**Danh sách liên kết đôi là gì? Cấu trúc dữ liệu của nó**

**Trong hướng dẫn này, mình sẽ giới thiệu các bạn một trong các danh sách liên kết thường gặp là danh sách liên kết đôi.**



Khi các bạn nắm được thành thạo danh sách liên kết đơn thì việc học danh sách liên kết đôi rất đơn giản. Về cơ bản nó là một danh sách liên kết đơn, vì vậy các bạn hãy học [DSLK đơn](https://freetuts.net/giai-thuat/danh-sach-lien-ket-don) trước khi qua bài này nhé.

**Mục lục**

* [1. Danh sách liên kết đôi là gì?](https://freetuts.net/danh-sach-lien-ket-doi-la-gi-3000.html#goto-h2-0)
* [2. Cấu trúc dữ liệu của danh sách liên kết đôi](https://freetuts.net/danh-sach-lien-ket-doi-la-gi-3000.html#goto-h2-1)
  + [Cấu trúc của một Node](https://freetuts.net/danh-sach-lien-ket-doi-la-gi-3000.html#goto-h3-0)
  + [Cấu trúc danh sách liên kết đôi](https://freetuts.net/danh-sach-lien-ket-doi-la-gi-3000.html#goto-h3-1)
* [3. Các thao tác trong danh sách liên kết đôi](https://freetuts.net/danh-sach-lien-ket-doi-la-gi-3000.html#goto-h2-2)
* [4. Kết luận](https://freetuts.net/danh-sach-lien-ket-doi-la-gi-3000.html#goto-h2-3)

**1. Danh sách liên kết đôi là gì?**

Danh sách liên kết đôi (**Doubly linked list**) là danh sách liên kết mà mỗi phần tử có hai liên kết đến phần tử liền trước và liền sau nó.

Khi duyệt các nút sẽ thực hiện theo hai chiều về trước và về sau thay vì thực hiền duyệt một chiều như danh sách liên kết đơn.

*Bài viết này được đăng tại [free tuts .net]*



Danh sách liên kết đôi có các thông số cần quan tâm:

* Giá trị (***data***).
* Mối liên kết tới Node khác (***pPre và pNext***).

Ở mối liên kết tới Node khác, thay vì chỉ có mỗi pNext trỏ tới phần tử sau nó như DSLK đơn, thì trong DSLK đôi cần có thêm pPrev để trỏ tới phần tử trước nó.

**2. Cấu trúc dữ liệu của danh sách liên kết đôi**

Tương tự như danh sách liên kết đơn, trong DSLK đôi cũng có cấu trúc dữ liệu, đây là điều kiện cần để có thể thực hiện các thao tác trên danh sách liên kết đôi.

**Cấu trúc của một Node**



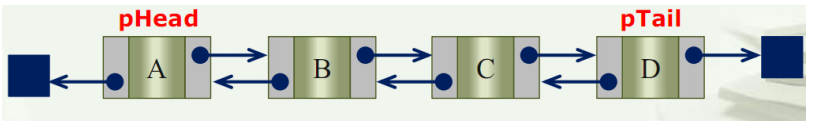
**Trong Node có:**

* **Data**là giá trị của Node.
* **pPre**là con trỏ, trỏ tới phần tử liền trước.
* **pNext**là con trỏ, trỏ tới phần tử liền sau.

Tổ chức biểu diễn một Node của danh sách liên kết đôi.

|  |  |
| --- | --- |
| 1  2  3  4  5 | struct Node  {  **int** data;      struct Node\* next;      struct Node\* prev;  }; |

**Cấu trúc danh sách liên kết đôi**



**Ta có:**

* **pHead**là Node đầu tiên trong DSLK đôi, nó luôn luôn quản lý Node đầu.
* **pTail**là Node cuối cùng trong DSLK đôi, nó luôn luôn quản lý Node cuối.
* **Node B**có hai con trỏ, trỏ đến A và C, tương tự các Node khác cũng vậy.

**Khai báo con trỏ đầu danh sách:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | /\* Khai báo Node đầu pHead và Node cuối pTail\*/  struct SingleList  {    Node \*pHead; //Node đầu pHead    Node \*pTail; // Node cuối pTail  }; |

**Khởi tạo giá trị cho pHead và pTail:**

|  |  |
| --- | --- |
| 1  2  3  4 | void Initialize(SingleList &list)  {    list.pHead=list.pTail=NULL;// khởi tạo giá trị cho Node đầu và Node cuối là Null  } |

**3. Các thao tác trong danh sách liên kết đôi**

Trong danh sách liên kết đôi cũng có các thao tác tương tự như danh sách liên kết đơn, các thao tác này được sử dụng rất nhiều khi làm việc với DSLK.

* Tạo Node mới trong DSLK đôi
* Chèn Node trong DSLK đôi
* Xoa Node trong DSLK đôi
* Duyệt DSLK đôi
* Tìm kiếm và sắp xếp trong DSLK đôi

Chèn Node trong danh sách liên kết đôi có hai trường hợp là: chèn vào đầu, chèn vào cuối.

Xóa Node trong danh sách liên kết đôi cũng có hai trường hợp là: xóa ở đầu và xóa ở cuối.

Ở DSLK đơn việc duyệt danh sách thực hiện duyệt từ đầu đến cuối, riêng DSLK đôi có thể thực hiện duyệt từ cuối về đầu.

**4. Kết luận**

Như vậy là mình đã giới thiệu về danh sách liên kết đôi, cấu trúc dữ liệu của nó và các thao tác trong danh sách. Khi làm việc với DSLK đôi, các bạn chú ý rằng nó có mối liên kết hai chiều, vì vậy các bạn hãy cận thận khi thay đổi mối liên kết của nó. Ở các bài tiếp theo mình sẽ thực hiện lần lượt các thao tác trong danh sách. Các bạn hãy chú ý theo dõi nhé!!!

**Tạo Node mới trong danh sách liên kết đôi**

Trong bài này mình sẽ giới thiệu đến các bạn cách tạo một Node trong danh sách liên kết đôi. Đây là một bước rất quan trọng để có thể thực hiện các thao tác khác.



Chúng ta sẽ cùng nhau tìm hiểu về điều kiện cần để có thể tạo được một Node và cách tạo nó như thế nào. Sau đó chúng ta sẽ viết một vài ví dụ tạo Node cụ thể.

**Mục lục**

* [1. Điều kiện cần để tạo một Node trong DLSK đôi](https://freetuts.net/tao-node-moi-trong-danh-sach-lien-ket-doi-3005.html#goto-h2-0)
* [2. Tạo Node mới trong danh sách liên kết đôi](https://freetuts.net/tao-node-moi-trong-danh-sach-lien-ket-doi-3005.html#goto-h2-1)
* [3. Ví dụ tạo Node trong C++](https://freetuts.net/tao-node-moi-trong-danh-sach-lien-ket-doi-3005.html#goto-h2-2)
* [Kết luận](https://freetuts.net/tao-node-moi-trong-danh-sach-lien-ket-doi-3005.html#goto-h2-3)

**1. Điều kiện cần để tạo một Node trong DLSK đôi**

Danh sách liên kết đôi là một cấu trúc dữ liệu, vì vậy để thao tác được với nó ta cần tạo cấu trúc dữ liệu cho nó đồng thời khởi tạo rỗng cho danh sách.

Khi các bạn đã tạo cấu trúc dữ liệu cho danh sách thì mọi thao tác thêm, sửa, xóa đều có thể thực hiện. Ở bài trước mình có giới thiệu về cấu trúc dữ liệu của nó, các bạn có thể xem lại để hiểu hơn.

