

Lab program 1

write a program to stimulate the working of stack using an array including push, pop, display programs
start and display appropriate.

```
#include <stdio.h>
int
#define STACK_SIZE 5
int top = -1;
int size s[10];
int item;
void push()
{
    if (top == stack - SIZE - 1)
    {
        printf("stack overflow \n");
        return;
    }
    else
    {
        top = top + 1;
        s[top] = item;
    }
}
int pop()
{
```

Lap program 1

write a program to stimulate the working of stack using an arry including push, pop, display programs.
stack display appropriate.

```
#include <stdio.h>
#define STACK-SIZE 5
int top = -1;
int size[10];
int item;
void push()
{
    if (top == STACK-SIZE - 1)
    {
        printf("stack overflow \n");
        return;
    }
    else
    {
        top = top + 1;
        size[top] = item;
    }
}
int pop()
{
```

```

if (top == -1)
{
    return -1;
}
else
{
    return s[top-1];
}

void display()
{
    int i;
    if (top == -1) {
        cout << "stack empty";
        return;
    }
    else
    {
        cout << "contents of stack \n";
        for (i=top; i>=0; i--)
        {
            cout << "%d \n", s[i];
        }
    }
}

```



```
    break;  
    case 3 : display();  
        break;  
    default : init();  
}  
}  
getch();  
}
```

program2

```
#include <stdio.h>
#include <string.h>
int F(char symbol)
{
    switch (symbol)
    {
        case '+': return 2;
        case '-': return 3;
        case '*': return 4;
        case '/': return 5;
        case '^': return 0;
        case '#': return -1;
        default : return 0;
    }
}
int G(char symbol)
{
    switch (symbol)
    {
        case '+': return 1;
        case '-': return 2;
        case '*': return 3;
        case '/': return 4;
        case '^': return 5;
    }
}
```

(0 number) (1 + or -) (2 * or /) (3 ^ or #) (4 dot) (5 blank)

(6 blank) (7 dot) (8 number) (9 + or -) (10 * or /) (11 ^ or #) (12 blank)

(13 blank) (14 dot) (15 number) (16 + or -) (17 * or /) (18 ^ or #) (19 blank)

(20 blank) (21 dot) (22 number) (23 + or -) (24 * or /) (25 ^ or #) (26 blank)

(27 blank) (28 dot) (29 number) (30 + or -) (31 * or /) (32 ^ or #) (33 blank)

(34 blank) (35 dot) (36 number) (37 + or -) (38 * or /) (39 ^ or #) (40 blank)

(41 blank) (42 dot) (43 number) (44 + or -) (45 * or /) (46 ^ or #) (47 blank)

(48 blank) (49 dot) (50 number) (51 + or -) (52 * or /) (53 ^ or #) (54 blank)

(55 blank) (56 dot) (57 number) (58 + or -) (59 * or /) (60 ^ or #) (61 blank)

(62 blank) (63 dot) (64 number) (65 + or -) (66 * or /) (67 ^ or #) (68 blank)

(69 blank) (70 dot) (71 number) (72 + or -) (73 * or /) (74 ^ or #) (75 blank)

(76 blank) (77 dot) (78 number) (79 + or -) (80 * or /) (81 ^ or #) (82 blank)

(83 blank) (84 dot) (85 number) (86 + or -) (87 * or /) (88 ^ or #) (89 blank)

(90 blank) (91 dot) (92 number) (93 + or -) (94 * or /) (95 ^ or #) (96 blank)

(97 blank) (98 dot) (99 number) (100 + or -) (101 * or /) (102 ^ or #) (103 blank)

(104 blank) (105 dot) (106 number) (107 + or -) (108 * or /) (109 ^ or #) (110 blank)

(111 blank) (112 dot) (113 number) (114 + or -) (115 * or /) (116 ^ or #) (117 blank)

(118 blank) (119 dot) (120 number) (121 + or -) (122 * or /) (123 ^ or #) (124 blank)

(125 blank) (126 dot) (127 number) (128 + or -) (129 * or /) (130 ^ or #) (131 blank)

(132 blank) (133 dot) (134 number) (135 + or -) (136 * or /) (137 ^ or #) (138 blank)

(139 blank) (140 dot) (141 number) (142 + or -) (143 * or /) (144 ^ or #) (145 blank)

(146 blank) (147 dot) (148 number) (149 + or -) (150 * or /) (151 ^ or #) (152 blank)

(153 blank) (154 dot) (155 number) (156 + or -) (157 * or /) (158 ^ or #) (159 blank)

(160 blank) (161 dot) (162 number) (163 + or -) (164 * or /) (165 ^ or #) (166 blank)

(167 blank) (168 dot) (169 number) (170 + or -) (171 * or /) (172 ^ or #) (173 blank)

(174 blank) (175 dot) (176 number) (177 + or -) (178 * or /) (179 ^ or #) (180 blank)

(181 blank) (182 dot) (183 number) (184 + or -) (185 * or /) (186 ^ or #) (187 blank)

(188 blank) (189 dot) (190 number) (191 + or -) (192 * or /) (193 ^ or #) (194 blank)

(195 blank) (196 dot) (197 number) (198 + or -) (199 * or /) (200 ^ or #) (201 blank)

(202 blank) (203 dot) (204 number) (205 + or -) (206 * or /) (207 ^ or #) (208 blank)

(209 blank) (210 dot) (211 number) (212 + or -) (213 * or /) (214 ^ or #) (215 blank)

(216 blank) (217 dot) (218 number) (219 + or -) (220 * or /) (221 ^ or #) (222 blank)

