TRIBHUVAN UNIVERSITY

**PATAN MULTIPLE CAMPUS**

PATANDHOKA, LALITPUR

**SUBJECT**: DATA STRUCTURES AND ALGORITHMS (BIT 201)

|  |  |
| --- | --- |
| **SUBMITTED BY** | **SUBMITTED TO** |
| **NAME:** SURESH DAHAL | **RAKESH KUMAR BACHCHAN** |
| **ROLL NO:** 23 |  |
| **CLASS:** BIT – II/I | **………………………..** |
| **DATE:** 2081/10/27 | **CHECKED BY** |

1. **Write a program to find sum of two 1-D arrays and store the sum of corresponding elements into third array & print all three arrays.**

**Algorithm:**

* Start
* Take the size of the arrays as input
* Declare three arrays of the given size
* Take input for the first and second arrays
* Add corresponding elements of both arrays and store them in the third array
* Print all three arrays
* End

**Example:**

Let the size of the arrays be 3

Array 1: 12 21 10

Array 2: 1 2 3

Output:

First Array: 12 21 10

Second Array: 1 2 3

Sum Array: 13 23 13

**Program:**

#include <stdio.h>

int main() {

int n;

printf("Enter the size of the arrays: ");

scanf("%d", &n);

int arr1[n], arr2[n], sum[n];

printf("Enter elements of first array:\n");

for (int i = 0; i < n; i++) {

scanf("%d", &arr1[i]);

}

printf("Enter elements of second array:\n");

for (int i = 0; i < n; i++) {

scanf("%d", &arr2[i]);

}

for (int i = 0; i < n; i++) {

sum[i] = arr1[i] + arr2[i];

}

printf("\nFirst Array: ");

for (int i = 0; i < n; i++) {

printf("%d ", arr1[i]);

}

printf("\nSecond Array: ");

for (int i = 0; i < n; i++) {

printf("%d ", arr2[i]);

}

printf("\nSum Array: ");

for (int i = 0; i < n; i++) {

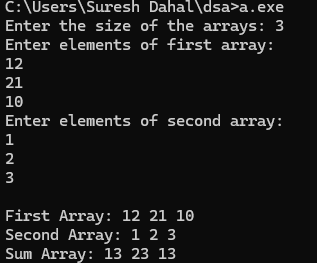
printf("%d ", sum[i]);

}

return 0;

}

**Output:**

****

**Conclusion:**

Hence the sum of two 1-D arrays are found and displayed using C programming language.

1. **Write a program to find the smallest and largest number in an array of size 10 without sorting the elements.**

**Algorithm:**

* Start
* Declare an array of size 10
* Take input for the 10 elements of the array
* Initialize two variables, min and max, with the first element of the array
* Traverse the array:
* If an element is smaller than min, update min
* If an element is larger than max, update max
* Print the smallest and largest numbers
* Stop

**Example:**

Input:

Array: 0 5 1 3 4 12 2 9 6 7

Output:

Smallest Number: 0

Largest Number: 12

**Program:**

#include <stdio.h>

int main() {

int arr[10];

printf("Enter 10 elements of the array:\n");

for (int i = 0; i < 10; i++) {

scanf("%d", &arr[i]);

}

int min = arr[0], max = arr[0];

for (int i = 1; i < 10; i++) {

if (arr[i] < min) {

min = arr[i];

}

if (arr[i] > max) {

max = arr[i];

}

}

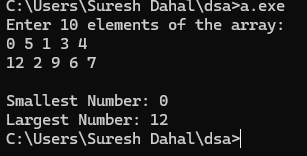
printf("\nSmallest Number: %d", min);

printf("\nLargest Number: %d", max);

return 0;

}

**Output:**



**Conclusion:**

Hence the smallest and the largest elements were found and displayed from the array of size 10 without sorting the array elements.

