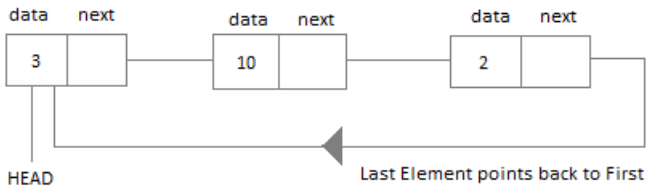
Bởi **Nguyễn Văn Hiếu**

This entry is part 7 of 9 in the series Cấu trúc dữ liệu

Danh sách liên kết vòng(Circular Linked List) là danh sách liên kết có thêm sự kết nối giữa 2 phần tử đầu tiên và phần tử cuối cùng để tạo thành vòng khép kín. Bài viết này Nguyễn Văn Hiếu sẽ hướng dẫn bạn cách cài đặt DSLK vòng trong C/C++ nhé.

Nội dung bài viết

1. Lý thuyết về danh sách liên kết vòng



Danh sách liên kết vòng có kết nối của phần tử cuối với head

Đó là sự khác biệt giữa DSLK vòng và danh sách liên kết đơn. Tất nhiên, bạn cũng có thể cài đặt DSLK vòng trên danh sách liên kết đôi. Tuy nhiên, trong bài này tôi sẽ hướng dẫn bạn cài đặt trên danh sách liên kết đơn trước nhé.

2. Cài đặt DSLK vòng đơn

Khai báo kiểu dữ liệu Node

Khai báo Node của DSLK vòng trong bài này giống với danh sách liên kết đơn

```
0
1 struct Node {
2     int data;
3     struct Node * next;
4 };
5
6 typedef struct Node NODE;
7
```

Tạo mới Node

```
0
1 NODE * CreateNewNode(int data) {
```

```
    return newNode;
}

```

Lấy số lượng Node

```
int Length(NODE * tail) {
    NODE * current = tail;
    int i = 1;
    if (tail == NULL) {
        return 0;
    } else {
        current = current -> next;
        while (current != tail) {
            i++;
            current = current -> next;
        }
    }
    return i;
}

```

Thêm vào đầu

```
NODE * InsertAtHead(NODE * tail, int data) {
    NODE * newNode = CreateNewNode(data);
    if (tail == NULL) {
        tail = newNode;
        newNode -> next = newNode;
    } else {
        newNode -> next = tail -> next;
        tail -> next = newNode;
    }
    return tail;
}

```

Thêm vào cuối

Trong **Circular Linked List**, để thêm vào cuối bạn có thể thêm vào đầu và trả về đầu mới là next của node mới thêm vào.

```
NODE * InsertAtEnd(NODE * tail, int data) {
    // simply insert at head and return the next node pointed by tail
    return InsertAtHead(tail, data) -> next;
}

```

Thêm vào vị trí bất kỳ

Các bạn lưu ý là vị trí, không phải chỉ số nhé. Do đó, phần tử đầu tiên có vị trí là 1 và phần tử cuối là len.

```
NODE * InsertAtArbitrary(NODE * tail, int data, int location) {
    int len = length(tail), i;
    if (location < 1 || location > len + 1) {
        printf("\nInvalid location to enter data\n");
    } else {
```

```
        for (i = 1; i != location; i++) current = current -> next;
        newNode -> next = current -> next;
        current -> next = newNode;
        if (location == len + 1) tail = newNode;
    }
    return tail;
}

```

Xóa theo giá trị chỉ định

```

NODE * DeleteByValue(NODE * tail, int data) {
    NODE * current = tail, * previous;
    if (tail == NULL) return tail;
    else if (tail == tail -> next) {
        if (tail -> data == data) {
            tail = NULL;
            free(current);
        }
        return tail;
    }
    do {
        previous = current;
        current = current -> next;
        if (current -> data == data) {
            previous -> next = current -> next;
            if (current == tail) tail = previous;
            free(current);
            current = previous -> next;
        }
    } while (current != tail);
    return tail;
}

