**Cấu trúc dữ liệu của danh sách liên kết đơn**

**Trong hướng dẫn này mình sẽ giới thiệu đến các bạn danh sách liên kết đơn là gì, cũng như cấu trúc dữ liệu của nó và cách khai báo danh sách.**



Danh sách liên kết đơn là loại DSLK đơn giản và dễ cài đặt nhất trong 3 loại mà chúng ta sẽ học trong series cấu trúc dữ liệu. Để hiểu được nó thì bạn phải có kiến thức nền tảng [C++](https://freetuts.net/hoc-c/c-can-ban) hoặc [C](https://freetuts.net/hoc-c/c-can-ban) nhé.

**Mục lục**

* [1. Danh sách liên kết đơn là gì?](https://freetuts.net/cau-truc-du-lieu-cua-danh-sach-lien-ket-don-2984.html#goto-h2-0)
* [2. Cấu trúc dữ liệu của DSLK đơn – quản lý bằng pHead](https://freetuts.net/cau-truc-du-lieu-cua-danh-sach-lien-ket-don-2984.html#goto-h2-1)
* [3. Cấu trúc dữ liệu của DSLK đơn – quản lý bằng pHead và pTail](https://freetuts.net/cau-truc-du-lieu-cua-danh-sach-lien-ket-don-2984.html#goto-h2-2)
* [Kết luận](https://freetuts.net/cau-truc-du-lieu-cua-danh-sach-lien-ket-don-2984.html#goto-h2-3)

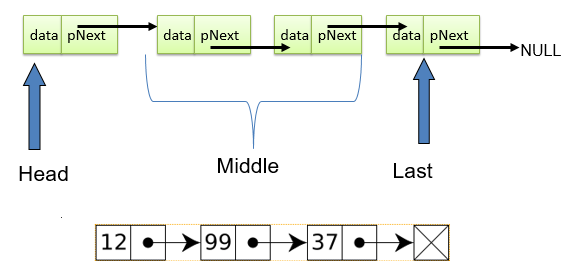
**1. Danh sách liên kết đơn là gì?**

Danh sách liên kết đơn là một cấu trúc dữ liệu động sử dụng con trỏ để cài đặt và trỏ đến các phần tử trong danh sách.

Trong DSLK đơn thì mỗi phần tử sẽ liên kết với phần tử đứng sau nó trong danh sách, ta gọi là các phần tử đó là **Node**. Mỗi node có hai thông số cần quan tâm, đó là:

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

* Giá trị (*data*)
* Mối liên kết tới node khác (*pNext*).



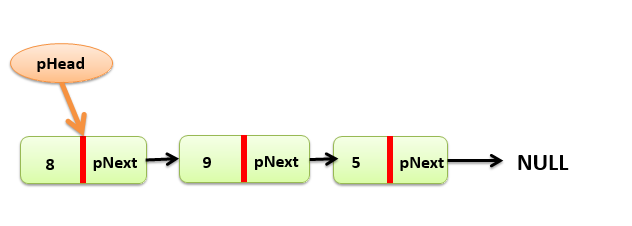
Như hình trên, ở node thứ hai có liên kết với node thứ nhất thông qua pNext, tương tự như vậy node thứ ba liên kết với node thứ hai cũng thông qua pNext.

**2. Cấu trúc dữ liệu của DSLK đơn – quản lý bằng pHead**

Trong danh sách liên kết đơn ta có mỗi phần tử liên kết với phần tử đứng sau nó trong danh sách. Mỗi phần tử gồm một Node và mỗi Node có hai thông tin: Giá trị (data) và con trỏ pNext trỏ tới phần tử kế tiếp.

Ở phần này chúng ta sẽ quản lý danh sách liên kết bằng 1 con trỏ pHead (trỏ tới phần tử đầu).

**Ví dụ**: ta có một danh sách liên kết đơn sử dụng pHead để quản lý



Trong danh sách ta cần xác định các thông tin sau:

* pHead là Node đầu của danh sách liên kết đơn.
* Các giá trị 8, 9, 5 phía bên trái gạch đỏ là giá trị (data) của Node.
* pNext là con trỏ, trỏ đến phần tử sau.

Dựa vào các thông tin trên, ta có cấu trúc dữ liệu sử dụng pHead để quản lý:

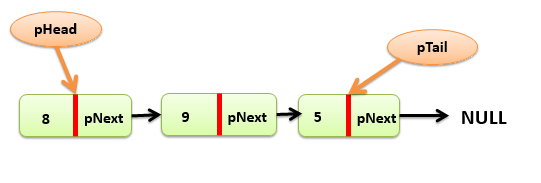
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | /\* Khai báo giá trị data và con trỏ pNext trỏ tới phần tử kế tiếp \*/  struct Node  {  **int** data;// giá trị data của node    Node \*pNext;// con trỏ pNext  };  /\* Khai báo Node đầu pHead \*/  struct SingleList  {    Node \*pHead; //Node đầu pHead  };  /\* khởi tạo giá trị cho Node \*/  void Initialize(SingleList &list)  {    list.pHead=NULL;// khởi tạo giá trị cho Node là Null  } |

Như vậy là chúng ta đã khai báo các thông tin có trong một danh sách liên kết đơn sử dụng pHead để quản lý.