**3. Using an array, perform following tasks (use switch case for many).**

**a) Insert: Insert a value from specified position.**

**b) Delete: Delete a value from a specified position.**

**c) Traverse: Print all elements from 0 to n-1.**

**d) Searching: Search a particular value in array.**

**Algorithm:**

* Start
* Input the number of elements (n) and array elements
* Repeat until exit:
* Display menu: Insert, Delete, Traverse, Search, Exit
* Switch(choice):
  + Case 1 (Insert):
    - If n == 100, print "Array full"
    - Else, shift elements and insert value at pos
  + Case 2 (Delete):
    - If pos is invalid, print "Invalid position"
    - Else, shift elements to remove value at pos
  + Case 3 (Traverse): Print all elements
  + Case 4 (Search): Loop through array, print index if found
  + Case 5 (Exit): Stop program
* End

**Example:**

**Input:**

Array: **1 2 3**  
Choose operation:

1. Insert **(Position: 1, Value: 12) → 1 12 2 3**
2. Delete **(Position: 1)** → **1 2 3**
3. Traverse → **1 2 3**
4. Search **(Value: 12)** → Found at index **1**

**Program:**

#include <stdio.h>

#define MAX 100

int main() {

int arr[MAX], n, choice, pos, value, i;

printf("Enter the number of elements in the array: ");

scanf("%d", &n);

printf("Enter %d elements:\n", n);

for (i = 0; i < n; i++) {

scanf("%d", &arr[i]);

}

while (1) {

printf("\nMenu:\n");

printf("1. Insert\n2. Delete\n3. Traverse\n4. Search\n5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1: // Insert

if (n == MAX) {

printf("Array is full! Cannot insert.\n");

break;

}

printf("Enter position (0 to %d) and value: ", n);

scanf("%d %d", &pos, &value);

if (pos < 0 || pos > n) {

printf("Invalid position!\n");

} else {

for (i = n; i > pos; i--) {

arr[i] = arr[i - 1];

}

arr[pos] = value;

n++;

printf("Value inserted successfully.\n");

}

break;

case 2: // Delete

printf("Enter position (0 to %d) to delete: ", n - 1);

scanf("%d", &pos);

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

printf("Invalid position!\n");

} else {

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

arr[i] = arr[i + 1];

}

n--;

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

}

break;

case 3: // Traverse

printf("Array elements: ");

for (i = 0; i < n; i++) {

printf("%d ", arr[i]);

}

printf("\n");

break;

case 4: // Search

printf("Enter value to search: ");

scanf("%d", &value);

int found = 0;

for (i = 0; i < n; i++) {

if (arr[i] == value) {

printf("Value found at index %d\n", i);

found = 1;

break;

}

}

if (!found) {

printf("Value not found in array.\n");

}

break;

case 5: // Exit

return 0;

default:

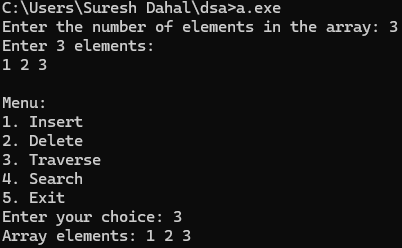
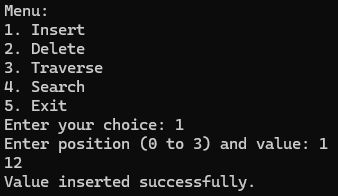
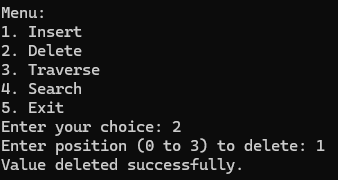
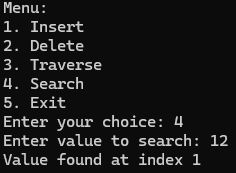
printf("Invalid choice! Try again.\n");

}

}

}

**Output:**

**Conclusion:**

Hence the basic array operations: insertion, deletion, traversal, and searching was implemented.

1. **Write a menu driven program to illustrate basic operation of stack using array. (PUSH, POP, TOP, DISPLAY ALL)**

**Algorithm:**

* Start
* Initialize an empty stack with top = -1
* Repeat until exit:
* Display menu: PUSH, POP, TOP, DISPLAY ALL, EXIT
* Switch(choice):
  + Case 1 (PUSH):
    - If top == MAX - 1, print "Stack Overflow"
    - Else, increment top and insert value
  + Case 2 (POP):
    - If top == -1, print "Stack Underflow"
    - Else, remove the top element and decrement top
  + Case 3 (TOP):
    - If top == -1, print "Stack is empty"
    - Else, print stack[top]
  + Case 4 (DISPLAY ALL):
    - If top == -1, print "Stack is empty"
    - Else, print elements from top to 0
  + Case 5 (EXIT): Stop program
* End

**Example:**

* PUSH 12
* TOP → Output: 12
* DISPLAY ALL → Output: 12
* POP → Output: 12 popped from stack
* DISPLAY ALL → Output: 12

**Program:**

#include <stdio.h>

#define MAX 100

int stack[MAX], top = -1;

void push() {

int value;

if (top == MAX - 1) {

printf("Stack Overflow!\n");