(223 blank) (224 dot) (225 number) (226 + or -) (227 * or /) (228 ^ or #) (229 blank)

(230 blank) (231 dot) (232 number) (233 + or -) (234 * or /) (235 ^ or #) (236 blank)

(237 blank) (238 dot) (239 number) (240 + or -) (241 * or /) (242 ^ or #) (243 blank)

(244 blank) (245 dot) (246 number) (247 + or -) (248 * or /) (249 ^ or #) (250 blank)

(251 blank) (252 dot) (253 number) (254 + or -) (255 * or /) (256 ^ or #) (257 blank)

(258 blank) (259 dot) (260 number) (261 + or -) (262 * or /) (263 ^ or #) (264 blank)

(265 blank) (266 dot) (267 number) (268 + or -) (269 * or /) (270 ^ or #) (271 blank)

(272 blank) (273 dot) (274 number) (275 + or -) (276 * or /) (277 ^ or #) (278 blank)

(279 blank) (280 dot) (281 number) (282 + or -) (283 * or /) (284 ^ or #) (285 blank)

(286 blank) (287 dot) (288 number) (289 + or -) (290 * or /) (291 ^ or #) (292 blank)

(293 blank) (294 dot) (295 number) (296 + or -) (297 * or /) (298 ^ or #) (299 blank)

(300 blank) (301 dot) (302 number) (303 + or -) (304 * or /) (305 ^ or #) (306 blank)

(307 blank) (308 dot) (309 number) (310 + or -) (311 * or /) (312 ^ or #) (313 blank)

(314 blank) (315 dot) (316 number) (317 + or -) (318 * or /) (319 ^ or #) (320 blank)

(321 blank) (322 dot) (323 number) (324 + or -) (325 * or /) (326 ^ or #) (327 blank)

(328 blank) (329 dot) (330 number) (331 + or -) (332 * or /) (333 ^ or #) (334 blank)

(335 blank) (336 dot) (337 number) (338 + or -) (339 * or /) (340 ^ or #) (341 blank)

(342 blank) (343 dot) (344 number) (345 + or -) (346 * or /) (347 ^ or #) (348 blank)

(349 blank) (350 dot) (351 number) (352 + or -) (353 * or /) (354 ^ or #) (355 blank)

(356 blank) (357 dot) (358 number) (359 + or -) (360 * or /) (361 ^ or #) (362 blank)

(363 blank) (364 dot) (365 number) (366 + or -) (367 * or /) (368 ^ or #) (369 blank)

(370 blank) (371 dot) (372 number) (373 + or -) (374 * or /) (375 ^ or #) (376 blank)

(377 blank) (378 dot) (379 number) (380 + or -) (381 * or /) (382 ^ or #) (383 blank)

(384 blank) (385 dot) (386 number) (387 + or -) (388 * or /) (389 ^ or #) (390 blank)

(391 blank) (392 dot) (393 number) (394 + or -) (395 * or /) (396 ^ or #) (397 blank)

(398 blank) (399 dot) (400 number) (401 + or -) (402 * or /) (403 ^ or #) (404 blank)

(405 blank) (406 dot) (407 number) (408 + or -) (409 * or /) (410 ^ or #) (411 blank)

(412 blank) (413 dot) (414 number) (415 + or -) (416 * or /) (417 ^ or #) (418 blank)

(419 blank) (420 dot) (421 number) (422 + or -) (423 * or /) (424 ^ or #) (425 blank)

(426 blank) (427 dot) (428 number) (429 + or -) (430 * or /) (431 ^ or #) (432 blank)

(433 blank) (434 dot) (435 number) (436 + or -) (437 * or /) (438 ^ or #) (439 blank)

(440 blank) (441 dot) (442 number) (443 + or -) (444 * or /) (445 ^ or #) (446 blank)

(447 blank) (448 dot) (449 number) (450 + or -) (451 * or /) (452 ^ or #) (453 blank)

(454 blank) (455 dot) (456 number) (457 + or -) (458 * or /) (459 ^ or #) (460 blank)

(461 blank) (462 dot) (463 number) (464 + or -) (465 * or /) (466 ^ or #) (467 blank)

(468 blank) (469 dot) (470 number) (471 + or -) (472 * or /) (473 ^ or #) (474 blank)

(475 blank) (476 dot) (477 number) (478 + or -) (479 * or /) (480 ^ or #) (481 blank)

(482 blank) (483 dot) (484 number) (485 + or -) (486 * or /) (487 ^ or #) (488 blank)

(489 blank) (490 dot) (491 number) (492 + or -) (493 * or /) (494 ^ or #) (495 blank)

(496 blank) (497 dot) (498 number) (499 + or -) (500 * or /) (501 ^ or #) (502 blank)

(503 blank) (504 dot) (505 number) (506 + or -) (507 * or /) (508 ^ or #) (509 blank)

(510 blank) (511 dot) (512 number) (513 + or -) (514 * or /) (515 ^ or #) (516 blank)

(517 blank) (518 dot) (519 number) (520 + or -) (521 * or /) (522 ^ or #) (523 blank)

(524 blank) (525 dot) (526 number) (527 + or -) (528 * or /) (529 ^ or #) (530 blank)

(531 blank) (532 dot) (533 number) (534 + or -) (535 * or /) (536 ^ or #) (537 blank)

(538 blank) (539 dot) (540 number) (541 + or -) (542 * or /) (543 ^ or #) (544 blank)

(545 blank) (546 dot) (547 number) (548 + or -) (549 * or /) (550 ^ or #) (551 blank)