```

Xóa theo vị trí chỉ định

```

NODE * DeleteByLocation(NODE * tail, int location) {
    NODE * current, * previous = tail;
    int len = Length(tail), i;
    if (location < 1 || location > len) {
        printf("Invalid Location to delete");
    } else if (len == 1) {
        tail = NULL;
        free(current);
    } else {
        current = tail -> next;
        for (i = 1; i < location; i++) {
            previous = current;
            current = current -> next;
        }
        previous -> next = current -> next;
        if (current == tail) tail = previous;
        free(current);
    }

    return tail;
}

```

```
0
1  NODE * Sort(NODE * tail) {
2      if (Length(tail) < 2) return tail;
3      NODE * ptr1 = tail -> next, * ptr2, * min;
4      int temp;
5      // selection sort implementation
6      while (ptr1 -> next != tail -> next) {
7          min = ptr1;
8          ptr2 = ptr1 -> next;
9          while (ptr2 != tail -> next) {
10             if (min -> data > ptr2 -> data) min = ptr2;
11             ptr2 = ptr2 -> next;
12         }
13         if (min != ptr1) {
14             temp = min -> data;
15             min -> data = ptr1 -> data;
16             ptr1 -> data = temp;
17         }
18         ptr1 = ptr1 -> next;
19     }
20     return tail;
21 }
22
```

3. Full code DSLK vòng đơn

```
0
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct Node {
5      int data;
6      struct Node * next;
7  };
8
9  typedef struct Node NODE;
10
11 NODE * CreateNewNode(int data) {
12     NODE * newNode = (NODE *) malloc (sizeof(NODE));
13     newNode -> data = data;
14     return newNode;
15 }
16
17 void Display(NODE * tail) {
18     NODE * current = tail;
19     if (tail != NULL) {
20         do {
21             current = current -> next;
22             printf(" %d -> ", current -> data);
23         } while (current != tail);
24     }
25 }
26
27 int Length(NODE * tail) {
28     NODE * current = tail;
29     int i = 1;
30     if (tail == NULL) {
31         return 0;
32     } else {
33         current = current -> next;
34         while (current != tail) {
35             i++;
```

```
    }
    return i;
}

NODE * InsertAtHead(NODE * tail, int data) {
    NODE * newNode = CreateNewNode(data);
    if (tail == NULL) {
        tail = newNode;
        newNode -> next = newNode;
    } else {
        newNode -> next = tail -> next;
        tail -> next = newNode;
    }
    return tail;
}

NODE * InsertAtEnd(NODE * tail, int data) {
    // simply insert at head and return the next node pointed by tail
    return InsertAtHead(tail, data) -> next;
}

NODE * InsertAtArbitrary(NODE * tail, int data, int location) {
    int len = Length(tail), i;
    if (location < 1 || location > len + 1) {
        printf("\nInvalid location to enter data\n");
    } else {
        if (tail == NULL) return InsertAtHead(tail, data);
        NODE * newNode = CreateNewNode(data), * current = tail;
        for (i = 1; i != location; i++) current = current -> next;
        newNode -> next = current -> next;
        current -> next = newNode;
        if (location == len + 1) tail = newNode;
    }
    return tail;
}

NODE * DeleteByValue(NODE * tail, int data) {
    NODE * current = tail, * previous;
    if (tail == NULL) return tail;
    else if (tail == tail -> next) {
        if (tail -> data == data) {
            tail = NULL;
            free(current);
        }
        return tail;
    }
    do {
        previous = current;
        current = current -> next;
        if (current -> data == data) {
            previous -> next = current -> next;
            if (current == tail) tail = previous;
            free(current);
            current = previous -> next;
        }
    } while (current != tail);
    return tail;
}

NODE * DeleteByLocation(NODE * tail, int location) {
    NODE * current, * previous = tail;
    int len = Length(tail), i;
    if (location < 1 || location > len) {
```

```
        tail = NULL;
        free(current);
    } else {
        current = tail -> next;
        for (i = 1; i < location; i++) {
            previous = current;
            current = current -> next;
        }
        previous -> next = current -> next;
        if (current == tail) tail = previous;
        free(current);
    }

    return tail;
}

NODE * sort(NODE * tail) {
    if (Length(tail) < 2) return tail;
    NODE * ptr1 = tail -> next, * ptr2, * min;
    int temp;
    // selection sort implementation
    while (ptr1 -> next != tail -> next) {
        min = ptr1;
        ptr2 = ptr1 -> next;
        while (ptr2 != tail -> next) {
            if (min -> data > ptr2 -> data) min = ptr2;
            ptr2 = ptr2 -> next;
        }
        if (min != ptr1) {
            temp = min -> data;
            min -> data = ptr1 -> data;
            ptr1 -> data = temp;
        }
        ptr1 = ptr1 -> next;
    }
    return tail;
}

int main() {
    NODE * cll = NULL;
    int option, data, location;
    while (1) {
        Display(cll);
        printf("\nlength = %d\n", Length(cll));
        printf("\n\nMENU OF CHOICE\n1. Insert at head\n2. Insert at end\n3. Insert at arbitrary location\n
        printf("Your choice: ");
        scanf("%d", &option);

        if (option == 1) {
            printf("Enter data to be inserted: ");
            scanf("%d", &data);
            cll = InsertAtHead(cll, data);
        } else if (option == 2) {
            printf("Enter data to be inserted at end: ");
            scanf("%d", &data);
            cll = InsertAtEnd(cll, data);
        } else if (option == 3) {
            printf("Enter data to be inserted: ");
            scanf("%d", &data);
            printf("Enter location to be inserted into: ");
            scanf("%d", &location);
            cll = InsertAtArbitrary(cll, data, location);
        } else if (option == 4) {
```

```
            cll = DeleteByValue(cll, data);
        } else if (option == 5) {
            printf("Enter location to be deleted: ");
            scanf("%d", &location);
            cll = DeleteByLocation(cll, location);
        } else if(option == 6) {
            sort(cll);
        } else if (option == 7) {
            break;
        }
    }
    return 0;
}

