**LABSHEET 4**

**PROBLEM 1**

Implementing printer scenario using simple queue.

Displaying current status.

New arrivals

Complete jobs.

**ALGORITHM:**

1. Creating a class printer\_queue with member function, enque, deque and output.
2. Create an array
3. Initialize front to -1 and rear to -1
4. Enqueue(element)
5. If rear is size -1 , queue is full , overflow error is shown.

Else

If front and rear are -1 , front is incremented and rear is incremented and element is added to the array.

Else rear is incremented and element is added to array.

1. Dequeue()
2. if front and rear are -1 , underflow message is shown.
3. ElseIf front is equal to rear and is not equal to -1, element in front is removed, and both front and rear are made -1

Else element is removed and front is incremented.

1. Output()
2. counter Is initialized to front and is printed till counter becomes equal to rear.
3. In the main, the options are generated for the above functions and executed until the user wishes to exits.

**CODE:**

#include <iostream>

using namespace std;

class printer\_queue

{

int queue[15], front, rear, size;

public:

printer\_queue():front(-1), rear(-1), size(15){}

void enqueue()

{

int comp\_no;

cout<<"Enter system number\n”;

cin>>comp\_no;

if(rear == size-1)

cout<<"Queue is full, OVERFLOW!\n”;

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

{

front++;

queue[++rear]=comp\_no;

}

else

{

queue[++rear] = comp\_no;

}

}

void dequeue()

{

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

cout<<"Empty queue, UNDERFLOW!\n”;

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

{

queue[front] = 0;

front = rear = -1;

}

else

queue[front++] = 0;

}

void output()

{

int i = front;

while(i <= rear)

{

cout<<queue[i]<<"\t"<<”\n”;

i++;

}

cout<<"rear is "<<rear<<"\t front is "<<front<<”\n”;

}

};

int main()

{

printer\_queue p;

int op;

cout<<"Select an option"<<”\n”;

cout<<"1. Using printer \n2. Job done \n3. Output\n4. Exit"<<\n”;

cin>>op;

while(op != 4)

{

switch(op)

{

case 1: p.enqueue();

break;

case 2: p.dequeue();

break;

case 3 : p.output();

break;

default: break;

}

cout<<"Select an option"<<”\n”;

cout<<"1. Using printer \n2. Job done \n3. Output\n4. Exit"<<”\n”;

cin>>op;

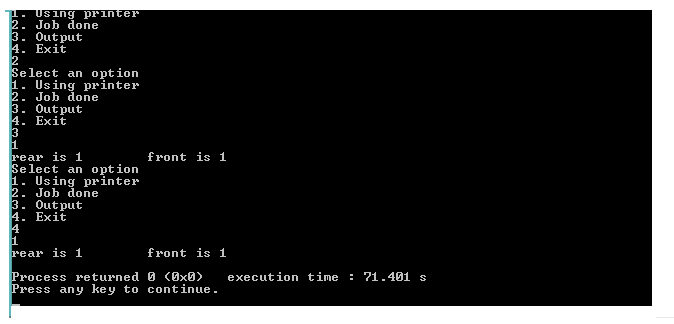
}

p.output();

return 0;

}

**TESTING INPUT AND OUTPUT:**



**PROBLEM 2 (USING SIMPLE QUEUE):**

Implementing the code for a ticket counter queue, with people lined up for tickets.

**ALGORITHM**

1. Class of ticket\_queue contains enque and deque as member functions and queue of strings.
2. A constructor initializes values for the data members.
3. Create an array
4. Initialize front to -1 and rear to -1
5. Enqueue(element)
6. If rear is size -1 , queue is full , overflow error is shown.
7. ElseIf front and rear are -1 , front is incremented and rear is incremented and element is added to the array.
8. Else rear is incremented and element is added to array.
9. Dequeue()
10. if front and rear are -1 , underflow message is shown.
11. ElseIf front is equal to rear and is not equal to -1, element in front is removed, and both front and rear are made -1
12. Else element is removed and front is incremented.
13. Output()
14. counter Is initialized to front and is printed till counter becomes equal to rear.
15. In the main, the options are generated for the above functions and executed until the user wishes to exits.