**3. Cấu trúc dữ liệu của DSLK đơn – quản lý bằng pHead và pTail**

Về cơ bản việc sử dụng pHead và pTail để quản lý DSLK đơn tương tự như việc sử dụng pHead. Tuy nhiên khi chúng ta sử dụng pHead và pTail thì việc insertLast() sẽ rất thuận tiện và dễ dàng.

Minh họa danh sách liên kết đơn sử dụng 2 con trỏ pHead và pTail. pHead luôn luôn quản lý node đầu, pTail luôn luôn quản lý node cuối.



Ta có pHead quản lý Node đầu và pTail quản lý Node cuối, vì vậy khi khai báo ở cấu trúc SingleList() chúng ta sẽ khai báo cả pHead và pTail.

|  |  |
| --- | --- |
| 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  }; |

Cũng như khi khởi tạo giá trị, thì chúng ta sẽ phải khởi tạo cho cả pHead và pTail bằng Null.

|  |  |
| --- | --- |
| 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  } |

**Full code:**cấu trúc dữ liệu trong DSLK đơn

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | /\* Khai báo giá trị data và con trỏ pNext trỏ tới phần tử kế tiếp \*/  struct Node  {  **int** data;// giá trị data của node    Node \*pNext;// con trỏ pNext  };  /\* 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 đầu và Node cuối \*/  void Initialize(SingleList &list)  {    list.pHead=list.pTail=NULL;// khởi tạo giá trị cho Node đầu và Node cuối là Null  } |

Việc sử dụng pHead và pTail có ưu điểm là không cần phải duyệt DSLk khi insertLast, nhưng bù lại nhược điểm của nó sẽ tốn bộ nhớ để khởi tạo biến pTail. Vì vậy các bạn hãy điều chỉnh việc sử dụng hai cách quản lý này cho phù hợp với bài toán.

**Kết luận**

Trong bài viết này mình đã giới thiệu về danh sách liên kết đơn và cấu trúc dữ liệu của nó. Ở bài tiếp theo chúng ta sẽ cùng nhau thực hiện tạo Node trong danh sách liên kết đơn. Các bạn hãy chú ý theo dõi nhé.

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

**Trong bài này mình sẽ giới thiệu các bạn cách tạo một Node mới trong dánh sách liên kết đơn. Đây là bước đầu tiên để bạn có thể tạo được một danh sách liên kết hoàn chỉnh.**



Chúng ta sẽ cùng nhau tìm hiểu xem trong hàm tạo Node cần chú ý đến những điểm nào và điều kiện cần để có thể tạo được một Node mới.

**Mục lục**

* [1. Điều kiện cần để tạo một Node mới trong DSLK đơn](https://freetuts.net/tao-node-moi-trong-danh-sach-lien-ket-don-2987.html#goto-h2-0)
* [2. Tạo Node mới trong DSLK đơn](https://freetuts.net/tao-node-moi-trong-danh-sach-lien-ket-don-2987.html#goto-h2-1)
* [Kết luận](https://freetuts.net/tao-node-moi-trong-danh-sach-lien-ket-don-2987.html#goto-h2-2)

**1. Điều kiện cần để tạo một Node mới trong DSLK đơn**

Như các bạn đã biết thì DSLK đơn là một cấu trúc dữ liệu, vì vậy để có thể tạo được một Node trong DSLK đơn chúng ta cần khai báo cấu trúc và khởi tạo giá trị cho các Node.

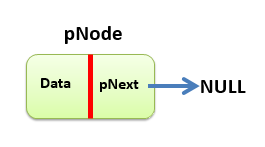
Ở bài trước chúng ta đã thực hiện tạo cấu trúc dữ liệu cho DSLK đơn bằng (*pHead*) và (*pHead, pTail)*. Đây chính là điều kiện cần để có thể tạo một Node mới.

*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  17 | /\* Khai báo giá trị data và con trỏ pNext trỏ tới phần tử kế tiếp \*/  struct Node  {  **int** data;// giá trị data của node    Node \*pNext;// con trỏ pNext  };  /\* 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 đầu và Node cuối \*/  void Initialize(SingleList &list)  {    list.pHead=list.pTail=NULL;// khởi tạo giá trị cho Node đầu và Node cuối là Null  } |

**2. Tạo Node mới trong DSLK đơn**

Sau khi chúng ta đã khai báo các cấu trúc dữ liệu trong DSLK đơn và khởi tạo giá trị Null cho Node. Như vậy là các bạn có thể tạo một Node mới cho danh sách được rồi.



