21/9/20

store
67

# DS Assignment - 1

Q. A university wants to automate their admission process. Students are admitted based on the marks scored in a qualifying exam.
A student is identified by student id, age and marks in qualifying exam. Data are valid if:
→ Age is greater than 20
→ Marks is between 0 and 100 (both inclusive)
A student qualifies for admission, if:
→ Age and marks are valid and
→ Marks is 65 or more.
WAP to represent the students seeking admission in the university.

Sol:-

```
#include <stdio.h>
struct student
{
   int id, age;
   float marks;
} stu[100];

void eligibility (i)
{
  if (stu[i].age <= 20 || stu[i].marks < 0 || stu[i].marks > 100)
  {
    printf ("\t\t Data Invalid \n");
  }
  else if (stu[i].marks >= 65)
  {
    printf ("\t\t eligible \n");
  }
  else
  {
```

```
        printf("\t\t not eligible \n");
    }
}

int main()
{
    int i, n;
    printf("\t\t\t UNIVERSITY ADMISSION");
    printf("\n\n\n no. of students - ");
    scanf("%d", &n);
    printf("\n\n Enter students information : \n\n");
    for(i=0; i<n; i++)
    {
        printf("Enter student id - ");
        scanf("%d", &stu[i].id);
        printf("Enter age - ");
        scanf("%d", &stu[i].age);
        printf("Enter marks - ");
        scanf("%f", &stu[i].marks);
        printf("\n");
        eligibility(i);
        printf("\n");
    }

    return 0;
}
```

28/10/20

store
67

DS LAB - 1

Q. Stack implementation using array.

Sol:-

```
#include <stdio.h>

int stack[100], i, size, choice, top, x;

void push()
{
    if(top>=size-1)
    {
      printf("\n\t Stack Overflow");
    }
    else
    {
      printf("Enter a value to be pushed: ");
      scanf("%d", &x);
      top++;
      stack[top] = x;
    }
}


void pop()
{
    if (top<=-1)
    {
      printf("\n\t Stack Underflow");
    }
    else
    {
      printf("\n\t %d is popped", stack[top]);
      top--;
    }
}
```

```
void display()
{
    if (top >= 0)
    {
        printf("\n The elements in the stack - \n");
        for(i=top; i>=0; i--)
        {
            printf("\n %d", stack[i]);
        }
    }
    else
    {
        printf("\n The stack is empty");
    }
}

int main()
{
    top = -1;
    printf("Enter the size of the stack - ");
    scanf("%d", &size);
    do
    {
    printf("\n Stack Operations - \n");
    printf("1. push\n2. pop\n3.display\n4. exit\n");
    printf("\nEnter your choice:");
    scanf("%d",&choice);
    switch(choice)
    {
        case 1: push();
                break;
```

```
        case 2: pop();
                break;
        case 3: display();
                break;
        case 4: printf("\n\t EXIT");
                break;
        default: printf("\n\t Invalid choice");
    }
    while (choice != 4);
    return 0;
}
```

5/10/20

# DS LAB - 2

Q. WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide).

Sol:-

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>

#define SIZE 100

char stack[SIZE];
int top=-1;

void push(char item)
{
    if(top>=SIZE-1)
    {
    printf("\n Stack Overflow");
    }
    else
    {
        top+=1;
        stack[top]=item;
    }
}

char pop()
{
   char item;
   if(top<0)
```

```
    {
      printf("\n Stack Underflow");
      getchar();
      exit(1);
    }
  else
    {
      item = stack[top];
      top -= 1;
      return(item);
    }
}

int is_operator(char symbol)
{
  if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' ||
     symbol == '-')
  {
    return 1;
  }
  else
    {
      return 0;
    }
}

int precedence(char symbol)
{
  if( symbol == '^')
  {
    return(3);
  }
```

```
else if ( symbol == '*' || symbol == '/')
 {
    return(2);
 }
else if ( symbol == '+' || symbol == '-')
 {
    return (1);
 }
 else
  {
    return(0);
  }
}

void InfixToPostfix (char Infix_exp[], char postfix_exp[])
 {
  int i,j;
  char item;
  char x;
  push ('(');
  strcat (infix_exp, ")");

   i = 0;
   j = 0;
   item = infix_exp[i];
  while (item != '\0').
   {
     if (item == '(')
      {
        push (item);
      }
     else if ( isdigit (item) || isalpha (item))
```

