  free(temp);  printf("%d deleted.\n", value);  return;  }  prev = temp;  temp = temp->next;  } while (temp != last->next);  printf("Element %d not found!\n", value);  }  // Function to display the circular linked list  void display() {  if (last == NULL) {  printf("List is empty.\n");  return;  }  struct Node \*temp = last->next;  printf("Circular Linked List: ");  do {  printf("%d ", temp->data);  temp = temp->next;  } while (temp != last->next);  printf("\n");  }  // Function to display menu and return choice  int menu() {  int choice;  printf("\n--- Circular Linked List Operations ---\n");  printf("1. Insert at Beginning\n");  printf("2. Insert at End\n");  printf("3. Delete\n");  printf("4. Display\n");  printf("5. Exit\n");  printf("Enter your choice: ");  scanf("%d", &choice);  return choice;  }  // Function to process circular linked list operations  void processCircularList() {  int choice, value;  while (1) {  choice = menu();  switch (choice) {  case 1:  printf("Enter value: ");  scanf("%d", &value);  insertBegin(value);  break;  case 2:  printf("Enter value: ");  scanf("%d", &value);  insertEnd(value);  break;  case 3:  printf("Enter value to delete: ");  scanf("%d", &value);  deleteNode(value);  break;  case 4:  display();  break;  case 5:  printf("Exiting program.\n");  exit(0);  default:  printf("Invalid choice! Try again.\n");  }  }  }  int main() {  processCircularList();  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 54 | Date: 26/09/2025 |
| Program Title : Perform the respective operations on Circular Linked Queue. | |
| /\*PROGRAM 54 OPERATIONS ON CIRCULAR LINKED QUEUE  @ALBIN MAMMEN MATHEW  Roll No: 08  01/10/2025  \*/  #include <stdio.h>  #include <stdlib.h>  struct Node {  int data;  struct Node \*next;  };  struct Node \*front = NULL;  struct Node \*rear = NULL;  // Function to insert (enqueue) an element  void enqueue(int value) {  struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));  newNode->data = value;  newNode->next = NULL;  if (front == NULL) {  front = rear = newNode;  rear->next = front; // circular link  } else {  rear->next = newNode;  rear = newNode;  rear->next = front;  }  printf("%d inserted into Circular Queue.\n", value);  }  // Function to delete (dequeue) an element  void dequeue() {  if (front == NULL) {  printf("Queue Underflow!\n");  return;  }  struct Node \*temp = front;  int value = temp->data;  // Single element case  if (front == rear) {  front = rear = NULL;  } else {  front = front->next;  rear->next = front;  }  free(temp);  printf("%d deleted from Circular Queue.\n", value);  }  // Function to display the circular queue  void display() {  if (front == NULL) {  printf("Queue is empty.\n");  return;  }  struct Node \*temp = front;  printf("Circular Queue elements: ");  do {  printf("%d ", temp->data);  temp = temp->next;  } while (temp != front);  printf("\n");  }  // Function to display menu and return choice  int menu() {  int choice;  printf("\n--- Circular Linked Queue Operations ---\n");  printf("1. Enqueue\n");  printf("2. Dequeue\n");  printf("3. Display\n");  printf("4. Exit\n");  printf("Enter your choice: ");  scanf("%d", &choice);  return choice;  }  // Function to process circular queue operations  void processCircularQueue() {  int choice, value;  while (1) {  choice = menu();  switch (choice) {  case 1:  printf("Enter value to insert: ");  scanf("%d", &value);  enqueue(value);  break;  case 2:  dequeue();  break;  case 3:  display();  break;  case 4:  printf("Exiting program.\n");  exit(0);  default:  printf("Invalid choice! Try again.\n");  }  }  }  int main() {  processCircularQueue();  return 0;  } | |
| Output | |
| Program No: 55 | Date: 26/09/2025 |
| Program Title : Perform the respective operations on Doubly Linked List. | |
