**DSA Practical – Sorting Algorithms**

|  |
| --- |
| NAME: Gaurav Shukla  ROLL NO.: 2420712  BRANCH: B. TECH (AI&DS)  SECTION: C |

WEEK 1 AIM: Write a program to implement the following sorting algorithms using user defined functions:

1. b. Bubble Sort (Ascending Order)
2. Selection Sort (Descending order)

TOOL USED: · ·

Online c complier

MS Word

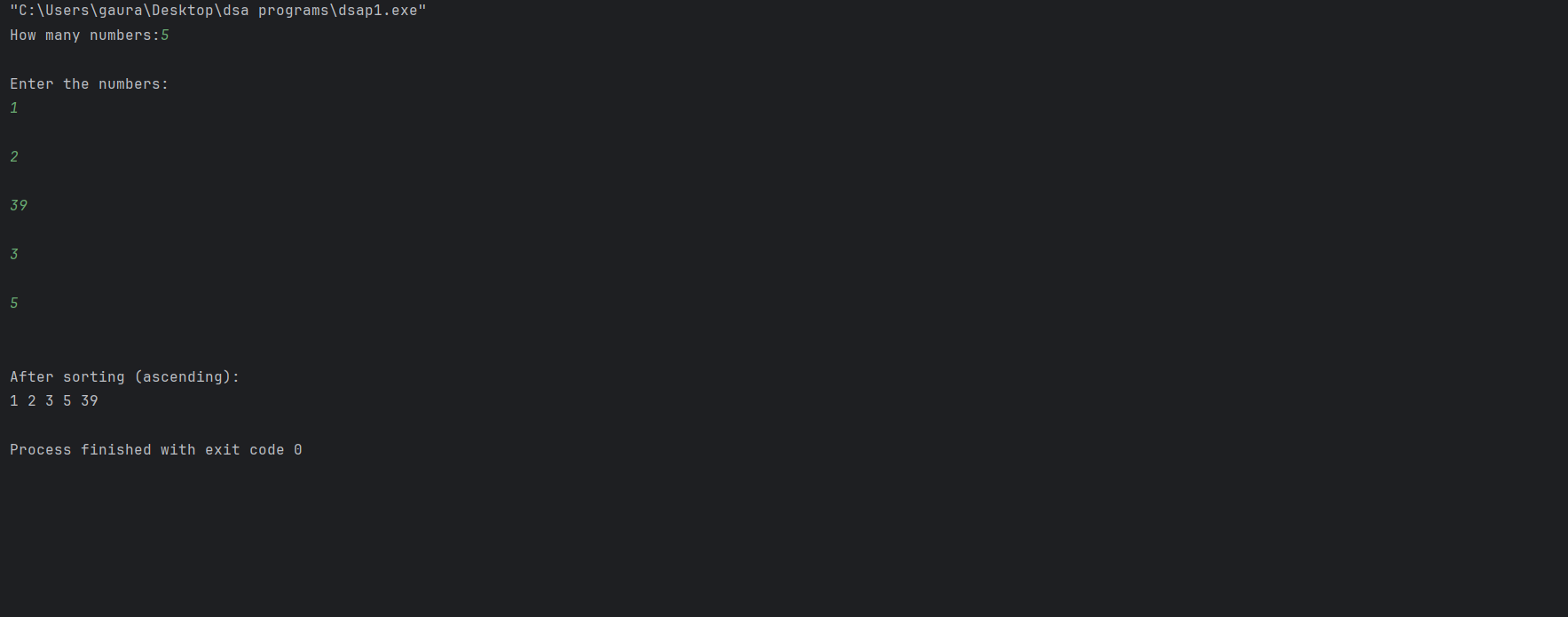
**FUNCTIONS USED:**

* **Bubble Sort Ascending(int arr[], int n)** – Sorts the array in ascending order using the Bubble Sort algorithm.
* **Selection Sort Descending(int arr[], int n)** – Sorts the array in descending order using the Selection Sort algorithm.
* **display(int arr[], int n)** – Prints the elements of the array.

Source code:

|  |
| --- |
| #include <stdio.h>  int main() {  int size;  int a[100];  int i, k, hold;  printf("How many numbers: ");  scanf("%d", &size);  printf("Enter the numbers:\n");  for(i = 0; i < size; i++) {  scanf("%d", &a[i]);  }  for(i = 0; i < size; i++) {  for(k = 0; k < size - i - 1; k++) {  if(a[k] > a[k+1]) {  hold = a[k];  a[k] = a[k+1];  a[k+1] = hold;  }  }  }  printf("\nAfter sorting (ascending):\n");  for(i = 0; i < size; i++) {  printf("%d ", a[i]);  }  printf("\n");  return 0;  } |

OUTPUT: Bubble sort (ascending order) :



**Code (b): Selection sort (descending order)**

|  |
| --- |
| **#include <stdio.h>**  **void sortDesc(int a[], int n) {**  **int i, j, big, temp;**  **for(i = 0; i < n-1; i++) {**  **big = i;**  **for(j = i+1; j < n; j++) {**  **if(a[j] > a[big]) {**  **big = j;**  **}**  **}**  **temp = a[i];**  **a[i] = a[big];**  **a[big] = temp;**  **}**  **}**  **int main() {**  **int x[100], num, i;**  **printf("How many numbers: ");**  **scanf("%d", &num);**  **printf("Enter numbers:\n");**  **for(i = 0; i < num; i++) {**  **scanf("%d", &x[i]);**  **}**  **sortDesc(x, num);**  **printf("Sorted (Descending):\n");**  **for(i = 0; i < num; i++) {**  **printf("%d ", x[i]);**  **}**  **printf("\n");**  **return 0;**  **}** |

**OUTPUT: Selection sort (descending order)**

****

**ERROR:**

Not printing the sorted array (so it looks like “no output”).