(552 blank) (553 dot) (554 number) (555 + or -) (556 * or /) (557 ^ or #) (558 blank)

(559 blank) (560 dot) (561 number) (562 + or -) (563 * or /) (564 ^ or #) (565 blank)

(566 blank) (567 dot) (568 number) (569 + or -) (570 * or /) (571 ^ or #) (572 blank)

(573 blank) (574 dot) (575 number) (576 + or -) (577 * or /) (578 ^ or #) (579 blank)

(580 blank) (581 dot) (582 number) (583 + or -) (584 * or /) (585 ^ or #) (586 blank)

(587 blank) (588 dot) (589 number) (590 + or -) (591 * or /) (592 ^ or #) (593 blank)

(594 blank) (595 dot) (596 number) (597 + or -) (598 * or /) (599 ^ or #) (600 blank)

(601 blank) (602 dot) (603 number) (604 + or -) (605 * or /) (606 ^ or #) (607 blank)

(608 blank) (609 dot) (610 number) (611 + or -) (612 * or /) (613 ^ or #) (614 blank)

(615 blank) (616 dot) (617 number) (618 + or -) (619 * or /) (620 ^ or #) (621 blank)

(622 blank) (623 dot) (624 number) (625 + or -) (626 * or /) (627 ^ or #) (628 blank)

(629 blank) (630 dot) (631 number) (632 + or -) (633 * or /) (634 ^ or #) (635 blank)

(636 blank) (637 dot) (638 number) (639 + or -) (640 * or /) (641 ^ or #) (642 blank)

(643 blank) (644 dot) (645 number) (646 + or -) (647 * or /) (648 ^ or #) (649 blank)

(650 blank) (651 dot) (652 number) (653 + or -) (654 * or /) (655 ^ or #) (656 blank)

(657 blank) (658 dot) (659 number) (660 + or -) (661 * or /) (662 ^ or #) (663 blank)

(664 blank) (665 dot) (666 number) (667 + or -) (668 * or /) (669 ^ or #) (670 blank)

(671 blank) (672 dot) (673 number) (674 + or -) (675 * or /) (676 ^ or #) (677 blank)

(678 blank) (679 dot) (680 number) (681 + or -) (682 * or /) (683 ^ or #) (684 blank)

(685 blank) (686 dot) (687 number) (688 + or -) (689 * or /) (690 ^ or #) (691 blank)

(692 blank) (693 dot) (694 number) (695 + or -) (696 * or /) (697 ^ or #) (698 blank)

(699 blank) (700 dot) (701 number) (702 + or -) (703 * or /) (704 ^ or #) (705 blank)

(706 blank) (707 dot) (708 number) (709 + or -) (710 * or /) (711 ^ or #) (712 blank)

(713 blank) (714 dot) (715 number) (716 + or -) (717 * or /) (718 ^ or #) (719 blank)

(720 blank) (721 dot) (722 number) (723 + or -) (724 * or /) (725 ^ or #) (726 blank)

(727 blank) (728 dot) (729 number) (730 + or -) (731 * or /) (732 ^ or #) (733 blank)

(734 blank) (735 dot) (736 number) (737 + or -) (738 * or /) (739 ^ or #) (740 blank)

(741 blank) (742 dot) (743 number) (744 + or -) (745 * or /) (746 ^ or #) (747 blank)

(748 blank) (749 dot) (750 number) (751 + or -) (752 * or /) (753 ^ or #) (754 blank)

(755 blank) (756 dot) (757 number) (758 + or -) (759 * or /) (760 ^ or #) (761 blank)

(762 blank) (763 dot) (764 number) (765 + or -) (766 * or /) (767 ^ or #) (768 blank)

(769 blank) (770 dot) (771 number) (772 + or -) (773 * or /) (774 ^ or #) (775 blank)

(776 blank) (777 dot) (778 number) (779 + or -) (780 * or /) (781 ^ or #) (782 blank)

(783 blank) (784 dot) (785 number) (786 + or -) (787 * or /) (788 ^ or #) (789 blank)

(790 blank) (791 dot) (792 number) (793 + or -) (794 * or /) (795 ^ or #) (796 blank)

(797 blank) (798 dot) (799 number) (800 + or -) (801 * or /) (802 ^ or #) (803 blank)

(804 blank) (805 dot) (806 number) (807 + or -) (808 * or /) (809 ^ or #) (810 blank)

(811 blank) (812 dot) (813 number) (814 + or -) (815 * or /) (816 ^ or #) (817 blank)

(818 blank) (819 dot) (820 number) (821 + or -) (822 * or /) (823 ^ or #) (824 blank)

(825 blank) (826 dot) (827 number) (828 + or -) (829 * or /) (830 ^ or #) (831 blank)

(832 blank) (833 dot) (834 number) (835 + or -) (836 * or /) (837 ^ or #) (838 blank)

(839 blank) (840 dot) (841 number) (842 + or -) (843 * or /) (844 ^ or #) (845 blank)

(846 blank) (847 dot) (848 number) (849 + or -) (850 * or /) (851 ^ or #) (852 blank)

(853 blank) (854 dot) (855 number) (856 + or -) (857 * or /) (858 ^ or #) (859 blank)

(860 blank) (861 dot) (862 number) (863 + or -) (864 * or /) (865 ^ or #) (866 blank)

(867 blank) (868 dot) (869 number) (870 + or -) (871 * or /) (872 ^ or #) (873 blank)

(874 blank) (875 dot) (876 number) (877 + or -) (878 * or /) (879 ^ or #) (880 blank)

(881 blank) (882 dot) (883 number) (884 + or -) (885 * or /) (886 ^ or #) (887 blank)

