|  |  |
| --- | --- |
| Assignment No. : | 09 |
| Title: | C++ program for simulating job queue ( Linear Queue). |
| Subject: | Data Structures Laboratory |
| Class: | S.Y. (C.S.E.) |
| Roll No.: |  |
| Assessment (Marks): |  |
| Signature and Date of Assessment: |  |

Experiment No.: 9

Title: C++ program for simulatingjob queue.

Objectives: a) To understand the concept of queue.

1. To understand various operation of queue.
2. To understand use of various operations of queue using array.
3. To understand the concept and use of priority queue.

Problem Queues are frequentlyused in computer programming, anda typical exampleis the creation ofa

Statement: job queue by an operating system. If the operating system does not use priorities, then the jobs

are processed in the order they enter the system. Write C++ program for simulating job queue. Write functions to addjob and delete job from queue.

|  |  |
| --- | --- |
| Outcomes: | * Understanding the concept of queue. * Understanding the various operation of queue. * Understanding use of various operations of queue using array.  Understanding the use of priority queue. |
| Theory: | Queue is a linear structure which follows a particular order in which the operations are |

performed. The order is First In First Out (FIFO). A good example of queue is any queueof consumers for a resource where the consumer that came first is served first.

The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

Array implementation Of Queue

For implementing queue, we need to keep track of two indices, front and rear. We enqueue an item at the rear and dequeue an item from front.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 7 | 2 |  |  | 1 |  |

* 14 5



Figure I : Representation of queue Operations on Queue:

Mainly the following four basic operations are performed on queue:

* enqueue(): Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition.
* dequeue(): Removes an item from the queue. The items are poplkd in the same order in which they are pushed. Ifthe queue is empty , then it is said to be an Underflow condition.
* isfull(): checks if queue is full.
* isempty(): checks if queue is empty.



isfull() Operation: isempty() Operation :

|  |  |  |
| --- | --- | --- |
| bool isfull()  MAXSIZE - 1) return true;  Else return false; | | bool isempty()    return true;  Else return false; |
| nqueue perat on(l.e. Inse int enqueue(int data)  if(isfull()) return 0; rear: rear+ l ; queue[rearl = data; return l ; | on): | equeue pera on( .e e etlon): int dequeue()  if(isempty()) return 0; int data = queue[frontl; front = front + l ; return data; |

Priority Queue is more specialized data structure than Queue. Like ordinary queue, priority queue has same method but with a major difference. In Priority queue items are ordered by key value so that item with the lowest value of key is at front and item with the highest value of key is at rear or vice versa.

A priority queue is used to handle the job queue of an operating system. So for the implementation of jobs of operating system priority queue is used. As the jobs are assigned in queue depending upon their priority.

Priority Queue is Collection of entities or elements in which:

 Addition of element is done on basis of priority.  Removal of element is done at FRONT.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Insert  New Element  (with priority)   |  | | --- | | 37 |  |  | | --- | | 21 | | 24 | | 61 | | 82 | | 97 |   Rear  Front | Rea r | |  | | --- | | 21 | | 24 | | 37 | | 61 | | 82 | | 97 | |

Figure 2: Insertion of element in priority queue depending upon priority.

Applications of Queue:

Queue is used when things don't have to be processed immediately, but have to be processed in First InFirst Out order like Breadth First Search. This property of Queue makes it also useful in following kind of scenarios.

1. When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling.
2. When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include 10 Buffers, pipes, file 10, etc.

Enqueue Operation

As queue maintains two data pointers, front and rear, its operatiorus are comparatively more difficult to implement than Queue.

The following steps should be taken to enqueue (insert) data into a queue:

Step 1 — Check if queue is full.

Step 2 — If queue is full, produce overflow error and exit.

Step 3 — If queue is not full, increment rear pointer to point next empty space.

Step 4 — Add data element to the queue location, where rear is pointing. Step 5 — return success.

Dequeue Operation

Accessing data from queue is a process of two tasks — access the data where front is pointing and remove the data after access.