```
{
    postfix_exp[j] =item;
    j++;
}
else if (is_operator (item) == 1)
{
    x = pop();
    while ( is_operator(x) == 1 && precedence (x) >= precedence (item))
    {
        postfix_exp[j] = x;
        j++;
        x = pop();
    }
    push(x);
    push (item);
}
else if (item == ')')
{
    x = pop();
    while (x != '(')
    {
        postfix_exp[j] = x;
        j++;
        x = pop();
    }
}
else
{
    printf("\nInvalid Infix expression.\n");
    getchar();
    exit(1);
}
```

```
    i++;
    item = infix_exp[i];
  }
  if (top > 0)
  {
    printf("\n Invalid infix expression.\n");
    getchar();
    exit(1);
  }
  postfix_exp[j] = '\0';
}

int main()
{
  char infix[SIZE], postfix[SIZE];
  printf("Enter Infix expression : ");
  gets(infix);

  InfixToPostfix(infix, postfix);
  printf("Postfix Expression : ");
  puts(postfix);

  return 0;
}
```

- Expected Output :-

```
Enter Infix expression : a+b
Postfix Expression : ab+
```

30/10/20

DS LAB - 3

store 67

Q. Circular Queue

Sol:-

```
#include <stdio.h>
#define MAX 1
int cqueue_arr[MAX]
int front = -1, rear = -1;
void insert (int item)
{
  if ((front == 0 && rear == MAX-1) || (front == rear+1))
  {
    printf(" Queue Overflow\n");
    return;
  }
  if (front == -1)
  {
    front = 0;
    rear = 0;
  }
  else
  {
    if (rear == MAX-1)
      rear = 0;
    else
      rear += 1;
  }
  cqueue_arr[rear] = item;
}

void deletion ()
{
  if (front == -1)
  {
    printf(" Queue Underflow.\n");
    return;
  }
```

```
printf("Element deleted from queue is : %d \n", queue_arr[front]);
if(front == rear)
{
  front = -1;
  rear = -1;
}
else
{
  if (front = MAX - 1)
    front = 0;
  else
    front = front + 1;
}
}

void display()
{
  int front_pos = front, rear_pos = rear;
  if (front == -1)
  {
    printf("Queue is empty \n");
    return;
  }
  printf("Queue elements : \n");
  if(front_pos <= rear_pos)
  {
    while (front_pos <= rear_pos)
    {
      printf("%d", queue_arr[front_pos]);
      front_pos++;
    }
  }
```

```
    else
    {
      while (front_pos <= MAX-1)
      {
        printf("%d", cqueue_arr[front_pos]);
        front_pos++;
      }
    front_pos = 0;
    while (front_pos <= rear_pos)
    {
      printf("%d", cqueue_arr[front_pos]);
      front_pos++;
    }
  }
  printf("\n");
}

int main()
{
  int choice, item;
  do
  {
    printf("1.Insert\n2.Delete\n3.Display\n4.Quit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    switch (choice)
    {
      case 1: printf("Input the element for insertion:");
              scanf("%d", &item);
              ~~break;~~ insert(item);
              break;
```

```
        case 2 : deletion();
                 break;
        case 3 : display();
                 break;
        case 4 :  break;
        default : printf("Wrong Choice\n");
    }
    while(choice != 4);
    return 0;
}
```

Q. Double ended queue.

Sol:-

```
#include<stdio.h>
#define qsize 2
int f=0, r=-1, ch;
int item, q[10];

int isfull()
{
  return(r==qsize-1)? 1:0;
}

int isempty()
{
  return(f>r)?1:0;
}

void insert_rear()
{
  if(isfull())
  {
    printf("queue Overflow\n");
    return;
  }
  r+=1;
  q[r]=item;
}

void delete_front()
{
  if(isempty())
  {
    printf("queue empty\n");
    return;
```

```
        printf("item deleted is %d\n", q[(f)++]);
        if(f>r)
        {
            f=0;
            r=-1;
        }
}

void insert_front()
{
    if (f!=0)
    {
        f--;
        q[f] = item;
        return;
    }
    else if ((f==0)&&(r==-1))
    {
        q[++(r)] = item;
        return;
    }
    else
    {
    }printf("insertion not possible\n");
}

void delete_rear()
{
    if(isempty())
    {
        printf("queue is empty\n");
        return;
    }
```

