

Written Assignment-1

1> Differentiate b/w linear and non-linear data structure.

Linear data structure

Non linear data structure

i> Elements are arranged sequentially

Elements are not arranged sequentially

ii> Each element is connected to its previous and next element

Elements can have multiple connections with other elements or may have none at all.

iii> Elements can be traversed in a single run since they are connected sequentially

Multiple runs may be required to traverse all elements of this data structure.

iv> They can be stored in memory easily, as memory itself is accessed sequentially.

Efficient methods of storing these data structures in memory must be developed.

v> All elements are present on a single level.

Elements may be present on multiple levels.

vi> Eg: Array, list, queue, stack

Eg: Graph, map, tree.

② Explain ADT of stack.

A stack is a ADT (Abstract Data type) which stores a collection of items. In a stack items can only be added and removed from one end, thus a stack follows LIFO principle.

It is based on stacks in real life, for example, a stack of books or plates. The insertion operation is called 'push' and the deletion operation is called 'pop'.

Other operations:

1. peek - Returns the element at the top of stack, without removing it.
2. isFull - Checks if stack is full or not.
3. isEmpty - Checks if stack is empty or not.
4. display - Shows all the elements in the stack.

To implement stack ADT, we need to use an array to hold the items and two variables: TOP and MAX. TOP stores the index of the element at top and MAX stores capacity. Then, the operations mentioned above are implemented as follows:

- 1> push(element):
 1. If stack is full, print "overflow"
 2. Otherwise set $TOP = TOP + 1$ & $array[TOP] = element$.
- 2> pop():
 1. If stack is empty, print "Underflow"
 2. Otherwise return $array[TOP]$ and set $TOP = TOP - 1$
- 3> isFull(): If TOP is $MAX - 1$ return True otherwise return False.
- 4> isEmpty(): If $TOP = -1$ return True otherwise return False.

⑦ WAP to implement priority queue with the following operations:
 1> Insert 2> Delete 3> Search 4> Display

```
#include <iostream>
using namespace std;
```

```
#define N 1000
```

```
void enqueue
```

```
int queue[N];
```

```
int size = 0, rear = -1, front = -1;
```

```
void enqueue() {
```

```
    int val;
```

```
    if (rear == N-1)
```

```
        cout << "Queue Overflow" << endl;
```

```
    else {
```

```
        if (front == -1)
```

```
            front = 0;
```

```
        cout << "Insert the element in queue:" << endl;
```

```
        cin >> val;
```

```
        rear++;
```

```
        size++;
```

```
        queue[rear] = val;
```

```
    }
```

```
}
```

```
void dequeue() {
```

```
    if (front == -1 || front > rear) {
```

```
        cout << "Queue Underflow" << endl;
```

```
        return;
```

```
    }
```



```
else {
```

```
cout << "Element deleted from queue is: " << queue[front]
```

```
front++;
```

```
size--;
```

```
}
```

```
}
```

```
void search () {
```

```
int i;
```

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

```
cout << "Queue is empty" << endl;
```

```
else {
```

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

```
if (queue[i] == element)
```

```
cout << "Element found in queue with pos: " << i;
```

```
}
```

```
}
```

```
void Display () {
```

```
if (front == -1)
```

```
cout << "Queue is empty" << endl;
```

```
else {
```

```
cout << "Queue elements are: ";
```

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

```
cout << queue[i] << " ";
```

```
cout << endl;
```

```
}
```

```
}
```

```
int main() {
```

```
    int ch
```

```
    cout << " 1) Insert" << endl;
```

```
    cout << " 2) Delete" << endl;
```

```
    cout << " 3) Search" << endl;
```

```
    cout << " 4) Display" << endl;
```

```
    do { cout << "Enter your choice : " << endl
```

```
        cin >> ch;
```

```
        switch (ch) {
```

```
            case 1 : enqueue();
```

```
            break;
```

```
            case 2 : dequeue();
```

```
            break;
```

```
            case 3 : search();
```

```
            break;
```

```
            case 4 : Display();
```

```
            break;
```

```
            default : cout << "Invalid" << endl;
```

```
        }
```

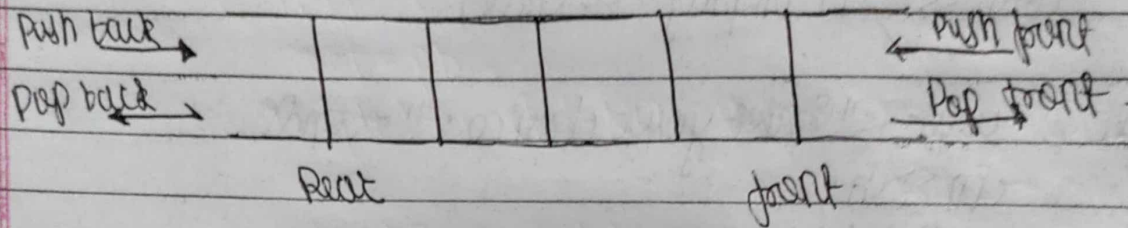
```
    } while (ch != 4);
```

```
    return 0;
```

```
}
```

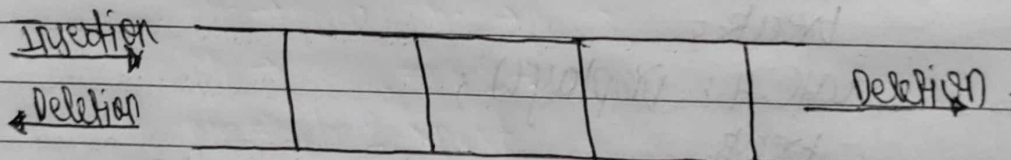

④ Write a short note on Double Ended Queue.

