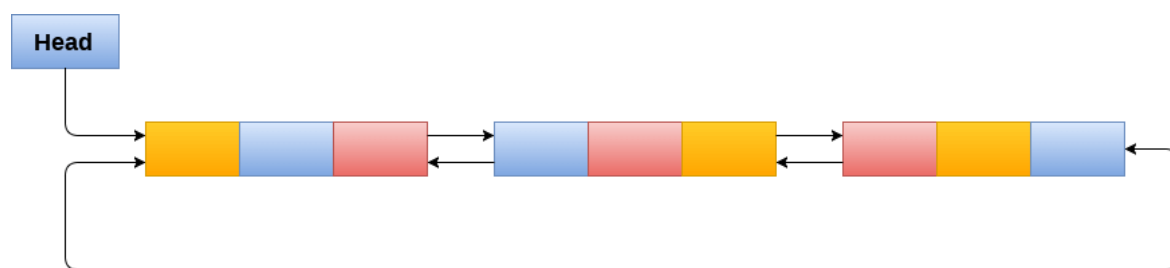


Circular Doubly Linked List

Circular doubly linked list is a more complexed type of data structure in which a node contain pointers to its previous node as well as the next node. Circular doubly linked list doesn't contain NULL in any of the node. The last node of the list contains the address of the first node of the list. The first node of the list also contain address of the last node in its previous pointer.

A circular doubly linked list is shown in the following figure.