| /\*PROGRAM 55 OPERATIONS ON DOUBLY LINKED LIST  @ALBIN MAMMEN MATHEW  Roll No: 08  01/10/2025  \*/  #include <stdio.h>  #include <stdlib.h>  struct Node {  int data;  struct Node \*prev;  struct Node \*next;  };  struct Node \*head = NULL;  // Function to insert at the beginning  void insertBegin(int value) {  struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));  newNode->data = value;  newNode->prev = NULL;  newNode->next = head;  if (head != NULL)  head->prev = newNode;  head = newNode;  printf("%d inserted at the beginning.\n", value);  }  // Function to insert at the end  void insertEnd(int value) {  struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));  newNode->data = value;  newNode->next = NULL;  if (head == NULL) {  newNode->prev = NULL;  head = newNode;  } else {  struct Node \*temp = head;  while (temp->next != NULL)  temp = temp->next;  temp->next = newNode;  newNode->prev = temp;  }  printf("%d inserted at the end.\n", value);  }  // Function to delete a node by value  void deleteNode(int value) {  if (head == NULL) {  printf("List is empty!\n");  return;  }  struct Node \*temp = head;  while (temp != NULL && temp->data != value)  temp = temp->next;  if (temp == NULL) {  printf("%d not found in the list.\n", value);  return;  }  if (temp->prev != NULL)  temp->prev->next = temp->next;  else  head = temp->next;  if (temp->next != NULL)  temp->next->prev = temp->prev;  free(temp);  printf("%d deleted from the list.\n", value);  }  // Function to display list from beginning  void displayForward() {  if (head == NULL) {  printf("List is empty.\n");  return;  }  struct Node \*temp = head;  printf("Doubly Linked List (Forward): ");  while (temp != NULL) {  printf("%d ", temp->data);  temp = temp->next;  }  printf("\n");  }  // Function to display list in reverse order  void displayBackward() {  if (head == NULL) {  printf("List is empty.\n");  return;  }  struct Node \*temp = head;  while (temp->next != NULL)  temp = temp->next;  printf("Doubly Linked List (Backward): ");  while (temp != NULL) {  printf("%d ", temp->data);  temp = temp->prev;  }  printf("\n");  }  // Function to display menu and return choice  int menu() {  int choice;  printf("\n--- Doubly Linked List Operations ---\n");  printf("1. Insert at Beginning\n");  printf("2. Insert at End\n");  printf("3. Delete\n");  printf("4. Display Forward\n");  printf("5. Display Backward\n");  printf("6. Exit\n");  printf("Enter your choice: ");  scanf("%d", &choice);  return choice;  }  // Function to process doubly linked list operations  void processDoublyLinkedList() {  int choice, value;  while (1) {  choice = menu();  switch (choice) {  case 1:  printf("Enter value: ");  scanf("%d", &value);  insertBegin(value);  break;  case 2:  printf("Enter value: ");  scanf("%d", &value);  insertEnd(value);  break;  case 3:  printf("Enter value to delete: ");  scanf("%d", &value);  deleteNode(value);  break;  case 4:  displayForward();  break;  case 5:  displayBackward();  break;  case 6:  printf("Exiting program.\n");  exit(0);  default:  printf("Invalid choice! Try again.\n");  }  }  }  int main() {  processDoublyLinkedList();  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 56 | Date: 03/10/2025 |
| Program Title : Perform the respective operations on Circular doubly linked list - store string values as data part. | |
| /\*PROGRAM 56 OPERATIONS ON CIRCULAR DOUBLY LINKED LIST  @ALBIN MAMMEN MATHEW  Roll No: 08  04/10/2025  \*/  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  struct Node {  char data[50];  struct Node \*prev;  struct Node \*next;  };  struct Node \*last = NULL;  // Function to insert at the end  void insertEnd(char value[]) {  struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));  strcpy(newNode->data, value);  if (last == NULL) {  newNode->next = newNode->prev = newNode;  last = newNode;  } else {  newNode->next = last->next;  newNode->prev = last;  last->next->prev = newNode;  last->next = newNode;  last = newNode;  }  printf("'%s' inserted at the end.