Forgetting to decide function return type (void).

Accessing out of range index (e.g., arr[n]).

**QUESTIONS:**

Differentiate between searching & sorting.

State the advantages of selection sort algorithm.

Is Bubble Sort a stable sorting algorithm?

**DSA Practical – Recursion**

|  |
| --- |
| NAME: Gaurav Shukla  ROLL NO.: 2420712  BRANCH: B. TECH (AI&DS)  SECTION: C |

**WEEK 2**

**AIM: Write a program to implement the following using recursion.**

**a. Sum of n numbers**

**b. Generate Fibonacci sequence**

**c. Solve Towers of Hanoi problem**

**TOOL USED:**

**C LION**

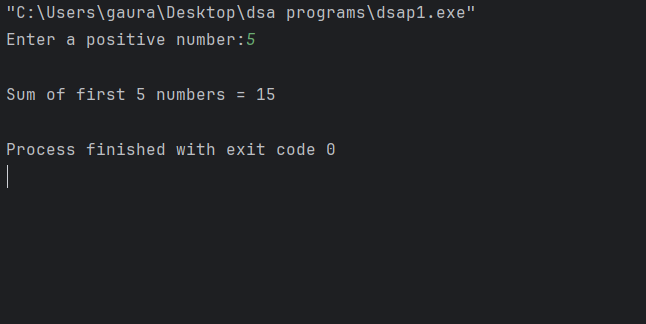
**ONLINE C COMPILER**

**MS Word**

**Code (a): Sum of n Numbers (using recursion)**

|  |
| --- |
| **#include <stdio.h>**  **int add(int n) {**  **if(n == 0)**  **return 0;**  **else**  **return n + add(n - 1);**  **}**  **int main() {**  **int num;**  **printf("Enter a positive number: ");**  **scanf("%d", &num);**  **printf("Sum of first %d numbers = %d\n", num, add(num));**  **return 0;**  **} return 0;**  **}** |

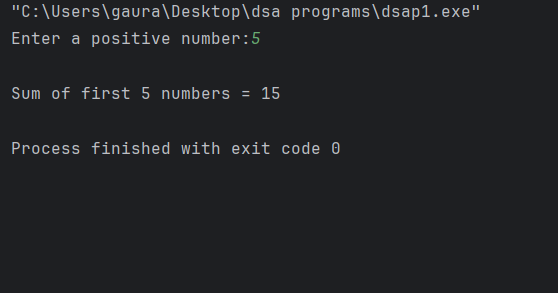
**OUTPUT:**

****

**Code(b): Generate Fibonacci sequence**

|  |
| --- |
| **#include <stdio.h>**  **int fibo(int x) {**  **if(x == 0)**  **return 0;**  **else if(x == 1)**  **return 1;**  **else**  **return fibo(x - 1) + fibo(x - 2);**  **}**  **int main() {**  **int n, k;**  **printf("How many terms: ");**  **scanf("%d", &n);**  **printf("Fibonacci series:\n");**  **for(k = 0; k < n; k++) {**  **printf("%d ", fibo(k));**  **}**  **printf("\n");**  **return 0;**  **}** |

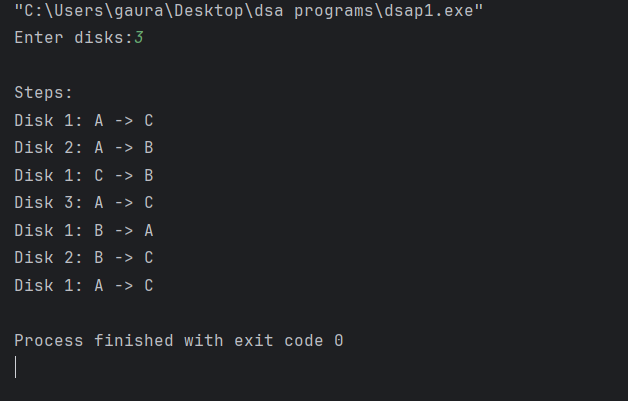
**OUTPUT:**

****

**Code(c): Solve Towers of Hanoi Problem**

|  |
| --- |
| **#include <stdio.h>**  **void hanoi(int n, char start, char end, char mid) {**  **if(n == 1) {**  **printf("Disk 1: %c -> %c\n", start, end);**  **return;**  **}**  **hanoi(n-1, start, mid, end);**  **printf("Disk %d: %c -> %c\n", n, start, end);**  **hanoi(n-1, mid, end, start);**  **}**  **int main() {**  **int disks;**  **printf("Enter disks: ");**  **scanf("%d", &disks);**  **printf("Steps:\n");**  **hanoi(disks, 'A', 'C', 'B');**  **return 0;**  **}** |

**OUTPUT:**

****

**ERROR:**

Missing / incorrect base case.

Wrong function return or missing return.

Wrong scanf() or parameter mistakes.

**QUESTIONS:**

What is recursion?

What is fibonacci sequence?

Difference between recursion & iteration.

**DSA Practical – Binary Search Tree operations**

|  |
| --- |
| Name: Gaurav Shukla  Roll no.: 2420712  Branch: B. Tech (AI&DS)  Section: C |

**WEEK 3**

**AIM: Binary Search Tree Operations (Recursive)**

a. Insert a node in BST

b. Delete a node from BST

c. Inorder traversal

**TOOL USED:**

.CLION

· Online c compiler

· Ms word

**FUNCTIONS USED:**

· CreateNode()

· Insert()

· Inorder()

**Source code:**

Code(a): Insert a node in BST

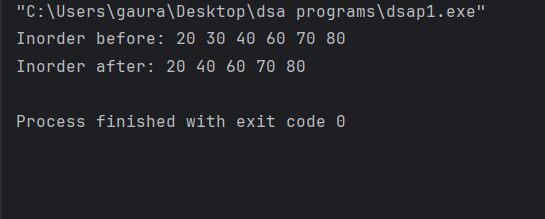
**OUTPUT:**

|  |
| --- |
|  |

Code(b): Delete a node from BST

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  struct node {  int data;  struct node \*left, \*right;  };  struct node\* makeNode(int val) {  struct node\* p = (struct node\*)malloc(sizeof(struct node));  p->data = val;  p->left = p->right = NULL;  return p;  }  struct node\* add(struct node\* root, int val) {  if(root == NULL)  return makeNode(val);  if(val < root->data)  root->left = add(root->left, val);  else if(val > root->data)  root->right = add(root->right, val);  return root;  }  struct node\* minNode(struct node\* root) {  while(root->left != NULL)  root = root->left;  return root;  }  struct node\* del(struct node\* root, int key) {  if(root == NULL)  return root;  if(key < root->data)  root->left = del(root->left, key);  else if(key > root->data)  root->right = del(root->right, key);  else {  if(root->left == NULL) {  struct node\* t = root->right;  free(root);  return t;  }  else if(root->right == NULL) {  struct node\* t = root->left;  free(root);  return t;  }  struct node\* t = minNode(root->right);  root->data = t->data;  root->right = del(root->right, t->data);  }  return root;  }  void inorder(struct node\* root) {  if(root != NULL) {  inorder(root->left);  printf("%d ", root->data);  inorder(root->right);  }  }  int main() {  struct node\* root = NULL;  root = add(root, 80);  root = add(root, 40);  root = add(root, 70);  root = add(root, 20);  root = add(root, 30);  root = add(root, 60);  printf("Inorder before: ");  inorder(root);  root = del(root, 30);  printf("\nInorder after: ");  inorder(root);  printf("\n");  return 0;  } |

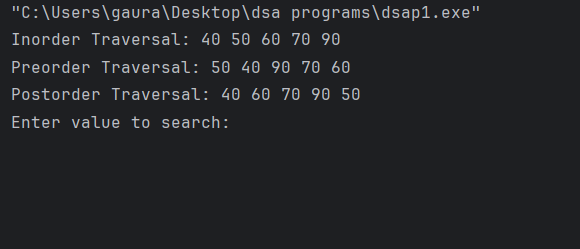
**OUTPUT:**

****

Code(c): Inorder transversal

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  struct Node {  int data;  struct Node \*left, \*right;  };  struct Node\* createNode(int value) {  struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));  newNode->data = value;  newNode->left = newNode->right = NULL;  return newNode;  }  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;  }  struct Node\* search(struct Node\* root, int key) {  if (root == NULL || root->data == key)  return root;  if (key < root->data)  return search(root->left, key);  return search(root->right, key);  }  void inorder(struct Node\* root) {  if (root != NULL) {  inorder(root->left);  printf("%d ", root->data);  inorder(root->right);  }  }  void preorder(struct Node\* root) {  if (root != NULL) {  printf("%d ", root->data);  preorder(root->left);  preorder(root->right);  }  }  void postorder(struct Node\* root) {  if (root != NULL) {  postorder(root->left);  postorder(root->right);  printf("%d ", root->data);  }  }  int main() {  struct Node\* root = NULL;  root = insert(root, 50);  root = insert(root, 90);  root = insert(root, 70);  root = insert(root, 60);  root = insert(root, 40);  printf("Inorder Traversal: ");  inorder(root);  printf("\n");  printf("Preorder Traversal: ");  preorder(root);  printf("\n");  printf("Postorder Traversal: ");  postorder(root);  printf("\n");  int key;  printf("Enter value to search: ");  scanf("%d", &key);  struct Node\* found = search(root, key);  if (found != NULL)  printf("%d found in BST.\n", key);  else  printf("%d not found in BST.\n", key);  return 0;  } |

**OUTPUT:**

****

ERRORS:

· Root not initialized (root= NULL)

· Missing base condition in recursion.

· Output formatting errors (\n missing)

QUESTIONS:

· What is a binary Search Tree?

· Why is recursion used in BST operations?

· What is inorder Traversal of BST?

|  |
| --- |
| **Name** : Gaurav Shukla  **Roll no**. : 2420712  **Branch** : BTech AI&DS  **Section** : C |

