

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Ayush Ranjan (1BM23CS058)

**in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
September 2024-January 2025**

**B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering**



This is to certify that the Lab work entitled “**DATA STRUCTURES**” carried out by **Ayush Ranjan (1BM23CS058)**, who is Bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST)** work prescribed for the said degree.

Prof.Sneha S
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Sl. No.	Experiment Title	Page No.
1	Stack Implementation using Arrays	4
2	Infix Arithmetic Expression to Postfix Expression	9
3	Linear Queue using Arrays	11
	Circular Queue using Arrays	15
4	Singly Linked List – Creation, Insertion, Deletion & Display	19
5	Singly Linked List – Sorting, Reversal & Concatenation	29
6	Stacks using Linked List	39
	Queue using Linked List	45
7	Doubly Linked List -Creation, Insertion, deletion & Display	52
8	Binary Search Tree	59
9	Breadth First Search	64
10	Depth First Search	68

Course outcomes:

CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyze data structure operations for a given problem
CO3	Design and develop solutions using the operations of linear and nonlinear data structure for a given specification.
CO4	Conduct practical experiments for demonstrating the operations of different data structures.

Lab program 1:

Write a program to simulate the working of stack using an array with the following: a) Push b) Pop c) Display The program should print appropriate messages for stack overflow, stack underflow

```
#include<stdio.h>

#include<stdlib.h>

#define SIZE 5

int stack[SIZE];

int top=-1;

void push(int a)
{
    if(top==SIZE-1)
    {
        printf("\nStack is full,overflow condition");
    }
    else
    {
        top++;
        stack[top]=a;
        printf("\nElement successfully pushed to stack");
    }
}

void pop()
{
    if(top== -1)
    {
        printf("\nStack is empty,underflow condition");
    }
    else
    {

```

```

        int ele = stack[top];

        printf("\nElement %d has been successfully popped",ele);

        top--;
    }
}

void display()
{
    if(top== -1)
    {
        printf("\nstack is empty,underflow condition");
    }
    else
    {
        for(int i=top;i>-1;i--)
        {
            printf("%d ",stack[i]);
        }
    }
}

void main()
{
    int c,e;

    while(1)
    {
        printf("\n\n1.push\n2.pop\n3.display\n4.exit\nEnter :");

        scanf("%d",&c);

        switch (c)
        {
            case 1: printf("\nEnter the element to push ");

                    scanf("%d",&e);

```

```

        push(e);

        break;

    case 2: pop();

        break;

    case 3: display();

        break;

    case 4: exit(1);

    default : printf("\nInvalid input");

}

}

}

```

```

C:\Users\adithy\Documents> gcc stack.c -o stack
C:\Users\adithy\Documents> .\stack
Enter the element to push: 1
Element successfully added to stack
1
Enter
2
Pop
Element successfully popped from stack
Enter
3
Display
1
Enter
4
Exit the stack
Enter the element to push: 2
Element successfully added to stack
1
2
Enter
5
Pop
Element successfully popped from stack
Enter
6
Display
1
2
Enter
7
Exit the stack
Enter the element to push: 3
Element successfully added to stack
1
2
3
Enter
8
Pop
Element successfully popped from stack
Enter
9
Display
1
2
3
Enter
10
Exit the stack
Enter the element to push: 4
Element successfully added to stack
1
2
3
4
Enter
11
Pop
Element successfully popped from stack
Enter
12
Display
1
2
3
4
Enter
13
Exit the stack
Enter the element to push: 5
Element successfully added to stack
1
2
3
4
5
Enter
14
Pop
Element successfully popped from stack
Enter
15
Display
1
2
3
4
5
Enter
16
Exit the stack

```

```

Enter the element to push 5
Stack is full,insertion condition
1.push
2.pop
3.display
4.exit
Enter 1:

Element 5 has been successfully pushed
1.push
2.pop
3.display
4.exit
Enter 1:

Element 4 has been successfully popped
1.push
2.pop
3.display
4.exit
Enter 1:

Element 5 has been successfully popped
1.push
2.pop
3.display
4.exit
Enter 1:

Element 2 has been successfully popped
1.push
2.pop
3.display
4.exit
Enter 1:

Element 1 has been successfully popped
1.push
2.pop
3.display
4.exit
Enter 1:

Stack is empty,insertion condition
1.push
2.pop
3.display
4.exit
Enter 1:

Enter the element to push 1

```

```

Element successfully pushed to stack
1.push
2.pop
3.display
4.exit
Enter 1:

Enter the element to push 2

Element successfully pushed to stack
1.push
2.pop
3.display
4.exit
Enter 1:

Enter the element to push 1

Element successfully pushed to stack
1.push
2.pop
3.display
4.exit
Enter 1:

Enter the element to push 4

Element successfully pushed to stack
1.push
2.pop
3.display
4.exit
Enter 1:

Enter the element to push 5

Element successfully pushed to stack
1.push
2.pop
3.display
4.exit
Enter 1:

Enter the element to push 4

Stack is full,insertion condition
1.push
2.pop
3.display
4.exit
Enter 1:

1.push
2.pop

```

Element 4 has been successfully popped

```
1.push  
2.pop  
3.display  
4.exit  
Enter 12
```

Element 4 has been successfully popped

```
1.push  
2.pop  
3.display  
4.exit  
Enter 12
```

Element 3 has been successfully popped

```
1.push  
2.pop  
3.display  
4.exit  
Enter 12
```

Element 2 has been successfully popped

```
1.push  
1.push  
1.push  
2.pop  
3.display  
4.exit  
Enter 12
```

```
4.exit  
Enter 12
```

```
Enter 12
```

Element 1 has been successfully popped

```
1.push  
1.push  
2.pop  
3.display  
4.exit  
Enter 12
```

Stack is empty,underflow condition

```
1.push  
2.pop  
3.display  
4.exit  
Enter 13  
3.display  
4.exit  
Enter 13  
Enter 13
```

HOMEWORKS QUESTIONS DEMO CODES TERMINAL FOR3

```
4 exit  
Enter 12
```

```
Enter 12
```

Element 1 has been successfully popped

```
1.push  
1.push  
1.pop  
3.display  
4.exit  
Enter 12
```

Stack is empty,underflow condition

```
1.push  
2.pop  
3.display  
4.exit  
Enter 13  
3.display  
4.exit  
Enter 13  
Enter 13
```

Stack is empty,underflow condition

```
1.push  
2.pop  
3.display  
4.exit  
Enter 16
```

Enter 14 input

```
1.push  
2.push  
3.display  
4.exit  
Enter 14  
PS C:\Users\ADMINI~1\Documents>
```


Lab program 2

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)