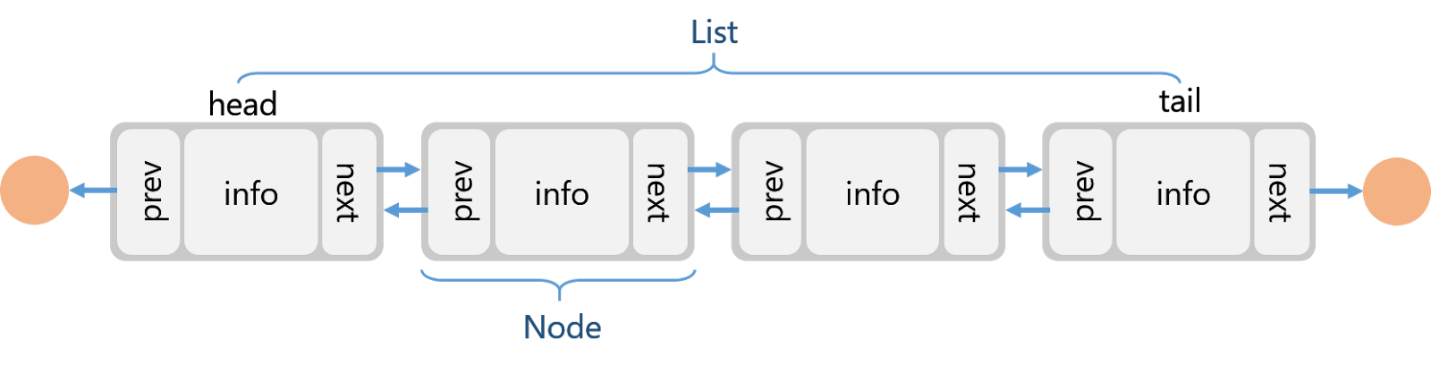
*Bài viết này được đăng tại [free tuts .net]*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | /\* tạo cấu trúc một Node bao gồm giá trị data và con trỏ pNext, pTail \*/  struct Node  {  **int** data; // giá trị của node      struct Node\* next;// con trỏ next trỏ đến phần tử sau      struct Node\* prev;// con trỏ prev trỏ đến phần tử trước  };  /\* Khai báo Node đầu pHead và Node cuối pTail\*/  struct SingleList  {    Node \*pHead; //Node đầu pHead    Node \*pTail; // Node cuối pTail  };  void Initialize(SingleList &list)  {    list.pHead=list.pTail=NULL;// khởi tạo giá trị cho Node đầu và Node cuối là Null  } |

Cụ thể chúng ta cần tạo một cấu trúc Node với hai thông số là giá trị **data**và mối liên kết (**next và prev**). Khai báo Node đầu và Node cuối cho danh sách liên kết đôi và cuối cùng là khởi tạo giá trị cho Node.

**2. Tạo Node mới trong danh sách liên kết đôi**

Khi các bạn đã tạo cấu trúc dữ liệu cho danh sách liên kết đôi thì lúc này các bạn đã có thể tạo Node mới được rồi. Việc tạo Node mới về cơ bản nó tương tự như tạo Node ở danh sách liên kết đơn.



Đầu tiên cần tạo một Node p bằng cách sử dụng cấu trúc tạo Node mà chúng ta đã tạo Node \*p = new Node

Sau khi đã có Node thì ta xét điều kiện, nếu p == null, tức là Node không có giá trị thì thoát khỏi hàm. Ngược lại nếu p != null thì thực hiện các bước sau đây:

* Cho con trỏ next trỏ về giá trị null.
* Cho con trỏ prev trỏ về giá trị null.
* Gán giá trị data của node bằng x.
* Return p.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | //Tạo mới một Node  Node \*creatNode(**int** x ){ //tao thong tin cho node      Node \*p = new Node; // tạo mới một node p      if(p == NULL) **exit**(1);// nếu p rỗng thì thoát khỏi hàm      p->next = NULL; // khi tạo mới một node thì p->next == p->prev == null      p->prev = NULL;      p->data = x;//gán giá trị data = x      return p;  } |

**3. Ví dụ tạo Node trong C++**

Ví dụ chúng ta có đề bài quản lý thông tin của một khách hàng bao gồm: tên (**char**), tuổi (**int**), quê quán (**char**).

Đầu tiên chúng ta sẽ tạo cấu trúc của khách hàng với các thông tin trên:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | /\* tạo cấu trúc khách hàng \*/  struct khachhang{  **char** ten[50]; // khai báo tên  **int** tuoi; // khai báo tuổi  **char** quequan[120]; // khai báo quê quán  }; |

Tiếp đến ta tạo cấu trúc dữ liệu của danh sách liên kết đôi, trong cấu trúc Node ta lấy giá trị từ struct khachhang.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | /\* tạo cấu trúc một Node bao gồm giá trị data và con trỏ pNext, pTail \*/  struct Node  {      khachhang \*data; // giá trị của node được lấy tù struct khachhang      struct Node\* next;// con trỏ next trỏ đến phần tử sau      struct Node\* prev;// con trỏ prev trỏ đến phần tử trước  };  /\* Khai báo Node đầu pHead và Node cuối pTail\*/  struct SingleList  {    Node \*pHead; //Node đầu pHead    Node \*pTail; // Node cuối pTail  };  //khởi tạo giá trị cho Node  void Initialize(SingleList \*&list)  {      list=new SingleList;      list->pHead=NULL;      list->pTail = NULL;  } |

Sau khi đã tạo các cấu trúc của khách hàng cũng như của danh sách liên kết đôi, ta bắt đầu thực hiện tạo một hàm nhập các thông tin của khách hàng.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | khachhang \*NhapKhachHang()  {      khachhang \*kh=new khachhang;      cout<<"Nhap ten khach hang:";  **gets**(kh->ten);        cout<<"Nhap tuoi khach hang:";      cin>>kh->tuoi;      cin.ignore();      cout<<"Nhap ten khach hang:";  **gets**(kh->quequan);      return kh;  } |

Cuối cùng ta tạo một Node khách hàng với tham số truyền vào là struct khachhang.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | //tạo node khách hàng  Node \*CreateNode(khachhang \*kh)  {      Node \*pNode=new Node;      if(pNode!=NULL)      {          pNode->data=kh;          pNode->next=NULL;      }      else      {          cout<<"cap phat bo nho that bai!!!";      }      return pNode;  } |

**Kết luận**

Như vậy là chúng ta đã tìm hiểu xong điều kiện cần để tạo Node và cách tạo Node trong danh sách liên kết đôi. Khi các bạn đã nắm rõ danh sách liên kết đơn thì việc học DSLK đôi rất dễ dàng. Ở bài tiếp theo mình sẽ hướng dẫn các bạn cách duyệt danh sách liên kết đơn để có thể duyệt và kiểm tra danh sách khi chúng ta thao tác với nó. Hãy chú ý theo dõi nhé !!!

**Duyệt danh sách liên kết đôi**

Trong bài này mình sẽ giới thiệu đến các bạn cách duyệt danh sách liên kết đôi. Đây là một bước quan trọng để kiểm tra các thao tác của bạn trên danh sách có được thực hiện như bạn mong muốn hay không.



Chúng ta sẽ cùng nhau tìm hiểu về hai cách duyệt danh sách liên kết đôi:

* Duyệt từ đầu đến cuối danh sách.
* Duyệt từ cuối về đầu danh sách.

Sỡ dĩ mình hướng dẫn bài này trước vì khi thao tác với danh sách như thêm, sửa, xóa thì các bạn có thể duyệt danh sách và kiểm tra ngay xem các thao các có đúng hay không và có gặp lỗi gì.

**Mục lục**

* [1. Duyệt từ đầu đến cuối danh sách](https://freetuts.net/duyet-danh-sach-lien-ket-doi-3006.html#goto-h2-0)
* [2. Duyệt từ cuối về đầu danh sách](https://freetuts.net/duyet-danh-sach-lien-ket-doi-3006.html#goto-h2-1)
* [3. Ví dụ về duyệt danh sách liên kết đôi](https://freetuts.net/duyet-danh-sach-lien-ket-doi-3006.html#goto-h2-2)
* [4. Kết luận](https://freetuts.net/duyet-danh-sach-lien-ket-doi-3006.html#goto-h2-3)

**1. Duyệt từ đầu đến cuối danh sách**

Trong phần này mình sẽ giới thiệu đến các bạn cách duyệt từ đầu đến cuối danh sách liên kết đôi.