A Double ended queue is an ADT and is a generalized form of a queue in which elements can be added and removed from the front as well as the back.

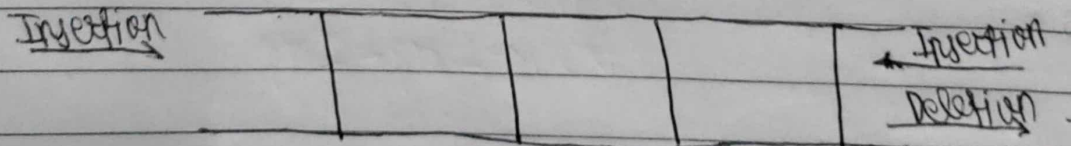


There are two types of deque:

- 1> Input restricted deque: In an input restricted deque, insertion can only be performed at one end but deletion can be performed at both ends.



- 2> Output restricted deque: In an output restricted deque, deletion can only be performed at one end, but insertion can be performed at both ends.



⑤ WAP to implement singly linked list application - Polynomial representation and addition.

```
#include <iostream>
using namespace std;
#include <malloc.h>
typedef struct node {
    int coeff, pow;
    struct node *next;
} NODE;
NODE* add (NODE* root 1, NODE* root 2, *input());
void display (NODE* root);
```

```
int main () {
    NODE* root 1, *root 2, *root;
    cout << "Enter details of polynomial 1 : \n";
    root 1 = input();
    cout << "Enter details of polynomial 2 : \n";
    root 2 = input();
    cout << "Polynomial 1 : "; display (root 1);
    cout << " \n Polynomial 2 : "; display (root 2);
    root = add (root 1, root 2);
    cout << " \n Addition of polynomials : "; display (root);
    return 0;
}
```

```
NODE *add (NODE* root 1, NODE* root 2) {
    NODE *ptr 1 = root 1, *ptr 2 = root 2, *ptr = NULL, *root = NULL;
    int i = 0;
```


Date

```
while (ptr → next != NULL || ptr2 → next != NULL) {
```

```
    if (ptr == NULL) {
```

```
        ptr = (NODE*) (malloc (sizeof (NODE)));
```

```
        ptr → next = NULL;
```

```
    }
```

```
    if (i == 0)
```

```
        root = ptr;
```

```
    if (ptr1 → next != NULL && (ptr2 → next == NULL || ptr1 → prev > ptr2 → prev)) {
```

```
        ptr → prev = ptr1 → prev;
```

```
        ptr → next = ptr1 → next;
```

```
        ptr1 = ptr1 → next;
```

```
    }
```

```
    else if (ptr2 → next != NULL && (ptr → next == NULL || ptr → prev < ptr2 → prev)) {
```

```
        ptr → prev = ptr2 → prev;
```

```
        ptr → next = ptr2 → next;
```

```
        ptr2 = ptr2 → next;
```

```
    }
```

```
    else {
```

```
        ptr → next = ptr1 → next + ptr2 → next;
```

```
        ptr → prev = ptr1 → prev;
```

```
        ptr1 = ptr1 → next;
```

```
        ptr2 = ptr2 → next;
```

```
    }
```

```
    ptr → next = (NODE*) (malloc (sizeof (NODE)));
```

```
    ptr = ptr → next;
```

```
    ptr → next = NULL;
```

```
    i++;
```

```
}
```

```
return root; }
```



```

NODE *input() {

```

NODE * root = NULL, * ptr = NULL;

774 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 8

cout << "Enter no. of terms in the polynomial: ";

$9n \gg n$

```
for (i=0; i<n; i++) {
```

cout << "Enter coefficient & power of term " << i << endl;

3. $\frac{1}{2} \ln \frac{1}{2} \gg \frac{1}{2} \ln \frac{1}{2} \gg \frac{1}{2} \ln \frac{1}{2}$

if (ptr == NULL) {

$ptr = (NODE *) malloc(sizeof(NODE));$

ptr \rightarrow result = NULL;

50

if (i == 0)

root = ptr;

$\text{ptr} \rightarrow \text{useff}_h = \text{useff}_h; \text{ptr} \rightarrow \text{paw} = \text{paw};$

```
ptr → next = (NODE *) (malloc (size of (NODE)));
```

```
pto => pto -> next;
```

ptr \rightarrow next = NULL;

3

zeker niet;

9

void display (NODE *root) {

Node #0 ptr = root,

```
while (ptr → next != NULL) {
```

cout << " " << ptr -> useful << " x^" << ptr -> piece;

ptr = ptr → result;

if (ptr \rightarrow next) \neq Null

$$\text{out} < \frac{1}{2} + 1$$

3

3