(888 blank) (889 dot) (890 number) (891 + or -) (892 * or /) (893 ^ or #) (894 blank)

(895 blank) (896 dot) (897 number) (898 + or -) (899 * or /) (900 ^ or #) (901 blank)

(902 blank) (903 dot) (904 number) (905 + or -) (906 * or /) (907 ^ or #) (908 blank)

(909 blank) (910 dot) (911 number) (912 + or -) (913 * or /) (914 ^ or #) (915 blank)

(916 blank) (917 dot) (918 number) (919 + or -) (920 * or /) (921 ^ or #) (922 blank)

(923 blank) (924 dot) (925 number) (926 + or -) (927 * or /) (928 ^ or #) (929 blank)

(930 blank) (931 dot) (932 number) (933 + or -) (934 * or /) (935 ^ or #) (936 blank)

(937 blank) (938 dot) (939 number) (940 + or -) (941 * or /) (942 ^ or #) (943 blank)

(944 blank) (945 dot) (946 number) (947 + or -) (948 * or /) (949 ^ or #) (950 blank)

(951 blank) (952 dot) (953 number) (954 + or -) (955 * or /) (956 ^ or #) (957 blank)

(958 blank) (959 dot) (960 number) (961 + or -) (962 * or /) (963 ^ or #) (964 blank)

(965 blank) (966 dot) (967 number) (968 + or -) (969 * or /) (970 ^ or #) (971 blank)

(972 blank) (973 dot) (974 number) (975 + or -) (976 * or /) (977 ^ or #) (978 blank)

(979 blank) (980 dot) (981 number) (982 + or -) (983 * or /) (984 ^ or #) (985 blank)

(986 blank) (987 dot) (988 number) (989 + or -) (990 * or /) (991 ^ or #) (992 blank)

(993 blank) (994 dot) (995 number) (996 + or -) (997 * or /) (998 ^ or #) (999 blank)

Case '^':

```
case '^': return 6;
case '(': return 9;
case ')': return 0;
default : return 7;
}
```

```
void infix_postfix (char infix[], char postfix[])
```

```
{ int top, i, j;
char s[20], symbol;
top = -1;
s[++top] = '#';
j = 0;
```

```
for ( i=0 ; i < strlen(infix) ; i++ )
```

```
{
```

```
symbol = infix[i];
```

```
while ( F(s[top]) > g(symbol) )
```

```
{
```

```
postfix[j] = s[top-1];
j++
```

```
}
```

```
if ( F(s[top]) != g(symbol) )
```

```
s[++top] = symbol;
```

```
else
```

```
top--;
```

```
}
```

```
while (s[top] != '#')  
{ postfin[i] = '0';  
}  
}  
int main ()  
{ char infix [20];  
char postfix [20];  
printf ("Enter the valid parenthesized infix  
expression\n");  
scanf ("%s", infix);  
infix -> postfin (infix, postfix);  
printf ("The postfix expression is\n");  
printf ("%s\n", postfix);  
return 0;  
}
```

Linear Queue

```
#include <stdio.h>
#define MAX 5

int queue[MAX];
int rear = -1, front = -1;
void insert()
{
    int add-item;
    if (rear == MAX - 1) // overflow
        printf("Queue Overflow\n");
    else
    {
        if (front == -1) // empty
            front = 0;
        printf("Enter the element to be inserted : ");
        scanf("%d", &add-item);
        rear = rear + 1;
        queue[rear] = add-item;
    }
}

void delete()
{
    if (front == -1 || front > rear)
        printf("Queue underflow\n");
    return;
}
```

else

{ printf ("Element deleted from queue is : %d\n",
queue [front]);
front = front + 1;
}

void display()

{ int i;

if (front == -1)

printf ("Queue is empty \n");

else

{ printf ("Queue is : \n");

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

printf ("%d\n", queue [i]);

printf ("\n");

}

main()

{

int choice;

while (1)

{

printf ("1 : INSERT \n");

printf ("2 : DELETE \n");

printf ("3 : DISPLAY \n");

printf ("4 : EXIT \n");

```
printf("Enter your choice : ");
scanf('%d', &choice);
switch(choice)
{
    case 1:
        insert();
        break;
    case 2:
        delete();
        break;
    case 3:
        display();
        break;
    case 4:
        exit(1);
    default:
        printf("Wrong choice\n");
}
}
```

program + queue = circular queue

Circular Queue

```

#include <stdio.h>
#define q-size 5
int item, front = 0, rear = -1, q[q-size], count = 0;
void insert_rear()
{
    if(count == q-size)
    {
        printf("Queue Overflow\n");
        return;
    }
    rear = (rear + 1) % q-size;
    q[rear] = item;
    count++;
    return;
}
int delete_front()
{
    if(count == 0)
        return -1;
    else
    {
        item = q[front];
        front = (front + 1) % q-size;
        count--;
        return item;
    }
}

```

```
}
```

```
void display()
```

```
{ int i, f;
```

```
if (counter == 0)
```

```
{ printf ("queue is empty\n"); }
```

```
return;
```

```
f=front;
```

```
printf ("contents of Queue:\n");
```

```
for (i=0; i<=count; i++)
```

```
{ printf ("%d\n", q[f]); }
```

```
f=(f+1)%q-size;
```

```
void main()
```

```
{
```

```
int choice;
```

```
for (;;)
```

```
{ printf ("1: Insert \n 2: DELETE \n 3: DISPLAY\n"); }
```

```
printf ("enter the choice \n ");
```

```
scanf ("%d", &choice);
```

```
switch (choice) {
```

```
case 1 : printf ("Enter an element to be Inserted \n");
    scanf ("%d"; &item);
    insert_read();
    break;

case 2 : item=deletel-front();
    if (item == -1)
        printf ("queue is empty \n");
    printf ("item deleted is %d \n"; item);
    break;

case 3 : display();
    break;

default : exit(0);
```

}

Program 5.