*Bài viết này được đăng tại [free tuts .net]*

void Print() {

// khởi tạo một node tạm temp, sử dụng node này thay thế cho head để duyệt danh sách

struct Node\* temp = head;

cout<<"\nForward: ";

//sử dụng vòng lặp while để duyệt danh sách

while(temp != NULL) {

cout<<temp->data;

temp = temp->next;

}

cout<<endl;

}

Để duyệt được danh sách, ta cần tạo một node tạm **temp**để thay thế cho node **head.**Vì node head luôn luôn quản lý node đầu tiên vì vậy ta không thể lấy head để duyệt danh sách, thay vào đó ta dùng biến temp để duyệt.

Sử dụng vòng lặp while với điều kiện temp != null để lặp từng phần tử trong danh sách. Cụ thể, khi temp != null ta thực hiện xuất data của node đó ra màn hình và tiếp tục trỏ con trỏ next đến Node tiếp theo. Cứ như vậy lặp hết danh sách, khi đó ta sẽ có một list các phần tử trong danh sách được hiển thị ra màn hình.

**2. Duyệt từ cuối về đầu danh sách**

Thay vì duyệt từ đầu đến cuối danh sách, ta cũng có thể duyệt từ cuối về đầu danh sách liên kết đôi.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | /\* duyệt từ cuối về đầu danh sách \*/  void ReversePrint() {    //tạo một node tạm temp thay thế cho node tail để duyệt danh sách      struct Node\* temp = tail;      //nếu danh sách rỗng thì thoát khỏi hàm      if(temp == NULL) return;      cout<<"\nReverse: ";      //ngược lại nếu temp != null thì thực hiện vòng lặp while      while(temp != NULL) {          cout<<temp->data;//xuất giá trị data của node ra màn hình          temp = temp->prev;// trỏ đến node kế tiếp      }      cout<<endl;  } |

Tương tự như duyệt danh sách từ đầu đến cuối, ta cần tạo một node tạm temp để thay thế cho node tail duyệt từ cuối về đầu danh sách.Vì node tail luôn luôn quản lý node cuối nên không thể sử dụng node tail để duyệt danh sách được.

Xét điều kiện nếu temp == null, tức là danh sách rỗng thì ta return và thoát khỏi ham. Ngược lại nếu temp != null thì ta thực hiện vòng lặp while để lặp từng phần tử trong danh sách. Cứ sau mỗi lần lặp ta thực hiện trỏ đến node kế tiếp.

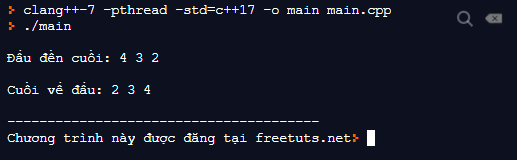
**3. Ví dụ về duyệt danh sách liên kết đôi**

Trong ví dụ này mình sẽ sử dụng một hàm chèn Node vào đầu đã viết sẵn để thực hiện duyệt cho các bạn xem, ở bài tiếp theo mình sẽ giải thích cụ thể hơn về hàm này. Mình sẽ duyệt danh sách theo hai chiều từ đầu đến cuối và từ cuối về đầu.

**Code mẫu:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  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 | #include <iostream>  using namespace std;  /\* tạo cấu trúc node \*/  struct Node  {  **int** data;      struct Node\* next;      struct Node\* prev;  };    struct Node \*head, \*tail; // Khởi tạo Node head global của dslk đôi.    /\* tạo node mới \*/  struct Node\* CreateNode(**int** x) {      struct Node\* newNode          = (struct Node\*)**malloc**(sizeof(struct Node));      newNode->data = x;      newNode->prev = NULL;      newNode->next = NULL;      return newNode;  }  /\* hiển thị từ đầu đến cuối \*/  void Print() {      struct Node\* temp = head;  **printf**("\nĐầu đến cuối: ");      while(temp != NULL) {  **printf**("%d ",temp->data);          temp = temp->next;      }  **printf**("\n");  }    /\* hiển thị từ cuối về đầu \*/  void ReversePrint() {      struct Node\* temp = tail;      if(temp == NULL) return; // empty list, exit      // Traversing backward using prev pointer  **printf**("\nCuối về đầu: ");      while(temp != NULL) {  **printf**("%d ",temp->data);          temp = temp->prev;      }  **printf**("\n");  }  /\* thêm node vào đầu danh sách \*/  void InsertAtHead(**int** x) {      struct Node\* newNode = CreateNode(x);      if(head == NULL) {          head = newNode;          tail = newNode;          return;      }      head->prev = newNode;      newNode->next = head;      head = newNode;  }  **int** main() {        head = NULL;   //gọi hàm thêm node vào đầu danh sách      InsertAtHead(2);      InsertAtHead(3);      InsertAtHead(4);      Print(); ReversePrint();        cout<<"\n---------------------------------------\n";      cout<<"Chương trình này được đăng tại freetuts.net";  } |

**Kết quả:**



**4. Kết luận**

Như vậy là chúng ta đã tìm hiểu xong cách duyệt danh sách từ đầu đến cuối và tử cuối về đầu. Đây là một bước rất quan trọng để các bạn có thể kiểm tra được các thao tác trên danh sách có đúng hay không và có lỗi gì để tìm cách khắc phục. Ở bài tiếp theo mình sẽ hướng dẫn cách thêm Node vào danh sách, các bạn chú ý theo dõi nhé !!!

**Chèn Node vào danh sách liên kết đôi**

**Trong hướng dẫn này mình sẽ giới thiệu các bạn cách chèn Node vào danh sách liên kết đôi. Đây làm một bước khá quan trọng trong việc thêm dữ liệu vào danh sách.**



Chúng ta sẽ cùng nhau tìm hiểu hai cách thêm Node vào danh sách liên kết đôi

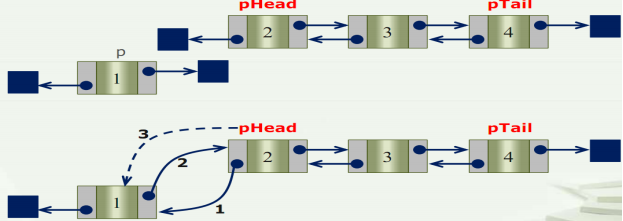
* Thêm Node vào đầu danh sách.
* Thêm Node vào cuối danh sách.

**Mục lục**

* [1. Thêm Node vào đầu danh sách](https://freetuts.net/chen-node-vao-danh-sach-lien-ket-doi-3007.html#goto-h2-0)
* [2. Thêm Node vào cuối danh sách](https://freetuts.net/chen-node-vao-danh-sach-lien-ket-doi-3007.html#goto-h2-1)
* [3. Ví dụ thêm Node vào danh sách liên kết đôi](https://freetuts.net/chen-node-vao-danh-sach-lien-ket-doi-3007.html#goto-h2-2)
* [4. Kết luận](https://freetuts.net/chen-node-vao-danh-sach-lien-ket-doi-3007.html#goto-h2-3)

**1. Thêm Node vào đầu danh sách**

Trong phần này chúng ta sẽ thêm Node vào đầu danh sách (**insertFirst**). Về cơ bản thì nó tương tự như danh sách liên kết đơn, cùng mình tìm hiểu xem cách thêm nó như thế nào nhé.



Đầu tiên ta sử dụng hàm CreateNode() để tạo một Node mới newNode.