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

int index1 = 0, pos = 0, top = -1, length;
char symbol, temp, infix[20], postfix[20], stack[20];
void infixToPostfix();
void push(char symbol);
char pop();
int pred(char symbol);
int main() {
    printf("Enter infix expression:\n");
    scanf("%s", infix);
    infixToPostfix();
    printf("\nInfix expression: %s", infix);
    printf("\nPostfix expression: %s\n", postfix);
    return 0;
}
void infixToPostfix() {
    length = strlen(infix);
    push('#'); // Push an initial dummy character to the stack
    while (index1 < length) {
        symbol = infix[index1];
        switch (symbol) {
            case '(':
                push(symbol);
                break;
            case ')':
                temp = pop();
                while (temp != '(') {
                    postfix[pos++] = temp;
                    temp = pop();
                }
                break;
            case '+':
            case '-':
            case '*':
            case '/':
            case '^':
                while (pred(stack[top]) >= pred(symbol)) {
                    temp = pop();
                    postfix[pos++] = temp;
                }
                push(symbol);
                break;
            default:
                postfix[pos++] = symbol;
        }
        index1++;
    }
    while (top > 0) {
        temp = pop();
        postfix[pos++] = temp;
    }
    postfix[pos] = '\0';
}
```

```

}

void push(char symbol) {
    top = top + 1;
    stack[top] = symbol;
}

char pop() {
    char symb;
    symb = stack[top];
    top = top - 1;
    return symb;
}

int pred(char symbol) {
    int p;
    switch (symbol) {
        case '^':
            p = 3;
            break;
        case '*':
        case '/':
            p = 2;
            break;
        case '+':
        case '-':
            p = 1;
            break;
        case '(':
            p = 0;
            break;
        case '#':
            p = -1;
            break;
        default:
            p = -1;
    }
    return p;
}

```

Output:

```

Enter infix expression:
7-8+(6-8)*11

Infix expression: 7-8+(6-8)*11
Postfix expression: 78-68-11*+

```

Lab Program 3

- a) WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display The program should print appropriate messages for queue empty and queue overflow conditions

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

```
#define SIZE 3
```

```
int queue[SIZE];
```

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

```
int is_full() {
```

```
    return (rear == SIZE - 1);
```

```
}
```

```
int is_empty() {
```

```
    return (front == -1);
```

```
}
```

```
void insert(int value) {
```

```
    if (is_full()) {
```

```
        printf("Queue Overflow\n");
```

```
        return;
```

```
    }
```

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

```
        front = 0;
```

```
    queue[++rear] = value;
```

```
    printf("Inserted %d into the queue.\n", value);
```

```
}
```

```
void delete() {  
    if (is_empty()) {  
        printf("Queue Underflow.\n");  
        return;  
    }  
    printf("Deleted %d from the queue.\n", queue[front]);  
    front++;  
    if (front > rear) {  
        front = -1;  
        rear = -1;  
    }  
}
```

```
void display() {  
    if (is_empty()) {  
        printf("Queue is empty!\n");  
        return;  
    }  
    printf("Queue elements: ");  
    for (int i = front; i <= rear; i++) {  
        printf("%d ", queue[i]);  
    }  
    printf("\n");  
}
```

```
int main() {
```

```

int choice, value;

printf("\nQueue Operations:\n");

printf("1. Insert\n");

printf("2. Delete\n");

printf("3. Display\n");

printf("4. Exit\n");

while (1) {

    printf("Enter your choice: ");

    scanf("%d", &choice);


    switch (choice) {

        case 1:

            printf("Enter the value to insert: ");

            scanf("%d", &value);

            insert(value);

            break;

        case 2:

            delete();

            break;

        case 3:

            display();

            break;

        case 4:

            printf("Exiting...\n");

            return 0;

        default:

            printf("Invalid choice! Please try again.\n");

    }

}

}

```

```

C:\Programs\X < C:\Programs\X
C:\Programs\X > cd C:\Programs\X
1. #include <stdio.h>
2. #include <stdlib.h>
3.
4. int queue[100];
5. int front = 0, rear = 0;
6.
7.
8. int isFull() {
9.     return (rear == 100);
10. }
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.

```

Queue Operations:

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice: 1

Queue Underflow.

Enter your choice: 2

Queue is empty.

Enter your choice: 3

Enter the value to insert: 1

Inserted 1 into the queue.

Enter your choice: 3

Enter the value to insert: 2

Inserted 2 into the queue.

Enter your choice: 3

Enter the value to insert: 3

Inserted 3 into the queue.

Enter your choice: 3

Enter the value to insert: 4

Inserted 4 into the queue.

Enter your choice: 3

Enter the value to insert: 5

Inserted 5 into the queue.

Queue Overflow.

Enter your choice: 2

Queue elements: 1 2 3 4 5

Enter your choice: 2

Deleted 1 from the queue.

Enter your choice: 3

Queue elements: 2 3 4 5

Enter your choice: 2

Deleted 2 from the queue.

Enter your choice: 3

Queue elements: 3 4 5

Enter your choice: 2

Deleted 3 from the queue.

Enter your choice: 3

Queue elements: 4 5

Enter your choice: 2

Deleted 4 from the queue.

Enter your choice: 3

Queue elements: 5

Enter your choice: 2

Deleted 5 from the queue.

Enter your choice: 3

Queue is empty.

Enter your choice: 4

Exiting...

b) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display The program should print appropriate messages for queue empty and queue overflow conditions

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

```
#define SIZE 3
```

```
int queue[SIZE];
```

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

```
int is_full() {
```

```
    return (front == (rear + 1) % SIZE);
```

```
}
```

```
int is_empty() {
```

```
    return (front == -1);
```

```
}
```

```
void insert(int n) {
```

```
    if (is_full()) {
```

```
        printf("Queue Overflow\n");
```

```
        return;
```

```
    }
```

```
    if (is_empty())
```

```
        front = 0;
```

```
        rear = 0;
```

```
    else
```

```
        rear = (rear + 1) % SIZE;
```

```
queue[rear] = n;
printf("Element %d inserted.\n", n);
}
```

```
void delete() {
    if (is_empty()) {
        printf("Queue Underflow.\n");
        return;
    }
    printf("Element %d deleted\n", queue[front]);
    if (front == rear){
        front = -1;
        rear = -1;
    }
    else
        front = (front + 1) % SIZE;
}
```

```
void display() {
    if (is_empty()) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements: ");
    int i = front;
    while (1) {
        printf("%d ", queue[i]);
        if (i == rear)
```



```

        break;
    i = (i + 1) % SIZE;
}
printf("\n");
}

int main() {
    int choice, value;

    printf("\nCircular Queue Operations:\n");
    printf("1. Insert\n");
    printf("2. Delete\n");
    printf("3. Display\n");
    printf("4. Exit\n");
    while (1) {
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &value);
                insert(value);
                break;
            case 2:
                delete();
                break;
            case 3:
                display();
                break;
            case 4:

```

```

        printf("Exiting...\n");

        return 0;

    default:

        printf("Invalid choice! Please try again.\n");