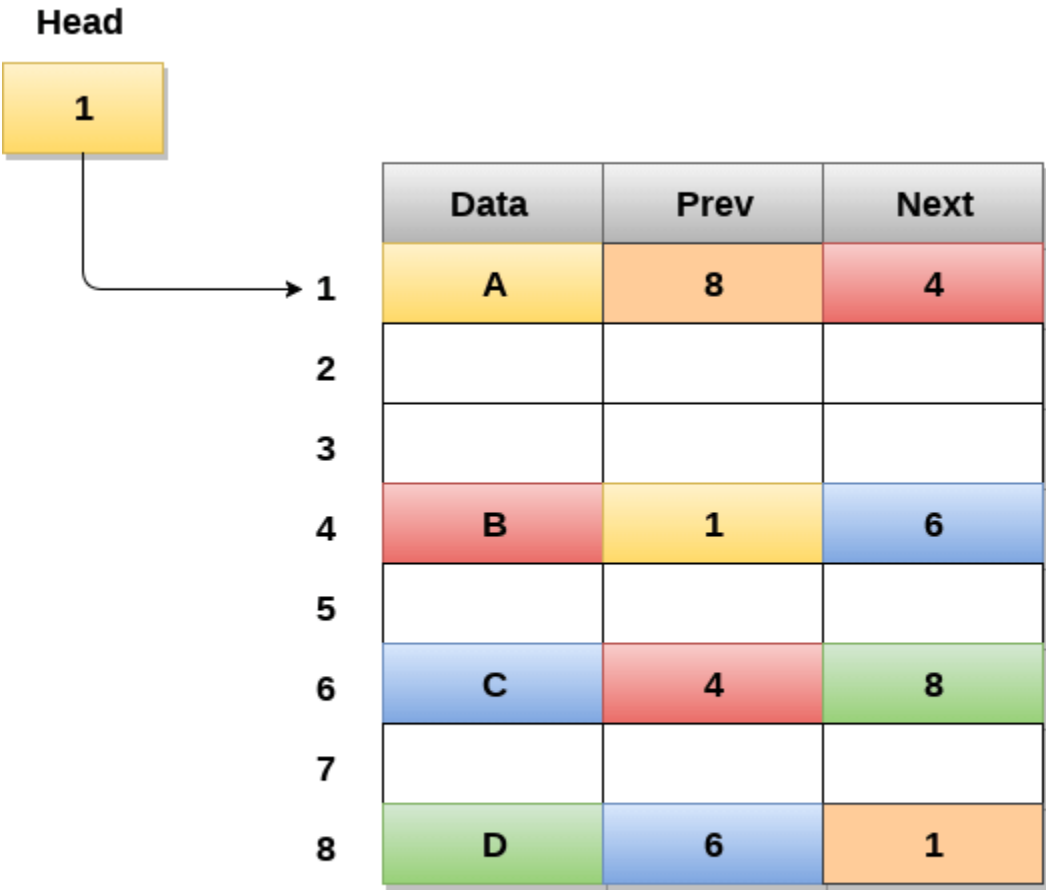
Circular Doubly Linked List

Due to the fact that a circular doubly linked list contains three parts in its structure therefore, it demands more space per node and more expensive basic operations. However, a circular doubly linked list provides easy manipulation of the pointers and the searching becomes twice as efficient.

Memory Management of Circular Doubly linked list

The following figure shows the way in which the memory is allocated for a circular doubly linked list. The variable head contains the address of the first element of the list i.e. 1 hence the starting node of the list contains data A is stored at address 1. Since, each node of the list is supposed to have three parts therefore, the starting node of the list contains address of the last node i.e. 8 and the next node i.e. 4. The last node of the list that is stored at address 8 and containing data as 6, contains address of the first node of the list as shown in the image i.e. 1. In circular doubly linked list, the last node is identified by the address of the first node which is stored in the next part of the last node therefore the node which contains the address of the first node, is actually the last node of the list.

25



26
27

Memory Representation of a Circular Doubly linked list

Operations on circular doubly linked list :

29 There are various operations which can be performed on circular doubly linked list. The
30 node structure of a circular doubly linked list is similar to doubly linked list. However, the
31 operations on circular doubly linked list is described in the following table.

SN	Operation	Description
1	Insertion at beginning	Adding a node in circular doubly linked list at the beginning.

2	Insertion at end	Adding a node in circular doubly linked list at the end.
3	Deletion at beginning	Removing a node in circular doubly linked list from beginning.
4	Deletion at end	Removing a node in circular doubly linked list at the end.

32 Traversing and searching in circular doubly linked list is similar to that in the circular singly
33 linked list.

34 C program to implement all the operations on circular 35 doubly linked list

```

361. #include<stdio.h>
372. #include<stdlib.h>
383. struct node
394. {
405.     struct node *prev;
416.     struct node *next;
427.     int data;
438. };
449. struct node *head;
4510. void insertion_beginning();
4611. void insertion_last();
4712. void deletion_beginning();
4813. void deletion_last();
4914. void display();
5015. void search();
5116. void main ()
5217. {
5318. int choice =0;
5419. while(choice != 9)
5520. {
5621.     printf("\n*****Main Menu*****\n");
5722.     printf("\nChoose one option from the following list ...\n");

```

```

5823.    printf("\n=====\\
59    n");
6024.    printf("\n1.Insert in Beginning\n2.Insert at last\n3.Delete from Beginning\n4.Delete from las
61    t\n5.Search\n6.Show\n7.Exit\n");
6225.    printf("\nEnter your choice?\n");
6326.    scanf("\n%d",&choice);
6427.    switch(choice)
6528.    {
6629.        case 1:
6730.            insertion_beginning();
6831.            break;
6932.        case 2:
7033.            insertion_last();
7134.            break;
7235.        case 3:
7336.            deletion_beginning();
7437.            break;
7538.        case 4:
7639.            deletion_last();
7740.            break;
7841.        case 5:
7942.            search();
8043.            break;
8144.        case 6:
8245.            display();
8346.            break;
8447.        case 7:
8548.            exit(0);
8649.            break;
8750.        default:
8851.            printf("Please enter valid choice..");
8952.    }
9053. }
9154. }