*Bài viết này được đăng tại [free tuts .net]*

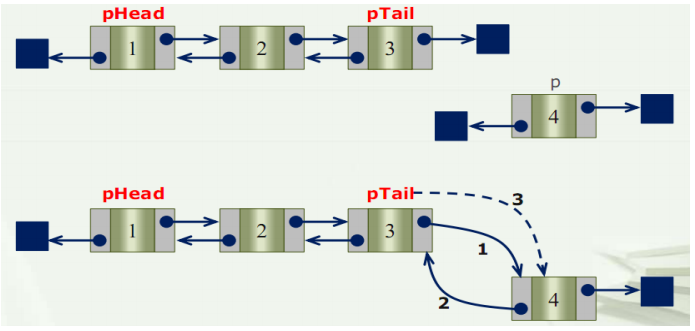
Tiếp đến ta xét điều kiện, nếu danh sách rỗng thì newNode vừa là Node đầu và Node cuối: head = newNode và tail = newNode. Nếu trong danh sách có phần tử thì ta thực hiện chèn vào đầu danh sách bằng cách:

* Thiết lập con trỏ prev của head trỏ đến newNode: head -> prev = newNode
* Thiết lập con trỏ next của newNode trỏ đến head: newNode -> next = head
* Dịch chuyển Node head về newNode, vì head luôn luôn quản lý Node đầu tiên trong danh sách: head = newNode

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | /\* thêm node vào đầu danh sách \*/  void InsertAtHead(**int** x) {    //sử dụng hàm tạo Node tạo một Node mới      struct Node\* newNode = CreateNode(x);     //nếu danh sách rỗng thì node chèn vào vừa là node đầu vừa là node cuối      if(head == NULL) {          head = newNode;          tail = newNode;          return;      }      //Nếu danh sách không rỗng thì, dịch chuyển Node head về node mới chèn, và cho con trỏ của newNode trỏ đến Node head      head->prev = newNode;      newNode->next = head;      head = newNode;  } |

**2. Thêm Node vào cuối danh sách**

Trong phần này chúng ta sẽ thực hiện chèn Node vào cuối danh sách, đây là cách thường được sử dụng rất nhiều.



Tương tự như thêm Node vào đầu danh sách, ta sử dụng hàm CreateNode() để tạo một Node mới newNode.

Sau đó cũng xét điều kiện, nếu danh sách rỗng thì newNode vừa là head vừa là tail. Nếu danh sách có phần tử thì ta thực hiện:

* Thiết lập con trỏ next của tail trỏ đến newNode.
* Thiết lập con trỏ prev của newNode trỏ đến tail.
* Dịch chuyển tail về newNode, vì tail luôn luôn quản lý Node cuối cùng.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | /\* thêm node vào cuối danh sách \*/  void InsertLast(**int** x) {    //sử dụng hàm tạo Node để tạo node mới newNode      struct Node\* newNode = CreateNode(x);      //Nếu danh sách rỗng thì newNode vừa là node đầu vừa là node cuối      if(head == NULL) {          head = newNode;          tail = newNode;          return;      }      //Nếu danh sách có phần tử thì,      tail->next = newNode;// con trỏ next của tail trỏ đến newNode      newNode->prev = tail;// con trỏ prev của newNode trỏ đến tail      tail = newNode;//dịch chuyển tail về newNode, vì tail luôn luôn quản lý phần tử cuối cùng trong danh sách  } |

**3. Ví dụ thêm Node vào danh sách liên kết đôi**

Trong ví dụ này chúng ta sẽ thực hiện một chương trình tạo một danh sách liên kết đôi với các thao tác: tạo, thêm vào đầu, thêm vào cuối, duyệt danh sách theo hai chiều.

Đầu tiên ta cần tạo cấu trúc dữ liệu cho danh sách liên kết đôi.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | /\* tạo cấu trúc node \*/  struct Node  {  **int** data;      struct Node\* next;      struct Node\* prev;  };    struct Node \*head, \*tail; // Khởi tạo Node head global của dslk đôi.    /\* tạo node mới \*/  struct Node\* CreateNode(**int** x) {      struct Node\* newNode          = (struct Node\*)**malloc**(sizeof(struct Node));      newNode->data = x;      newNode->prev = NULL;      newNode->next = NULL;      return newNode;  } |

Tiếp đến ta viết hàm tạo Node CreateNode().

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | /\* tạo node mới \*/  struct Node\* CreateNode(**int** x) {      struct Node\* newNode          = (struct Node\*)**malloc**(sizeof(struct Node));      newNode->data = x;      newNode->prev = NULL;      newNode->next = NULL;      return newNode;  } |

Để kiểm tra được kết quả của chương trình ta cần một hàm duyệt danh sách: duyệt từ đầu đến cuối (Print()) và duyệt từ cuối về đầu (ReversePrint()).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | /\* hiển thị từ cuối về đầu \*/  void ReversePrint() {      struct Node\* temp = tail;      if(temp == NULL) return; // empty list, exit      // Traversing backward using prev pointer  **printf**("\nCuối về đầu: ");      while(temp != NULL) {  **printf**("%d ",temp->data);          temp = temp->prev;      }  **printf**("\n");  }  /\* thêm node vào đầu danh sách \*/  void InsertFirst(**int** x) {    //sử dụng hàm tạo Node tạo một Node mới      struct Node\* newNode = CreateNode(x);     //nếu danh sách rỗng thì node chèn vào vừa là node đầu vừa là node cuối      if(head == NULL) {          head = newNode;          tail = newNode;          return;      }      //Nếu danh sách không rỗng thì, dịch chuyển Node head về node mới chèn, và cho con trỏ của newNode trỏ đến Node head      head->prev = newNode;      newNode->next = head;      head = newNode;  } |

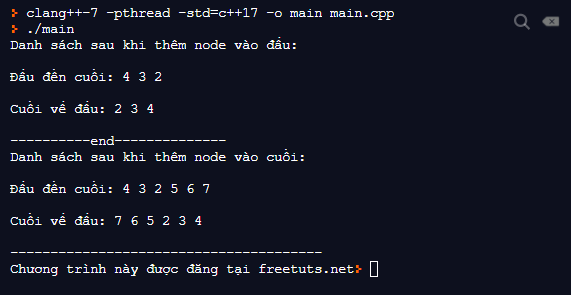
Và cuối cùng là hai cách thêm Node vào danh sách mà ta đã tìm hiểu ở trên.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | /\* thêm node vào đầu danh sách \*/  void InsertFirst(**int** x) {    //sử dụng hàm tạo Node tạo một Node mới      struct Node\* newNode = CreateNode(x);     //nếu danh sách rỗng thì node chèn vào vừa là node đầu vừa là node cuối      if(head == NULL) {          head = newNode;          tail = newNode;          return;      }      //Nếu danh sách không rỗng thì, dịch chuyển Node head về node mới chèn, và cho con trỏ của newNode trỏ đến Node head      head->prev = newNode;      newNode->next = head;      head = newNode;  }  /\* thêm node vào cuối danh sách \*/  void InsertLast(**int** x) {    //sử dụng hàm tạo Node để tạo node mới newNode      struct Node\* newNode = CreateNode(x);      //Nếu danh sách rỗng thì newNode vừa là node đầu vừa là node cuối      if(head == NULL) {          head = newNode;          tail = newNode;          return;      }      //Nếu danh sách có phần tử thì,      tail->next = newNode;// con trỏ next của tail trỏ đến newNode      newNode->prev = tail;// con trỏ prev của newNode trỏ đến tail      tail = newNode;//dịch chuyển tail về newNode, vì tail luôn luôn quản lý phần tử cuối cùng trong danh sách  } |