    }

}

}

```

```

C:\Users\Aditya> g++ 2022.cpp -std=c++11
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
20. 20
21. 21
22. 22
23. 23
24. 24
25. 25
26. 26
27. 27
28. 28
29. 29
30. 30
31. 31
32. 32
33. 33
34. 34
35. 35
36. 36
37. 37
38. 38
39. 39
40. 40
41. 41
42. 42
43. 43
44. 44
45. 45
46. 46
47. 47
48. 48
49. 49
50. 50
51. 51
52. 52
53. 53
54. 54
55. 55
56. 56
57. 57
58. 58
59. 59
60. 60
61. 61
62. 62
63. 63
64. 64
65. 65
66. 66
67. 67
68. 68
69. 69
70. 70
71. 71
72. 72
73. 73
74. 74
75. 75
76. 76
77. 77
78. 78
79. 79
80. 80
81. 81
82. 82
83. 83
84. 84
85. 85
86. 86
87. 87
88. 88
89. 89
90. 90
91. 91
92. 92
93. 93
94. 94
95. 95
96. 96
97. 97
98. 98
99. 99
100. 100
101. 101
102. 102
103. 103
104. 104
105. 105
106. 106
107. 107
108. 108
109. 109
110. 110
111. 111
112. 112
113. 113
114. 114
115. 115
116. 116
117. 117
118. 118
119. 119
120. 120
121. 121
122. 122
123. 123
124. 124
125. 125
126. 126
127. 127
128. 128
129. 129
130. 130
131. 131
132. 132
133. 133
134. 134
135. 135
136. 136
137. 137
138. 138
139. 139
140. 140
141. 141
142. 142
143. 143
144. 144
145. 145
146. 146
147. 147
148. 148
149. 149
150. 150
151. 151
152. 152
153. 153
154. 154
155. 155
156. 156
157. 157
158. 158
159. 159
160. 160
161. 161
162. 162
163. 163
164. 164
165. 165
166. 166
167. 167
168. 168
169. 169
170. 170
171. 171
172. 172
173. 173
174. 174
175. 175
176. 176
177. 177
178. 178
179. 179
180. 180
181. 181
182. 182
183. 183
184. 184
185. 185
186. 186
187. 187
188. 188
189. 189
190. 190
191. 191
192. 192
193. 193
194. 194
195. 195
196. 196
197. 197
198. 198
199. 199
200. 200
201. 201
202. 202
203. 203
204. 204
205. 205
206. 206
207. 207
208. 208
209. 209
210. 210
211. 211
212. 212
213. 213
214. 214
215. 215
216. 216
217. 217
218. 218
219. 219
220. 220
221. 221
222. 222
223. 223
224. 224
225. 225
226. 226
227. 227
228. 228
229. 229
230. 230
231. 231
232. 232
233. 233
234. 234
235. 235
236. 236
237. 237
238. 238
239. 239
240. 240
241. 241
242. 242
243. 243
244. 244
245. 245
246. 246
247. 247
248. 248
249. 249
250. 250
251. 251
252. 252
253. 253
254. 254
255. 255
256. 256
257. 257
258. 258
259. 259
260. 260
261. 261
262. 262
263. 263
264. 264
265. 265
266. 266
267. 267
268. 268
269. 269
270. 270
271. 271
272. 272
273. 273
274. 274
275. 275
276. 276
277. 277
278. 278
279. 279
280. 280
281. 281
282. 282
283. 283
284. 284
285. 285
286. 286
287. 287
288. 288
289. 289
290. 290
291. 291
292. 292
293. 293
294. 294
295. 295
296. 296
297. 297
298. 298
299. 299
300. 300
301. 301
302. 302
303. 303
304. 304
305. 305
306. 306
307. 307
308. 308
309. 309
310. 310
311. 311
312. 312
313. 313
314. 314
315. 315
316. 316
317. 317
318. 318
319. 319
320. 320
321. 321
322. 322
323. 323
324. 324
325. 325
326. 326
327. 327
328. 328
329. 329
330. 330
331. 331
332. 332
333. 333
334. 334
335. 335
336. 336
337. 337
338. 338
339. 339
340. 340
341. 341
342. 342
343. 343
344. 344
345. 345
346. 346
347. 347
348. 348
349. 349
350. 350
351. 351
352. 352
353. 353
354. 354
355. 355
356. 356
357. 357
358. 358
359. 359
360. 360
361. 361
362. 362
363. 363
364. 364
365. 365
366. 366
367. 367
368. 368
369. 369
370. 370
371. 371
372. 372
373. 373
374. 374
375. 375
376. 376
377. 377
378. 378
379. 379
380. 380
381. 381
382. 382
383. 383
384. 384
385. 385
386. 386
387. 387
388. 388
389. 389
390. 390
391. 391
392. 392
393. 393
394. 394
395. 395
396. 396
397. 397
398. 398
399. 399
400. 400
401. 401
402. 402
403. 403
404. 404
405. 405
406. 406
407. 407
408. 408
409. 409
410. 410
411. 411
412. 412
413. 413
414. 414
415. 415
416. 416
417. 417
418. 418
419. 419
420. 420
421. 421
422. 422
423. 423
424. 424
425. 425
426. 426
427. 427
428. 428
429. 429
430. 430
431. 431
432. 432
433. 433
434. 434
435. 435
436. 436
437. 437
438. 438
439. 439
440. 440
441. 441
442. 442
443. 443
444. 444
445. 445
446. 446
447. 447
448. 448
449. 449
450. 450
451. 451
452. 452
453. 453
454. 454
455. 455
456. 456
457. 457
458. 458
459. 459
460. 460
461. 461
462. 462
463. 463
464. 464
465. 465
466. 466
467. 467
468. 468
469. 469
470. 470
471. 471
472. 472
473. 473
474. 474
475. 475
476. 476
477. 477
478. 478
479. 479
480. 480
481. 481
482. 482
483. 483
484. 484
485. 485
486. 486
487. 487
488. 488
489. 489
490. 490
491. 491
492. 492
493. 493
494. 494
495. 495
496. 496
497. 497
498. 498
499. 499
500. 500
501. 501
502. 502
503. 503
504. 504
505. 505
506. 506
507. 507
508. 508
509. 509
510. 510
511. 511
512. 512
513. 513
514. 514
515. 515
516. 516
517. 517
518. 518
519. 519
520. 520
521. 521
522. 522
523. 523
524. 524
525. 525
526. 526
527. 527
528. 528
529. 529
530. 530
531. 531
532. 532
533. 533
534. 534
535. 535
536. 536
537. 537
538. 538
539. 539
540. 540
541. 541
542. 542
543. 543
544. 544
545. 545
546. 546
547. 547
548. 548
549. 549
550. 550
551. 551
552. 552
553. 553
554. 554
555. 555
556. 556
557. 557
558. 558
559. 559
560. 560
561. 561
562. 562
563. 563
564. 564
565. 565
566. 566
567. 567
568. 568
569. 569
570. 570
571. 571
572. 572
573. 573
574. 574
575. 575
576. 576
577. 577
578. 578
579. 579
580. 580
581. 581
582. 582
583. 583
584. 584
585. 585
586. 586
587. 587
588. 588
589. 589
590. 590
591. 591
592. 592
593. 593
594. 594
595. 595
596. 596
597. 597
598. 598
599. 599
600. 600
601. 601
602. 602
603. 603
604. 604
605. 605
606. 606
607. 607
608. 608
609. 609
610. 610
611. 611
612. 612
613. 613
614. 614
615. 615
616. 616
617. 617
618. 618
619. 619
620. 620
621. 621
622. 622
623. 623
624. 624
625. 625
626. 626
627. 627
628. 628
629. 629
630. 630
631. 631
632. 632
633. 633
634. 634
635. 635
636. 636
637. 637
638. 638
639. 639
640. 640
641. 641
642. 642
643. 643
644. 644
645. 645
646. 646
647. 647
648. 648
649. 649
650. 650
651. 651
652. 652
653. 653
654. 654
655. 655
656. 656
657. 657
658. 658
659. 659
660. 660
661. 661
662. 662
663. 663
664. 664
665. 665
666. 666
667. 667
668. 668
669. 669
670. 670
671. 671
672. 672
673. 673
674. 674
675. 675
676. 676
677. 677
678. 678
679. 679
680. 680
681. 681
682. 682
683. 683
684. 684
685. 685
686. 686
687. 687
688. 688
689. 689
690. 690
691. 691
692. 692
693. 693
694. 694
695. 695
696. 696
697. 697
698. 698
699. 699
700. 700
701. 701
702. 702
703. 703
704. 704
705. 705
706. 706
707. 707
708. 708
709. 709
710. 710
711. 711
712. 712
713. 713
714. 714
715. 715
716. 716
717. 717
718. 718
719. 719
720. 720
721. 721
722. 722
723. 723
724. 724
725. 725
726. 726
727. 727
728. 728
729. 729
730. 730
731. 731
732. 732
733. 733
734. 734
735. 735
736. 736
737. 737
738. 738
739. 739
740. 740
741. 741
742. 742
743. 743
744. 744
745. 745
746. 746
747. 747
748. 748
749. 749
750. 750
751. 751
752. 752
753. 753
754. 754
755. 755
756. 756
757. 757
758. 758
759. 759
760. 760
761. 761
762. 762
763. 763
764. 764
765. 765
766. 766
767. 767
768. 768
769. 769
770. 770
771. 771
772. 772
773. 773
774. 774
775. 775
776. 776
777. 777
778. 778
779. 779
780. 780
781. 781
782. 782
783. 783
784. 784
785. 785
786. 786
787. 787
788. 788
789. 789
790. 790
791. 791
792. 792
793. 793
794. 794
795. 795
796. 796
797. 797
798. 798
799. 799
800. 800
801. 801
802. 802
803. 803
804. 804
805. 805
806. 806
807. 807
808. 808
809. 809
810. 810
811. 811
812. 812
813. 813
814. 814
815. 815
816. 816
817. 817
818. 818
819. 819
820. 820
821. 821
822. 822
823. 823
824. 824
825. 825
826. 826
827. 827
828. 828
829. 829
830. 830
831. 831
832. 832
833. 833
834. 834
835. 835
836. 836
837. 837
838. 838
839. 839
840. 840
841. 841
842. 842
843. 843
844. 844
845. 845
846. 846
847. 847
848. 848
849. 849
850. 850
851. 851
852. 852
853. 853
854. 854
855. 855
856. 856
857. 857
858. 858
859. 859
860. 860
861. 861
862. 862
863. 863
864. 864
865. 865
866. 866
867. 867
868. 868
869. 869
870. 870
871. 871
872. 872
873. 873
874. 874
875. 875
876. 876
877. 877
878. 878
879. 879
880. 880
881. 881
882. 882
883. 883
884. 884
885. 885
886. 886
887. 887
888. 888
889. 889
890. 890
891. 891
892. 892
893. 893
894. 894
895. 895
896. 896
897. 897
898. 898
899. 899
900. 900
901. 901
902. 902
903. 903
904. 904
905. 905
906. 906
907. 907
908. 908
909. 909
910. 910
911. 911
912. 912
913. 913
914. 914
915. 915
916. 916
917. 917
918. 918
919. 919
920. 920
921. 921
922. 922
923. 923
924. 924
925. 925
926. 926
927. 927
928. 928
929. 929
930. 930
931. 931
932. 932
933. 933
934. 934
935. 935
936. 936
937. 937
938. 938
939. 939
940. 940
941. 941
942. 942
943. 943
944. 944
945. 945
946. 946
947. 947
948. 948
949. 949
950. 950
951. 951
952. 952
953. 953
954. 954
955. 955
956. 956
957. 957
958. 958
959. 959
960. 960
961. 961
962. 962
963. 963
964. 964
965. 965
966. 966
967. 967
968. 968
969. 969
970. 970
971. 971
972. 972
973. 973
974. 974
975. 975
976. 976
977. 977
978. 978
979. 979
980. 980
981. 981
982. 982
983. 983
984. 984
985. 985
986. 986
987. 987
988. 988
989. 989
990. 990
991. 991
992. 992
993. 993
994. 994
995. 995
996. 996
997. 997
998. 998
999. 999
1000. 1000

```

Lab Program 4

WAP to Implement Singly Linked List with following operations

a) Create LinkedList.

b) Insertion of a node at first position, at any position and at end of list.

c) Deletion of first element, specified element and last element in the list.

Display the contents of the linked list.

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

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