Để tạo Node mới chúng ta thực hiện các bước sau:

* Đầu tiên chúng ta sẽ sử dụng \*pNode để tạo một Node mới.
* Tiếp đến xét điều kiện, nếu **pNode**có giá trị thì gán giá trị **data**đó cho tham số d và dùng **pNext**trỏ đến giá trị Null. Nếu **pNode**không có giá trị thì hiển thị thông báo ra màn hình.
* Sau khi xét điều kiện thì return pNode và kết thúc hàm.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | /\* tạo Node trong danh sách liên kết đơn \*/  Node \*CreateNode(**int** d)  {      Node \*pNode=new Node; //sử dụng pNode để tạo một Node mới      if(pNode!=NULL) // Nếu pNode != Null, tức là pNode có giá trị thì      {         pNode->data=d; // gán giá trị data cho d         pNode->pNext=NULL;// và cho con trỏ pNext trỏ tới giá trị Null      }      else // Nếu pNode == Null, tức là pNode không có giá trị thì xuất thông tin      {        cout<<"Error allocated memory";      }      return pNode;//trả về pNode  } |

Việc tạo Node mới trong DSLK đơn sử dụng **pHead**hay **pHead**và **pTail**đều giống nhau. Khi dùng **pHead**và **pTail**để quản lý danh sách liên kết, cấu trúc Node không đổi nhưng cấu trúc danh sách liên kết sẽ bổ sung thêm **pTail**.

**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 | /\* Khai báo giá trị data và con trỏ pNext trỏ tới phần tử kế tiếp \*/  struct Node  {  **int** data;// giá trị data của node    Node \*pNext;// con trỏ pNext  };    /\* 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 đầu và Node cuối \*/  void Initialize(SingleList &list)  {    list.pHead=list.pTail=NULL;// khởi tạo giá trị cho Node đầu và Node cuối là Null  }    /\* tạo Node trong danh sách liên kết đơn \*/  Node \*CreateNode(**int** d)  {      Node \*pNode=new Node; //sử dụng pNode để tạo một Node mới      if(pNode!=NULL) // Nếu pNode != Null, tức là pNode có giá trị thì      {         pNode->data=d; // gán giá trị data cho d         pNode->pNext=NULL;// và cho con trỏ pNext trỏ tới giá trị Null      }      else // Nếu pNode == Null, tức là pNode không có giá trị thì xuất thông tin      {        cout<<"Error allocated memory";      }      return pNode;//trả về pNode  } |

**Kết luận**

Như vậy là chúng ta đã tìm hiểu xong cách tạo Node mới trong danh sách liên kết đơn và điều kiện cần để tạo được Node. Ở các bài tiếp theo chúng ta sẽ cùng nhau thực hiện các thao tác thêm, sữa, xóa đối với DSLK đơn. Hãy chú ý theo dõi nhé!!!

**Chèn Node vào danh sanh liên kết đơn**

**Trong bài này mình sẽ giới thiệu đến các bạn cách chèn các Node vào danh sách liên kết đơn.**



Để hiểu được bài này các bạn phải biết được cấu trúc dữ liệu của DSLK đơn ở bài trước mình đã giới thiệu.

Chúng ta sẽ cùng nhau tìm hiểu về 3 cách chèn một Node vào danh sách liên kết đơn:

* Chèn Node vào đầu danh sách.
* Chèn Node vào cuối danh sách.
* Chèn Node vào giữa danh sách.

**Mục lục**

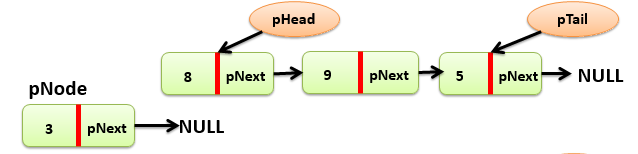
* [1. Chèn Node vào đầu danh sách liên kết đơn](https://freetuts.net/chen-node-vao-danh-sanh-lien-ket-don-2994.html#goto-h2-0)
* [2. Chèn Node vào cuối danh sách liên kết đơn](https://freetuts.net/chen-node-vao-danh-sanh-lien-ket-don-2994.html#goto-h2-1)
* [3. Chèn Node vào giữa danh sách liên kết đơn](https://freetuts.net/chen-node-vao-danh-sanh-lien-ket-don-2994.html#goto-h2-2)
* [4. Ví dụ chèn Node vào danh sách liên kết đơn](https://freetuts.net/chen-node-vao-danh-sanh-lien-ket-don-2994.html#goto-h2-3)

**1. Chèn Node vào đầu danh sách liên kết đơn**

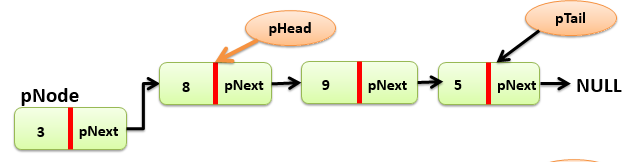
Trong phần này mình sẽ giới thiệu cách chèn Node vào đầu danh sách liên kết đơn.