**Full code:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  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 | #include <iostream>  using namespace std;  /\* tạo cấu trúc node \*/  struct Node  {  **int** data;      struct Node\* next;      struct Node\* prev;  };    struct Node \*head, \*tail; // Khởi tạo Node head global của dslk đôi.    /\* tạo node mới \*/  struct Node\* CreateNode(**int** x) {      struct Node\* newNode          = (struct Node\*)**malloc**(sizeof(struct Node));      newNode->data = x;      newNode->prev = NULL;      newNode->next = NULL;      return newNode;  }  /\* hiển thị từ đầu đến cuối \*/  void Print() {      struct Node\* temp = head;  **printf**("\nĐầu đến cuối: ");      while(temp != NULL) {  **printf**("%d ",temp->data);          temp = temp->next;      }  **printf**("\n");  }    /\* hiển thị từ cuối về đầu \*/  void ReversePrint() {      struct Node\* temp = tail;      if(temp == NULL) return; // empty list, exit      // Traversing backward using prev pointer  **printf**("\nCuối về đầu: ");      while(temp != NULL) {  **printf**("%d ",temp->data);          temp = temp->prev;      }  **printf**("\n");  }  /\* thêm node vào đầu danh sách \*/  void InsertFirst(**int** x) {    //sử dụng hàm tạo Node tạo một Node mới      struct Node\* newNode = CreateNode(x);     //nếu danh sách rỗng thì node chèn vào vừa là node đầu vừa là node cuối      if(head == NULL) {          head = newNode;          tail = newNode;          return;      }      //Nếu danh sách không rỗng thì, dịch chuyển Node head về node mới chèn, và cho con trỏ của newNode trỏ đến Node head      head->prev = newNode;      newNode->next = head;      head = newNode;  }  /\* thêm node vào cuối danh sách \*/  void InsertLast(**int** x) {    //sử dụng hàm tạo Node để tạo node mới newNode      struct Node\* newNode = CreateNode(x);      //Nếu danh sách rỗng thì newNode vừa là node đầu vừa là node cuối      if(head == NULL) {          head = newNode;          tail = newNode;          return;      }      //Nếu danh sách có phần tử thì,      tail->next = newNode;// con trỏ next của tail trỏ đến newNode      newNode->prev = tail;// con trỏ prev của newNode trỏ đến tail      tail = newNode;//dịch chuyển tail về newNode, vì tail luôn luôn quản lý phần tử cuối cùng trong danh sách  }  **int** main() {        head = NULL;   //gọi hàm thêm node vào đầu danh sách        InsertFirst(2);      InsertFirst(3);      InsertFirst(4);      cout<<"Danh sách sau khi thêm node vào đầu: \n";      Print();      ReversePrint();      cout<<"\n----------end--------------\n";   //gọi hàm thêm node vào cuối danh sách      InsertLast(5);      InsertLast(6);      InsertLast(7);      cout<<"Danh sách sau khi thêm node vào cuối: \n";      Print();      ReversePrint();        cout<<"\n---------------------------------------\n";      cout<<"Chương trình này được đăng tại freetuts.net";  } |

**Kết quả:**



**4. Kết luận**

Như vậy là chúng ta đã tìm hiểu xong hai cách thêm Node vào danh sách liên kết đôi đó là thêm vào đầu danh sách và thêm vào cuối danh sách. Cũng như thực hiện chương trình thêm Node vào danh sách trong C++. Ở bài tiếp theo mình sẽ hướng dẫn các bạn cách xóa Node trong danh sách liên kết đôi, hãy chú ý theo dõi nhé !!!

**Xóa Node trong danh sách liên kết đôi**

Trong hướng dẫn này mình sẽ giới thiệu các bạn cách xóa Node trong danh sách liên kết đôi. Đây là một bước khi chúng ta cần xóa một Node nào đó khỏi danh sách theo yêu cầu của bài toán chẳng hạn.



Chúng ta sẽ cùng nhau tìm hiểu về hai cách xóa Node trong danh sách liên kết đôi:

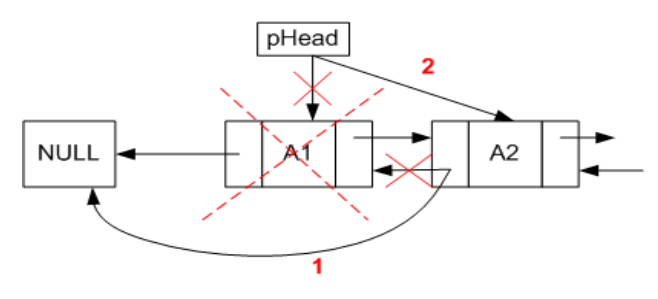
* Xóa Node đầu trong danh sách
* Xóa Node cuối trong danh sách

**Mục lục**

* [1. Xóa Node đầu trong danh sách](https://freetuts.net/xoa-node-trong-danh-sach-lien-ket-doi-3008.html#goto-h2-0)
* [2. Xóa Node cuối trong danh sách](https://freetuts.net/xoa-node-trong-danh-sach-lien-ket-doi-3008.html#goto-h2-1)
* [3. Ví dụ xóa Node trong danh sách liên kết đôi](https://freetuts.net/xoa-node-trong-danh-sach-lien-ket-doi-3008.html#goto-h2-2)
* [4. Kết luận](https://freetuts.net/xoa-node-trong-danh-sach-lien-ket-doi-3008.html#goto-h2-3)

**1. Xóa Node đầu trong danh sách**

Trong phần này chúng ta sẽ tìm hiểu về cách xóa Node đầu trong danh sách liên kết đôi.



Trước khi xóa Node đầu khỏi danh sách ta cần kiểm tra xem trong danh sách có phần tử nào hay không. Nếu danh sách rỗng thì ta return rồi thoát khỏi hàm, ngược lại nếu danh sách có phần tử thì ta thực hiện các bước sau:

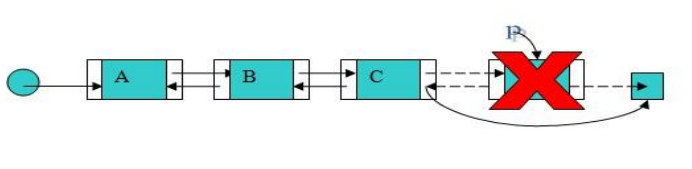
*Bài viết này được đăng tại [free tuts .net]*

* Dịch chuyển Node head đến Node kế tiếp: head = head -> next
* Cho con trỏ prev của head trỏ đến null: head -> prev = NULL

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | /\* Xóa node đầu trong danh sách \*/  void DelFirst() {    //Nếu danh sách rỗng thì return và thoát khỏi hàm      if(head == NULL) {          return;      }      //ngược lại nếu danh sách có phần tử thì      head = head->next;//thực hiện dịch chuyển node head đến node kế tiếp      head->prev = NULL;//sau đó cho con trỏ prev của head = null  } |

**2. Xóa Node cuối trong danh sách**

Trong phần này mình sẽ giới thiệu các bạn cách xóa Node cuối trong danh sách liên kết đôi.



Tương tự như xóa Node đầu trong danh sách, đầu tiên ta cần kiểm tra xem danh sách có phần tử hay không. Nếu không thì ta return rồi thoát khỏi hàm, ngược lại nếu danh sách có phần tử thì ta thực hiện các bước sau:

* Dịch chuyển Node tail về Node trước đó: tail = tail -> prev
* Cho con trỏ next của tail trỏ về null: tail -> next = NULL

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | /\* Xóa node cuối trong danh sách \*/  void DelLast() {    //Nếu danh sách rỗng thì return và thoát khỏi hàm      if(head == NULL) {          return;      }      //Ngược lại nếu danh sách có phần tử thì      tail = tail->prev;//thực hiện dịch chuyển tail về node trước đó      tail->next = NULL;//sau đó cho con trỏ next của tail == null  } |

**3. Ví dụ xóa Node trong danh sách liên kết đôi**

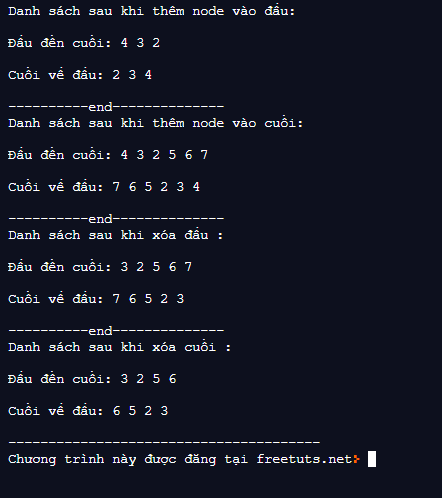
Trong ví dụ này ta sẽ thực hiện thêm Node vào danh sách, sau đó thực hiện xóa Node đầu và Node cuối trong danh sách. Để tiết kiệm thời gian mình lấy lại ví dụ ở bài trước (thêm Node trong danh sách) làm dữ liệu để thực hiện xóa Node.