**CODE:**

#include<iostream>

using namespace std;

class ticket\_queue

{

string queue[10];

int front, rear, size;

public:

ticket\_queue():front(-1), rear(-1), size(10){}

void enqueue()

{

string name;

cout<<"Enter the name"<<”\n”;

cin>>name;

if(rear == size-1)

cout<<"Queue is full, OVERFLOW!"<<”\n”;

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

front++;

queue[++rear]=name;

}

else {

queue[++rear] = name;

}

}

void dequeue() {

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

cout<<"Empty queue, UNDERFLOW!"<<”\n”;

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

{

queue[front] = 'NULL';

front = rear = -1;

}

else

queue[front++] = 'NULL';

}

void output()

{

int i = front;

while(i <= rear)

{

cout<<queue[i]<<"\t"<<”\n”;

i++;

}

cout<<"rear is "<<rear<<"\t front is "<<front<<”\n”;

}

};

int main()

{

ticket\_queue t;

int op;

cout<<"Select an option"<<”\n”;

cout<<"1. Need a ticket \n2. Got a ticket \n3. Present queue status\n4. Exit"<<”\n”;

cin>>op;

while(op != 4)

{

switch(op)

{

case 1: t.enqueue();

break;

case 2: t.dequeue();

break;

case 3 : t.output();

break;

default: break;

}

cout<<"Select an option"<<”\n”;

cout<<"1. Need a ticket \n2. Got a ticket \n3. Present queue status\n4. Exit"<<”\n”;

cin>>op;

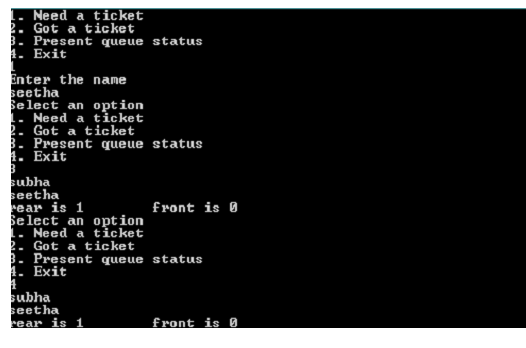
}

t.output();

return 0;

}

**TESTING INPUT AND OUTPUT:**



**PROBLEM 3**

Implementing circular queue using array.

**ALGORITHM:**

1. Class , circular has member functions insertion, deletion, output, member variables, front , rear, size
2. constructor initializes front , rear to -1 and size too.
3. Insertion()
4. if the queue is full I.e front = (rear+1) % size, overflow error
5. else if front and rear are -1 then increment front and rear, store the element in the queue.
6. Else rear becomes (rear+1)%size, and element Is stored in the array .
7. Deletion()
8. if the queue is empty I.e front and rear point to -1, show underflow message.
9. Elseif front and rear are equal remove the element in the front and make rear and front -1
10. else remove the element in the front and front becomes (front+1)% size
11. output()
12. counter is initialized to front While the counter is not equal to rear, elements are displayed
13. counter becomes ((counter +1)%size)
14. In the main we have a menu driven program which only exists if the user wishes to .

**CODE**

#include <iostream>

using namespace std;

class circular

{

char c\_queue[10];

int front, rear, size;

public:

circular():front(-1), rear(-1), size(10){}

void insertion(char ch)

{

if(front == ((rear+1)%size))

cout<<"OVERFLOW!"<<”\n”;

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

{

front++;

c\_queue[++rear] = ch;

}

else

{

rear = (rear+1)%size;

c\_queue[rear] = ch;

}

}

void deletion ()

{

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

cout<<"UNDERFLOW!"<<endl;

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

{

c\_queue[front] == '#';

front = rear = -1;

}

else

{

c\_queue[front] == '#';

front = (front+1)%size;

}

}

void output()

{

int i = front;

while(i != rear)

{

cout<<c\_queue[i]<<"\t"<<”\n”;

i = ((i+1)%size);

}

cout<<"rear is "<<rear<<"\t front is "<<front<<”\n”;

};

int main()