} else {

printf("Enter value to push: ");

scanf("%d", &value);

stack[++top] = value;

printf("%d pushed to stack.\n", value);

}

}

void pop() {

if (top == -1) {

printf("Stack Underflow!\n");

} else {

printf("%d popped from stack.\n", stack[top--]);

}

}

void topElement() {

if (top == -1) {

printf("Stack is empty!\n");

} else {

printf("Top element: %d\n", stack[top]);

}

}

void display() {

if (top == -1) {

printf("Stack is empty!\n");

} else {

printf("Stack elements: ");

for (int i = top; i >= 0; i--) {

printf("%d ", stack[i]);

}

printf("\n");

}

}

int main() {

int choice;

while (1) {

printf("\n1. PUSH\n2. POP\n3. TOP\n4. DISPLAY ALL\n5. EXIT\nEnter choice: ");

scanf("%d", &choice);

switch (choice) {

case 1: push(); break;

case 2: pop(); break;

case 3: topElement(); break;

case 4: display(); break;

case 5: return 0;

default: printf("Invalid choice! Try again.\n");

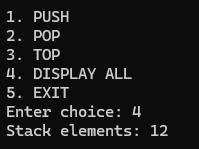
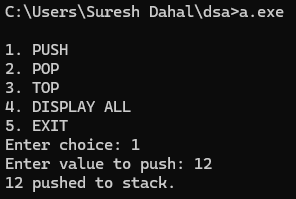
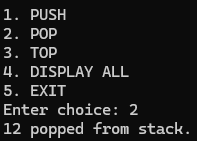
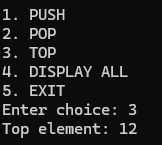
}

}

return 0;

}

**Output:**

**Conclusion:**

Hence the basic operations of stack i.e. push, pop, top and display all were implemented in C programming.

1. **Write a program to reverse give string using stack.**

**Algorithm:**

* Start
* Initialize an empty stack with top = -1
* Input the string
* Push each character of the string onto the stack
* Pop characters from the stack one by one and store them in a new string
* Print the reversed string
* End

**Example:**

Input:

Suresh

Stack (After Pushing Characters):

{S, u, r, e, s, h}

Popped Characters (Reversed String):

{h, s, e, r, u, S}

Output:

Reversed String: hseruS

**Program:**

#include <stdio.h>

#include <string.h>

#define MAX 100

char stack[MAX];

int top = -1;

void push(char ch) {

if (top == MAX - 1) {

printf("Stack Overflow!\n");

} else {

stack[++top] = ch;

}

}

char pop() {

if (top == -1) {

printf("Stack Underflow!\n");

return '\0';

} else {

return stack[top--];

}

}

void reverseString(char str[]) {

int len = strlen(str);

for (int i = 0; i < len; i++) {

push(str[i]);

}

for (int i = 0; i < len; i++) {

str[i] = pop();

}

}

int main() {

char str[MAX];

printf("Enter a string: ");

scanf("%s", str);

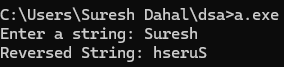
reverseString(str);

printf("Reversed String: %s\n", str);

return 0;

}

**Output:**

****

**Conclusion:**

Hence we have used stack to reverse a string using the **LIFO (Last In, First Out)** property.

1. **Write a program to convert decimal to binary using string.**

**Algorithm:**

 Start

 Input the decimal number

 Initialize an empty string to store binary

 Repeat the following until the decimal number becomes 0:

* Divide the decimal number by 2
* Store the remainder (either 0 or 1) in the string

 Reverse the string to get the correct binary representation

 Print the binary string

 End

**Example:**

Input:

13

Process:

13 / 2 = 6 remainder 1

6 / 2 = 3 remainder 0

3 / 2 = 1 remainder 1

1 / 2 = 0 remainder 1

Binary (before reversing): {1, 0, 1, 1}

Binary (after reversing): 1101

**Program:**

#include <stdio.h>

#include <string.h>

void decimalToBinary(int n) {

char binary[32];

int index = 0;

if (n == 0) {

printf("Binary: 0\n");

return;

}

// Convert decimal to binary and store remainders

while (n > 0) {

binary[index++] = (n % 2) + '0'; // Store '0' or '1' as char

n = n / 2;

}

binary[index] = '\0'; // Null-terminate the string

// Reverse the string to get the correct binary representation

int start = 0;

int end = index - 1;

while (start < end) {

char temp = binary[start];

binary[start] = binary[end];

binary[end] = temp;

start++;

end--;

}

printf("Binary: %s\n", binary);

}

int main() {

int decimal;

printf("Enter a decimal number: ");

scanf("%d", &decimal);

decimalToBinary(decimal);

return 0;

}

**Output:**



**Conclusion:**

Hence we have converted the decimal number into the binary by iteratively dividing the decimal by 2, storing the remainders in a string and reversing the string.