**Aim: Write a program to Simulate the working of a Singly circular linked list providing the following operations**

**a. Delete from the beginning/end**

**b. Delete a given element**

**c. Display &Insert is mandatory**

**Tools Used**: Clion/c , MS Word , GCC compiler

**Function Used:**

**• insert**

**• deleteFromBeginning**

**• deleteFromEnd**

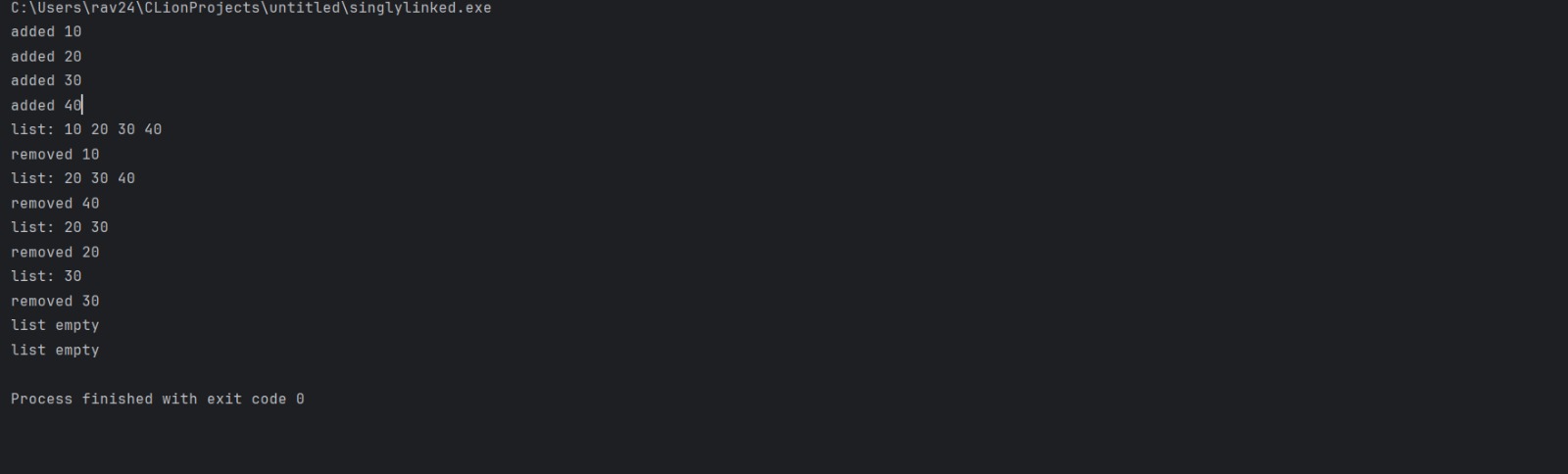
**• DeleteElement**

**• display**

**Code (a) : Delete from the beginning/end**

|  |
| --- |
| **#include <stdio.h>**  **#include <stdlib.h>**  **struct node {**  **int data;**  **struct node \*next;**  **};**  **struct node \*head = NULL;**  **void insert(int val) {**  **struct node \*n = (struct node\*)malloc(sizeof(struct node));**  **n->data = val;**  **n->next = NULL;**  **if (head == NULL) {**  **head = n;**  **} else {**  **struct node \*t = head;**  **while (t->next != NULL) {**  **t = t->next;**  **}**  **t->next = n;**  **}**  **printf("added %d\n", val);**  **}**  **void delBeg() {**  **if (head == NULL) {**  **printf("list empty\n");**  **return;**  **}**  **struct node \*temp = head;**  **printf("removed %d\n", head->data);**  **head = head->next;**  **free(temp);**  **}**  **void delEnd() {**  **if (head == NULL) {**  **printf("list empty\n");**  **return;**  **}**  **if (head->next == NULL) {**  **printf("removed %d\n", head->data);**  **free(head);**  **head = NULL;**  **return;**  **}**  **struct node \*p = head;**  **struct node \*q = NULL;**  **while (p->next != NULL) {**  **q = p;**  **p = p->next;**  **}**  **printf("removed %d\n", p->data);**  **q->next = NULL;**  **free(p);**  **}**  **void display() {**  **if (head == NULL) {**  **printf("list empty\n");**  **return;**  **}**  **struct node \*t = head;**  **printf("list: ");**  **while (t != NULL) {**  **printf("%d ", t->data);**  **t = t->next;**  **}**  **printf("\n");**  **}**  **int main() {**  **insert(10);**  **insert(20);**  **insert(30);**  **insert(40);**  **display();**  **delBeg();**  **display();**  **delEnd();**  **display();**  **delBeg();**  **display();**  **delEnd();**  **display();**  **delBeg();**  **return 0;**  **}** |

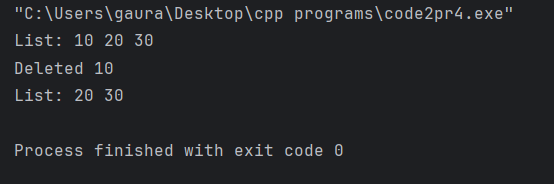
**Output: Delete from the beginning/end**

****

**Code (b) : Delete a given element**

|  |
| --- |
| **#include <stdio.h>**  **#include <stdlib.h>**  **struct node {**  **int data;**  **struct node \*next;**  **};**  **struct node \*head = NULL;**  **void insert(int val) {**  **struct node \*n = (struct node\*)malloc(sizeof(struct node));**  **n->data = val;**  **if (head == NULL) {**  **head = n;**  **n->next = head;**  **} else {**  **struct node \*t = head;**  **while (t->next != head) {**  **t = t->next;**  **}**  **t->next = n;**  **n->next = head;**  **}**  **}**  **void deleteBeg() {**  **if (head == NULL) {**  **printf("List empty\n");**  **return;**  **}**  **if (head->next == head) {**  **printf("Deleted %d\n", head->data);**  **free(head);**  **head = NULL;**  **return;**  **}**  **struct node \*temp = head;**  **struct node \*last = head;**  **while (last->next != head) {**  **last = last->next;**  **}**  **printf("Deleted %d\n", head->data);**  **head = head->next;**  **last->next = head;**  **free(temp);**  **}**  **void display() {**  **if (head == NULL) {**  **printf("List empty\n");**  **return;**  **}**  **struct node \*t = head;**  **printf("List: ");**  **do {**  **printf("%d ", t->data);**  **t = t->next;**  **} while (t != head);**  **printf("\n");**  **}**  **int main() {**  **insert(10);**  **insert(20);**  **insert(30);**  **display();**  **deleteBeg();**  **display();**  **return 0;**  **}** |