Sau khi viết hàm xóa Node đầu DelFirst() và xóa Node cuối DelLast(), ta chỉ cần gọi nó và thực hiện xóa.

**Full code:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  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  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 | #include <iostream>  using namespace std;  /\* tạo cấu trúc node \*/  struct Node  {  **int** data;      struct Node\* next;      struct Node\* prev;  };    struct Node \*head, \*tail; // Khởi tạo Node head global của dslk đôi.    /\* tạo node mới \*/  struct Node\* CreateNode(**int** x) {      struct Node\* newNode          = (struct Node\*)**malloc**(sizeof(struct Node));      newNode->data = x;      newNode->prev = NULL;      newNode->next = NULL;      return newNode;  }  /\* hiển thị từ đầu đến cuối \*/  void Print() {      struct Node\* temp = head;  **printf**("\nĐầu đến cuối: ");      while(temp != NULL) {  **printf**("%d ",temp->data);          temp = temp->next;      }  **printf**("\n");  }    /\* hiển thị từ cuối về đầu \*/  void ReversePrint() {      struct Node\* temp = tail;      if(temp == NULL) return; // empty list, exit      // Traversing backward using prev pointer  **printf**("\nCuối về đầu: ");      while(temp != NULL) {  **printf**("%d ",temp->data);          temp = temp->prev;      }  **printf**("\n");  }  /\* thêm node vào đầu danh sách \*/  void InsertFirst(**int** x) {    //sử dụng hàm tạo Node tạo một Node mới      struct Node\* newNode = CreateNode(x);     //nếu danh sách rỗng thì node chèn vào vừa là node đầu vừa là node cuối      if(head == NULL) {          head = newNode;          tail = newNode;          return;      }      //Nếu danh sách không rỗng thì, dịch chuyển Node head về node mới chèn, và cho con trỏ của newNode trỏ đến Node head      head->prev = newNode;      newNode->next = head;      head = newNode;  }  /\* thêm node vào cuối danh sách \*/  void InsertLast(**int** x) {    //sử dụng hàm tạo Node để tạo node mới newNode      struct Node\* newNode = CreateNode(x);      //Nếu danh sách rỗng thì newNode vừa là node đầu vừa là node cuối      if(head == NULL) {          head = newNode;          tail = newNode;          return;      }      //Nếu danh sách có phần tử thì,      tail->next = newNode;// con trỏ next của tail trỏ đến newNode      newNode->prev = tail;// con trỏ prev của newNode trỏ đến tail      tail = newNode;//dịch chuyển tail về newNode, vì tail luôn luôn quản lý phần tử cuối cùng trong danh sách  }  /\* Xóa node đầu trong danh sách \*/  void DelFirst() {    //Nếu danh sách rỗng thì return và thoát khỏi hàm      if(head == NULL) {          return;      }      //ngược lại nếu danh sách có phần tử thì      head = head->next;//thực hiện dịch chuyển node head đến node kế tiếp      head->prev = NULL;//sau đó cho con trỏ prev của head = null  }  /\* Xóa node cuối trong danh sách \*/  void DelLast() {    //Nếu danh sách rỗng thì return và thoát khỏi hàm      if(head == NULL) {          return;      }      //Ngược lại nếu danh sách có phần tử thì      tail = tail->prev;//thực hiện dịch chuyển tail về node trước đó      tail->next = NULL;//sau đó cho con trỏ next của tail == null  }  **int** main() {        head = NULL;   //gọi hàm thêm node vào đầu danh sách        InsertFirst(2);      InsertFirst(3);      InsertFirst(4);      cout<<"Danh sách sau khi thêm node vào đầu: \n";      Print();      ReversePrint();      cout<<"\n----------end--------------\n";   //gọi hàm thêm node vào cuối danh sách      InsertLast(5);      InsertLast(6);      InsertLast(7);      cout<<"Danh sách sau khi thêm node vào cuối: \n";      Print();      ReversePrint();      cout<<"\n----------end--------------\n";    //gọi hàm xóa đầu danh sách      DelFirst();      cout<<"Danh sách sau khi xóa đầu : \n";      Print();      ReversePrint();      cout<<"\n----------end--------------\n";    //gọi hàm xóa cuối danh sách      DelLast();      cout<<"Danh sách sau khi xóa cuối : \n";      Print();      ReversePrint();        cout<<"\n---------------------------------------\n";      cout<<"Chương trình này được đăng tại freetuts.net";  } |

**Kết quả:**



**4. Kết luận**

Như vậy là chúng ta đã tìm hiểu xong cách xóa Node đầu và xóa Node cuối trong danh sách liên kết đôi. Cũng như áp dụng các thao tác thêm, xóa Node vào ví dụ cụ thể. Ở bài tiếp theo chúng ta sẽ thực hiện tìm kiếm và sắp xếp danh sách liên kết đôi, các bạn hãy chú ý theo dõi nhé !!!

**Tìm kiếm phần tử k trong danh sách liên kết đôi**

Trong hướng dẫn này mình sẽ giới thiệu các bạn cách tìm kiếm phần tử k trong danh sách liên kết đôi. Đây là một trong các thao tác phổ biến nhất và được sử dụng rất nhiều trong lập trình.



Chúng ta sẽ cùng nhau tìm hiểu cách tìm kiếm một phần tử k trong danh sách. Sau đó áp dụng chúng vào ví dụ cụ thể trong C++.

**Mục lục**

* [1. Tìm kiếm phần tử k trong DSLK đôi](https://freetuts.net/tim-kiem-phan-tu-k-trong-danh-sach-lien-ket-doi-3009.html#goto-h2-0)
* [2. Ví dụ tìm kiếm phần tử k trong DSLK đôi](https://freetuts.net/tim-kiem-phan-tu-k-trong-danh-sach-lien-ket-doi-3009.html#goto-h2-1)
* [3. Kết luận](https://freetuts.net/tim-kiem-phan-tu-k-trong-danh-sach-lien-ket-doi-3009.html#goto-h2-2)

**1. Tìm kiếm phần tử k trong DSLK đôi**

Trong phần này chúng ta sẽ cùng nhau tìm hiểu về cách tìm phần tử k trong danh sách liên kết đôi. Người dùng nhập vào số cần tìm k và hàm thực hiện tìm kiếm xem số k đó có trong danh sách hay không. Nếu có thì trả về true và ngược lại nếu không có thì trả về false.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | /\* Tìm kiếm phần tử k trong danh sách \*/  **bool** search(Node l, **int** k ){ // thanm số truyền vào bao gồm một danh sách và phần tử k      Node \*p = head;//tạo một node tạm p để thay thế cho node head    //sử dụng vòng lặp while để lặp từng phần tử trong danh sách      while(p!=NULL) {          if(p->data == k )return true;//nếu giá trị tại node hiện tại == k thì return true          else p = p->next;//Nếu không == k thì trỏ đến node kế tiếp    }    return false;//Kết thúc vòng lặp vẫn ko tìm thấy thì return false  } |

Tham số truyền vào của hàm bao gồm: danh sách liên kết đôi **l** và số **k**cần tìm. Ta cần tạo một Node p để thay thế cho Node head thực hiện duyệt từng phần tử trong danh sách, vì Node head luôn luôn quản lý node đầu nên không thể sử dụng head để duyệt.

*Bài viết này được đăng tại [free tuts .net]*

Để duyệt từng phần tử trong danh sách ta sử dụng vòng lặp while với điều kiện p != NULL. Trong vòng lặp nếu giá trị data của node hiện tại == k thì return true, ngược lại nếu != k thì ta trỏ đến Node kế tiếp rồi thực hiện so sánh.