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

typedef struct ((alloc) node) {
    int data;
    struct node *link;
} node;

node *root = NULL;

void add_at_end() {
    node *temp;
    temp = (node *) malloc (sizeof(node));
    printf ("Enter the node element \n");
    scanf ("%d", &temp->data);
    temp->link = NULL;
    if (root == NULL)
        root = temp;
    else {
        node *p = root;
        while (p->link != NULL)
            p = p->link;
        p->link = temp;
    }
}
```

```
void add-at-begin ()  
{  
    node *temp;  
    temp = (node*) malloc(sizeof(node));  
    printf("Enter node element in ");  
    scanf("%d", &temp->data);  
    temp->link = NULL;  
    if (root == NULL)  
    {  
        root = temp;  
    }  
    else  
    {  
        temp->link = root;  
        root = temp;  
    }  
}
```

```
int length()  
{  
    node *p;  
    p = root;  
    int i = 0;  
    while (p != NULL)  
    {  
        i++;  
        p = p->link;  
    }  
}
```

```
return L;
}

void add-after(C) {
    node *p, *temp;
    int loc, i=L;
    printf ("Enter the location \n");
    scanf ("%d", &loc);
    if (loc > lengthL)
    {
        printf ("Invalid location, the list has %d nodes", lengthL);
    }
    else
    {
        p = root;
        while (i < loc)
        {
            p = p->link;
            i++;
        }
        temp = (node *)malloc (sizeof (node));
        printf ("Enter the node element \n");
        scanf ("%d", &temp->data);
        temp->link = NULL;
        temp->link = p->link;
        p->link = temp;
    }
}
```

```

void deletec()
{
    int loc;
    node *temp;
    printf("Enter the location of node to be deleted\n");
    scanf("%d", &loc);
    if (loc > length())
    {
        printf("There is no such node\n");
    }
    else if (loc == 1)
    {
        temp = root;
        root = temp -> link;
        temp -> link = NULL;
        free(temp);
    }
    else
    {
        node *p = root, *q;
        int i = 1;
        while (i < loc - 1)
        {
            p = p -> link;
            i++;
        }
        q = p -> link;
        p -> link = q -> link;
        q -> link = NULL;
        free(q);
    }
}

```

```
void display()
{
    node *temp = root;
    if (temp == NULL)
    {
        printf("NO nodes in the list \n");
    }
    else
    {
        while (temp != NULL)
        {
            printf("./d\n", temp->data);
            temp = temp->link;
        }
    }
}
```

```
int main()
{
    int op, len;
    while (1)
    {
        printf("1. Add in begin \n2. add at end \n");
        printf("3. Add after a node \n4. Delete node \n5. Display \n");
        printf("6. length of list \n7. Exit \n");
        scanf("./d", &op);
        switch (op)
        {
            case 1 : add_at_begin();
            break;
            case 2 : add_at_end();
            break;
```

```
case 3 : add-after();  
    break;  
case 4 : delete();  
    break;  
case 5 : display();  
    break;  
case 6 : len = length();  
    printf("the length is %d \n", len);  
    break;  
case 7 : exit(0);  
    break;  
default : printf("No such Operation\n");  
}  
}  
return 0;  
}
```

Program

```
#include <stdio.h>
#include <stdlib.h>
struct node{
    int info;
    struct node* link;
};

typedef struct node* NODE;
NODE getnode()
{
    NODE x;
    x = (NODE) malloc(sizeof(struct node));
    if (x == NULL)
    {
        printf("Memory is full");
        return exit(0);
    }
    return x;
}

NODE delete_front(NODE first)
{
    if (first == NULL)
        printf("List is empty");
    return;
    NODE temp;
    temp = first->link;
    free(first);
    return temp;
}
```

```
NODE * delete_rear ( NODE * first )
```

```
{
```

```
    if ( first == NULL )
```

```
{
```

```
    printf (" List is empty ");
```

```
    return ;
```

```
}
```

```
NODE temp, prev ;
```

```
temp = first ;
```

```
while ( temp != NULL )
```

```
{ prev = temp ;
```

```
temp = temp -> link ;
```

```
}
```

```
prev -> link = NULL ;
```

```
free ( temp ) ;
```

```
return first ;
```

```
}
```

```
NODE * insert ( NODE * first )
```

```
{ int info ; queue ( info, first ) ; }
```

```
NODE temp ; temp = getnode ( ) ;
```

```
printf (" Enter the item : " ) ;
```

```
scanf ("%d", &info ) ;
```

```
temp -> info = info ;
```

```
temp -> link = first ;
```

```
return temp ;
```

```
} .
```

~~NODE~~ delete-at-pos (NODE first)

{

int pos;

NODE temp, prev;

printf ("enter the position: ");

scanf ("%d", &pos);

for temp = first

for (int i=0; i <= pos; j ~~++temp~~ i++)

{ prev = temp;

{ temp = temp -> link;

} if (temp == NULL)

{ printf ("deletion not possible"); return first; }

prev -> link = temp -> link;

free (temp);

return first;

}