1. **Write a program to evaluate postfix expression using stack.**

**Algorithm:**

 Start

 Initialize an empty stack.

 For each character in the postfix expression (from left to right):

* If the character is a number, push it onto the stack.
* If the character is an operator (such as +, -, \*, /):
  + Pop two numbers from the stack.
  + Perform the operation on the two numbers.
  + Push the result back onto the stack.

 After processing the entire expression, the result will be the only number left in the stack.

 Print the result.

 End.

**Example:**

**Postfix Expression:**

5 3 + 8 2 - \*

#### **Process:**

* Push 5 and 3 onto the stack.
* Encounter +, pop 3 and 5, add them (5 + 3 = 8), and push the result (8) back onto the stack.
* Push 8 and 2 onto the stack.
* Encounter -, pop 2 and 8, subtract them (8 - 2 = 6), and push the result (6) back onto the stack.
* Encounter \*, pop 6 and 8, multiply them (8 \* 6 = 48), and push the result (48) back onto the stack.

The final result in the stack is 48.

**Output:**

Result: 48

**Program:**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#define MAX 50

int performOperation(int a, int b, char op) {

switch (op) {

case '+': return a + b;

case '-': return a - b;

case '\*': return a \* b;

case '/': return a / b;

default: return 0;

}

}

// Function to evaluate the postfix expression

int evaluatePostfix(char\* expr) {

int stack[MAX];

int top = -1;

int i = 0;

int a, b, result;

// Loop through each character in the postfix expression

while (expr[i] != '\0') {

// Skip spaces

if (expr[i] == ' ') {

i++;

continue;

}

// If the character is a digit, handle multi-digit numbers

if (isdigit(expr[i])) {

int num = 0;

// Process full multi-digit number eg: 23

while (isdigit(expr[i])) {

num = num \* 10 + (expr[i] - '0');

i++;

}

stack[++top] = num; // Push the full number onto the stack

}

// If the character is an operator

else if (expr[i] == '+' || expr[i] == '-' || expr[i] == '\*' || expr[i] == '/') {

b = stack[top--];

a = stack[top--];

result = performOperation(a, b, expr[i]);

stack[++top] = result; // Push the result back to the stack

i++;

} else {

i++; // Ignore invalid characters (you can add error handling if needed)

}

}

// The result will be the only element left in the stack

return stack[top];

}

int main() {

char expr[MAX];

printf("Enter postfix expression: ");

fgets(expr, MAX, stdin);

// Remove newline character if present

if (expr[strlen(expr) - 1] == '\n') {

expr[strlen(expr) - 1] = '\0';

}

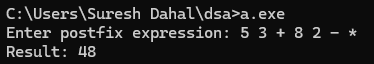
int result = evaluatePostfix(expr);

printf("Result: %d\n", result);

return 0;

}

**Output:**



**Conclusion:**

Hence we have used stack to evaluate postfix expressions. This approach ensures correct order of operations, as the stack manages operands and operators dynamically during traversal.

1. **Write a program to convert infix expression into postfix expression.**

**Algorithm:**

 Initialize:

* Create a stack to store operators.
* Create an array for the output (postfix expression).

 Scan the Infix Expression:

* Read each character from the left to the right.

 If the character is an operand:

* Add it directly to the output.

 If the character is an operator:

* While the stack is not empty and the operator at the top of the stack has higher or equal precedence:
  + Pop the stack and add the operator to the output.
* Push the current operator to the stack.

 If the character is a left parenthesis ( or {:

* Push it to the stack.

 If the character is a right parenthesis ) or }:

* Pop from the stack and add operators to the output until you encounter a left parenthesis or {.
* Remove the left parenthesis or { from the stack.

 Repeat steps 3 to 6 until the end of the expression.

 Pop remaining operators from the stack and add them to the output.

 Return the postfix expression.