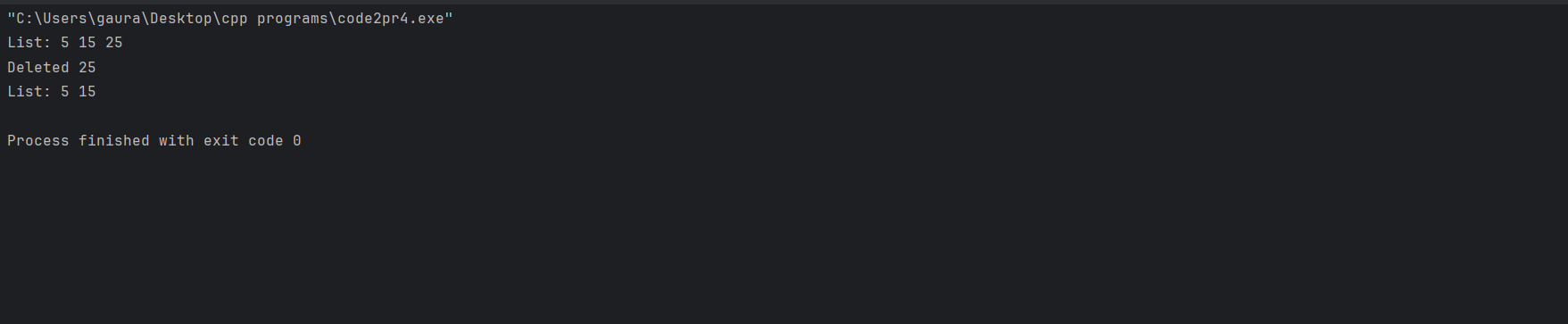
**Output: Delete a given element**

****

**Code (c) : Display &Insert is mandatory**

|  |
| --- |
| **#include <stdio.h>**  **#include <stdlib.h>**  **struct node {**  **int data;**  **struct node \*next;**  **};**  **struct node \*head = NULL;**  **void insert(int val) {**  **struct node \*n = (struct node\*)malloc(sizeof(struct node));**  **n->data = val;**  **if (head == NULL) {**  **head = n;**  **n->next = head;**  **} else {**  **struct node \*t = head;**  **while (t->next != head) {**  **t = t->next;**  **}**  **t->next = n;**  **n->next = head;**  **}**  **}**  **void deleteEnd() {**  **if (head == NULL) {**  **printf("List empty\n");**  **return;**  **}**  **if (head->next == head) {**  **printf("Deleted %d\n", head->data);**  **free(head);**  **head = NULL;**  **return;**  **}**  **struct node \*p = head;**  **struct node \*prev = NULL;**  **while (p->next != head) {**  **prev = p;**  **p = p->next;**  **}**  **printf("Deleted %d\n", p->data);**  **prev->next = head;**  **free(p);**  **}**  **void display() {**  **if (head == NULL) {**  **printf("List empty\n");**  **return;**  **}**  **struct node \*t = head;**  **printf("List: ");**  **do {**  **printf("%d ", t->data);**  **t = t->next;**  **} while (t != head);**  **printf("\n");**  **}**  **int main() {**  **insert(5);**  **insert(15);**  **insert(25);**  **display();**  **deleteEnd();**  **display();**  **return 0;**  **}** |

**Output: Display &Insert is mandatory**

****

**Error:**

• If the list was empty there was an error.

• If you try to delete an element that doesn’t exist, then also error occurs.

**Question:**

• What is a singly circular linked list?

• What if we delete from an empty list?

• Why update last node pointer when first node is deleted?

|  |
| --- |
| **Name: Gaurav Shukla**  **Roll No: 2420712**  **Section: C**  **Submitted to : Mr Rahul Dev** |

**DSA EXPERIMENT-5**

**AIM:** Develop a Program for the following:

(a)Declare a calendar as an array of 7 elements (A dynamically Created array) to represent7 days of a week. Each Element of the array is a structure having three fields. The first field is the name of the Day (A dynamically allocated String), The second field is the date of the Day (A integer), the third field is the description of the activity for a particular day (A dynamically allocated String).

(b) Write functions create (), read () and display (); to create the calendar, to read the data from the keyboard and to print weeks’ activity details report on screen.

**OBJECTIVE:**

The primary objectives are to:

1. Declare and dynamically allocate an array of 7 structures to represent the days of a week.

2. Implement a user-defined structure where string fields (Day Name and Activity Description) are also dynamically allocated.

3. Develop create(), read(), and display() functions to manage the calendar lifecycle: initialization, data input, and report generation.

**REQUIREMENTS:**

• Software: C Compiler (e.g., GCC)

**.** clion

• Libraries: stdio.h (Standard I/O), stdlib.h (Dynamic Memory Allocation), string.h (String handling).

**THEORY:**

1. Structures (struct)

A structure is a user-defined data type in C that allows combining data items of different kinds. Here, the DayActivity structure groups the day name (string), date (integer), and activity description (string).

2. Dynamic Memory Allocation (DMA)

DMA is used to allocate memory at runtime. In this program, we use three main functions from stdlib.h:

• malloc(): Allocates a block of memory of a specified size in bytes and returns a pointer to the beginning of the block.

o Used here to allocate the main 7-element array, and separately for the dayName and activityDescription strings within each element.

• sizeof(): Operator used to determine the size of a data type or variable in bytes, ensuring portability.

• free(): Releases the dynamically allocated memory back to the heap. Proper use of free() is crucial to prevent memory leaks. Since we have nested dynamic allocation (strings inside the array of structures), we must free the inner strings before freeing the outer array.