\n", value);  }  // Function to insert at the beginning  void insertBegin(char value[]) {  struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));  strcpy(newNode->data, value);  if (last == NULL) {  newNode->next = newNode->prev = newNode;  last = newNode;  } else {  newNode->next = last->next;  newNode->prev = last;  last->next->prev = newNode;  last->next = newNode;  }  printf("'%s' inserted at the beginning.\n", value);  }  // Function to delete a node  void deleteNode(char value[]) {  if (last == NULL) {  printf("List is empty!\n");  return;  }  struct Node \*temp = last->next;  struct Node \*prev = last;  // Single node case  if (last == last->next && strcmp(last->data, value) == 0) {  free(last);  last = NULL;  printf("'%s' deleted.\n", value);  return;  }  do {  if (strcmp(temp->data, value) == 0) {  prev->next = temp->next;  temp->next->prev = prev;  if (temp == last)  last = prev;  free(temp);  printf("'%s' deleted.\n", value);  return;  }  prev = temp;  temp = temp->next;  } while (temp != last->next);  printf("'%s' not found in the list.\n", value);  }  // Function to display the circular doubly linked list forward  void displayForward() {  if (last == NULL) {  printf("List is empty.\n");  return;  }  struct Node \*temp = last->next;  printf("Circular Doubly Linked List (Forward): ");  do {  printf("%s ", temp->data);  temp = temp->next;  } while (temp != last->next);  printf("\n");  }  // Function to display the list backward  void displayBackward() {  if (last == NULL) {  printf("List is empty.\n");  return;  }  struct Node \*temp = last;  printf("Circular Doubly Linked List (Backward): ");  do {  printf("%s ", temp->data);  temp = temp->prev;  } while (temp != last);  printf("\n");  }  // Function to display menu and return choice  int menu() {  int choice;  printf("\n--- Circular Doubly Linked List Operations ---\n");  printf("1. Insert at Beginning\n");  printf("2. Insert at End\n");  printf("3. Delete\n");  printf("4. Display Forward\n");  printf("5. Display Backward\n");  printf("6. Exit\n");  printf("Enter your choice: ");  scanf("%d", &choice);  getchar(); // clear newline  return choice;  }  // Function to process Circular Doubly Linked List operations  void processCircularDoublyLinkedList() {  int choice;  char value[50];  while (1) {  choice = menu();  switch (choice) {  case 1:  printf("Enter string to insert: ");  scanf(" %[^\n]", value);  insertBegin(value);  break;  case 2:  printf("Enter string to insert: ");  scanf(" %[^\n]", value);  insertEnd(value);  break;  case 3:  printf("Enter string to delete: ");  scanf(" %[^\n]", value);  deleteNode(value);  break;  case 4:  displayForward();  break;  case 5:  displayBackward();  break;  case 6:  printf("Exiting program.\n");  exit(0);  default:  printf("Invalid choice! Try again.\n");  }  }  }  int main() {  processCircularDoublyLinkedList();  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 57 | Date: 03/10/2025 |
| Program Title : Implement linear search for finding an element in an array. | |
| /\*PROGRAM 57 LINEAR SEARCH  @ALBIN MAMMEN MATHEW  Roll No: 08  05/10/2025  \*/  #include <stdio.h>  // Function to perform linear search  int linearSearch(int arr[], int n, int key) {  int i;  for (i = 0; i < n; i++) {  if (arr[i] == key)  return i; // Return index if element found  }  return -1; // Return -1 if not found  }  int main() {  int n,i, key;  printf("Enter number of elements: ");  scanf("%d", &n);  int arr[n];  printf("Enter %d elements:\n", n);  for (int i = 0; i < n; i++)  scanf("%d", &arr[i]);  printf("Enter element to search: ");  scanf("%d", &key);  int pos = linearSearch(arr, n, key);  if (pos != -1)  printf("Element %d found at position %d\n", key, pos + 1);  else  printf("Element %d not found.\n", key);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 58 | Date: 03/10/2025 |