**Example:**

Consider infix expression: A + B \* (C - D) / E

1. Start with an empty stack and output.
2. Read A: Add it to the output → Output: A.
3. Read +: Push it to the stack → Stack: +.
4. Read B: Add it to the output → Output: A B.
5. Read \*: Push it to the stack → Stack: + \*.
6. Read (: Push it to the stack → Stack: + \* (.
7. Read C: Add it to the output → Output: A B C.
8. Read -: Push it to the stack → Stack: + \* ( -.
9. Read D: Add it to the output → Output: A B C D.
10. Read ): Pop the stack until we reach (. Add - to the output → Output: A B C D -.
    * Now the stack is + \*.
11. Read /: Since / has higher precedence than +, push it to the stack → Stack: + \* /.
12. Read E: Add it to the output → Output: A B C D - E.
13. End of the expression:
    * Pop remaining operators from the stack and add them to the output:
      + \* → Output: A B C D - E / \*.
      + + → Output: A B C D - E / \* +.

**Program:**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#define MAX 100

*// Function to get precedence of operators*

int precedence(char op) {

if (op == '+' || op == '-')

return 1;

else if (op == '\*' || op == '/')

return 2;

return 0; *// For any invalid operator*

}

*// Function to check if the character is an operator*

int isOperator(char c) {

return (c == '+' || c == '-' || c == '\*' || c == '/');

}

*// Function to convert infix to postfix*

void infixToPostfix(char\* expr, char\* result) {

char stack[MAX];

int top = -1, k = 0; *// k is for result, top is for stack*

for (int i = 0; i < strlen(expr); i++) {

char c = expr[i];

*// If operand, add it to result*

if (isalnum(c)) {

result[k++] = c;

}

*// If '(', push it to stack*

else if (c == '(') {

stack[++top] = c;

}

*// If ')', pop until '('*

else if (c == ')') {

while (top != -1 && stack[top] != '(') {

result[k++] = stack[top--];

}

top--; *// Pop '(' from stack*

}

*// If operator, pop from stack to result*

else if (isOperator(c)) {

while (top != -1 && precedence(stack[top]) >= precedence(c)) {

result[k++] = stack[top--];

}

stack[++top] = c;

}

}

*// Pop all remaining operators in stack*

while (top != -1) {

result[k++] = stack[top--];

}

result[k] = '\0'; *// Null-terminate the result*

}

int main() {

char expr[MAX], result[MAX];

*// Input infix expression*

printf("Enter infix expression: ");

fgets(expr, MAX, stdin);

*// Remove newline character from input string*

if (expr[strlen(expr) - 1] == '\n') {

expr[strlen(expr) - 1] = '\0';

}

*// Convert infix to postfix*

infixToPostfix(expr, result);

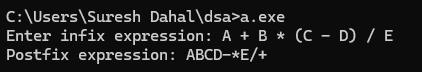
*// Output the postfix expression*

printf("Postfix expression: %s\n", result);

return 0;

}

**Output:**



**Conclusion:**

Hence we have written a program that converts an infix expression to a postfix expression using a stack.

1. **Write a program to insert and delete items from a linear queue.**

**Algorithm:**

* Initialization: Define a queue array, a front pointer, and a rear pointer to track the first and last elements.
* Insert Operation (Enqueue):
  + Check if the queue is full (i.e., rear == MAX - 1).
  + If not full, increment the rear pointer and insert the new element at queue[rear].
* Delete Operation (Dequeue):
  + Check if the queue is empty (i.e., front == -1 or front > rear).
  + If not empty, increment the front pointer to remove the front element.
* Display Operation: Traverse the queue from front to rear and print all elements.
* Exit: Exit the program when required.

**Example:**

For a queue with 5 positions, initially empty:

1. Insert 10
2. Insert 20
3. Delete (remove 10)
4. Insert 30
5. Display all elements

**Program:**

#include <stdio.h>

#include <stdlib.h>

#define MAX 5

int queue[MAX];

int front = -1, rear = -1;

int isFull() {

return rear == MAX - 1;

}

int isEmpty() {

return front == -1 || front > rear;

}

void enqueue(int value) {

if (isFull()) {

printf("Queue is full! Cannot insert.\n");

return;

}

if (front == -1) {

front = 0;

}

queue[++rear] = value;

printf("Inserted %d\n", value);

}

void dequeue() {

if (isEmpty()) {

printf("Queue is empty! Cannot delete.\n");

return;

}

printf("Deleted %d\n", queue[front]);

front++;

if (front > rear) {

front = rear = -1;

}

}

void display() {

if (isEmpty()) {

printf("Queue is empty!\n");

return;

