|  |  |
| --- | --- |
|  | // c program to implement operations on multiple queue  #include<stdio.h>  #include<conio.h> |
|  | # define max 20 |
|  |  |
|  | int insq (int queue[max], int qno, int rear[], int limit[], int \*data) { |
|  | if (rear[qno] == limit[qno]) |
|  | return(-1); |
|  | else { |
|  | rear[qno]++; |
|  | queue[ rear[qno] ] = \*data; |
|  | return(1); |
|  | } |
|  | } |
|  |  |
|  | int delq (int queue[max], int qno, int front[], int rear[], int \*data) { |
|  | if( front[qno] == rear[qno] ) |
|  | return(-1); |
|  | else { |
|  | front[qno]++; //... front[qno] = front[qno] + 1; |
|  | \*data = queue[ front[qno] ]; |
|  | return(1); |
|  | } |
|  | } |
|  |  |
|  | int getQueueNumber(int n) { |
|  | int qNo=0; |
|  | Inva: |
|  | printf("\n Enter a Logical Queue Number (1 to %d) : ", n); |
|  | scanf("%d", &qNo); |
|  | if (qNo<1 || qNo >n) { |
|  | printf(" Invalid Queue Number. Please try again.\n"); |
|  | goto Inva; |
|  | } |
|  | return qNo; |
|  | } |
|  |  |
|  | void main() { |
|  | int queue[max], data; |
|  | int bott[10], limit[10], f[10], r[10]; |
|  | int i, n, qno, size, option, reply; |
|  |  |
|  | printf("\n C Language program to implement the Multiple Queues \n"); |
|  | printf("\n How Many Queues ? : "); |
|  | scanf("%d", &n); |
|  | size = max / n; |
|  | bott[0] = -1; |
|  | for(i = 1; i < n; i++) |
|  | bott[i] = bott[i-1] + size; |
|  | for(i = 0; i < n; i++) |
|  | limit[i] = bott[i] + size; |
|  |  |
|  | for(i = 0; i < n; i++) |
|  | f[i] = r[i] = bott[i]; |
|  |  |
|  | do { |
|  | printf("\n\n C Language program to implement the Multiple Queues \n"); |
|  | printf("\n 1. Insert in a Queue"); |
|  | printf("\n 2. Delete from a Queue"); |
|  | printf("\n 3. Print from a Queue"); |
|  | printf("\n 3. Exit \n"); |
|  | printf("\n Select proper option ( 1 / 2 / 3 / 4) : "); |
|  | scanf("%d", &option); |
|  | switch(option) { |
|  | case 1 : //... Insert |
|  | qno = getQueueNumber(n); |
|  | printf("\n Enter Data : "); |
|  | scanf("%d", &data); |
|  | reply = insq(queue, qno-1, r, limit, &data); |
|  | if( reply == -1) |
|  | printf("\n Queue %d is Full \n", qno); |
|  | else |
|  | printf("\n %d is inserted in a Queue No. %d \n", data, qno); |
|  | break; |
|  | case 2 : //... Delete |
|  | qno = getQueueNumber(n); |
|  | reply = delq(queue, qno-1, f, r, &data); |
|  | if( reply == -1) |
|  | printf("\n Queue %d is Empty \n", qno); |
|  | else |
|  | printf("\n %d is deleted from Queue No. %d \n", data, qno); |
|  | break; |
|  | case 3: |
|  | qno = getQueueNumber(n); |
|  | printf("\n Elements of Queue %d are as : ", qno); |
|  | if (f[qno-1]==r[qno-1]) { |
|  | printf("\n Queue is empty"); |
|  | break; |
|  | } |
|  | for (i=f[qno-1]+1; i<=r[qno-1]; i++) |
|  | printf("%d\t", queue[i]); |
|  | printf("\n"); |
|  | break; |
|  | case 4 : |
|  | break; |
|  | default: |
|  | printf("\n Invalid input. Please try again."); |
|  | } |
|  | }while(option!=4); |
|  | } |