| Program Title : Apply binary search for searching an element in an array of integers. | |
| /\*PROGRAM 58 BINARY SEARCH TREE IN ARRAY OF INTEGERS  @ALBIN MAMMEN MATHEW  Roll No: 08  05/10/2025  \*/  #include <stdio.h>  // Function to perform binary search  int binarySearch(int arr[], int n, int key) {  int low = 0, high = n - 1;  while (low <= high) {  int mid = (low + high) / 2;  if (arr[mid] == key)  return mid; // Element found  else if (arr[mid] < key)  low = mid + 1; // Search right half  else  high = mid - 1; // Search left half  }  return -1; // Element not found  }  int main() {  int n,i, key;  printf("Enter number of elements: ");  scanf("%d", &n);  int arr[n];  printf("Enter %d sorted elements:\n", n);  for (i = 0; i < n; i++)  scanf("%d", &arr[i]);  printf("Enter element to search: ");  scanf("%d", &key);  int pos = binarySearch(arr, n, key);  if (pos != -1)  printf("Element %d found at position %d\n", key, pos + 1);  else  printf("Element %d not found.\n", key);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 59 | Date: 03/10/2025 |
| Program Title : Apply binary search for searching an element in an array of Strings. | |
| /\*PROGRAM 59 BINARY SEARCH TREE IN ARRAY OF STRINGS  @ALBIN MAMMEN MATHEW  Roll No: 08  05/10/2025  \*/  #include <stdio.h>  #include <string.h>  // Function to perform binary search on strings  int binarySearchString(char arr[][50], int n, char key[]) {  int low = 0, high = n - 1;  while (low <= high) {  int mid = (low + high) / 2;  int cmp = strcmp(arr[mid], key);  if (cmp == 0)  return mid; // String found  else if (cmp < 0)  low = mid + 1; // Search right half  else  high = mid - 1; // Search left half  }  return -1; // Not found  }  int main() {  int i,n;  char key[50];  printf("Enter number of strings: ");  scanf("%d", &n);  char arr[n][50];  printf("Enter %d strings in sorted order:\n", n);  for (i = 0; i < n; i++)  scanf("%s", arr[i]);  printf("Enter string to search: ");  scanf("%s", key);  int pos = binarySearchString(arr, n, key);  if (pos != -1)  printf("String '%s' found at position %d\n", key, pos + 1);  else  printf("String '%s' not found.\n", key);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 60 | Date: 03/10/2025 |
| Program Title : Binary search tree insertion and display Traversal using inorder, preorder and postorder using recursion. | |
| /\*PROGRAM 60 BINARY SEARCH TREE INSERTION AND DISPLAY TRAVERSAL USING INORDER, PREORDER, POSTORDER WITH RECURSION  @ALBIN MAMMEN MATHEW  Roll No: 08  17/10/2025  \*/  #include <stdio.h>  #include <stdlib.h>  // Structure to represent a node in the binary search tree  struct Node {  int data;  struct Node \*left, \*right;  };  // Function to create a new node  struct Node\* createNode(int value) {  struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));  newNode->data = value;  newNode->left = newNode->right = NULL;  return newNode;  }  // Function to insert a node into BST  struct Node\* insert(struct Node\* root, int value) {  if (root == NULL)  return createNode(value);  if (value < root->data)  root->left = insert(root->left, value);  else if (value > root->data)  root->right = insert(root->right, value);  return root;  }  // Recursive inorder traversal  void inorder(struct Node\* root) {  if (root != NULL) {  inorder(root->left);  printf("%d ", root->data);  inorder(root->right);  }  }  // Recursive preorder traversal  void preorder(struct Node\* root) {  if (root != NULL) {  printf("%d ", root->data);  preorder(root->left);  preorder(root->right);  }  }  // Recursive postorder traversal  void postorder(struct Node\* root) {  if (root != NULL) {  postorder(root->left);  postorder(root->right);  printf("%d ", root->data);  }  }  int main() {  struct Node\* root = NULL;  int i,n, value;  printf("Enter number of nodes: ");  scanf("%d", &n);  printf("Enter %d values:\n", n);  for (i = 0; i < n; i++) {  scanf("%d", &value);  root = insert(root, value);  }  printf("\nInorder Traversal: ");  inorder(root);  printf("\nPreorder Traversal: ");  preorder(root);  printf("\nPostorder Traversal: ");  postorder(root);  printf("\n");  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 61 | Date: 10/10/2025 |