}

printf("Queue elements: ");

for (int i = front; i <= rear; i++) {

printf("%d ", queue[i]);

}

printf("\n");

}

int main() {

int choice, value;

while (1) {

printf("\nMenu:\n");

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

printf("2. Dequeue (Delete)\n");

printf("3. Display\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

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 the program.\n");

exit(0);

default:

printf("Invalid choice! Try again.\n");

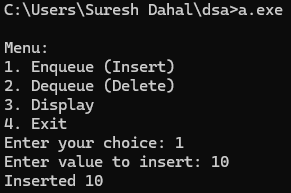
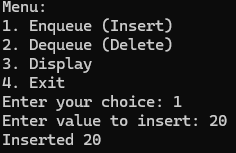
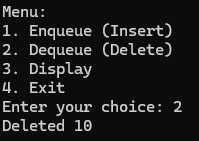
}

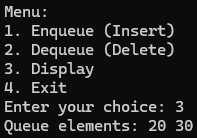
}

return 0;

}

**Output:**

****

**Conclusion:**

Hence we have implemented enqueue, dequeue, and display operations in linear queue handling both overflow (when the queue is full) and underflow (when the queue is empty).

1. **Write a program to implement a circular queue.**

**Algorithm:**

* Initialize:
* Define an array to hold the queue elements.
* Initialize front and rear pointers to -1.
* Enqueue (Insert):
* Check if the queue is full using the condition (rear + 1) % MAX == front.
* If full, print an error message and return.
* Otherwise, increment rear using circular logic: (rear + 1) % MAX.
* If the queue is empty (i.e., front == -1), set front = 0.
* Insert the element at the rear position.
* Dequeue (Delete):
* Check if the queue is empty using the condition front == -1.
* If empty, print an error message and return.
* Remove the element at the front position.
* If front equals rear, reset both to -1 (queue is now empty).
* Otherwise, increment front using circular logic: (front + 1) % MAX.
* Display:
* Check if the queue is empty.
* If not, loop from front to rear and print each element.
* Handle the circular nature by using modulo arithmetic: i = (i + 1) % MAX

**Example**

1. Insert 10
2. Insert 20
3. Delete (remove 10)
4. Insert 30
5. Display all elements

**Program:**

#include <stdio.h>

#define MAX 5

int queue[MAX];

int front = -1, rear = -1;

int isFull() {

return (rear + 1) % MAX == front;

}

int isEmpty() {

return front == -1;

}

void enqueue(int value) {

if (isFull()) {

printf("Queue is full! Cannot insert.\n");

return;

}

if (front == -1) {

front = 0; // First insertion

}

rear = (rear + 1) % MAX; // Circular increment

queue[rear] = value;

printf("Inserted %d\n", value);

}

void dequeue() {

if (isEmpty()) {

printf("Queue is empty! Cannot delete.\n");

return;

}

printf("Deleted %d\n", queue[front]);

if (front == rear) {

front = rear = -1; // Reset the queue when empty

} else {

front = (front + 1) % MAX;

}

}

void display() {

if (isEmpty()) {

printf("Queue is empty!\n");

return;

}

printf("Queue elements: ");

int i = front;

while (i != rear) {

printf("%d ", queue[i]);

i = (i + 1) % MAX;

}

printf("%d\n", queue[rear]);

}

int main() {

int choice, value;

while (1) {

printf("\nMenu:\n");

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

printf("2. Dequeue (Delete)\n");

printf("3. Display\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

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 the program.\n");

return 0;

default:

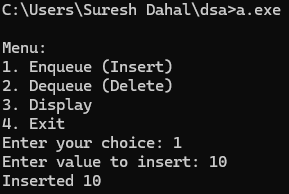
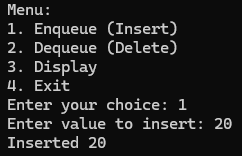
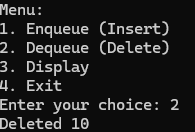
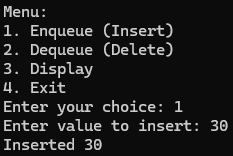
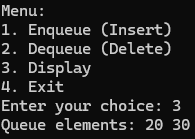
printf("Invalid choice! Try again.\n");

} }

return 0;

}

**Output:**

**Conclusion:**

Hence we have implemented the enqueue, dequeue, and display operation in the circular queue. This implementation avoids the limitation of a linear queue, where space cannot be reused after dequeuing.