```
struct Node {
```

```
    int data;
```

```
    struct Node* next;
```

```
};
```

```
// Create a new node
```

```
struct Node* create_node(int data) {
```

```
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
```

```
    new_node->data = data;
```

```
    new_node->next = NULL;
```

```
    return new_node;
```

```
}
```

```
// Insert node at the beginning
```

```
void insert_at_beginning(struct Node** head, int data) {
```

```
    struct Node* new_node = create_node(data);
```

```
    new_node->next = *head;
```

```
    *head = new_node;
```

```
}
```

```

// Insert node at the end
void insert_at_end(struct Node** head, int data) {
    struct Node* new_node = create_node(data);
    if (*head == NULL) {
        *head = new_node;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = new_node;
}

// Insert node at a specific position
void insert_at_position(struct Node** head, int data, int position) {
    if (position < 0) return; // Invalid position
    if (position == 0) {
        insert_at_beginning(head, data);
        return;
    }
    struct Node* new_node = create_node(data);
    struct Node* temp = *head;
    for (int i = 0; i < position - 1; i++) {
        if (temp == NULL) return; // Position out of range
        temp = temp->next;
    }
    new_node->next = temp->next;
    temp->next = new_node;
}

```

// Delete node at the beginning

```
void delete_at_beginning(struct Node** head) {  
    if (*head != NULL) {  
        struct Node* temp = *head;  
        *head = (*head)->next;  
        free(temp);  
    }  
}
```

// Delete node at the end

```
void delete_at_end(struct Node** head) {  
    if (*head == NULL) return;  
    if ((*head)->next == NULL) {  
        free(*head);  
        *head = NULL;  
        return;  
    }  
    struct Node* temp = *head;  
    while (temp->next && temp->next->next != NULL) {  
        temp = temp->next;  
    }  
    free(temp->next);  
    temp->next = NULL;  
}
```

// Delete node with a specific key

```
void delete_at_key(struct Node** head, int key) {  
    if (*head == NULL) return;  
    if ((*head)->data == key) {
```

```

    struct Node* temp = *head;