Cứ như vậy lặp cho đến hết danh sách, nếu vẫn không tìm thấy thì ta return false và kết thúc hàm.

**2. Ví dụ tìm kiếm phần tử k trong DSLK đôi**

Trong ví dụ này chúng ta sẽ thực hiện tìm kiếm một số k xem có tồn tại trong danh sách hay không. Nếu nó thì xuất ra đã tìm thấy, nếu không thì xuất ra không tìm thấy cho người dùng biết.

Đầu tiên ta cần có cấu trúc dữ liệu của danh sách liên kết đôi và khởi tạo cho cho danh sách.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | /\* tạo cấu trúc node \*/  struct Node  {  **int** data;      struct Node\* next;      struct Node\* prev;  };  struct Node \*head, \*tail; // Khởi tạo Node head global của dslk đôi.    /\* tạo node mới \*/  struct Node\* CreateNode(**int** x) {      struct Node\* newNode          = (struct Node\*)**malloc**(sizeof(struct Node));      newNode->data = x;      newNode->prev = NULL;      newNode->next = NULL;      return newNode;  } |

Tiếp đến ta cần một hàm InsertLast() để thêm các Node vào danh sách, khi đó ta mới có dữ liệu để thực hiện tìm kiếm.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | /\* thêm node vào cuối danh sách \*/  void InsertLast(**int** x) {    //sử dụng hàm tạo Node để tạo node mới newNode      struct Node\* newNode = CreateNode(x);      //Nếu danh sách rỗng thì newNode vừa là node đầu vừa là node cuối      if(head == NULL) {          head = newNode;          tail = newNode;          return;      }      //Nếu danh sách có phần tử thì,      tail->next = newNode;// con trỏ next của tail trỏ đến newNode      newNode->prev = tail;// con trỏ prev của newNode trỏ đến tail      tail = newNode;//dịch chuyển tail về newNode, vì tail luôn luôn quản lý phần tử cuối cùng trong danh sách  } |

Sau đó ta viết hàm duyệt danh sách Print() để hiển thị danh sách ra màn hình.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | /\* hiển thị từ đầu đến cuối \*/  void Print() {      struct Node\* temp = head;  **printf**("\nĐầu đến cuối: ");      while(temp != NULL) {  **printf**("%d ",temp->data);          temp = temp->next;      }  **printf**("\n");  }    /\* hiển thị từ cuối về đầu \*/  void ReversePrint() {      struct Node\* temp = tail;      if(temp == NULL) return; // empty list, exit      // Traversing backward using prev pointer  **printf**("\nCuối về đầu: ");      while(temp != NULL) {  **printf**("%d ",temp->data);          temp = temp->prev;      }  **printf**("\n");  } |

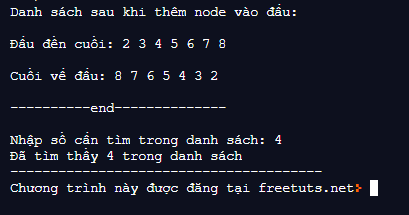
Và cuối cùng là hàm tiềm kiếm search() để thực hiện tìm kiếm số k được nhập từ bàn phím.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | **bool** search(Node l, **int** k ){      Node \*p = head;        while(p!=NULL) {          if(p->data == k )return true;          else p = p->next;    }    return false;  } |

**Full code:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  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  101  102  103  104  105  106  107  108  109  110  111  112  113 | #include <iostream>  using namespace std;  /\* tạo cấu trúc node \*/  struct Node  {  **int** data;      struct Node\* next;      struct Node\* prev;  };  struct Node \*head, \*tail; // Khởi tạo Node head global của dslk đôi.    /\* tạo node mới \*/  struct Node\* CreateNode(**int** x) {      struct Node\* newNode          = (struct Node\*)**malloc**(sizeof(struct Node));      newNode->data = x;      newNode->prev = NULL;      newNode->next = NULL;      return newNode;  }  /\* hiển thị từ đầu đến cuối \*/  void Print() {      struct Node\* temp = head;  **printf**("\nĐầu đến cuối: ");      while(temp != NULL) {  **printf**("%d ",temp->data);          temp = temp->next;      }  **printf**("\n");  }    /\* hiển thị từ cuối về đầu \*/  void ReversePrint() {      struct Node\* temp = tail;      if(temp == NULL) return; // empty list, exit      // Traversing backward using prev pointer  **printf**("\nCuối về đầu: ");      while(temp != NULL) {  **printf**("%d ",temp->data);          temp = temp->prev;      }  **printf**("\n");  }  /\* thêm node vào đầu danh sách \*/  void InsertFirst(**int** x) {    //sử dụng hàm tạo Node tạo một Node mới      struct Node\* newNode = CreateNode(x);     //nếu danh sách rỗng thì node chèn vào vừa là node đầu vừa là node cuối      if(head == NULL) {          head = newNode;          tail = newNode;          return;      }      //Nếu danh sách không rỗng thì, dịch chuyển Node head về node mới chèn, và cho con trỏ của newNode trỏ đến Node head      head->prev = newNode;      newNode->next = head;      head = newNode;  }  /\* thêm node vào cuối danh sách \*/  void InsertLast(**int** x) {    //sử dụng hàm tạo Node để tạo node mới newNode      struct Node\* newNode = CreateNode(x);      //Nếu danh sách rỗng thì newNode vừa là node đầu vừa là node cuối      if(head == NULL) {          head = newNode;          tail = newNode;          return;      }      //Nếu danh sách có phần tử thì,      tail->next = newNode;// con trỏ next của tail trỏ đến newNode      newNode->prev = tail;// con trỏ prev của newNode trỏ đến tail      tail = newNode;//dịch chuyển tail về newNode, vì tail luôn luôn quản lý phần tử cuối cùng trong danh sách  }    **bool** search(Node l, **int** k ){      Node \*p = head;        while(p!=NULL) {          if(p->data == k )return true;          else p = p->next;    }    return false;  }    **int** main() {      Node l;  **int** k, x, lc;      head = NULL;   //gọi hàm thêm node vào đầu danh sách        InsertLast(2);      InsertLast(3);      InsertLast(4);      InsertLast(5);      InsertLast(6);      InsertLast(7);      InsertLast(8);      cout<<"Danh sách sau khi thêm node vào đầu: \n";      Print();      ReversePrint();      cout<<"\n----------end--------------\n";      //tìm kiếm node      cout<<"\nNhập số cần tìm trong danh sách: "; cin>> k;      if(search(l,k) == true){        cout<<"Đã tìm thấy "<<k<<" trong danh sách";      }      else{        cout<<"không tìm thấy "<<k<<" trong danh sách";      }        cout<<"\n---------------------------------------\n";      cout<<"Chương trình này được đăng tại freetuts.net";  } |

**Kết quả:**



**3. Kết luận**

Như vậy là chúng ta đã tìm hiểu xong cách tìm một số k trong danh sách liên kết đôi. Như đã nói đây là một thao tác được sử dụng rất nhiều trong lập trình, vì vậy các bạn hãy chú ý và thực hiện thật thành thạo nó nhé. Ở bài tiếp theo mình sẽ thực hiện một vài bài tập liên quan đến danh sách liên kết đôi.

**Gộp hai danh sách liên kết đôi**

Trong hướng dẫn này mình sẽ giới thiệu đến các bạn cách nối hai danh sách liên kết đôi thành một danh sách liên kết đôi khác.



Chúng ta sẽ cùng nhau tìm hiểu về cách nối hai danh sách liên kết đôi. Để làm được bài này các bạn cần nắm vững kiến thức về danh sách liên kết đôi. Các thao tác tạo cấu trúc dữ liệu, thêm, duyệt danh sách liên kết đôi.