```
NODE delete_item (NODE *first)
{
    if (first == NULL)
    {
        printf ("list is empty\n");
        return first;
    }
    NODE temp = first; int flag = 0; NODE prev = NULL;
    NODE ptr;
    while (temp != NULL)
    {
        if (temp->info == key)
        {
            ptr = temp->link;
            prev->link = ptr;
            free (temp);
            flag = 1;
        }
        prev = temp;
        temp = temp->link;
    }
    if (flag == 0)
    {
        printf ("Element is not found in list\n");
        return first;
    }
}
```

```
int main()
{
    int choice, item, pos;
    NODE first = NULL;
    for (;;)
        printf (" 1: Insert \n 2: Delete front \n 3:
Delete rear \n 4: Delete at position \n 5:
Delete specified item \n");
    printf ("Enter the choice ");
    scanf ("%d", &choice);
    switch (choice)
    {
        case 1: insert (first);
        break;
        case 2: deleteFront (first);
        break;
        case 3: deleteRear (first);
        break;
        case 4: deleteAtPos (first);
        break;
        case 5: printf ("enter the item: ");
        scanf ("%d", &item);
        first = deleteItem (first, item);
        break;
        case 6: exit(0);
        default : break;
    }
}
```

Program :-

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int info;
    struct node* link;
};
typedef struct node* NODE;
NODE getnode() {
    NODE n;
    n = (NODE) malloc (sizeof (struct node));
    if (n == NULL)
        { printf ("memory is full"); exit(0); }
    return n;
}
NODE insert(NODE first)
{
    int item;
    NODE temp;
    temp = getnode();
    temp->info = item;
    temp->link = first;
    printf ("enter the item to insert : ");
    scanf ("%d", &item);
    temp->link = item;
}
```

```
void reverse(NODE first)
{
    NODE cur = first; // cur = head
    NODE prev = NULL; // previous node
    NODE next = NULL;
    while (cur != NULL) {
        next = cur->link; // store next node
        cur->link = prev; // reverse link
        prev = cur; // move previous node
        cur = next;
    }
    first = prev; // new head
}
```

```
3 void sort (NODE first)
{
    NODE cur, indent, temp;
    if (first == NULL) {
        return;
    }
```

while (cur != NULL)

```
{  
    indent = cur->indent;  
    if (cur->info > indent->info)  
    {  
        temp = cur->info;  
        cur->info = indent->info;  
        indent->info = temp;  
    }  
    indent = indent->link;  
}  
}
```

display (NODE first)

```
{  
    NODE temp = first;  
    while (temp != NULL)  
    {  
        printf ("%d\n", temp->data);  
        temp = temp->link;  
    }  
}
```

```

void concat (NODE first, NODE second)
{
    if (first == NULL)
        return second;
    if (second == NULL)
        return first;
    while (NODE temp = first)
        while (temp != NULL)
            temp = temp->link;
        temp->link = second;
        return first;
}

void main ()
{
    int choice; NODE first = NULL; NODE second = NULL;
    for (;;)
    {
        printf ("1: INSERT first \n 2: INSERT second \n
3: Delete first \n 4: Delete second \n 5: Reverse first
\n 6: reverse second \n 7: sort first \n 8: sort
second \n 9: concat \n 10: display \n");
        scanf ("%d", &choice);
    }
}

```

switch
case
switch (choice)

{
case 1 : insert (first);
 break;

case 2 : insert (second);
 break;

case 3 : delete-node (first);
 break;

case 4 : delete-node (second);
 break;

case 5 : reverse (first);
 break;

case 6 : reverse (second);
 break;

case 7 : ~~sort~~ (first);
 break;

case 8 : sort (second);
 break;

case 9 : concat (first, second);
 break;

case 10 : display (first);
 break;

program 8

implementation of stack ;

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
struct node{
```

```
    int info;
```

```
    struct node* link;
```

```
};
```

```
typedef structnode* NODE;
```

```
NODE getnode()
```

```
{
```

```
    NODE *x;
```

```
x = (NODE) malloc(sizeof(struct node));
```

```
if (x == NULL)
```

```
{ printf("memory is full");
```

```
    exit(0);
```

```
}
```

```
return x;
```

```
}
```

```
void push (NODE first)
```

```
{ NODE temp , cur; int item;
```

```
cur = getnode();
```

```
printf("enter the item: ");
```

```
scanf("%d", &item);
```

```
if (first == NULL) return cur;
temp = first;
while (temp != NULL)
    temp = temp -> link;
    temp -> link = cur;
return first;
}
```

```
NODE pop(NODE first)
{
    if (first == NULL) { printf("List is empty"); return; }
    NODE temp, prev = NULL;
    temp = first;
    while (temp != NULL) { prev = temp;
        temp = temp -> link;
        prev -> link = NULL;
        free(temp);
    }
    return first;
}
```

```
NODE display( NODE first )  
{  
    NODE temp ;  
    if ( first == NULL ) { printf( "Empty list" ); return first ; }  
    temp = first ;  
    while ( temp != NULL ) {  
        printf( "%d \t", temp->info ) ;  
        temp = temp->link ;  
    }  
    return first ;  
}
```

```
void main ()  
{  
    int choice ;  
    NODE first = NULL ;  
    for ( ; )  
    {  
        printf( "1: Push \n 2: Pop \n 3: DISPLAY" ) ;  
        scanf( "%d" , &choice ) ;  
        printf( "Enter your choice: " ) ;  
        scanf( "%d" , &choice ) ;  
        switch ( choice )  
        {
```

```
case 1 : first = push(first);
break;
case 2 : po first = pop(first);
break;
case 3 : first = display(first);
break;
default: break;
}
}
```

Statement 2 : ~~if (first == null)~~ {
 System.out.println("Stack Underflow");
 return;

Implementation Queue :-