    *head = (*head)->next;

    free(temp);

    return;
}

struct Node* temp = *head;

while (temp->next != NULL && temp->next->data != key) {

    temp = temp->next;

}

if (temp->next == NULL) return;

struct Node* to_delete = temp->next;

temp->next = temp->next->next;

free(to_delete);

}

// Delete node before the key

void delete_before_key(struct Node** head, int key) {

    if (*head == NULL || (*head)->next == NULL) return;

    if ((*head)->next->data == key) {

        struct Node* temp = *head;

        *head = (*head)->next;

        free(temp);

        return;

    }

    struct Node* temp = *head;

    while (temp->next != NULL && temp->next->next != NULL) {

        if (temp->next->next->data == key) {

            struct Node* to_delete = temp->next;

            temp->next = temp->next->next;

            free(to_delete);

        }

        temp = temp->next;

    }

}

```

```

        return;
    }

    temp = temp->next;
}
}

```

// Delete node after the key

```

void delete_after_key(struct Node** head, int key) {
    struct Node* temp = *head;
    while (temp != NULL && temp->data != key) {
        temp = temp->next;
    }
    if (temp != NULL && temp->next != NULL) {
        struct Node* to_delete = temp->next;
        temp->next = temp->next->next;
        free(to_delete);
    }
}

```

// Display the list

```

void display(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

```

// Free all nodes to avoid memory leaks

```

void free_list(struct Node** head) {
    struct Node* temp;
    while (*head != NULL) {
        temp = *head;
        *head = (*head)->next;
        free(temp);
    }
}

int main() {
    struct Node* head = NULL;

    int data, key;

    printf("Choice :
\n1.insert_at_beginning\n2.insert_at_end\n3.insert_at_position\n4.delete_at_beginning\n5.delete
_at_end\n6.delete_at_key\n7.delete_before_key\n8.delete_after_key\n9.display\n10.exit\n");

    int c;
    while (1) {
        printf("Enter choice: ");
        scanf("%d", &c);
        switch (c) {
            case 1:
                printf("Enter the data: ");
                scanf("%d", &data);
                insert_at_beginning(&head, data);
                break;
            case 2:
                printf("Enter the data: ");
                scanf("%d", &data);
                insert_at_end(&head, data);

```



```

        break;
case 3:
    printf("Enter the data and position: ");
    scanf("%d%d", &data, &key);
    insert_at_position(&head, data, key);
    break;
case 4:
    delete_at_beginning(&head);
    break;
case 5:
    delete_at_end(&head);
    break;
case 6:
    printf("Enter the key to delete: ");
    scanf("%d", &key);
    delete_at_key(&head, key);
    break;
case 7:
    printf("Enter the key to delete before: ");
    scanf("%d", &key);
    delete_before_key(&head, key);
    break;
case 8:
    printf("Enter the key to delete after: ");
    scanf("%d", &key);
    delete_after_key(&head, key);
    break;
case 9:
    display(head);
    break;

```

```
    case 10:  
        exit(0);  
    default:  
        printf("Invalid choice...\n");  
    }  
}  
return 0;  
}
```

```

Choice :
1.insert at beginning
2.insert_at_end
3.insert_at_position
4.delete at beginning
5.delete_at_end
6.delete_at_key
7.delete before key
6.delete_at_key
7.delete_before_key
8.delete after key
9.display
9.display
10.exit
10.exit
Enter choice:1
Enter choice:1
Enter the data : 1
Enter choice:2
Enter the data : 1
Enter choice:2
Enter the data : 2
Enter the data : 2
Enter choice:2
Enter the data : 3
Enter choice:2
Enter the data : 3
Enter choice:2
Enter the data : 4
Enter the data : 4
Enter choice:2
Enter choice:2
Enter the data : 5
Enter the data : 5
Enter choice:2
Enter the data : 6
Enter choice:2
Enter the data : 6
Enter choice:3
Enter choice:3
Enter the data and position : 3
4
Enter the data and position : 3
4
Enter choice:9
1 -> 2 -> 3 -> 4 -> 3 -> 5 -> 6 -> NULL
Enter choice:4
Enter choice:9
1 -> 2 -> 3 -> 4 -> 3 -> 5 -> 6 -> NULL
Enter choice:4
Enter choice:9
2 -> 3 -> 4 -> 3 -> 5 -> 6 -> NULL
Enter choice:9
2 -> 3 -> 4 -> 3 -> 5 -> 6 -> NULL
Enter choice:5
Enter choice:5

```

```

Enter choice:2
Enter choice:2
Enter the data : 5
Enter the data : 5
Enter choice:2
Enter the data : 6
Enter choice:2
Enter the data : 6
Enter choice:3
Enter choice:3
Enter the data and position : 1
4
Enter the data and position : 1
4
Enter choice:9
1 -> 2 -> 3 -> 4 -> 3 -> 5 -> 6 -> NULL
Enter choice:4
Enter choice:9
1 -> 2 -> 3 -> 4 -> 3 -> 5 -> 6 -> NULL
Enter choice:4
Enter choice:9
2 -> 3 -> 4 -> 3 -> 5 -> 6 -> NULL
Enter choice:9
2 -> 3 -> 4 -> 3 -> 5 -> 6 -> NULL
Enter choice:5
Enter choice:5
Enter choice:9
Enter choice:9
2 -> 3 -> 4 -> 3 -> 5 -> NULL
2 -> 3 -> 4 -> 3 -> 5 -> NULL
Enter choice:7
Enter choice:7
Enter the key to delete before :4
Enter the key to delete before :4
Enter choice:9
Enter choice:9
2 -> 3 -> 4 -> 3 -> 5 -> NULL
2 -> 3 -> 4 -> 3 -> 5 -> NULL
Enter choice:6
Enter choice:6
Enter the key to delete :3
Enter choice:9
2 -> 4 -> 3 -> 5 -> NULL
Enter choice:8
Enter the key to delete after :2
Enter choice:9
2 -> 3 -> 5 -> NULL
Enter choice:

```

Lab Program 5

WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.

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

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

```
struct Node {
```

```
    int data;
```

```
    struct Node* next;
```

```
};
```

```
struct Node* insertAtEnd(struct Node* head, int value) {
```

```
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
```

```
    newNode->data = value;
```

```
    newNode->next = NULL;
```

```
    if (!head) return newNode;
```

```
    struct Node* temp = head;
```

```
    while (temp->next) temp = temp->next;
```

```
    temp->next = newNode;
```

```
    return head;
```

```
}
```

```
void printList(struct Node* head) {
```

```
    while (head) {
```

```
        printf("%d -> ", head->data);
```

```
        head = head->next;
```

```
    }
```

```
    printf("NULL\n");
```

```
}
```

```

struct Node* sortList(struct Node* head) {
    if (!head || !head->next) return head;

    struct Node* current = head;
    while (current) {
        struct Node* next = current->next;
        while (next) {
            if (current->data > next->data) {
                int temp = current->data;
                current->data = next->data;
                next->data = temp;
            }
            next = next->next;
        }
        current = current->next;
    }
    return head;
}

```

