1. When we delete the first node in double circular linked list what will be taken as the base address?
2. How we will get to know that we have reached the first node in circular linked list?
3. To free up allocated memory --- we need to free head and iterate

**STACK**

LIFO

* Stack will grow from bottom to top
* When t == b 🡺 container is empty
* When t == limit 🡺 container is full
* Stack have limitations in size
* Insert/**push** --- adding element by checking whether the **stack is full or not**
* Remove/pop/**pull** ---deleting element by checking whether the **stack is empty or not**

**NOTE: Count the number of nodes in Linked list**

How to avoid multiple definitions of variables

**QUEUE**

FIFO

* Front == rear 🡺 empty queue (rear is increasing)
* Rear == MAX 🡺 queue full
* Front increases till MAX while deleting
* When front reaches rear reset front and rear i.e f =0, r = 0 during removing element
* Push here is enqueue and pop here is dequeue

1 #include <stdio.h>

2

3 #define MAX 5

4

5 int rear = 0;

6 int front = 0;

7

8 int q[MAX];

9

10 void enqueue(int);

11 void dequeue();

12 void dispQueue();

13

14 int main()

15 {

16 enqueue(10);

17 enqueue(20);

18 enqueue(30);

19 enqueue(40);

20 enqueue(50);

21 enqueue(60);

22

23 dispQueue();

24

25 dequeue();

26 dequeue();

27 dequeue();

28 dequeue();

29 dequeue();

30 dequeue();

31

32 dispQueue();

33 }

34

35 void dispQueue()

36 {

37 int i;

38 if(front == rear)

39 {

40 printf("\nEmpty Queue\n");

41 return;

42 }

43 printf("\nQueue is\n");

44 for(i=front;i<rear;i++)

45 printf("%d",q[i]);

46 printf("\n\n");

47 }

48

49 void enqueue(int val)

50 {

51 if(rear == MAX)

52 {

53 printf("\nQueue is full\n");

54 return;

55 }

56

57 q[rear] = val;

58 rear++;

59 }

60

61 void dequeue()

62 {

63 if(front == rear)

64 {

65 front = 0;

66 rear = 0;

67 printf("\nEmpty queue\n"); // Reset the queue

68 return;

69 }

70 printf("\n%d is dequeued from the queue",q[front]);

71 front++;

72 }

O/P

user50@trainux01:~/practice/try/day17$ gcc queue.c

user50@trainux01:~/practice/try/day17$ ./a.out

Queue is full

Queue is

1020304050

10 is dequeued from the queue

20 is dequeued from the queue

30 is dequeued from the queue

40 is dequeued from the queue

50 is dequeued from the queue

Empty queue

Empty Queue

Create the employee records dynamically and display it using stack operation --- imp

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Define an employee structure

struct Employee {

int id;

char name[50];

float salary;

};

// Stack structure to hold employee records

struct Stack {

struct Employee\* data;

int top;

int capacity;

};

// Function to create and initialize the stack

struct Stack\* createStack(int capacity) {

struct Stack\* stack = (struct Stack\*)malloc(sizeof(struct Stack));

stack->capacity = capacity;

stack->top = -1;

stack->data = (struct Employee\*)malloc(capacity \* sizeof(struct Employee));

return stack;

}

// Function to check if the stack is full

int isFull(struct Stack\* stack) {

return stack->top == stack->capacity - 1;

}

// Function to check if the stack is empty

int isEmpty(struct Stack\* stack) {

return stack->top == -1;

}

// Function to push an employee record onto the stack

void push(struct Stack\* stack, struct Employee emp) {

if (isFull(stack)) {

printf("Stack is full. Cannot push more records.\n");

return;

}

stack->data[++(stack->top)] = emp;

}

// Function to pop an employee record from the stack

struct Employee pop(struct Stack\* stack) {

if (isEmpty(stack)) {

printf("Stack is empty. Cannot pop.\n");

struct Employee empty = {0, "", 0.0};

return empty; // Return an empty employee record

}

return stack->data[(stack->top)--];

}

// Function to display all employee records in the stack

void displayStack(struct Stack\* stack) {

if (isEmpty(stack)) {

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

return;

}

printf("Employee Records (from top to bottom):\n");

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

printf("ID: %d, Name: %s, Salary: %.2f\n",

stack->data[i].id, stack->data[i].name, stack->data[i].salary);

}

}

// Function to free the memory allocated for the stack

void freeStack(struct Stack\* stack) {

free(stack->data);

free(stack);

}

int main() {

// Create a stack with capacity for 5 employee records

struct Stack\* stack = createStack(5);

// Dynamically add employee records

struct Employee e1 = {1, "Alice", 50000.00};

struct Employee e2 = {2, "Bob", 60000.00};

struct Employee e3 = {3, "Charlie", 55000.00};

struct Employee e4 = {4, "David", 65000.00};

struct Employee e5 = {5, "Eve", 70000.00};

// Push employee records onto the stack

push(stack, e1);

push(stack, e2);

push(stack, e3);

push(stack, e4);

push(stack, e5);

// Display all employee records in the stack

displayStack(stack);

// Pop one employee record and display the updated stack

printf("\nPopping an employee record...\n");

pop(stack);

displayStack(stack);

// Free the memory allocated for the stack

freeStack(stack);

return 0;

}

**C UNIT**

**Expression handling for validating the phone numbers**

A pointer to a array of characters might be a case when diff comes between empty string and not null

Sunny test case – positive(data should be received by other and match) and

Rainy test case – data1 or 1data… will not match