```

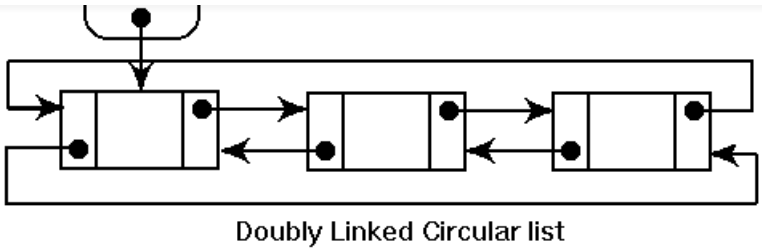
Kết quả chạy:

```
0
1  MENU OF CHOICE
2  1. Insert at head
3  2. Insert at end
4  3. Insert at arbitrary location
5  4. Delete by value
6  5. Delete by location
7  6. Sort
8  7. Exit
9  Your choice: 1
10 Enter data to be inserted: 1
11  1 ->
12 length = 1
13
14
15 MENU OF CHOICE
16 1. Insert at head
17 2. Insert at end
18 3. Insert at arbitrary location
19 4. Delete by value
20 5. Delete by location
21 6. Sort
22 7. Exit
23 Your choice: 1
24 Enter data to be inserted: 2
25  2 ->  1 ->
26 length = 2
27
28
29 MENU OF CHOICE
30 1. Insert at head
31 2. Insert at end
32 3. Insert at arbitrary location
33 4. Delete by value
34 5. Delete by location
35 6. Sort
36 7. Exit
37 Your choice: 1
38 Enter data to be inserted: 3
39  3 ->  2 ->  1 ->
40 length = 3
41
42
43 MENU OF CHOICE
44 1. Insert at head
```

```
3. Insert at arbitrary location
4. Delete by value
5. Delete by location
6. Sort
7. Exit
Your choice: 1
Enter data to be inserted: 4
 4 ->  3 ->  2 ->  1 ->
length = 4


MENU OF CHOICE
1. Insert at head
2. Insert at end
3. Insert at arbitrary location
4. Delete by value
5. Delete by location
6. Sort
7. Exit
Your choice: 1
Enter data to be inserted: 5
 5 ->  4 ->  3 ->  2 ->  1 ->
length = 5


MENU OF CHOICE
1. Insert at head
2. Insert at end
3. Insert at arbitrary location
4. Delete by value
5. Delete by location
6. Sort
7. Exit
Your choice: 6
 1 ->  2 ->  3 ->  4 ->  5 ->
length = 5


MENU OF CHOICE
1. Insert at head
2. Insert at end
3. Insert at arbitrary location
4. Delete by value
5. Delete by location
6. Sort
7. Exit
Your choice:

```

4. Danh sách liên kết vòng đôi

Với dslk vòng đôi, nếu bạn nào quan tâm có thể tham khảo source code sau đây. Mình xin phép không đi sâu giải thích nữa.



Hình ảnh mô phỏng DSLK vòng đôi(Circular Doubly Linked List)

```
0
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct node{
5      int data;
6      struct node *llink;
7      struct node *rlink;
8  };
9
10 typedef struct node *Node;
11
12 Node CreateNode(int data){
13     Node temp = (Node) malloc(sizeof(struct node));
14     temp->llink = temp;
15     temp->rlink = temp;
16     temp->data = data;
17     return temp;
18 }
19 void DeleteNode(Node temp)
20 {
21     printf("%d deleted\n", temp->data);
22     free(temp);
23     return;
24 }
25
26 void InsertFront(Node head, int data)
27 {
28     Node newNode = CreateNode(data);
29     Node first = head->rlink;
30     head->rlink = newNode;
31     newNode->rlink = first;
32     newNode->llink = head;
33     if(first !=NULL)
34         first->llink = newNode;
35 }
36 void DeleteRear(Node head)
37 {
38     if(head->rlink == head)
39     {
40         puts("List is empty");
41         return;
42     }
43     Node last = head->llink;
44     Node slast = last->llink;
45     slast->rlink = head;
46     head->llink = slast;
47     DeleteNode(last);
48 }
49 void Display(Node head)
50 {
51     Node current = head->rlink;
52     if(current == head)
```

```
        return;
    }
    puts("Contents of the Circular Doubly Linked list are");
    while(current!=head)
    {
        printf("%d ", current->data);
        current = current->rlink;
    }
    puts("");
}

int main()
{
    int choice, ex=0, data;
    Node head = CreateNode(0);
    while(!ex)
    {
        printf("Enter your choice\n1 Insert front\n2 Delete rear\n3 Display\n4 Exit\n>>> ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
                printf("Enter the data: ");
                scanf("%d", &data);
                InsertFront(head, data);
                break;
            case 2:
                DeleteRear(head);
                break;
            case 3:
                Display(head);
                break;
            default:
                ex=1;
        }
    }
    return 0;
}

```

Kết quả chạy:

```
0
1  Enter your choice
2  1 Insert front
3  2 Delete rear
4  3 Display
5  4 Exit
6  >>> 1
7  Enter the data: 5
8  Enter your choice
9  1 Insert front
10 2 Delete rear
11 3 Display
12 4 Exit
13 >>> 1
14 Enter the data: 4
15 Enter your choice
16 1 Insert front
17 2 Delete rear
18 3 Display
19 4 Exit
20 >>> 1
```

```
1 Insert front
2 Delete rear
3 Display
4 Exit
>>> 1
Enter the data: 2
Enter your choice
1 Insert front
2 Delete rear
3 Display
4 Exit
>>> 3
Contents of the Circular Doubly Linked list are
2 3 4 5
Enter your choice
1 Insert front
2 Delete rear
3 Display
4 Exit
>>>
```

Như vậy, sau bài này mình đã hoàn thành phần hướng dẫn về cấu trúc dữ liệu danh sách liên kết. Ở bài viết tiếp theo, mình sẽ hướng dẫn các bạn kiến thức về cấu trúc dữ liệu Cây(Tree). Mong được các bạn quan tâm và chia sẻ cho bạn bè của mình. Thân ái!

Series Navigation

<< Cài đặt danh sách liên kết đôi trong C/C++

Cây nhị phân – Binary Tree >>