```
#include <sfedio.h>
#include <stdlib.h>

struct node
{
    int info;
    struct node * link;
};

type def struct node * 'NODE';

NODE getnode()
{
    NODE n;
    n = (NODE) malloc(sizeof(struct node));
    if (n == NULL)
    {
        printf("memory is full");
        exit(0);
    }
    return n;
}
```

```
NODE insert (NODE first, int item) {
    NODE temp, cur;
    cur = getnode();
    cur->info = item;
    cur->link = NULL;
    if (first == NULL)
        return first;
    }
    temp = first;
    while (temp != NULL) {
        if (temp->info == item)
            return temp;
        temp = temp->link;
    }
    temp->link = cur;
    return first;
}
```

```
NODEC delete_(NODE first)
{ if(first == NODLL) { printf("list is empty"); return; }
  NODE temp = first->link;
  free(first);
  return temp;
```

~~Not~~ void display(NODEfirst)

```
{ NODE temp = first;
  if(temp == NULL) { printf("empty list"); return; }
  while(temp != NULL) { printf("%d\n", temp->data);
    temp = temp->link;
  }
```

while (temp != NULL)

```
{ printf("%d\n", temp->data);
  temp = temp->link;
```

}

~~return~~

}

```

int main()
{
    int choice, item;
    NODE first = NULL;

    for(;;)
    {
        printf("1: Insert\n2: Delete\n3: Display");
        printf("enter the item: ");
        scanf("%d", &choice);

        switch(choice)
        {
            case 1 : printf("enter the item: ");
                        scanf("%d", &item);
                        first = insert(first, item);
                        break;

            case 2 : Delete(first); break;

            case 3 : Display(first); break;

            default: break;
        }
    }
}

```

program 9

#include < stdio.h>
#include < stdlib.h>

```
struct node {  
    int info;  
    struct node *rlink;  
    struct node *llink;  
};
```

typedef struct node* NODE;
NODE getnode()
{
 NODE x;
 x = (NODE)malloc(sizeof(struct node));
 if (x==NULL)
 {
 printf("Memory full\n");
 exit(0);
 }
 return x;
}

void freenode(NODE x){
 free(x);
}

NODE insert_front(int item, NODE head){
 NODE temp, cur; // head is base
 temp = getnode(); // tail is base
 temp->info = item;
 cur = head->rlink; // head doesn't have
 head->rlink = temp; // head contains
 temp->llink = head; // tail contains
 temp->rlink = cur; // tail doesn't have
 cur->llink = temp; // tail contains
 return head;
}

| L009

| IBM 19CS162

~~QUESTION~~ NODE insert_rear (int item, NODE head)

{ NODE temp, cur ;

temp = getnode () ;

temp -> info = item ;

cur = head -> llink ;

head -> llink = temp ;

temp -> rlink = head ;

temp -> llink = cur ;

cur -> rlink = temp ;

return head ;

}

NODE delete_front (NODE head)

{

NODE cur, next ;

if (head -> rlink == head)

{ printf ("dq empty \n") ;

return head ;

}

cur = head -> rlink ;

next = cur -> rlink ;

head -> rlink = next ;

next -> llink = head ;

printf ("node deleted is .d %c \n", cur -> info) ;

freinode (cur) ;

return head ;

}

```

NODE ddelete_rear(NODE head)
{
    NODE cur, prev;
    if (head->tlink == head)
    {
        printf("dq empty\n");
        return head;
    }
    cur = head->tlink;
    prev = cur->llink;
    head->tlink = prev;
    prev->llink = head;
    printf("the node deleted is %d", cur->info);
    free(cur);
    return head;
}

```

void display(NODE head)

```

{
    NODE temp;
    if (head->tlink == head)
    {
        printf("dq empty\n");
        return;
    }
    printf("contents of dq\n");
    temp = head->tlink;

```

```
while (temp != NULL)
    while (temp != head)
        printf ("%d\n", temp->info);
        temp = temp->rlink;
    }
    printf ("n");
}
```

```
int length (NODE first)
```

```
{ NODE temp = first->rlink;
    int ct = 0;
    while (temp != first)
        {
            ct++;
            temp = temp->rlink;
        }
    printf ("length of the list is %d ", ct);
    return ct;
}
```

```
NODE search (NODE first)
```

```
{ NODE temp = first->rlink;
    int count = 0, key, flag = 0;
    printf ("Enter the key: ");
    scanf ("%d", &key);
    while (first != NULL)
    {
        if (key == first->info)
            flag = 1;
        first = first->rlink;
    }
    if (flag == 1)
        printf ("Key found\n");
    else
        printf ("Key not found\n");
}
```

```

        count++;
        if (!temp->info == key)
        {
            flag = 1;
            printf ("key %d found in position %d", key,
            records stored, count);
        }
        temp = temp->rlink; // end of main loop
    }
    if (flag == 0)
        printf ("%d is not found in list");
    return first;
}

NODE insert_after (NODE first)
{
    int key, item, flag=0;
    printf ("Enter the element: ");
    scanf ("%d", &key);
    NODE temp = first->rlink;
    NODE ptr = getnode();
    while (temp != first)
    {
        if (temp->info == key)
        {
            printf ("Enter the item need to be inserted: ");
            scanf ("%d", &item);
            ptr->info = item;
            ptr->rlink = temp->rlink;
            ptr->llink = temp;
            temp->rlink = ptr;
        }
    }
}

```

```
    flag = 1;
    return first;
}
temp = temp->rlink;
if (flag == 0)
    printf ("There is no such element \"%s\"");
    return first;
}
}
```