**Mục lục**

* [1. Gợi ý cách thực hiện](https://freetuts.net/gop-hai-danh-sach-lien-ket-doi-3010.html#goto-h2-0)
* [2. Nối hai danh sách liên kết đôi](https://freetuts.net/gop-hai-danh-sach-lien-ket-doi-3010.html#goto-h2-1)
* [3. Kết luận](https://freetuts.net/gop-hai-danh-sach-lien-ket-doi-3010.html#goto-h2-2)

**1. Gợi ý cách thực hiện**

Trong bài toán ta cần nối hai danh sách liên kết đôi thành một danh sách liên kết đôi khác. Cụ thể ta sẽ nối hai danh sách, một danh sách là số chẵn và một danh sách là số lẻ.

* Tạo cấu trúc Node.
* Tạo cấu trúc dữ liệu cho các danh sách số chẵn, số lẽ và danh sách để nối hai danh sách này.
* Tạo danh sách liên kết đôi với thao tác thêm Node vào cuối danh sách.
* Viết hàm hiển thị danh sách.
* Viết hàm nối danh sách.
* Hàm main để chạy chương trình và kiểm tra kết quả.

**2. Nối hai danh sách liên kết đôi**

Chúng ta sẽ thực hiện lần lượt các bước như mình đã gợi ý ở trên. Như đã nói, để hiểu được các bạn cần có kiến thức về danh sách liên kết.

*Bài viết này được đăng tại [free tuts .net]*

Đầu tiên ta tạo cấu trúc cho Node với giá trị data và cỏn trỏ next, prev.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | /\* tạo cấu trúc node \*/  struct node {  **int** data;     struct node \*prev;// tạo con trỏ prev trỏ về phần tử phía trước     struct node \*next;// tạo con trỏ next trỏ về phần tử phía sau  }; |

Tiếp đến ta cần tạo cấu trúc dữ liệu cho các danh sách liên kết đôi. Ở đây ta cần ba danh sách, một cho các số lẻ, một cho các số chẵn và danh sách còn lại để gộp hai danh sách này.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | /\* tạo cấu trúc dữ liệu cho danh sách sau khi nối \*/  struct node \*list = NULL;  struct node \*list\_last = NULL;  /\* tạo cấu trúc dữ liệu cho danh sách các số chẵn \*/  struct node \*even = NULL;  struct node \*even\_last = NULL;  /\* tạo cấu trúc dữ liệu cho danh sách các số lẻ \*/  struct node \*odd = NULL;  struct node \*odd\_last = NULL;  /\* tạo cấu trúc dữ liệu node hiện tại \*/  struct node \*current = NULL; |

Sau khi tạo xong cấu trúc dữ liệu cho danh sách, ta thực hiện tạo danh sách liên kết đôi và khởi tạo giá trị cho danh sách. Trong hàm này mình sử dụng InsertLast để thêm Node vào danh sách, đây là cách thường dùng khi muốn thêm node.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | // Nếu head trống thì tạo list mới     if(data%2 == 0) {        if(even == NULL) {           even = link;           return;        }else {           current = even;             while(current->next != NULL)              current = current->next;             // chèn node vào cuối danh sách           current->next = link;           even\_last = link;           link->prev = current;        }     }else {        if(odd == NULL) {           odd = link;           return;        }else {           current = odd;             while(current->next!=NULL)              current = current->next;           // chèn node vào cuối danh sách           current->next = link;           odd\_last = link;           link->prev = current;        }     }  } |

Để hiển thị danh sách ta cần tạo một hàm hiển thị, trong phần này mình viết hàm duyệt danh sách từ đầu đến cuối.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | /\* hiển thị danh sách \*/  void printList(struct node \*head) {     struct node \*ptr = head;       cout<<"\n[head] \t";     //dùng vòng lặp while lặp từ phần tử đầu đến phần tử cuối của list     while(ptr != NULL) {        cout<<ptr->data<<"\t";        ptr = ptr->next;     }     cout<<" [tail]\n";  } |

Và cuối cùng ta tạo một hàm combine() để nối hai danh sách các số chẵn và số lẻ.

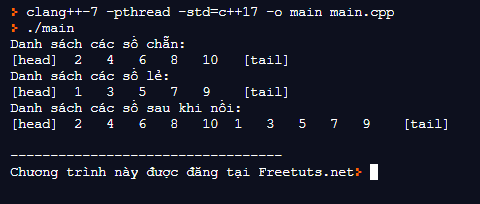
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | /\* nối hai danh sách \*/  void combine() {     struct node \*link;     list = even;     link = list;     while(link->next!= NULL) {        link = link->next;     }     link->next = odd;     odd->prev = link;     // gắn con trỏ next tới danh sách mới     while(link->next!= NULL) {        link = link->next;     }     list\_last = link;  } |

Phần này ta muốn cho danh sách các số lẻ nối vào đuôi danh sách các số chẵn. Vì vậy ta sẽ cho con trỏ prev của danh sách số lẻ trỏ về danh sách số chẵn và ngược lại cho con trỏ next của danh sách chẵn trỏ đến danh sách lẻ.

**Full code:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  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  101  102  103  104  105  106  107  108  109  110 | #include<iostream>  using namespace std;  /\* tạo cấu trúc node \*/  struct node {  **int** data;     struct node \*prev;     struct node \*next;  };  /\* tạo cấu trúc dữ liệu cho danh sách sau khi nối \*/  struct node \*list = NULL;  struct node \*list\_last = NULL;  /\* tạo cấu trúc dữ liệu cho danh sách các số chẵn \*/  struct node \*even = NULL;  struct node \*even\_last = NULL;  /\* tạo cấu trúc dữ liệu cho danh sách các số lẻ \*/  struct node \*odd = NULL;  struct node \*odd\_last = NULL;  /\* tạo cấu trúc dữ liệu node hiện tại \*/  struct node \*current = NULL;    /\* tạo danh sách liên kết đôi \*/  void insert(**int** data) {     // cấp phát bộ nhớ cho node mới;     struct node \*link = (struct node\*) **malloc**(sizeof(struct node));     link->data = data;     link->prev = NULL;     link->next = NULL;       // Nếu head trống thì tạo list mới     if(data%2 == 0) {        if(even == NULL) {           even = link;           return;        }else {           current = even;             while(current->next != NULL)              current = current->next;             // chèn node vào cuối danh sách           current->next = link;           even\_last = link;           link->prev = current;        }     }else {        if(odd == NULL) {           odd = link;           return;        }else {           current = odd;             while(current->next!=NULL)              current = current->next;           // chèn node vào cuối danh sách           current->next = link;           odd\_last = link;           link->prev = current;        }     }  }    /\* hiển thị danh sách \*/  void printList(struct node \*head) {     struct node \*ptr = head;       cout<<"\n[head] \t";     //dùng vòng lặp while lặp từ phần tử đầu đến phần tử cuối của list     while(ptr != NULL) {        cout<<ptr->data<<"\t";        ptr = ptr->next;     }     cout<<" [tail]\n";  }    /\* nối hai danh sách \*/  void combine() {     struct node \*link;     list = even;     link = list;     while(link->next!= NULL) {        link = link->next;     }     link->next = odd;     odd->prev = link;     // gắn con trỏ next tới danh sách mới     while(link->next!= NULL) {        link = link->next;     }     list\_last = link;  }    **int** main() {  **int** i;       for(i=1; i<=10; i++)        insert(i);       cout<<"Danh sách các số chẵn: ";     printList(even);       cout<<"Danh sách các số lẻ: ";     printList(odd);       combine();     cout<<"Danh sách các số sau khi nối: ";     printList(list);       cout<<"\n----------------------------------\n";     cout<<"Chương trình này được đăng tại Freetuts.net";  } |

**Kết quả:**



**3. Kết luận**

Như vậy là chúng ta đã thực hiên xong chương trình nối hai danh sách liên kết đôi trong C++. Đây là một bài tập áp dụng các kiến thức trong DSLK, các bạn hãy luyện tập thật nhiều để có thể thành thạo nó. Hãy chú ý các bài hướng dẫn hấp dẫn khác của mình nhé !!!