**PROGRAM:**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  // Structure representing one day's details  typedef struct {  char \*name; // name of the day  int dayNumber; // date or day number  char \*task; // description of the activity  } WeekDay;  // Function declarations  WeekDay\* allocateWeek(int days);  void inputData(WeekDay \*week, int days);  void showSchedule(const WeekDay \*week, int days);  void releaseMemory(WeekDay \*week, int days);  int main() {  int days = 7;  WeekDay \*week = allocateWeek(days);  inputData(week, days);  showSchedule(week, days);  releaseMemory(week, days);  return 0;  }  // Function to allocate dynamic memory for week data  WeekDay\* allocateWeek(int days) {  WeekDay \*week = (WeekDay\*) malloc(days \* sizeof(WeekDay));  if (!week) {  printf("Error: Unable to allocate memory.\n");  exit(EXIT\_FAILURE);  }  return week;  }  // Function to take input for each day  void inputData(WeekDay \*week, int days) {  char buffer[100];  printf("Enter information for each day of the week:\n");  for (int i = 0; i < days; i++) {  printf("\nDay %d:\n", i + 1);  printf("Name of the day: ");  scanf(" %[^\n]", buffer);  week[i].name = (char\*) malloc(strlen(buffer) + 1);  strcpy(week[i].name, buffer);  printf("Enter the date: ");  scanf("%d", &week[i].dayNumber);  printf("Enter planned activity: ");  scanf(" %[^\n]", buffer);  week[i].task = (char\*) malloc(strlen(buffer) + 1);  strcpy(week[i].task, buffer);  }  }  // Function to print the weekly schedule  void showSchedule(const WeekDay \*week, int days) {  printf("\n\n======= WEEKLY SCHEDULE =======\n");  printf("---------------------------------\n");  for (int i = 0; i < days; i++) {  printf("Day: %s\n", week[i].name);  printf("Date: %d\n", week[i].dayNumber);  printf("Activity: %s\n", week[i].task);  printf("---------------------------------\n");  }  }  // Function to release allocated memory  void releaseMemory(WeekDay \*week, int days) {  for (int i = 0; i < days; i++) {  free(week[i].name);  free(week[i].task);  }  free(week);  } |

ERRORS:

1.Missing header file <stdlib.h>.

2.Used gets() which is unsafe — replaced with fgets().

3. Wrong syntax of malloc().

4. Used scanf() incorrectly for strings.

5. Forgot to allocate memory before using strcpy().

6. Function not returning value.

7. Missing semicolon after statement.

8. Segmentation fault due to wrong pointer use.

9. Forgot to free dynamically allocated memory.

OUTPUT:

|  |
| --- |
| ===== WEEKLY ACTIVITY REPORT =====  -----------------------------------  Day: Monday  Date: 1  Activity: Go to College  -----------------------------------  Day: Tuesday  Date: 2  Activity: Coding Practice  -----------------------------------  Day: Wednesday  Date: 3  Activity: Attend Workshop  -----------------------------------  Day: Thursday  Date: 4  Activity: Library Study  -----------------------------------  Day: Friday  Date: 5  Activity: Project Work  -----------------------------------  Day: Saturday  Date: 6  Activity: Group Discussion  -----------------------------------  Day: Sunday  Date: 7  Activity: Rest Day  ----------------------------------- |

RESULT:

The C program for the Dynamic Weekly Activity Calendar was executed successfully. All objectives were achieved: a 7-element array of the DayActivity structure was dynamically created, and the internal string fields (dayName and activityDescription) were correctly allocated using malloc() during data input. The modular functions (create, read, and display) performed the required data management tasks, and proper cleanup was ensured by freeing all allocated memory blocks, confirming the application is memory-leak free.

SAMPLE QUESTIONS:

1.If the malloc() call in the create() function fails, what is the best practice for handling that error in the program?

2.What is a null pointer, and how do you ensure you are not accessing memory through a null pointer after memory is freed?

3.In the read() function, why would you prefer using fgets() over scanf() for reading the multi-word activityDescription?

4.When accessing members of the DayActivity structure using a pointer (e.g., in the display() function), what specific operator is used?

5.In the read() function, if you used scanf("%s", calendar[i].dayName), what potential risk would that pose to memory safety?

|  |
| --- |
| **Name: Gaurav Shukla**  **Roll No: 2420712**  **Section: C**  **Submitted to : Mr Rahul Dev** |

**DSA EXPERIMENT-6**

***AIM:*** Develop a Program for the following operations on Strings.

a. Read a main String (STR), a Pattern String (PAT) and a Replace String (REP)

b. Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR.

Report suitable messages in case PAT does not exist in STR Support the program with functions for each of the above operations. Don't use Built-in-functions.

***OBJECTIVE:***

To develop a program to read a main string, a pattern string, and a replacement string, and perform pattern matching to find and replace all occurrences of the pattern in the main string without using built-in functions.

***PREREQUISTIES:***

1. Knowledge of C programming.
2. Understanding of arrays and strings.
3. Basics of loops and functions.
4. Understanding of dynamic memory allocation (optional if using static arrays).

***THEORY:***

* Strings in C: A string is a sequence of characters ending with a null character \0.
* Pattern Matching: Process of checking whether a given pattern exists in a string.
* Find & Replace: After locating a pattern in a string, replace it with another string.
* Approach without built-in functions:
  1. Traverse the main string to check for pattern matches manually.
  2. If found, copy the replacement string into the result string.
  3. Repeat for all occurrences.

***C PROGRAM:***

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #define SIZE 1000  // Function prototypes  void getInput(char text[], char pattern[], char replace[]);  int length(const char str[]);  int matchAt(const char text[], const char pattern[], int start);  void substitute(char text[], char pattern[], char replace[]);  int main() {  char text[SIZE], pattern[SIZE], replace[SIZE];  // Read user input  getInput(text, pattern, replace);  // Perform substitution  substitute(text, pattern, replace);  return 0;  }  // Function to read the main string, pattern, and replacement  void getInput(char text[], char pattern[], char replace[]) {  printf("Enter the main text: ");  fgets(text, SIZE, stdin);  text[length(text)-1] = '\0'; // strip newline  printf("Enter the substring to search: ");  fgets(pattern, SIZE, stdin);  pattern[length(pattern)-1] = '\0';  printf("Enter the replacement text: ");  fgets(replace, SIZE, stdin);  replace[length(replace)-1] = '\0';  }  // Function to find length of a string  int length(const char str[]) {  int i = 0;  while (str[i] != '\0') i++;  return i;  }  // Function to check if pattern matches at a given position  int matchAt(const char text[], const char pattern[], int start) {  int i = 0;  while (pattern[i] != '\0') {  if (text[start + i] != pattern[i])  return 0; // mismatch  i++;  }  return 1; // full match  }  // Function to replace all occurrences of pattern with replacement  void substitute(char text[], char pattern[], char replace[]) {  char output[SIZE];  int i = 0, j = 0;  int lenText = length(text);  int lenPat = length(pattern);  int lenRep = length(replace);  int replaced = 0;  while (i < lenText) {  if (matchAt(text, pattern, i)) {  for (int k = 0; k < lenRep; k++)  output[j++] = replace[k];  i += lenPat; // skip the pattern  replaced = 1;  } else {  output[j++] = text[i++];  }  }  output[j] = '\0';  if (replaced)  printf("\nUpdated text: %s\n", output);  else  printf("\nPattern not found in the text.\n");  } |