```
printf("item deleted is %d \n", q[(r)--]);
if (f>r)
{
    f=0;
    r=-1;
}
}

void display()
{
  int i;
  if (isempty())
  {
    printf("queue empty\n");
    return;
  }
  for(i=f; i<=r; i++)
  {
    printf("%d \n", q[i]);
  }
}

void main()
{
  for(;;)
  {
    printf("1.insert-rear \n 2. insert_front \n 3. delete_rear \n
          4. delete_front \n 5. display \n 6. exit \n");
    printf("enter choice: ");
    scanf("%d", &ch);
    switch(ch)
    {
```

```
        case 1 : printf("enter the item: ");
                 scanf("%d", &item);
                 insert_rear();
                 break;
        case 2 : printf("enter the item : ");
                 scanf("%d", &item);
                 insert_front();
                 break;
        case 3 : delete_rear();
                 break;
        case 4 : delete_front();
                 break;
        default : exit(0);
    }
}

return 0;
}
```

2/11/20

store 67

# DS Assignment - 2

Q. Ascending Priority Queue

Sol:-

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAX 2

int pq[MAX]
int count=0;
int d=0;

void insert (int data)
{
    int i=0;
    if (count == MAX)
    {
        printf("Queue Overflow\n");
        return;
    }
    if (count == 0)
    {
        pq[count++] = data;
    }
    else
    {
        for(i= count-1; i>=0; i--)
        {
            if (data < pq[i])
            {
                pq[i+1] = pq[i];
```

```
                else
                {
                    break;
                }
            }
            pq[i+1] = data;
            count++;
        }
}
int removeData()
{
    return pq[d++];
}
void display()
{
    int i;
    if (count == 0)
    {
        printf("Queue is empty\n");
        return;
    }
    printf("Contents of the queue: ");
    for (i=d; i<count; i++)
    {
        printf("%d", pq[i]);
    }
    printf("\n");
}
```

```
int main()
{
   int choice, item;
   for(;;)
    {
      printf("\n1. insert \n2. delete \n3. display \n4. exit \n");
      printf("enter your choice : ");
      scanf("%d", &choice);
      switch(choice)
      {
        case 1 : printf("enter the item to be inserted : ");
                 scanf("%d", &item); insert(item);
                 break;
        case 2 : printf
                 item = removeData();
                 if(item == -1)
                  {
                    printf("Queue is empty \n");
                  }
                 else
                  {
                    printf("item deleted = %d", item);
                  }
                 break;
        case 3 : display();
                 break;
        default : exit(0);
      }
    }
   return 0;
}
```

Q Multiple Priority Queue

Sol:-

```
#include<stdio.h>
#define N 1

int queue[3][N];
int front[3]={0,0,0};
int rear[3]={-1,-1,-1};
int item, ptr;
void main()
{
  int ch;
  while(1)
  {
    printf("\n1.insert\n");
    printf("\n2.delete\n");
    printf("\n3.display\n");
    printf("\n4.exit\n");
    printf("\nenter your choice\n");
    scanf("%d",&ch);
    switch(ch)
    {
      case 1: printf("\nenter priority number\n");
              scanf("%d",&ptr);
              if(ptr>0&&ptr<4)
              {
                pq_insert(ptr-1);
              }
              else
                printf("\nWrong Input\n");
              break;
```

```
        case 2 : pqdelete();
                 break;
        case 3 : display();
                 break;
        case 4 : exit(0);
      }
   }
   getch();
}

pqinsert(int pr)
{
  if(rear[pr] == n-1)
    printf("\n Queue Overflow\n");
  else
   {
    printf("\n enter the item\n");
    scanf("%d", &item);
    rear[pr]++;
    queue[pr][rear[pr]] = item;
   }
  return;
}

pqdelete()
{
   int i;
   for(i=0; i<3; i++)
    {
      if(rear[i] == front[i] - 1)
      {
        printf("Queue empty\n");
      }
```

```
        else
        {
         printf("deleted item is %d of queue %d \n", queue[i][front[i]]
                                                    , i+1);

         front[i]++;
         return;
        }
    }
}

display()
{
  int i,j;
  for(i=0; i<3; i++)
  {
     if(rear[i] == front[i]-1)
      {
        printf("Queue empty %d \n",i+1);
      }
     else
     {
       printf("\n Queue %d :", i+1);
       for(j=front[i]; j<=rear[i]; j++)
        {
          printf("%d \t", queue[i][j]);
        }
     }
  }
return;
}
```