```

```

9255. void insertion_beginning()
9356. {
9457.     struct node *ptr,*temp;
9558.     int item;
9659.     ptr = (struct node *)malloc(sizeof(struct node));
9760.     if(ptr == NULL)
9861.     {
9962.         printf("\nOVERFLOW");
10063.     }
10164.     else
10265.     {
10366.         printf("\nEnter Item value");
10467.         scanf("%d",&item);
10568.         ptr->data=item;
10669.         if(head==NULL)
10770.         {
10871.             head = ptr;
10972.             ptr -> next = head;
11073.             ptr -> prev = head;
11174.         }
11275.         else
11376.         {
11477.             temp = head;
11578.             while(temp -> next != head)
11679.             {
11780.                 temp = temp -> next;
11881.             }
11982.             temp -> next = ptr;
12083.             ptr -> prev = temp;
12184.             head -> prev = ptr;
12285.             ptr -> next = head;
12386.             head = ptr;
12487.         }
12588.         printf("\nNode inserted\n");

```

```

12689. }
12790.
12891. }
12992. void insertion_last()
13093. {
13194.     struct node *ptr,*temp;
13295.     int item;
13396.     ptr = (struct node *) malloc(sizeof(struct node));
13497.     if(ptr == NULL)
13598.     {
13699.         printf("\nOVERFLOW");
137100.     }
138101.     else
139102.     {
140103.         printf("\nEnter value");
141104.         scanf("%d",&item);
142105.         ptr->data=item;
143106.         if(head == NULL)
144107.         {
145108.             head = ptr;
146109.             ptr -> next = head;
147110.             ptr -> prev = head;
148111.         }
149112.         else
150113.         {
151114.             temp = head;
152115.             while(temp->next != head)
153116.             {
154117.                 temp = temp->next;
155118.             }
156119.             temp->next = ptr;
157120.             ptr -> prev=temp;
158121.             head -> prev = ptr;
159122.             ptr -> next = head;

```

```

160123.     }
161124.     }
162125.     printf("\nnode inserted\n");
163126. }
164127.
165128. void deletion_beginning()
166129. {
167130.     struct node *temp;
168131.     if(head == NULL)
169132.     {
170133.         printf("\n UNDERFLOW");
171134.     }
172135.     else if(head->next == head)
173136.     {
174137.         head = NULL;
175138.         free(head);
176139.         printf("\nnode deleted\n");
177140.     }
178141.     else
179142.     {
180143.         temp = head;
181144.         while(temp -> next != head)
182145.         {
183146.             temp = temp -> next;
184147.         }
185148.         temp -> next = head -> next;
186149.         head -> next -> prev = temp;
187150.         free(head);
188151.         head = temp -> next;
189152.     }
190153.
191154. }
192155. void deletion_last()
193156. {

```

```

194157.    struct node *ptr;
195158.    if(head == NULL)
196159.    {
197160.        printf("\n UNDERFLOW");
198161.    }
199162.    else if(head->next == head)
200163.    {
201164.        head = NULL;
202165.        free(head);
203166.        printf("\nnode deleted\n");
204167.    }
205168.    else
206169.    {
207170.        ptr = head;
208171.        if(ptr->next != head)
209172.        {
210173.            ptr = ptr -> next;
211174.        }
212175.        ptr -> prev -> next = head;
213176.        head -> prev = ptr -> prev;
214177.        free(ptr);
215178.        printf("\nnode deleted\n");
216179.    }
217180. }
218181.
219182. void display()
220183. {
221184.     struct node *ptr;
222185.     ptr=head;
223186.     if(head == NULL)
224187.     {
225188.         printf("\nnothing to print");
226189.     }
227190.     else

```



```

228191.    {
229192.        printf("\n printing values ... \n");
230193.
231194.        while(ptr -> next != head)
232195.        {
233196.
234197.            printf("%d\n", ptr -> data);
235198.            ptr = ptr -> next;
236199.        }
237200.        printf("%d\n", ptr -> data);
238201.    }
239202.
240203. }
241204.
242205. void search()
243206. {
244207.     struct node *ptr;
245208.     int item,i=0,flag=1;
246209.     ptr = head;
247210.     if(ptr == NULL)
248211.     {
249212.         printf("\nEmpty List\n");
250213.     }
251214.     else
252215.     {
253216.         printf("\nEnter item which you want to search?\n");
254217.         scanf("%d",&item);
255218.         if(head ->data == item)
256219.         {
257220.             printf("item found at location %d",i+1);
258221.             flag=0;
259222.         }
260223.         else
261224.         {