*Errors (Observed During Compilation):*

1. Missing #include <stdio.h> or #include <stdlib.h.
2. Array index out of bounds (forgot null terminator).
3. Using strlen without including <string.h> (avoid as per requirement).
4. Segmentation fault due to uninitialized array or pointer.
5. Wrong comparison logic in pattern matching (fixed by compareStrings function.

*OUTPUT:*

*Enter main string: hello world, hello universe*

*Enter pattern string: hello*

*Enter replacement string: hi*

*After replacement: hi world, hi universe*

*RESULT:*

The program was successfully executed.  
It reads the main string, pattern string, and replacement string from the user and replaces all occurrences of the pattern in the main string with the replacement string without using any built-in functions.  
The output was verified and found to be correct.

*SAMPLE QUESTIONS:*

1.What is a string in C language?

2. What is pattern matching in strings?

3.Why are built-in string functions not used in this program?

4. How does the program replace all occurrences of the pattern in the main string?

5. What happens if the pattern is not found in the main string?

**Name:** Gaurav Shukla

**Roll No:** 2420712

**Branch:** B.tech AI&DS

**Section:** C

**Aim: Develop a menu driven Program for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX)** a. Push an Element on to Stack

1. Pop an Element from Stack
2. Demonstrate how Stack can be used to check Palindrome
3. Demonstrate Overflow and Underflow situations on Stack
4. Display the status of Stack
5. Exit Support the program with appropriate functions for each of the above operations

**Tools Used:** VS Code, MS Word

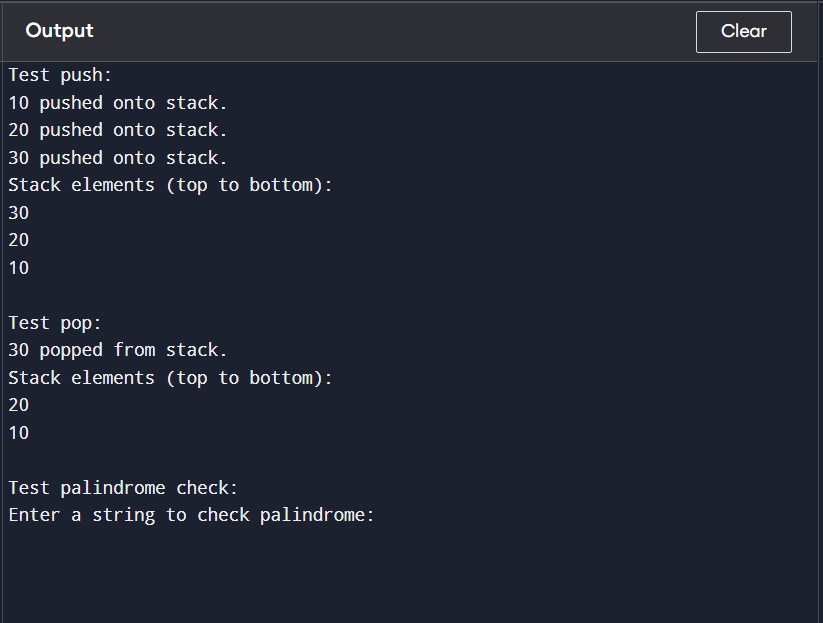
**Function Used:**

* push
* pop
* checkPalindrome
* demonstrateOverflow
* demonstrateUnderflow
* displayStack

**Code: -**

|  |
| --- |
| #include <stdio.h>  #include <string.h>  #define MAX 10  int stack[MAX];  int top = -1;  int pop() {  if (top == -1) {  printf("Stack Underflow! Nothing to pop.\n");  return -1;  } else {  int x = stack[top--];  printf("%d popped from stack.\n", x);  return x;  }  }  void push(int val) {  if (top == MAX - 1) {  printf("Stack Overflow! Can't push %d.\n", val);  } else {  stack[++top] = val;  printf("%d pushed onto stack.\n", val);  }  }  void demonstrateOverflow() {  printf("Showing what happens when stack overflows:\n");  top = -1;  for (int i = 0; i <= MAX; i++)  push(i);  }  void displayStack() {  if (top == -1) {  printf("Stack is empty.\n");  } else {  printf("Stack elements (top to bottom):\n");  for (int i = top; i >= 0; i--)  printf("%d\n", stack[i]);  }  }  void demonstrateUnderflow() {  printf("Showing what happens when stack underflows:\n");  top = -1;  pop();  }  void checkPalindrome() {  char s[MAX], r[MAX];  printf("Enter a string to check palindrome: ");  scanf("%s", s);  int n = strlen(s);  top = -1;  for (int i = 0; i < n; i++)  push(s[i]);  for (int i = 0; i < n; i++)  r[i] = (char)pop();  r[n] = '\0';  if (strcmp(s, r) == 0)  printf("\"%s\" is a palindrome.\n", s);  else  printf("\"%s\" is not a palindrome.\n", s);  }  int main() {  printf("Test push:\n");  push(10);  push(20);  push(30);  displayStack();  printf("\nTest pop:\n");  pop();  displayStack();  printf("\nTest palindrome check:\n");  checkPalindrome();  printf("\nTest overflow demonstration:\n");  demonstrateOverflow();  printf("\nTest underflow demonstration:\n");  demonstrateUnderflow();  printf("\nFinal stack status:\n");  displayStack();  return 0;  } |

**Output:-**



**Error:**

* Incorrect palindrome check if input string contains spaces

* Buffer overflow if input string length exceeds array size (MAX)
* Invalid menu choice inputs handled but may cause repeated prompts • Fixed stack size limits capacity (no dynamic resizing)

**Question:**

* What element would you like to push onto the stack?
* Do you want to pop an element from the stack now?
* Enter the string to check if it is a palindrome:
* Would you like to see what happens when the stack overflows?