| Program Title : Binary search tree insertion and display in-order without using recursion. | |
| /\*PROGRAM 61 BINARY SEARCH TREE INSERTION AND DISPLAY TRAVERSAL USING INORDER WITHOUT RECURSION  @ALBIN MAMMEN MATHEW  Roll No: 08  17/10/2025  \*/  #include <stdio.h>  #include <stdlib.h>  // Structure to represent a node in the BST  struct Node {  int data;  struct Node \*left, \*right;  };  // Function to create a new node  struct Node\* createNode(int value) {  struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));  newNode->data = value;  newNode->left = newNode->right = NULL;  return newNode;  }  // Function to insert a node into BST  struct Node\* insert(struct Node\* root, int value) {  if (root == NULL)  return createNode(value);  if (value < root->data)  root->left = insert(root->left, value);  else if (value > root->data)  root->right = insert(root->right, value);  return root;  }  // Inorder traversal without recursion using stack  void inorderNonRecursive(struct Node\* root) {  struct Node\* stack[100];  int top = -1;  struct Node\* current = root;  while (current != NULL || top != -1) {  while (current != NULL) {  stack[++top] = current;  current = current->left;  }  current = stack[top--];  printf("%d ", current->data);  current = current->right;  }  }  int main() {  struct Node\* root = NULL;  int n, i, value;  printf("Enter number of nodes: ");  scanf("%d", &n);  printf("Enter %d values:\n", n);  for (i = 0; i < n; i++) {  scanf("%d", &value);  root = insert(root, value);  }  printf("\nInorder Traversal (Without Recursion): ");  inorderNonRecursive(root);  printf("\n");  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 62 | Date: 10/10/2025 |
| Program Title : Binary search tree insertion and display pre-order without using recursion. | |
| /\*PROGRAM 62 BINARY SEARCH TREE INSERTION AND DISPLAY TRAVERSAL USING PREORDER WITHOUT RECURSION  @ALBIN MAMMEN MATHEW  Roll No: 08  17/10/2025  \*/  #include <stdio.h>  #include <stdlib.h>  // Structure to represent a node in the BST  struct Node {  int data;  struct Node \*left, \*right;  };  // Function to create a new node  struct Node\* createNode(int value) {  struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));  newNode->data = value;  newNode->left = newNode->right = NULL;  return newNode;  }  // Function to insert a node into the BST  struct Node\* insert(struct Node\* root, int value) {  if (root == NULL)  return createNode(value);  if (value < root->data)  root->left = insert(root->left, value);  else if (value > root->data)  root->right = insert(root->right, value);  return root;  }  // Preorder traversal without recursion using stack  void preorderNonRecursive(struct Node\* root) {  if (root == NULL)  return;  struct Node\* stack[100];  int top = -1;  stack[++top] = root;  while (top != -1) {  struct Node\* current = stack[top--];  printf("%d ", current->data);  if (current->right)  stack[++top] = current->right;  if (current->left)  stack[++top] = current->left;  }  }  int main() {  struct Node\* root = NULL;  int n, i, value;  printf("Enter number of nodes: ");  scanf("%d", &n);  printf("Enter %d values:\n", n);  for (i = 0; i < n; i++) {  scanf("%d", &value);  root = insert(root, value);  }  printf("\nPreorder Traversal (Without Recursion): ");  preorderNonRecursive(root);  printf("\n");  return 0;  } | |
| Output | |
| Program No: 63 | Date: 10/10/2025 |
| Program Title : Binary search tree insertion and display post-order without using recursion. | |