```

int main() {
    struct Node* head = NULL;

    int choice, value;

    do {
        printf("\n1. Insert\n2. Sort\n3. Display\n4. Exit\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:

```

```

        printf("Enter value to insert: ");
        scanf("%d", &value);
        head = insertAtEnd(head, value);
        break;
    case 2:
        head = sortList(head);
        printf("List sorted.\n");
        break;
    case 3:
        printf("Linked list: ");
        printList(head);
        break;
    case 4:
        printf("Exiting program.\n");
        break;
    default:
        printf("Invalid choice. Try again.\n");
    }
} while (choice != 4);

return 0;
}

```

```

1. Insert
2. Sort
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 4

1. Insert
2. Sort
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 2

1. Insert
2. Sort
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 7

1. Insert
2. Sort
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 1

1. Insert
2. Sort
3. Display
4. Exit
Enter your choice: 4
Linked list: 4 -> 2 -> 7 -> 1 -> NULL

1. Insert
2. Sort
3. Display
4. Exit
Enter your choice: 2
List sorted.

1. Insert
2. Sort
3. Display
4. Exit
Enter your choice: 3
Linked list: 1 -> 2 -> 4 -> 7 -> NULL

1. Insert
2. Sort
3. Display
4. Exit
Enter your choice: 4
Exiting program.

```



```

#include <stdio.h>

#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};

struct Node* insertAtEnd(struct Node* head, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    if (!head) return newNode;

    struct Node* temp = head;
    while (temp->next) temp = temp->next;
    temp->next = newNode;
    return head;
}

void printList(struct Node* head) {
    while (head) {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

```

```

struct Node* reverseList(struct Node* head) {
    struct Node* prev = NULL;
    struct Node* current = head;
    struct Node* next = NULL;

    while (current) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}

int main() {
    struct Node* head = NULL;
    int value;

    printf("Enter values to create a linked list (-1 to stop): ");
    do {
        scanf("%d", &value);
        if (value != -1) head = insertAtEnd(head, value);
    } while (value != -1);

    printf("Original List: ");
    printList(head);

    head = reverseList(head);

    printf("Reversed List: ");

```

```
printList(head);
```

```
return 0;
```

```
}
```

```
Enter values to create a linked list (-1 to stop): 1 2 3 4 5 -1
Original List: 1 > 2 > 3 > 4 > 5 > NULL
Reversed List: 5 > 4 > 3 > 2 > 1 > NULL
```

```

#include <stdio.h>

#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};

struct Node* insertAtEnd(struct Node* head, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    if (!head) return newNode;

    struct Node* temp = head;
    while (temp->next) temp = temp->next;
    temp->next = newNode;
    return head;
}

void printList(struct Node* head) {
    while (head) {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

```

```

struct Node* concatenateLists(struct Node* head1, struct Node* head2) {
    if (!head1) return head2;
    if (!head2) return head1;

    struct Node* temp = head1;
    while (temp->next) temp = temp->next;
    temp->next = head2;
    return head1;
}

```

```

int main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;

    int choice, value;

    printf("Creating List 1:\n");
    do {
        printf("Enter value to insert (-1 to stop): ");
        scanf("%d", &value);
        if (value != -1) list1 = insertAtEnd(list1, value);
    } while (value != -1);

    printf("Creating List 2:\n");
    do {
        printf("Enter value to insert (-1 to stop): ");
        scanf("%d", &value);
        if (value != -1) list2 = insertAtEnd(list2, value);
    } while (value != -1);
}

```

```

printf("List 1: ");
printList(list1);

printf("List 2: ");
printList(list2);

list1 = concatenateLists(list1, list2);

printf("Concatenated List: ");
printList(list1);

return 0;
}

```

```

Creating List 1:
Enter value to insert ( 1 to stop): 1
Enter value to insert (-1 to stop): 2
Enter value to insert (-1 to stop): 3
Enter value to insert (-1 to stop): 4
Enter value to insert (-1 to stop): -1
Creating List 2:
Enter value to insert ( 1 to stop): 5
Enter value to insert ( 1 to stop): 6
Enter value to insert ( 1 to stop): 7
Enter value to insert ( 1 to stop): 1
List 1: 1 -> 2 -> 3 -> 4 -> NULL
List 2: 5 -> 6 -> 7 -> NULL
Concatenated List: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> NULL

```

WAP to Implement Single Link List to simulate Stack & Queue Operations.

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

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

```
struct Node {  
    int data;  
    struct Node* next;  
};
```

```
struct Queue {  
    struct Node* front;  
    struct Node* rear;  
};
```

```
struct Node* createNode(int data) {  
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));  
    node->data = data;  
    node->next = NULL;  
    return node;  
}
```

```
struct Queue* createQueue() {  
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));  
    queue->front = NULL;  
    queue->rear = NULL;  
    return queue;  
}
```

```
int isEmpty(struct Queue* queue) {
```

```

    return queue->front == NULL;
}

void enqueue(struct Queue* queue, int data) {
    struct Node* node = createNode(data);
    if (queue->rear == NULL) {
        queue->front = queue->rear = node;
        return;
    }
    queue->rear->next = node;
    queue->rear = node;
}

int dequeue(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue underflow\n");
        return NULL;
    }
    struct Node* temp = queue->front;
    int data = temp->data;
    queue->front = queue->front->next;
    if (queue->front == NULL) queue->rear = NULL;
    free(temp);
    return data;
}

void display(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty\n");
    }
}

```



```

        return;
    }
    struct Node* temp = queue->front;
    printf("Queue contents:\n");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    struct Queue* queue = createQueue();
    int choice, value;

    while (1) {
        printf("\nQueue Operations Menu:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to enqueue: ");
                scanf("%d", &value);
                enqueue(queue, value);
                printf("Enqueued: %d\n", value);

```

```
        break;
    case 2:
        value = dequeue(queue);
        if (value != NULL) {
            printf("Dequeued: %d\n", value);
        }
        break;
    case 3:
        display(queue);
        break;
    case 4:
        printf("Exiting program.\n");
        exit(0);
    default:
        printf("Invalid choice! Please try again.\n");
    }
}
return 0;
}
```

```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 1
Enqueued: 1

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 2
Enqueued: 2

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 3
Enqueued: 3

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue contents:
1 2 3

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued: 1
```

```
|
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue contents:
2 4
```

```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued: 2
```

```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued: 4
```

```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
Queue is empty
```

```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Queue underflow
```

```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
Exiting program.
```

```

#include <stdio.h>

#include <stdlib.h>


// Define the Node structure
struct Node {
    int data;
    struct Node* next;
};


// Function to create a new node
struct Node* createNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->next = NULL;
    return node;
}


// Function to check if the stack is empty
int isEmpty(struct Node* top) {
    return top == NULL;
}


// Function to push an element onto the stack
void push(struct Node** top, int data) {
    struct Node* node = createNode(data);
    node->next = *top;
    *top = node;
    printf("\nPushed %d onto the stack.", data);
}

```