{

circular c;

int op;

char ch;

cout<<"Select an option"<<”\n”;

cout<<"1. Enter an element \n2. Delete an element \n3. Present queue status\n4. Exit"<<

“\n”;

cin>>op;

while(op != 4)

{

switch(op)

{

case 1:

{

cout<<"Enter"<<endl;cin>>ch; c.insertion(ch);

}

break;

case 2: c.deletion();

break;

case 3 : c.output();

break;

default: break;

}

cout<<"Select an option"<<”\n”;

cout<<"1. Enter an element \n2. Delete an element \n3. Present queue status\n4. Exit"<<”\n”;

cin>>op;

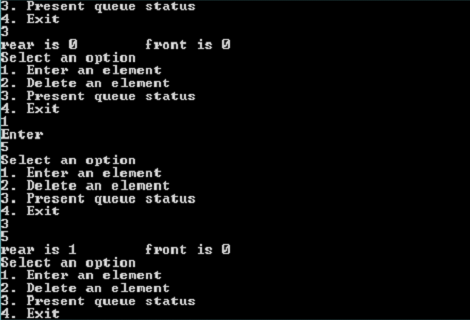
}

c.output();

return 0;

}

**TESTING INPUT AND OUTPUT:**



**PROBLEM 4**

Implement “Round Robin” scheduling algorithm using circular queue.

Round robin is the scheduling algorithm used by the CPU during execution of the processes. Round robin is designed specifically for time sharing systems. Processes along with its total time for execution are kept in a queue. Each process will be taken from the queue and is executed for a fixed amount of time, called “quantum. Process execution will be done cyclically if it is not complete in one “quantum.” Meanwhile new processes are added to the tail of the queue.

Example: Following are the set of jobs and execution time. Let quantum= 5 seconds

|  |  |
| --- | --- |
| Process | Execution Time |
| P1 (front) | 10 seconds |
| P2 | 5 Seconds |
| P3 | 15 second |
| P4(rear) | 10 seconds |

P1 executes for 5 seconds ie one quantum; then P2, P3…..Whichever process completes the execution within 5 second will be pop out from queue. Others will remain and wait for their next turn. ]

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P4 | P1 | P3 | P4 | P3 |

*Execution sequence*

ALGORITHM

1) Create a structure for storing process name and execution time.

2) Use the structure array as the circular queue.

3) Get the value of quantum from user.

4) Get a few processes and the execution time from user and enqueue them to the queue.

Enqueue:

i) IF FRONT = (REAR+1)%SIZE AND ENQUE IS CALLED RETURN "OVERFLOW".

ii) ELSE IF REAR AND FRONT ARE -1 INCREMENT FRONT AND REAR BY ONE AND ENTER INTO QUEUE[REAR].

iii) ELSE INCREMENT REAR=(REAR+1)%SIZE AND ENTER INTO QUEUE[REAR].

5) Start the scheduling

6) If front == -1 ask the user if he wants to add more processes. If yes add the process else break.

7) If front != -1 process the queue

Process:

i) DECREMENT THE VALUE OF QUANTUM FROM EACH PROCESS

ii) CHECK WHETHER ANY OF THE PROCESS IS COMPLETED OR NOT

iii) IF YES, POP PROCESS AND REARRANGE THE QUEUE AND DECREMENT REAR BY 1

8) Ask the user if he wants to add another process. If yes add the process.

CODE

#include<iostream>

#include<stdlib.h>

#include<cstring>

using namespace std;

int size = 5, front = -1,rear = -1,quantum;

struct process

{

char name[10];

int time;

}queue[5];

void Enqueue(process ele)

{

if( front==(rear+1)%size )

{

cout<<"Overflow\n";

}

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

{

front++;

rear++;

strcpy(queue[rear].name,ele.name);

queue[rear].time=ele.time;

}

else

{

rear=(rear+1)%size;

strcpy(queue[rear].name,ele.name);

queue[rear].time=ele.time;

}

}

void Display()

{

cout<<"\nProcess left:\n";

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

{

cout<<queue[i].name<<" ";

}

}

void Process()

{

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

{

queue[i].time -= quantum;

}

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

{

if(queue[i].time <= 0)

{

cout<<"\nProcess "<<queue[i].name<<" complete!";

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

{

strcpy(queue[j].name,queue[j+1].name);

queue[j].time=queue[j+1].time;

}

rear--;

Display();

}

}

if(rear == -1)

front = -1;

}

int main()

{

cout<<"\nRound Robin Scheduling";

cout<<"\nEnter value of quantum:";

cin>>quantum;

process pro;

char ch='y';

while(ch == 'y')

{

cout<<"\nEnter process name: ";

cin>>pro.name;

cout<<"\nEnter execution time: ";

cin>>pro.time;

Enqueue(pro);

cout<<"\nDo you want to add another process?(y/n): ";

cin>>ch;

}

cout<<"\nSheduling...\n";

Display();

while(1)

{

if(front == -1)

{

cout<<"\nProcessing Complete!\n";

cout<<"\nDo you want to add another process?(y/n): ";

cin>>ch;

if(ch == 'y')

{

goto A;

}

else

break;

}

Process();

cout<<"\nDo you want to add another process?(y/n): ";

cin>>ch;

if(ch == 'y')

{

A: cout<<"\nEnter process name: ";

cin>>pro.name;

cout<<"\nEnter execution time: ";

cin>>pro.time;

Enqueue(pro);

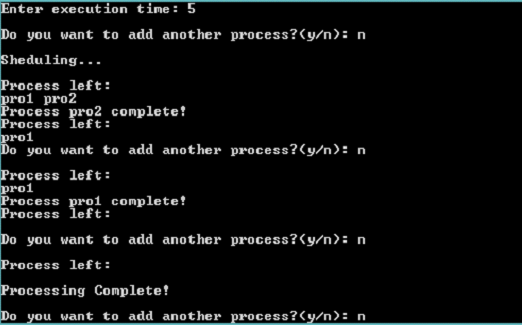
}

Display();

}

}

**TESTING INPUT AND OUTPUT:**



**PROBLEM 5**

Implementing deque using array.

**ALGORITHM**

1. class double\_ended has data members and array, front , rear and size , member functions enqueue\_front()
2. enqueue\_rear(), dequeue\_front(), dequeue\_rear(), and output().
3. constructor initializes front , rear to -1 and size too.
4. Enqueue\_rear(element)
5. If rear is size -1 , queue is full , overflow error is shown.
6. ElseIf front and rear are -1 , front is incremented and rear is incremented and element is added to the array.
7. Else rear is incremented and element is added to array.
8. Dequeue\_front()
9. if front and rear are -1 , underflow message is shown.
10. ElseIf front is equal to rear and is not equal to -1, element in front is removed, and both front and rear are made -1
11. Else element is removed and front is incremented.
12. enqueue\_front(element)
13. if front and rear is -1 increment front and rear, insert the element in the array.
14. ElseIf front is zero, say cannot insert.
15. Else decrement front and store the element in the array
16. dequeue\_rear()
17. if rear is -1 then show error message cannot empty.
18. Elseif rear is size-1 or rear is equal to front then remove the element at rear and change rear to -1
19. else remove element at rear and decrement rear
20. Output()
21. counter Is initialized to front and is printed till counter becomes equal to rear.
22. In the main, the options are generated for the above functions and executed until the user wishes to exits.

**CODE**

#include <iostream>

using namespace std;

class double\_ended

{

int deque[10], front, rear, size;

public:

double\_ended():front(-1), rear(-1), size(10){}

void enque\_front(int a)

{

if(front == 0)

cout<<"Cannot insert!"<<”\n”;

else if(front == -1)

{

deque[++front] = a;

rear++;

}

else

{

deque[--front] = a;

}

}

void enque\_rear(int a)

{

if(rear == size-1)

cout<<"Cannot insert!\n”;

else if(rear == -1)

{

deque[++rear] = a;

front++;

}

else

{

deque[++rear] = a;

}

}

void deque\_front()

{

if(front == -1)

{

cout<<"Cannot empty the queue, already empty!"<<”\n”;

}

else

{

if((front == size-1)||(front == rear))

{

deque[front] = -1;

// to indicate that there is no element present

front = -1;

rear = -1;

}

else

{

deque[front++] = -1;

}

}

}

void deque\_rear()

{

if(rear== -1)

{

cout<<"Cannot empty the queue, already empty!\n”;

}

else

{

if ((front == size-1)||(front == rear))

{

deque[rear] = -1;

front = -1;

rear = -1;

}

else

{

deque[rear--] = -1;

}

}

}

void output()

{

int i = front;

while(i <= rear)

{

cout<<deque[i]<<"\t"<<”\n”;

i++;

}

cout<<"rear is "<<rear<<"\t front is "<<front<<”\n”;

}

};

int main()

{

double\_ended de;

int op, a;

cout<<"Select an option"<<”\n”;

cout<<"1. Enqueue at front \n2. Dequeue at front \n3. Enqueue at rear \n4. Dequeue at rear \n5. Output status \n6. Exit"<<”\n”;

cin>>op;

while(op != 6)

{

switch(op)

{

case 1:

{

cout<<"Enter"<<”\n”;

cin>>a;

de.enque\_front(a);

}

break;

case 2: de.deque\_front();

break;

case 3 :

{

cout<<"Enter\n”;

cin>>a;

de.enque\_rear(a);

}

break;

case 4: de.deque\_rear();

break;

case 5: de.output();

break;

default:

break;

}

cout<<"Select an option"<<”\n”;

cout<<"1. Enqueue at front \n2. Dequeue at front \n3. Enqueue at rear \n4. Dequeue at rear \n5. Output status \n6. Exit"<<”\n”;

cin>>op;

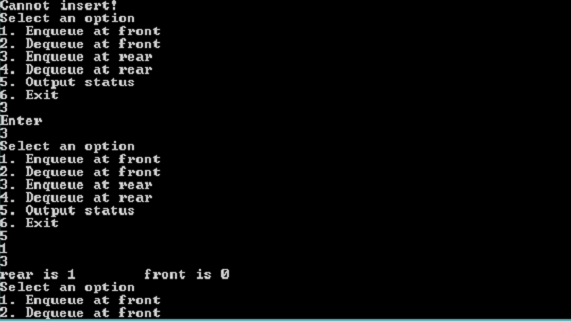
}

de.output();

return 0;

}

**TESTING INPUT AND OUTPUT:**



**PROBLEM 6**

Checking whether a word is a palindrome or not using deque .

**ALGORITHM:**

1. Get the string in a deque.
2. Take a flag=1.
3. Check whether the element in front is same as the element at the rear.
4. If not, break and assign flag=0.
5. Else, pop both from front and rear and increment f by 1 and decrement r by 1.
6. If flag=1, the string is palindrome else it is not a palindrome.

**CODE:**

#include <iostream>

#include <string>

using namespace std;

class palindrome

{

int front, rear, length, flag;

string deque;

public:

palindrome ():front(-1), rear(-1), flag(0){}

void input()

{

cout<<"Enter a word\n”;

cin>>deque;

length = deque.size();

front = 0;

rear = length-1;

}

char deque\_front()

{

if(front == -1)

{

//cout<<"Cannot empty the queue, already empty!"<<<”\n”;

}

else

{

if((front == length-1)||(front == rear))

{

char ch = deque[front];

front = -1;

rear = -1;

return ch;

}

else

{

return deque[front++];

}

}

}

char deque\_rear()

{

if(rear== -1)

{

//cout<<"Cannot empty the queue, already empty!"<<”\n”;

}

else {

if ((front == length-1)||(front == rear))

{

char ch = deque[rear];

front = -1;

rear = -1;

return ch;

}

else

{

return deque[rear--];

}

}

}

void check()

{

while(rear != -1)

{

if(deque\_front() != deque\_rear())

{

flag = 1;

}

if(front == rear)

{

rear = -1;

front = -1;

}

}

}

void output()

{

if(flag == 0)

cout<<"It is a Palindrome"<<”\n”;

else

cout<<"It is not a Palindrome"<<”\n”;

}

};

int main()

{

palindrome p;

p.input();

p.check();

p.output();

return 0;

}

**TESTING INPUT AND OUTPUT:**