| /\*PROGRAM 63 BINARY SEARCH TREE INSERTION AND DISPLAY TRAVERSAL USING POSTORDER WITHOUT RECURSION  @ALBIN MAMMEN MATHEW  Roll No: 08  17/10/2025  \*/  #include <stdio.h>  #include <stdlib.h>  // Structure to represent a node in the BST  struct Node {  int data;  struct Node \*left, \*right;  };  // Function to create a new node  struct Node\* createNode(int value) {  struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));  newNode->data = value;  newNode->left = newNode->right = NULL;  return newNode;  }  // Function to insert a node into the BST  struct Node\* insert(struct Node\* root, int value) {  if (root == NULL)  return createNode(value);  if (value < root->data)  root->left = insert(root->left, value);  else if (value > root->data)  root->right = insert(root->right, value);  return root;  }  // Postorder traversal without recursion using two stacks  void postorderNonRecursive(struct Node\* root) {  if (root == NULL)  return;  struct Node\* stack1[100];  struct Node\* stack2[100];  int top1 = -1, top2 = -1;  stack1[++top1] = root;  while (top1 != -1) {  struct Node\* current = stack1[top1--];  stack2[++top2] = current;  if (current->left)  stack1[++top1] = current->left;  if (current->right)  stack1[++top1] = current->right;  }  while (top2 != -1)  printf("%d ", stack2[top2--]->data);  }  int main() {  struct Node\* root = NULL;  int n, i, value;  printf("Enter number of nodes: ");  scanf("%d", &n);  printf("Enter %d values:\n", n);  for (i = 0; i < n; i++) {  scanf("%d", &value);  root = insert(root, value);  }  printf("\nPostorder Traversal (Without Recursion): ");  postorderNonRecursive(root);  printf("\n");  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 64 | Date: 10/10/2025 |
| Program Title : Binary search tree insertion using names and display the names in ascending order using inorder traversal. | |
| /\*PROGRAM 64 BINARY SEARCH TREE INSERTION USING NAMES AND DISPLAY USING INORDER  @ALBIN MAMMEN MATHEW  Roll No: 08  17/10/2025  \*/  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  // Structure to represent a node with a name  struct Node {  char name[50];  struct Node \*left, \*right;  };  // Function to create a new node  struct Node\* createNode(char name[]) {  struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));  strcpy(newNode->name, name);  newNode->left = newNode->right = NULL;  return newNode;  }  // Function to insert a name into the BST  struct Node\* insert(struct Node\* root, char name[]) {  if (root == NULL)  return createNode(name);  if (strcmp(name, root->name) < 0)  root->left = insert(root->left, name);  else if (strcmp(name, root->name) > 0)  root->right = insert(root->right, name);  return root;  }  // Inorder traversal to display names in ascending order  void inorder(struct Node\* root) {  if (root != NULL) {  inorder(root->left);  printf("%s\n", root->name);  inorder(root->right);  }  }  int main() {  struct Node\* root = NULL;  int i, n;  char name[50];  printf("Enter number of names: ");  scanf("%d", &n);  printf("Enter %d names:\n", n);  for (i = 0; i < n; i++) {  scanf("%s", name);  root = insert(root, name);  }  printf("\nNames in Ascending Order:\n");  inorder(root);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 65 | Date: 10/10/2025 |
| Program Title : Demonstrate the data structure of adjacent matrix using arrays. | |
| /\*PROGRAM 65 ADJACENCY MATRIX USING ARRAYS  @ALBIN MAMMEN MATHEW  Roll No: 08  17/10/2025  \*/  #include <stdio.h>  int main() {  int vertices, edges;  int src, dest, i, j;  printf("Enter number of vertices: ");  scanf("%d", &vertices);  int adjMatrix[vertices][vertices];  // Initialize all elements to 0  for (i = 0; i < vertices; i++) {  for (j = 0; j < vertices; j++) {  adjMatrix[i][j] = 0;  }  }  printf("Enter number of edges: ");  scanf("%d", &edges);  printf("Enter edges (source destination):\n");  for (i = 0; i < edges; i++) {  scanf("%d %d", &src, &dest);  adjMatrix[src][dest] = 1;  adjMatrix[dest][src] = 1; // For undirected graph  }  // Display the adjacency matrix  printf("\nAdjacency Matrix Representation:\n");  for (i = 0; i < vertices; i++) {  for (j = 0; j < vertices; j++) {  printf("%d ", adjMatrix[i][j]);  }  printf("\n");  }  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 66 | Date: 10/10/2025 |