```
// Function to pop an element from the stack
int pop(struct Node** top) {
    if (isEmpty(*top)) {
        printf("Stack underflow\n");
        return -1; // Return -1 to indicate the stack is empty
    }
    struct Node* temp = *top;
    int data = temp->data;
    *top = (*top)->next;
    free(temp);
    return data;
}
```

```
// Function to display the elements in the stack
void display(struct Node* top) {
    if (isEmpty(top)) {
        printf("Stack is empty\n");
        return;
    }
    struct Node* temp = top;
    printf("\nStack: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
```

```
// Main function with switch-based menu
int main() {
```

```

struct Node* stack = NULL;

int choice, value;

while (1) {
    printf("\nStack Operations Menu:\n");
    printf("1. Push\n");
    printf("2. Pop\n");
    printf("3. Display\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter value to push: ");
            scanf("%d", &value);
            push(&stack, value);
            break;
        case 2:
            value = pop(&stack);
            if (value != -1) { // Check for valid pop operation
                printf("Popped: %d\n", value);
            }
            break;
        case 3:
            display(stack);
            break;
        case 4:
            printf("Exiting program.\n");
            exit(0);
    }
}

```

```
        default:
            printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}
```



```
Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 1

Pushed 1 onto the stack.
Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 2

Pushed 2 onto the stack.
Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 3

Pushed 3 onto the stack.
Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 4

Pushed 4 onto the stack.
Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3

Stack: 4 3 2 1
```

Stack Operations Menu:

1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 2

Popped: 4

Stack Operations Menu:

1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 2

Popped: 3

Stack Operations Menu:

1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 2

Popped: 2

Stack Operations Menu:

1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 2

Popped: 1

Stack Operations Menu:

1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 3

Stack is empty

Stack Operations Menu:

1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 3

Stack is empty

Stack Operations Menu:

1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 2

Stack underflow

Stack Operations Menu:

1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 4

Exiting program.

Lab Program 7

WAP to Implement doubly link list with primitive operations

- a) Create a doubly linked list.
- b) Insert a new node to the left of the node.
- c) Delete the node based on a specific value
- d) Display the contents of the list

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

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

```
typedef struct node {
```

```
    struct node* prev;
```

```
    int data;
```

```
    struct node* next;
```

```
}node;
```

```
node* createnode(int data){
```

```
    node* newnode=(node*)malloc(sizeof(node));
```

```
    newnode->prev=NULL;
```

```
    newnode->next=NULL;
```

```
    newnode->data=data;
```

```
    return newnode;
```

```
}
```

```
struct node* createDoublyLinkedList() {
```

```
    return NULL;
```

```
}
```

```
int isempty(node* head){
```

```
    return head==NULL;
```

```
}
```

```
node* insert_at_beginning(int data, node* head){  
    node* newnode=createnode(data);  
    newnode->next=head;  
    if(head!=NULL)  
        head->prev=newnode;  
    head=newnode;  
    printf("%d has been successfully inserted.\n",data);  
    return head;  
}
```

```
node* insert_to_left(int data, int key, node* head) {  
    node* temp = head;  
  
    while (temp != NULL && temp->data != key) {  
        temp = temp->next;  
    }
```

```
    if (temp == NULL) {  
        printf("Key not found\n");  
        return head;  
    }
```

```
    node* newnode = createnode(data);  
    newnode->next = temp;  
    newnode->prev = temp->prev;
```

```
    if (temp->prev != NULL) {  
        temp->prev->next = newnode;
```

```

    } else {
        head = newnode;
    }

    temp->prev = newnode;

    printf("%d has been successfully inserted left of %d\n", data, key);
    return head;
}

```

```

node* deletenode(int key,node* head){
    if(isempty(head)){
        printf("List is empty hence cannot delete a node\n ");
        return head;
    }
    node* temp=head;
    while(temp!=NULL && temp->data!=key){
        temp=temp->next;
    }
    if(temp==NULL){
        printf("Key not found\n");
        return head;
    }
    if (temp->prev != NULL) {
        temp->prev->next = temp->next;
    } else {
        head = temp->next;
    }
}

```

```

    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }
    printf("The node has been deleted\n");
    free(temp);
    return head;
}

```

```

void display(node* head){
    node* temp=head;
    if(isempty(head)){
        printf("List is empty\n");
        return;
    }
    printf("List elements : ");
    while(temp->next!=NULL){
        printf("%d <-> ",temp->data);
        temp=temp->next;
    }
    printf("%d -> NULL\n",temp->data);
}

```

```

int main() {
    node* head = createDoublyLinkedList();
    int choice, value, key;
    printf("\nDoubly Linked List Operations:\n");
    printf("1. Insert at the beginning\n");
    printf("2. Insert to the left of a specific node\n");
    printf("3. Delete a node by value\n");
    printf("4. Display the list\n");
}

```

```

printf("5. Exit\n");
while (1) {
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter the new value to insert: ");
            scanf("%d", &value);
            head=insert_at_beginning(value,head);
            break;
        case 2:
            if(!isempty(head)){
                printf("Enter the key value: ");
                scanf("%d", &key);
                printf("Enter the new value to insert: ");
                scanf("%d", &value);
                head=insert_to_left(value,key,head);
            }
            else
                printf("List is empty hence cannot insert to left.\n");
            break;
        case 3:
            if(!isempty(head)){
                printf("Enter the value of the node to delete: ");
                scanf("%d", &value);
                head=deletenode(value,head);
            }
            else

```



```
        printf("List is empty hence cannot delete.\n");
        break;

    case 4:
        display(head);
        break;

    case 5:
        printf("Exiting program.\n");
        exit(0);

    default:
        printf("Invalid choice. Please try again.\n");
    }
}
return 0;
}
```

```
C:\Python\Python310\Python.exe -x + -
Doubly Linked List Operations:
1. Insert at the beginning
2. Insert to the left of a specific node
3. Delete a node by value
4. Display the list
5. Exit
Enter your choice: 2
List is empty hence cannot insert to left.
Enter your choice: 1
Enter the new value to insert: 1
1 has been successfully inserted.
Enter your choice: 2
Enter the key value: 1
Enter the new value to insert: 3
3 has been successfully inserted left of 1
Enter your choice: 4
List elements : 3 <-> 1 -> NULL
Enter your choice: 2
Enter the key value: 1
Enter the new value to insert: 2
2 has been successfully inserted left of 1
Enter your choice: 4
List elements : 3 <-> 2 <-> 1 -> NULL
Enter your choice: 3
Enter the value of the node to delete: 2
The node has been deleted
Enter your choice: 3
Enter the value of the node to delete: 2
Key not found
Enter your choice: 3
Enter the value of the node to delete: 3
The node has been deleted
Enter your choice: 3
Enter the value of the node to delete: 1
The node has been deleted
Enter your choice: 4
List is empty
Enter your choice: 3
List is empty hence cannot delete.
Enter your choice: 7
Invalid choice. Please try again.
Enter your choice: 5
Exiting program.