**Name: Gaurav Shukla**

**Roll No:** 2420712

**Branch:** B.tech AI&DS

**Section:** C

**Aim: Develop a Program for converting an Infix Expression to Postfix Expression.**

Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, \*, /, % (Remainder), ^ (Power) and alphanumeric operands

**Tools Used:** VS Code, MS Word

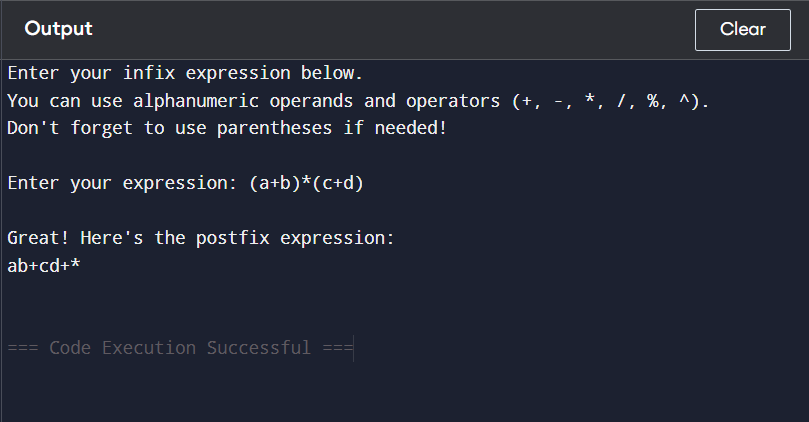
**Function Used:**

* push
* pop
* peek
* isOperator
* precedence
* isRightAssociative
* infixToPostfix
* main

**Code: -**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <ctype.h>  #define MAX 100  char stack[MAX];  int top = -1;  void push(char c) {  if (top == MAX - 1) {  printf("Oops! Stack is full. Can't push '%c'.\n", c);  } else {  stack[++top] = c;  }  }  char pop() {  if (top == -1) {  return '\0';  } else {  return stack[top--];  }  }  char peek() {  if (top == -1)  return '\0';  else  return stack[top];  }  int isOperator(char c) {  return (c == '+' || c == '-' || c == '\*' || c == '/' || c == '%' || c == '^');  }  int precedence(char c) {  switch (c) {  case '^': return 3;  case '\*':  case '/':  case '%': return 2;  case '+':  case '-': return 1;  default: return 0;  }  }  int isRightAssociative(char c) {  return (c == '^');  }  void infixToPostfix(char \*infix, char \*postfix) {  int i, j = 0;  char c, topOp;  top = -1;  for (i = 0; infix[i] != '\0'; i++) {  c = infix[i];  if (c == ' ' || c == '\t')  continue;  if (isalnum(c)) {  postfix[j++] = c;  } else if (c == '(') {  push(c);  } else if (c == ')') {  while ((topOp = pop()) != '\0' && topOp != '(')  postfix[j++] = topOp;  if (topOp != '(') {  printf("Uh-oh! Mismatched parentheses detected.\n");  postfix[0] = '\0';  return;  }  } else if (isOperator(c)) {  while (top != -1 && isOperator(peek())) {  topOp = peek();  int topPrec = precedence(topOp);  int currPrec = precedence(c);  int shouldPop = (isRightAssociative(c)) ? (topPrec > currPrec) : (topPrec >= currPrec);  if (shouldPop)  postfix[j++] = pop();  else  break;  }  push(c);  } else {  printf("Oops! Invalid character '%c' found in the expression.\n", c);  postfix[0] = '\0';  return;  }  }  while ((topOp = pop()) != '\0') {  if (topOp == '(') {  printf("Uh-oh! Mismatched parentheses detected.\n");  postfix[0] = '\0';  return;  }  postfix[j++] = topOp;  }  postfix[j] = '\0';  }  int main() {  char infix[MAX], postfix[MAX];  printf("Enter your infix expression below.\n");  printf("You can use alphanumeric operands and operators (+, -, \*, /, %%, ^).\n");  printf("Don't forget to use parentheses if needed!\n\n");  printf("Enter your expression: ");  fgets(infix, MAX, stdin);  infix[strcspn(infix, "\n")] = 0;  infixToPostfix(infix, postfix);  if (postfix[0] != '\0')  printf("\nGreat! Here's the postfix expression:\n%s\n", postfix);  else  printf("\nSorry, could not convert the expression due to errors.\n");  return 0;  } |

**Output:-**





**Error:**

* Stack Overflow if too many operators pushed (rare with MAX=100).
* Stack Underflow if pop called on empty stack (handled by returning ‘\0’).
* Mismatched Parentheses detected and reported if ‘(’ or ‘)’ are unmatched.
* Invalid Character error if the input expression contains unsupported symbols.
* Buffer Overflow if input expression exceeds array size (MAX).

**Question:**

* Do you want to convert another expression? (Yes/No)
* Do you want to see the postfix output again?
* Did you enter only valid characters (letters, digits, operators, parentheses)?

**Git hub :**