| Program Title : Demonstrate the data structure of adjacent matrix  using linked lists. | |
| /\*PROGRAM 66 ADJACENCY MATRIX USING LINKED LISTS  @ALBIN MAMMEN MATHEW  Roll No: 08  18/10/2025  \*/  #include <stdio.h>  #include <stdlib.h>  // Structure Definition  struct Node {  int data;  struct Node\* right; // pointer to next column node  struct Node\* down; // pointer to next row node  };  // Function Prototypes  struct Node\* newNode(int data);  struct Node\* createLinkedMatrix(int mat[][10], int n);  void displayMatrix(struct Node\* head);  void freeMatrix(struct Node\* head);  // Function Implementations  // Create a new node  struct Node\* newNode(int data) {  struct Node\* temp = (struct Node\*)malloc(sizeof(struct Node));  temp->data = data;  temp->right = NULL;  temp->down = NULL;  return temp;  }  // Create linked matrix representation from a 2D array  struct Node\* createLinkedMatrix(int mat[][10], int n) {  struct Node\* head = NULL;  struct Node\* prevRowStart = NULL; // first node of previous row  struct Node\* currRowStart = NULL; // first node of current row  struct Node\* prev = NULL;  int i, j, k;  for (i = 0; i < n; i++) {  prev = NULL;  for (j = 0; j < n; j++) {  // Create new node  struct Node\* newnode = newNode(mat[i][j]);  if (head == NULL)  head = newnode; // first node of matrix  if (prev == NULL)  currRowStart = newnode;  else  prev->right = newnode; // link nodes in the same row  // Link vertically with previous row  if (prevRowStart != NULL) {  struct Node\* temp = prevRowStart;  for (k = 0; k < j; k++)  temp = temp->right;  temp->down = newnode;  }  prev = newnode;  }  prevRowStart = currRowStart;  }  return head;  }  // Display the linked matrix  void displayMatrix(struct Node\* head) {  struct Node\* row = head;  while (row != NULL) {  struct Node\* col = row;  while (col != NULL) {  printf("%d ", col->data);  col = col->right;  }  printf("\n");  row = row->down;  }  }  // Free the allocated memory  void freeMatrix(struct Node\* head) {  struct Node\* row = head;  while (row != NULL) {  struct Node\* col = row;  struct Node\* nextRow = row->down;  while (col != NULL) {  struct Node\* temp = col;  col = col->right;  free(temp);  }  row = nextRow;  }  }  int main() {  int i, j, n;  int mat[10][10];  printf("Enter number of vertices: ");  scanf("%d", &n);  printf("Enter adjacency matrix (%d x %d):\n", n, n);  for (i = 0; i < n; i++) {  for (j = 0; j < n; j++) {  scanf("%d", &mat[i][j]);  }  }  struct Node\* head = createLinkedMatrix(mat, n);  printf("\nAdjacency Matrix represented using Linked Lists:\n");  displayMatrix(head);  freeMatrix(head);  return 0;  } | |
| Output | |

|  |  |
| --- | --- |
| Program No: 67 | Date: 10/10/2025 |
| Program Title : Project-Minesweeper | |
| /\*PROGRAM 67 MINESWEEPER  @ALBIN MAMMEN MATHEW  Roll No: 08  18/10/2025  \*/  #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #define SIZE 8  #define MINES 10  typedef struct {  int isMine; // 1 if mine, 0 otherwise  int revealed; // 1 if revealed, 0 hidden  int flagged; // 1 if flagged, 0 otherwise  int neighborMines; // number of nearby mines  } Cell;  typedef struct MineNode {  int r, c;  struct MineNode \*next;  } MineNode;  typedef struct {  int r, c;  char action; // 'R' for reveal, 'F' for flag  } Move;  Cell board[SIZE][SIZE];  MineNode \*mineHead = NULL; //linked list head for mines  Move moveStack[SIZE \* SIZE]; //stack for undo  int top = -1;  int flagCount = 0; // Total flags placed by user  //function to save move  void pushMove(int r, int c, char action) {  if (top < SIZE \* SIZE - 1)  moveStack[++top] = (Move){r, c, action};  }  //function to undo last move  Move popMove() {  