```

```

262225.     while (ptr->next != head)
263226.     {
264227.         if(ptr->data == item)
265228.         {
266229.             printf("item found at location %d ",i+1);
267230.             flag=0;
268231.             break;
269232.         }
270233.         else
271234.         {
272235.             flag=1;
273236.         }
274237.         i++;
275238.         ptr = ptr -> next;
276239.     }
277240. }
278241. if(flag != 0)
279242. {
280243.     printf("Item not found\n");
281244. }
282245. }
283246.
284247. }
```

285 Output:

```

286 *****Main Menu*****
287
288 Choose one option from the following list ...
289
290 =====
291
292 1.Insert in Beginning
293 2.Insert at last
294 3.Delete from Beginning
295 4.Delete from last
296 5.Search
297 6.Show
298 7.Exit
299
300 Enter your choice?
```

```

301 1
302
303 Enter Item value123
304
305 Node inserted
306
307 *****Main Menu*****
308
309 Choose one option from the following list ...
310
311 =====
312
313 1.Insert in Beginning
314 2.Insert at last
315 3.Delete from Beginning
316 4.Delete from last
317 5.Search
318 6.Show
319 7.Exit
320
321 Enter your choice?
322 2
323
324 Enter value234
325
326 node inserted
327
328 *****Main Menu*****
329
330 Choose one option from the following list ...
331
332 =====
333
334 1.Insert in Beginning
335 2.Insert at last
336 3.Delete from Beginning
337 4.Delete from last
338 5.Search
339 6.Show
340 7.Exit
341
342 Enter your choice?
343 1
344
345 Enter Item value90
346
347 Node inserted
348
349 *****Main Menu*****
350
351 Choose one option from the following list ...
352
353 =====
354
355 1.Insert in Beginning
356 2.Insert at last
357 3.Delete from Beginning

```

```

358 4.Delete from last
359 5.Search
360 6.Show
361 7.Exit
362
363 Enter your choice?
364 2
365
366 Enter value80
367
368 node inserted
369
370 *****Main Menu*****
371
372 Choose one option from the following list ...
373
374 =====
375
376 1.Insert in Beginning
377 2.Insert at last
378 3.Delete from Beginning
379 4.Delete from last
380 5.Search
381 6.Show
382 7.Exit
383
384 Enter your choice?
385 3
386
387 *****Main Menu*****
388
389 Choose one option from the following list ...
390
391 =====
392
393 1.Insert in Beginning
394 2.Insert at last
395 3.Delete from Beginning
396 4.Delete from last
397 5.Search
398 6.Show
399 7.Exit
400
401 Enter your choice?
402 4
403
404 node deleted
405
406 *****Main Menu*****
407
408 Choose one option from the following list ...
409
410 =====
411
412 1.Insert in Beginning
413 2.Insert at last
414 3.Delete from Beginning

```

```

415 4.Delete from last
416 5.Search
417 6.Show
418 7.Exit
419
420 Enter your choice?
421 6
422
423     printing values ...
424 123
425
426 *****Main Menu*****
427
428 Choose one option from the following list ...
429
430 =====
431
432 1.Insert in Beginning
433 2.Insert at last
434 3.Delete from Beginning
435 4.Delete from last
436 5.Search
437 6.Show
438 7.Exit
439
440 Enter your choice?
441 5
442
443 Enter item which you want to search?
444 123
445 item found at location 1
446 *****Main Menu*****
447
448 Choose one option from the following list ...
449
450 =====
451
452 1.Insert in Beginning
453 2.Insert at last
454 3.Delete from Beginning
455 4.Delete from last
456 5.Search
457 6.Show
458 7.Exit
459
460 Enter your choice?
461 7
462

```