```
NODE insert_after (NODE first)
```

```
{ int key, item, flag = 0;
    printf ("Enter the element : ");
    scanf ("%d", &key);
    NODE temp = first->rlink;
    NODE *ptr = getnode();
    while (temp != first)
    {
        if (temp->info == key)
        {
            printf ("Enter the item need to be inserted");
            scanf ("%d", &item);
            ptr->info = item;
            ptr->rlink = temp;
            ptr->llink = temp->llink;
            ptr->llink = ptr;
            flag = 1;
        }
        temp = temp->rlink;
    }
    if (flag == 0)
        printf ("Element not found");
    return first;
}
```

```
3. temp = temp->rlink;
```

3

```

NODE * head, *last;
int item, choice, len;
head = getNode();
head->rlink = head;
head->llink = head;
for (j; )
{
    printf("1: insert front\n2: insert rear\n3:
        4: delete front\n5: display\n6: exit\n7: delete duplicate items\n8:
        insert before\n9: insert after\n");
    printf("enter the choice\n");
    scanf("%d", &choice);
    switch (choice)
    {
        case 1 : printf("enter the item at front end\n");
        scanf("%d", &item);
        last = insertFront(item, head);
        break;
        case 2 : printf("enter the item at rear end\n");
        scanf("%d", &item);
        last = insertRear(item, head);
        break;
        case 3 : last = deleteFront(head);
        break;
        case 4 : last = deleteFront(head);
        break;
    }
}

```

```
case 5 : display(head) ;  
        break ;  
case 7 : delete_dup(head) ;  
        break ;  
case 8 : head = length(head) ;  
        break ;  
case 9 : head = search(head) ;  
        break ;  
case 10 : head = insert_after(head) ;  
        break ;  
case 11 : head = insert_before(head) ;  
        break ;  
default : break  
}  
}  
}
```

Program 10 Binary Search Tree

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

struct node {
    int key;
    struct node *left, *right;
};

struct node *newNode(int item) {
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}

void inorder(struct node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d\t", root->key);
        inorder(root->right);
    }
    if (root == NULL)
        {
            printf("Tree is empty\n");
            return;
        }
}
```

```
void preorder (struct node *root),
```

```
{ if (root != NULL) {
```

```
    printf ("%d \t", root->key);
```

```
    preorder (root->left);
```

```
    preorder (root->right);
```

```
}
```

```
if (root == NULL)
```

```
{
```

```
    printf ("Tree is empty \n");
```

```
    return;
```

```
}
```

```
void postorder (struct node *root)
```

```
{
```

```
if (root != NULL) {
```

```
    return;
```

```
    postorder (root->left);
```

```
    postorder (root->right);
```

```
    printf ("%d \t", root->key);
```

```
)
```

```
}
```

```

struct node * insert ( struct node * node , int key )
{
    if ( node == NULL )
        return newNode ( key );
    if ( key < node->key )
        node->left = insert ( node->left , key );
    else if ( key > node->key )
        node->right = insert ( node->right , key );
    return node ;
}

int main ( )
{
    int ch ;
    int item ;
    struct node * root = NULL ;
    for ( ; )
    {
        printf (" 1 : Create tree \n 2 : Insert \n 3 : InOrder \n 4 : "
            " Preorder \n 5 : Postorder \n " );
        printf (" Enter your choice : " );
        scanf ("%d" , &ch );
        switch (ch )
        {
            case 1 : printf (" Enter the name : " );
                search ( &root , &item );
                root = insert ( root , item );
                break;
        }
    }
}

```

```

struct node * insert (struct node * node , int key)
{
    if (node == NULL)
        return newNode(key);
    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);
    return node;
}

int main()
{
    int ch;
    int item;
    struct node * root = NULL;
    for(;;)
    {
        printf("1: Create tree\n 2: Insert\n 3: InOrder\n 4:
               preorder\n 5: postorder\n");
        printf("Enter your choice:");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1: printf("Enter the name :");
                      scanf("%s", &item);
                      root = insert(root, item);
                      break;
        }
    }
}

```

```

case 2 : printf ("Enter the item: ");
scanf ("%d", &item);
insert (root, item);
break;
case 3 : inorder (root);
break;
case 4 : preorder (root);
break;
case 5 : postorder (root);
break;
default : break;
}
return 0;
}

```

int main () {
 struct node *root = NULL;
 int choice, item;

do {
 cout << "1. Insertion" << endl;
 cout << "2. Deletion" << endl;
 cout << "3. Inorder Traversal" << endl;
 cout << "4. Preorder Traversal" << endl;
 cout << "5. Postorder Traversal" << endl;
 cout << "6. Exit" << endl;
 cout << "Enter your choice: ";

cin >> choice;
 switch (choice) {
 case 1 : insert (root, item);
 break;
 }
}

cout << "Enter the item to be inserted: ";
cin >> item;
insert (root, item);
cout << "Item inserted successfully" << endl;
cout << "Do you want to continue (y/n): ";
cin >> choice;
if (choice == 'n')
 break;
}

cout << "Enter the item to be deleted: ";
cin >> item;
delete (root, item);
cout << "Item deleted successfully" << endl;
cout << "Do you want to continue (y/n): ";
cin >> choice;
if (choice == 'n')
 break;
}

cout << "Inorder Traversal of the tree: ";
inorder (root);
cout << endl;
cout << "Preorder Traversal of the tree: ";
preorder (root);
cout << endl;
cout << "Postorder Traversal of the tree: ";
postorder (root);
cout << endl;
cout << "Exiting..." << endl;
}