The following steps are taken to perform dequeue operation:

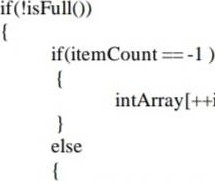
Step 1 — Check if queue is empty.

Step 2 — If queue is empty, produce underflow error and exit.

Step 3 — If queue is not empty, access data where front is pointing. Step 3 — Increment front pointer to point next available data element Step 5 — return success.

Insertion operation of priority queue: void insert(int data) int i = 0;

// if queue is empty, insert the data

intArray[++itemCountl = data;

// start from the right end of the queue itemCount++•, for(i = itemCount - I ; i O; )

// if data is larger, shift existing item to right end if(data > intArray[il) intArray[i+l I = intArray[il; else break; intArray[i+l I = data; // insert the data

Enter how many jobs: 5

Enter Job (job priority I -9) : 7 Enter Job (job priority 1-9) : 3 Enter Job (job priority -9) : 4 Enter Job (job priority I -9) : I

Enter Job (job priority I -9) : 2

Dequeuejobs:

Job: 1 Job: 2 Job: 3

Job: 4

Job: 7

|  |  |
| --- | --- |
| Conclusion: | Thus, we implemented simulation ofjobs using priority queue. |
| Questions: | Explain queue ADT.   1. Explain linear and priority queue ADT. 2. Explain array representation of queue. |

4. What are the applications of linear and priority queue?

Code:-

#include <iostream>

using namespace std;

#define n 5

class Queue

{

    int Q[n], front = -1, rear = -1;

public:

    void Enqueue()

    {

        int c;

        if (rear == n - 1)

        {

            cout << "QUEUE OVERFLOW " << endl;

        }

        else if (front == -1 && rear == -1)

        {

            front++;

            rear++;

            cout << "ENTER AN ELEMENT: " << endl;

            cin >> c;

            Q[rear] = c;

        }

        else

        {

            rear++;

            cout << "ENTER AN ELEMENT" << endl;

            cin >> c;

            Q[rear] = c;

        }

    }

public:

    void Dequeue()

    {

        if (front == -1 && rear == -1)

        {

            cout << "QUEUE UNDERFLOW, NO ELEMENTS TO DELETE " << endl;

        }

        else if (front == rear && front > -1)

        {

            cout << "THE ELEMENT DELETED IS: " << Q[front] << endl;

            front = front - 1;

            rear = rear - 1;

        }

        else

        {

            cout << "THE ELEMENT DELETED IS: " << Q[front] << endl;

            front++;

        }

    }

public:

    void Display()

    {

        if (front == -1)

        {

            cout << "NO ELEMENTS TO DISPLAY" << endl;

        }

        else

        {

            cout << "THE ELEMENTS ARE: " << endl;

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

            {

                cout << Q[i] << endl;

            }

        }

    }

};

int main()

{

    Queue Q;

    int d;

    cout << "1.ENQUEUE/ENTER." << endl;

    cout << "2.DEQUEUE/DELETE." << endl;

    cout << "3.DISPLAY." << endl;

    cout << "4.EXIT." << endl;

    do

    {

        cout << "ENTER YOUR CHOICE: " << endl;

        cin >> d;

        switch (d)

        {

        case 1:

            Q.Enqueue();

            break;

        case 2:

            Q.Dequeue();

            break;

        case 3:

            Q.Display();

            break;

        case 4:

            cout << "EXIT" << endl;

            break;

        default:

            cout << "INVALID INPUT." << endl;

        }

    } while (d != 4);

}

Output:-

1.ENQUEUE/ENTER.

2.DEQUEUE/DELETE.

3.DISPLAY.

4.EXIT.

ENTER YOUR CHOICE:

1

ENTER AN ELEMENT:

1

ENTER YOUR CHOICE:

1

ENTER AN ELEMENT

2

ENTER YOUR CHOICE:

1

ENTER AN ELEMENT

3

ENTER YOUR CHOICE:

3

THE ELEMENTS ARE:

1

2

3

ENTER YOUR CHOICE: