DSA Lab

Week 6 Assignment Submission

Swamiraju Satya Praveen Varma

200905044

Batch B1

10

1) Implement an ascending priority queue.

Note: An ascending priority queue is a collection of items into which items can be inserted arbitrarily and from which only the smallest item can be removed. If apq is an ascending priority queue, the operation pqinsert(apq, x) inserts element x into apq and pqmindelete(apq) removes the minimum element from apq and returns its value.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
typedef struct
{
  int items[MAX];
  int front, rear;
} PQUEUE;
void pqinsert (PQUEUE * pq, int);
void pqmindelete (PQUEUE * pq);
void create (PQUEUE * pq);
void ins (PQUEUE * pq, int);
void pqdisplay (PQUEUE * pq);
void
create (PQUEUE * pq)
{
```

```
  pq->front = -1;
  pq->rear = -1;
}

void
pqinsert (PQUEUE * pq, int x)
{
  if (pq->rear >= MAX - 1)
    {
      printf ("Queue Overflow\n");
      return;
    }
  if ((pq->front == -1) && (pq->rear == -1))
    {
      pq->front++;
      pq->rear++;
      pq->items[pq->rear] = x;
      return;
    }
  else
    ins (pq, x);
  pq->rear++;
}

void
ins (PQUEUE * pq, int x)
{
  int i, j;
  for (i = 0; i <= pq->rear; i++)
    {
```

```
      if (x <= pq->items[i])
          {
            for (j = pq->rear + 1; j > i; j--)
              {
                pq->items[j] = pq->items[j - 1];
              }
            pq->items[i] = x;
            return;
          }
    }
  pq->items[i] = x;
}

void
pqmindelete (PQUEUE * pq)
{
  int i;
  if ((pq->front == -1) && (pq->rear == -1))
    {
      printf ("Queue Empty\n");
      return;
    }
  for (i = 0; i < pq->rear; i++)
    {
      pq->items[i] = pq->items[i + 1];
    }
  pq->items[i] = -99;
  pq->rear--;
  if (pq->rear == -1)
    pq->front = -1;
```

```
  return;
}

void
pqdisplay (PQUEUE * pq)
{
  if ((pq->front == -1) && (pq->rear == -1))
    {
      printf ("\nQueue is empty");
      return;
    }
  for (; pq->front <= pq->rear; pq->front++)
    {
      printf (" %d ", pq->items[pq->front]);
    }
  pq->front = 0;
}

int
main ()
{
  PQUEUE *pq;
  pq = malloc (sizeof (PQUEUE));
  int n, ch;
  printf ("1)Insert an element into queue\n");
  printf ("2)Delete an element from queue\n");
  printf ("3)Display queue elements\n");
  printf ("4)Exit\n");
  create (pq);
  while (1)
```

```
    {
      printf ("Enter your choice : ");
      scanf ("%d", &ch);
      switch (ch)
          {
          case 1:
            printf ("Enter value to be inserted : ");
            scanf ("%d", &n);
            pqinsert (pq, n);
            break;
          case 2:
            pqmindelete (pq);
            break;
          case 3:
            pqdisplay (pq);
            printf ("\n");
            break;
          case 4:
            exit (0);
          default:
            printf ("\nChoice is incorrect, Enter a correct choice\n");
          }
    }
}
```

Sample input/output:

```
Student@prg19: ~/200905044/DSAL/Week6
File  Edit  View  Search  Terminal  Help
Student@prg19:~/200905044/DSAL/Week6$ gcc l6q1.c -o q1
Student@prg19:~/200905044/DSAL/Week6$ ./q1
1)Insert an element into queue
2)Delete an element from queue
3)Display queue elements
4)Exit
Enter your choice : 3

Queue is empty
Enter your choice : 1
Enter value to be inserted : 10
Enter your choice : 1
Enter value to be inserted : 5
Enter your choice : 1
Enter value to be inserted : 20
Enter your choice : 3
 5  10  20
Enter your choice : 2
Enter your choice : 3
 10  20
Enter your choice : 1
Enter value to be inserted : 30
Enter your choice : 3
 10  20  30
Enter your choice : 5

Choice is incorrect, Enter a correct choice
Enter your choice : 4
Student@prg19:~/200905044/DSAL/Week6$
```

2) Implement a queue of strings using an output restricted dequeue (no deleteRight).

Note: An output-restricted deque is one where insertion can be made at both ends, but deletion can be made from one end only, where as An input-restricted deque is one where deletion can be made from both ends, but insertion can be made at one end only.

Code:

```
#include <stdio.h>

#include <stdlib.h>

#define MAX_SIZE 5

#define MAX_STR 10

typedef struct

{

  char arr[MAX_SIZE][MAX_STR];
```

```
  int front, rear;
} outresq;
void
init (outresq * s)
{
  s->front = s->rear = -1;
}

int
isEmpty (outresq * s)
{
  if (s->rear == -1)
    {
      return 1;
    }
  return 0;
}

int
isFull (outresq * s)
{
  if ((s->rear + 1) % MAX_SIZE == s->front)
    {
      return 1;
    }
  return 0;
}

void
insertright (outresq * s, char x[])
```

```
{
  int i;
  if (isEmpty (s))
    {
      s->rear = s->front = 0;
      for (i = 0; x[i] != '\0'; i++)
          {
            s->arr[s->rear][i] = x[i];
          }
      s->arr[s->rear][i] = '\0';
    }
  else
    {
      s->rear = (s->rear + 1) % MAX_SIZE;
      for (i = 0; x[i] != '\0'; i++)
          {
            s->arr[s->rear][i] = x[i];
          }
      s->arr[s->rear][i] = '\0';
    }
}

void
insertleft (outresq * s, char x[])
{
  int i;
  if (isEmpty (s))
    {
      s->rear = s->front = 0;
      for (i = 0; x[i] != '\0'; i++)
```

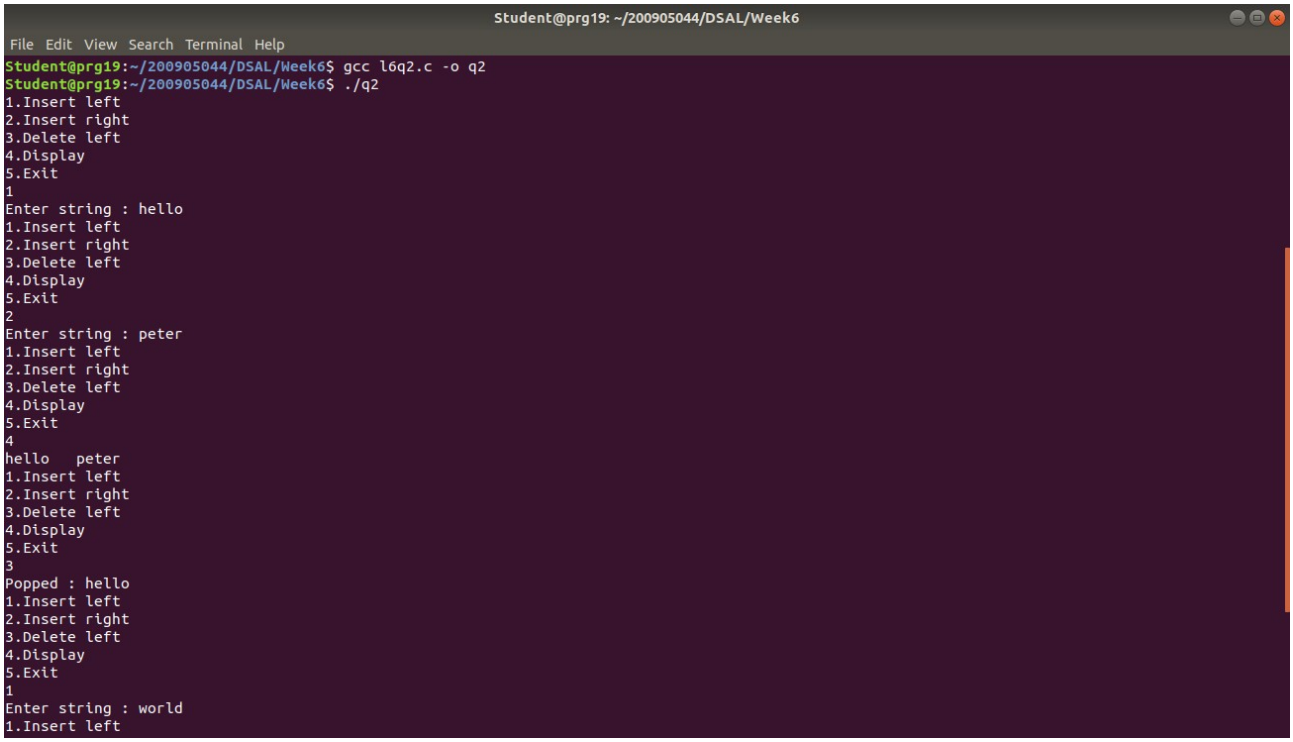
```
        {
          s->arr[s->front][i] = x[i];
        }
    s->arr[s->front][i] = '\0';
  }
  else
  {
    s->front = (s->front - 1 + MAX_SIZE) % MAX_SIZE;;
    for (i = 0; x[i] != '\0'; i++)
        {
          s->arr[s->front][i] = x[i];
        }
    s->arr[s->front][i] = '\0';
  }
}

char *
deleteleft (outresq * s)
{
  char *str;
  str = s->arr[s->front];
  if (s->rear == s->front)
  {
    init (s);
  }
  else
  {
    s->front = (s->front + 1) % MAX_SIZE;
  }
  return str;
```

```
}

void
displaydq (outresq * s)
{
  if (isEmpty (s))
    {
      printf ("Queue is empty\n");
      return;
    }
  int temp;
  for (temp = (s->front) % MAX_SIZE; temp != (s->rear);
       temp = (temp + 1) % MAX_SIZE)
    {
      printf ("%s\t", s->arr[temp]);
    }
  printf ("%s\n", s->arr[s->rear]);
}

int
main ()
{
  outresq s;
  init (&s);
  int ch;
  char str[MAX_STR];
  do
    {
      printf
          ("1.Insert left\n2.Insert right\n3.Delete left\n4.Display\n5.Exit\n");
```

```
scanf ("%d", &ch);
if (ch == 1)
    {
      if (isFull (&s))
        {
          printf ("Overflow\n");
        }
      else
        {
          printf ("Enter string : ");
          scanf ("%s", str);
          insertleft (&s, str);
        }
    }
else if (ch == 2)
    {
      if (isFull (&s))
        {
          printf ("Overflow\n");
        }
      else
        {
          printf ("Enter string : ");
          scanf (" %s", str);
          insertright (&s, str);
        }
    }
else if (ch == 3)
    {
      if (!isEmpty (&s))
```

```
            {
              char *pop = deleteleft (&s);
              printf ("Popped : %s\n", pop);
            }
          else
            {
              printf ("Underflow\n");
            }
        }
    else if (ch == 4)
        {
          displaydq (&s);
        }
  }
 while (ch != 5);
}
```

Sample input/output:



```
Student@prg19: ~/200905044/DSAL/Week6
File  Edit  View  Search  Terminal  Help
5.Exit
2
Enter string : peter
1.Insert left
2.Insert right
3.Delete left
4.Display
5.Exit
4
hello   peter
1.Insert left
2.Insert right
3.Delete left
4.Display
5.Exit
3
Popped : hello
1.Insert left
2.Insert right
3.Delete left
4.Display
5.Exit
1
Enter string : world
1.Insert left
2.Insert right
3.Delete left
4.Display
5.Exit
4
world   peter
1.Insert left
2.Insert right
3.Delete left
4.Display
5.Exit
5
Student@prg19:~/200905044/DSAL/Week6$
```

3) Write a program to check whether given string is a palindrome using a dequeue.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 30
typedef struct DQUEUE
{
  char item[MAX];
  int rear, front;
} DQUEUE;
void
init (DQUEUE * dq)
{
  dq->rear = -1;
  dq->front = -1;
```

```
}

int
isEmpty (DQUEUE * dq)
{
  if (dq->rear == -1)
    return (1);
  return (0);
}

int
full (DQUEUE * dq)
{
  if ((dq->rear + 1) % MAX == dq->front)
    return (1);
  return (0);
}

void
insertRight (DQUEUE * dq, char x)
{
  if (isEmpty (dq))
    {
      dq->rear = 0;
      dq->front = 0;
      dq->item[0] = x;
    }
  else
    {
      dq->rear = (dq->rear + 1) % MAX;
```