Process returned 0 (0x0)   execution time : 0.62.681 s
Press any key to continue.
```

Activate Windows
Go to Settings to activate Windows.

Lab Program 8

Write a program

- a) To construct a binary Search tree.
- b) To traverse the tree using all the methods i.e., inorder, preorder and post order
- c) To display the elements in the tree.

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

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

```
struct Node {  
    int data;  
    struct Node* left;  
    struct Node* right;  
};
```

```
struct Node* createNode(int data) {  
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));  
    newNode->data = data;  
    newNode->left = newNode->right = NULL;  
    return newNode;  
}
```

```
struct Node* insert(struct Node* root, int data) {  
    if (root == NULL) {  
        return createNode(data);  
    }  
    if (data < root->data) {  
        root->left = insert(root->left, data);  
    } else if (data > root->data) {  
        root->right = insert(root->right, data);  
    }
```

```
    }  
    return root;  
}
```

```
void inorderTraversal(struct Node* root) {  
    if (root == NULL) {  
        return;  
    }  
    inorderTraversal(root->left);  
    printf("%d ", root->data);  
    inorderTraversal(root->right);  
}
```

```
void preorderTraversal(struct Node* root) {  
    if (root == NULL) {  
        return;  
    }  
    printf("%d ", root->data);  
    preorderTraversal(root->left);  
    preorderTraversal(root->right);  
}
```

```
void postorderTraversal(struct Node* root) {  
    if (root == NULL) {  
        return;  
    }  
    postorderTraversal(root->left);  
    postorderTraversal(root->right);  
    printf("%d ", root->data);  
}
```

```

int main() {
    struct Node* root = NULL;
    int choice, data;

    while (1) {
        printf("\nBinary Search Tree Operations:\n");
        printf("1. Insert a node\n");
        printf("2. In-order traversal\n");
        printf("3. Pre-order traversal\n");
        printf("4. Post-order traversal\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &data);
                root = insert(root, data);
                printf("Node %d inserted.\n", data);
                break;

            case 2:
                printf("In-order traversal: ");
                inorderTraversal(root);
                printf("\n");
                break;

            case 3:

```

```
    printf("Pre-order traversal: ");  
    preorderTraversal(root);  
    printf("\n");  
    break;  
  
case 4:  
    printf("Post-order traversal: ");  
    postorderTraversal(root);  
    printf("\n");  
    break;  
  
case 5:  
    printf("Exiting...\n");  
    exit(0);  
  
default:  
    printf("Invalid choice, please try again.\n");  
}  
}  
  
return 0;  
}
```

```
C:\247\bl.exe
Enter the number of vertices : 6
Enter the adjacency matrix:
0 1 1 0 0 1
1 0 1 1 0 0
1 1 0 0 0 0
0 1 0 0 1 0
0 0 0 1 0 0
1 0 0 0 0 0
Enter the starting vertex: 6
BFS Traversal: 6 1 2 3 4 5

Process returned 0 (0x0)   execution time : 74.770 s
Press any key to continue.
|
```

Lab Program 9

Write a program to traverse a graph using BFS method

```
#include <stdio.h>

#include <stdlib.h>

#define MAX 100

struct Queue {
    int items[MAX];
    int front, rear;
};

struct Queue* createQueue() {
    struct Queue* q = (struct Queue*)malloc(sizeof(struct Queue));
    q->front = -1;
    q->rear = -1;
    return q;
}

int isEmpty(struct Queue* q) {
    return q->front == -1;
}

void enqueue(struct Queue* q, int value) {
    if (q->rear == MAX - 1) {
        printf("Queue is full\n");
    } else {
        if (q->front == -1) {
```



```

        q->front = 0;
    }
    q->items[++q->rear] = value;
}
}

```

```

int dequeue(struct Queue* q) {
    int item;
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return -1;
    } else {
        item = q->items[q->front];
        if (q->front == q->rear) {
            q->front = q->rear = -1;
        } else {
            q->front++;
        }
        return item;
    }
}

```

```

void bfs(int graph[MAX][MAX], int startVertex, int n) {
    int visited[MAX] = {0};
    struct Queue* q = createQueue();

    visited[startVertex] = 1;
    enqueue(q, startVertex);

    printf("BFS Traversal: ");
}

```

```

while (!isEmpty(q)) {
    int currentVertex = dequeue(q);
    printf("%d ", currentVertex);

    for (int i = 1; i <= n; i++) {
        if (graph[currentVertex][i] == 1 && !visited[i]) {
            visited[i] = 1;
            enqueue(q, i);
        }
    }
}

printf("\n");
}

int main() {
    int n, startVertex;
    int graph[MAX][MAX];

    printf("Enter the number of vertices : ");
    scanf("%d", &n);

    printf("Enter the adjacency matrix:\n");
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
}

```

```

printf("Enter the starting vertex: ");

scanf("%d", &startVertex);

bfs(graph, startVertex, n);

return 0;
}

```

```

C:\247\bst.exe
Enter the key to be inserted: 20
Key 20 inserted successfully!

Enter your choice: 1
Enter the key to be inserted: 40
Key 40 inserted successfully!

Enter your choice: 1
Enter the key to be inserted: 60
Key 60 inserted successfully!

Enter your choice: 1
Enter the key to be inserted: 80
Key 80 inserted successfully!

Enter your choice: 2
In-order traversal: 20 30 40 50 60 70 80

Enter your choice: 3
Pre-order traversal: 50 30 20 40 70 60 80

Enter your choice: 4
Post-order traversal: 20 40 30 60 80 70 50

Enter your choice: 5
Exiting...

Process returned 0 (0x0)   execution time : 0.2889 s
Press any key to continue.

```

Lab Program 9

Write a program to check whether given graph is connected or not using DFS method.

```
#include <stdio.h>

#define MAX_NODES 100

int adjacencyMatrix[MAX_NODES][MAX_NODES];
int visited[MAX_NODES];
int nodes;

// Function for DFS
void DFS(int vertex) {
    visited[vertex] = 1;
    printf("%d ", vertex); // Print visited node

    for (int i = 0; i < nodes; i++) {
        if (adjacencyMatrix[vertex][i] == 1 && !visited[i]) {
            DFS(i);
        }
    }
}

// Function to check if the graph is connected
int isConnected() {
    // Initialize visited array to 0
    for (int i = 0; i < nodes; i++) {
        visited[i] = 0;
    }
}
```

```

// Start DFS from node 0
DFS(0);

// Check if all nodes are visited
for (int i = 0; i < nodes; i++) {
    if (!visited[i]) {
        return 0; // Graph is not connected
    }
}
return 1; // Graph is connected
}

int main() {
    printf("Enter the number of nodes: ");
    scanf("%d", &nodes);

    printf("Enter the adjacency matrix:\n");
    for (int i = 0; i < nodes; i++) {
        for (int j = 0; j < nodes; j++) {
            scanf("%d", &adjacencyMatrix[i][j]);
        }
    }

    // Check connectivity
    if (isConnected()) {
        printf("\nThe graph is connected.\n");
    } else {
        printf("\nThe graph is not connected.\n");
    }
}

```

```
    return 0;  
}
```

```
Enter the number of nodes: 4  
Enter the adjacency matrix:  
0 1 1 0  
1 0 0 1  
1 0 0 0  
0 1 0 0  
0 1 3 2  
The graph is connected.
```