3.

|  |  |  |  |
| --- | --- | --- | --- |
| **Step** | **INPUT** | **STACK** | **OUTPUT** |
| 1 | A | Empty | A |
| 2 | \* | \* | A |
| 3 | B | \* | AB |
| 4 | / | / | AB\* |
| 5 | ( | /( | AB\* |
| 6 | C | /( | AB\*C |
| 7 | - | /(- | AB\*C |
| 8 | D | /(- | AB\*CD |
| 9 | ) | / | AB\*CD- |
| 10 | + | + | AB\*CD-/ |
| 11 | E | + | AB\*CD-/E |
| 12 | \* | +\* | AB\*CD-/E |
| 13 | F | +\* | AB\*CD-/EF |
| 14 |  | Empty | AB\*CD-/EF\*+ |

Steps 1 to 3 : 1 Mark Steps 4 to 6 : 1 Mark Steps: 7 to 9 : 1 Mark Steps: 10 to 12 : 1 Mark Steps: 13 to 14: 1 Mark

4. QUEUE\_SIZE is the size of the queue and initially q.count == 0

struct queue (1 Mark)

{

int f;

int r;

int item[QUEUE\_SIZE];

int count;

};

typedef struct queue QUEUE;

int qfull (QUEUE q) (0.5 Marks)

{

return(q.count == QUEUE\_SIZE ) ? 1 : 0;

}

int qempty(QUEUE q) (0.5 Marks)

{

return(q.count == 0) ? 1 : 0 ;

}

void insert(int item, QUEUE q) ( 1.5 Marks)

if (qfull(q))

{

printf(“Queue is full”);

return;

}

else

{

q.r = q.r + 1 % QUEUE\_SIZE;

q.item[q.r] == item;

(q.count)++;

}

int delete(QUEUE q) (1.5 Marks)

if (qempty(q))

{

printf(“Queue is empty”);

return 0;

}

else

{

item = q.item[q.f];

q.f = q.f+ 1 % QUEUE\_SIZE;

(q.count)--;

return item; }