if (top >= 0)  return moveStack[top--];  else  return (Move){-1, -1, 'X'}; //empty stack  }  // Adding Mines  void addMine(int r, int c) {  MineNode \*newNode = (MineNode \*)malloc(sizeof(MineNode));  newNode->r = r;  newNode->c = c;  newNode->next = mineHead;  mineHead = newNode;  }  //Counting Neighbors  int countMines(int r, int c) {  int i, j, count = 0;  for (i = -1; i <= 1; i++)  for (j = -1; j <= 1; j++) {  int nr = r + i, nc = c + j; //neighbor coordinates  if (nr >= 0 && nr < SIZE && nc >= 0 && nc < SIZE && board[nr][nc].isMine)  count++;  }  return count;  }  //Reveal Function  void reveal(int r, int c) {  if (r < 0 || r >= SIZE || c < 0 || c >= SIZE || board[r][c].revealed || board[r][c].flagged)  return;  board[r][c].revealed = 1;  pushMove(r, c, 'R'); //stores move  int i, j;  if (board[r][c].neighborMines == 0)  for (i = -1; i <= 1; i++)  for (j = -1; j <= 1; j++)  if (!(i == 0 && j == 0))  reveal(r + i, c + j); //reveals neighboring tiles  }  //Printing the board  void printBoard(int showMines) {  int i, j;  printf("\n ");  for (i = 0; i < SIZE; i++) // print column numbers  printf("%d ", i);  printf("\n");  for (i = 0; i < SIZE; i++) {  printf("%d ", i); //print row number  for (j = 0; j < SIZE; j++) {  if (board[i][j].flagged)  printf("F "); //flag  else if (board[i][j].revealed)  printf("%d ", board[i][j].neighborMines); //show count  else if (showMines && board[i][j].isMine)  printf("\* "); //show mines when game is over  else  printf(". "); //hidden cell  }  printf("\n");  }  }  // Function to check Win  int checkWin() {  int safeCells = SIZE \* SIZE - MINES;  int revealedCells = 0;  int i, j;  for (i = 0; i < SIZE; i++)  for (j = 0; j < SIZE; j++)  if (board[i][j].revealed && !board[i][j].isMine)  revealedCells++;  return revealedCells == safeCells; //player wins the match  }  int main() {  srand(time(0)); //random seed for mine positions  int i, j;  // Initialize board  for (i = 0; i < SIZE; i++)  for (j = 0; j < SIZE; j++)  board[i][j] = (Cell){0, 0, 0, 0};  // Place mines dynamically  int placed = 0;  while (placed < MINES) {  int r = rand() % SIZE, c = rand() % SIZE;  if (!board[r][c].isMine) {  board[r][c].isMine = 1;  addMine(r, c); //store in linked list  placed++;  }  }  // Calculate neighbors  for (i = 0; i < SIZE; i++)  for (j = 0; j < SIZE; j++)  if (!board[i][j].isMine)  board[i][j].neighborMines = countMines(i, j);  //INSTRUCTIONS  printf("\n?? MINESWEEPER GAME ??\nCommands:\n");  printf("R row col -> Reveal cell\n");  printf("F row col -> Flag/Unflag cell\n");  printf("U -> Undo last move\n");  printf("Q -> Quit game\n");  // GAME LOOP  char cmd;  while (1) {  printBoard(0);  printf("\nEnter command: ");  scanf(" %c", &cmd);  if (cmd == 'Q' || cmd == 'q') {  printf("\nExiting game. Goodbye!\n");  break;  }  if (cmd == 'U' || cmd == 'u') {  Move last = popMove();  if (last.action == 'R')  board[last.r][last.c].revealed = 0;  else if (last.action == 'F')  board[last.r][last.c].flagged = !board[last.r][last.c].flagged;  continue;  }  int r, c;  scanf("%d %d", &r, &c);  if (r < 0 || r >= SIZE || c < 0 || c >= SIZE) {  printf("Invalid input!\n");  continue;  }  if (cmd == 'F' || cmd == 'f') {  if (!board[r][c].flagged) { // trying to place a flag  if (flagCount >= MINES) {  printf("?? You’ve already used all %d flags!\n", MINES);  continue;  }  board[r][c].flagged = 1;  flagCount++;  printf("?? Flag placed! Flags used: %d/%d\n", flagCount, MINES);  } else { // removing a flag  board[r][c].flagged = 0;  flagCount--;  printf("?? Flag removed! Flags used: %d/%d\n", flagCount, MINES);  }  pushMove(r, c, 'F');  continue;  }  if (cmd == 'R' || cmd == 'r') {  if (board[r][c].isMine) {  printf("\n?? Boom! You hit a mine!\n");  printBoard(1);  break;  }  reveal(r, c);  }  if (checkWin()) {  printf("\n?? Congratulations! You cleared all safe cells!\n");  printBoard(1);  break;  }  }  return 0;  } | |
| Output | |