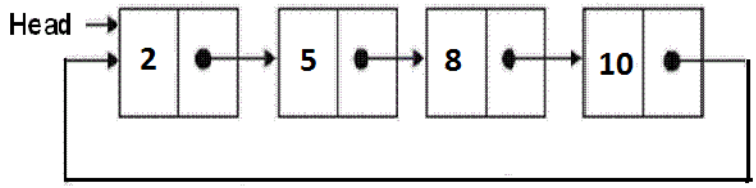
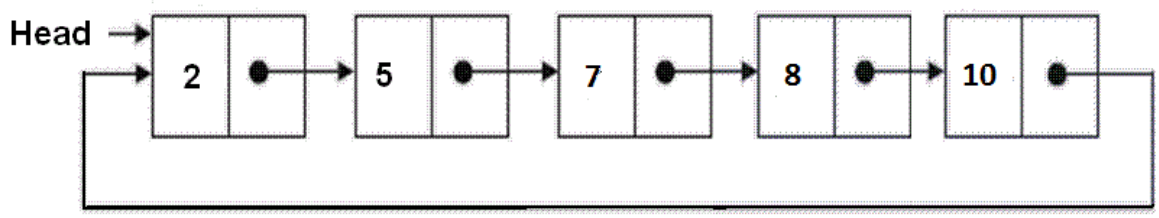
Sorted insert for circular linked list

Difficulty Level: Rookie

Write a C function to insert a new value in a sorted Circular Linked List (CLL). For example, if the input CLL is following.



After insertion of **7**, the above CLL should be changed to following



Algorithm:

Allocate memory for the newly inserted node and put data in the newly allocated node. Let the pointer to the new node be new_node. After memory allocation, following are the three cases that need to be handled.

```
1) Linked List is empty:
    a)  since new_node is the only node in CLL, make a self loop.
          new_node->next = new_node;
    b) change the head pointer to point to new node.
          *head_ref = new_node;
2) New node is to be inserted just before the head node:
  (a) Find out the last node using a loop.
         while(current->next != *head_ref)
            current = current->next;
  (b) Change the next of last node.
         current->next = new_node;
  (c) Change next of new node to point to head.
```

```
          new_node->next = *head_ref;
    (d) change the head pointer to point to new node.
          *head_ref = new_node;
 3) New node is to be  inserted somewhere after the head:
    (a) Locate the node after which new node is to be inserted.
          while ( current->next!= *head_ref &&
              current->next->data < new_node->data)
          {   current = current->next;   }
    (b) Make next of new_node as next of the located pointer
          new_node->next = current->next;
    (c) Change the next of the located pointer
          current->next = new_node;
```

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

/* structure for a node */
struct node
{
  int data;
  struct node *next;
};

/* function to insert a new_node in a list in sorted way.
   Note that this function expects a pointer to head node
   as this can modify the head of the input linked list */
void sortedInsert(struct node** head_ref, struct node* new_node)
{
  struct node* current = *head_ref;

  // Case 1 of the above algo
  if (current == NULL)
  {
     new_node->next = new_node;
     *head_ref = new_node;
  }

  // Case 2 of the above algo
  else if (current->data >= new_node->data)
  {
    /* If value is smaller than head's value then
      we need to change next of last node */
    while(current->next != *head_ref)
        current = current->next;
    current->next = new_node;
    new_node->next = *head_ref;
    *head_ref = new_node;
  }

  // Case 3 of the above algo
  else
  {
    /* Locate the node before the point of insertion */
    while (current->next!= *head_ref && current->next->data < new_node->data)
      current = current->next;

    new_node->next = current->next;
    current->next = new_node;
  }
}
```

```
/* Function to print nodes in a given linked list */
void printList(struct node *start)
{
  struct node *temp;

  if(start != NULL)
  {
    temp = start;
    printf("\n");
    do {
      printf("%d ", temp->data);
      temp = temp->next;
    } while(temp != start);
  }
}

/* Driver program to test above functions */
int main()
{
  int arr[] = {12, 56, 2, 11, 1, 90};
  int list_size, i;

  /* start with empty linked list */
  struct node *start = NULL;
  struct node *temp;

  /* Create linked list from the array arr[].
    Created linked list will be 1->2->11->56->12 */
  for(i = 0; i< 6; i++)
  {
    temp = (struct node *)malloc(sizeof(struct node));
    temp->data = arr[i];
    sortedInsert(&start, temp);
  }

  printList(start);
  getchar();
  return 0;
}
```

Run on IDE

Output:

1 2 11 12 56 90

Time Complexity: O(n) where n is the number of nodes in the given linked list.

Case 2 of the above algorithm/code can be optimized. Please see this comment from Pavan. To implement the suggested change we need to modify the case 2 to following.

```
// Case 2 of the above algo
else if (current->data >= new_node->data)
{
  // swap the data part of head node and new node
  swap(&(current->data), &(new_node->data));  // assuming that we have a function swap(int

  new_node->next = (*head_ref)->next;
  (*head_ref)->next = new_node;
}
```

Run on IDE

Please write comments if you find the above code/algorithm incorrect, or find other ways to solve the same problem.

22 Comments Category: Linked Lists

Related Posts:

- Merge two sorted linked lists such that merged list is in reverse order
- Compare two strings represented as linked lists
- Rearrange a given linked list in-place.
- Sort a linked list that is sorted alternating ascending and descending orders?
- Select a Random Node from a Singly Linked List
- Merge Sort for Doubly Linked List
- Point to next higher value node in a linked list with an arbitrary pointer
- Swap nodes in a linked list without swapping data

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.