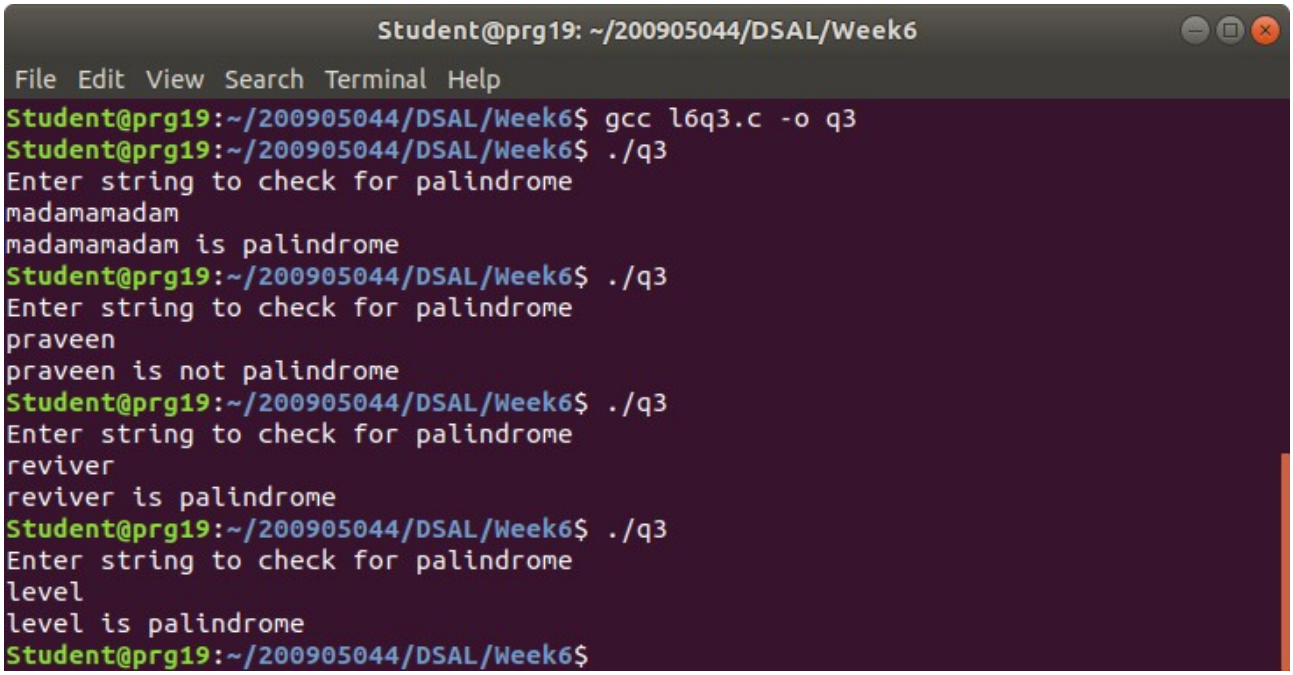
```
      dq->item[dq->rear] = x;
    }
}

void
insertLeft (DQUEUE * dq, char x)
{
  if (isEmpty (dq))
    {
      dq->rear = 0;
      dq->front = 0;
      dq->item[0] = x;
    }
  else
    {
      dq->front = (dq->front - 1 + MAX) % MAX;
      dq->item[dq->front] = x;
    }
}

char
deleteFront (DQUEUE * dq)
{
  char x;
  x = dq->item[dq->front];
  if (dq->rear == dq->front)
/*delete the last element */
    init (dq);
  else
    dq->front = (dq->front + 1) % MAX;
```

```
  return (x);
}

char
deleteRight (DQUEUE * dq)
{
  char x;
  x = dq->item[dq->rear];
  if (dq->rear == dq->front)
    init (dq);
  else
    dq->rear = (dq->rear - 1 + MAX) % MAX;
  return (x);
}

void
print (DQUEUE * dq)
{
  if (isEmpty (dq))
    {
      printf ("\nQueue is empty!!");
      exit (0);
    }
  int i;
  i = dq->front;
  while (i != dq->rear)
    {
      printf ("\n%c", dq->item[i]);
      i = (i + 1) % MAX;
    }
```

```
  printf ("\n%c\n", dq->item[dq->rear]);
}

int
main ()
{
  int i, x, n;
  int op = 0;
  char c[20];
  DQUEUE q;
  init (&q);
  printf ("Enter string to check for palindrome\n");
  scanf ("%s", c);
  n = strlen (c);
  for (i = 0; i < n; i++)
    {
      insertLeft (&q, c[i]);
    }
  for (i = 0; i < n / 2; i++)
    {
      if (deleteFront (&q) != deleteRight (&q))
          {
            op = 1;
            break;
          }
    }
  if (op == 0)
    printf ("%s is palindrome\n", c);
  else
    printf ("%s is not palindrome\n", c);
```

```
  return 0;
}
```

Sample input/output:



THANK YOU!