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

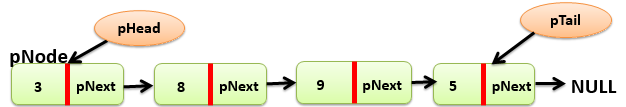
Để chèn được Node, việc đầu tiên các bạn phải có một danh sách liên kết và một Node đã được tạo sẵn.



Một Node khi mới tạo sẽ có giá trị data và con trỏ pNext -> Null. Nếu pHead == Null thì khí đó pNode được thêm vào sẽ là phần tử đầu tiên luôn. Ngược lại nếu pHead != Null, thì việc đơn giản ta chỉ cần thay đổi con trỏ pNext -> pHead



pHead luôn luôn quản lý Node đầu tiên, vì vậy sau khi trỏ đến pHead, ta phải đưa pHead về đầu danh sách: pHead -> pNode.



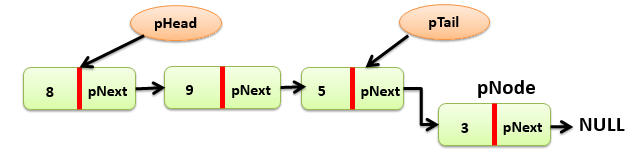
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | /\* chèn Node đầu danh sách \*/  void InsertFirst(SingleList &list,**int** d)  {    Node \*pNode=CreateNode(d); //tạo một node mới  //Nếu pHead == null thì pNode chính là pHead    if(list.pHead==NULL)    {      list.pHead=list.pTail=pNode;    }  // Ngược lại ta phải chèn pNode vào pHead sau đó đưa pHead ra đầu danh sách    else    {      pNode->pNext=list.pHead;      list.pHead=pNode;    }  } |

**2. Chèn Node vào cuối danh sách liên kết đơn**

Tương tự như chèn Node vào đầu danh sách, để chèn Node vào danh sách liên kết đơn ta chỉ cần thay đổi mối liên kết giữa Node cần chèn và Node trước đó.

Trong trường hợp pTail == Null thì Node được thêm vào là Node cuối luôn, không cần phải thay đổi con trỏ pNext.

Còn trong trường hợp pTail != Null, khi đó ta chỉ cần thay đổi list.pTail->pNext=pNode;. Tức là thay đổi con trỏ pNext của pTail liên kết đến pNode.



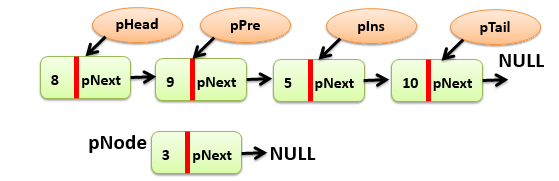
Vì pTail luôn luôn quản lý Node cuối cùng, vì vậy ta phải đưa pTail về phần tử cuối cùng: list.pTail=pNode;



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | /\* chèn node vào cuối danh sách \*/  void InsertLast(SingleList &list,**int** d)  {    Node \*pNode=CreateNode(d); // tạo node mới  // Nếu pTail == null thì pNode chính là pTail    if(list.pTail==NULL)    {      list.pHead=list.pTail=pNode;    }  // Ngược lại nếu pTail != null thì chèn pNode vào pTail và đưa pTail ra cuối danh sách    else    {      list.pTail->pNext=pNode;      list.pTail=pNode;    }  } |

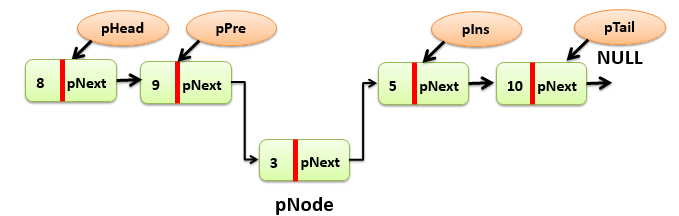
**3. Chèn Node vào giữa danh sách liên kết đơn**

Ở trên chúng ta đã thực hiện hai cách đó là chèn Node vào đầu danh sách và chèn Node vào cuối danh sách. Bây giờ chúng ta sẽ thực hiện chèn Node vào giữa danh sách (vị trí bất kỳ trong danh sách).



Giả sử chúng ta có pPre là Node đứng trước Node cần chèn và pIns là Node đứng sau Node cần chèn. Khi đó, để chèn pNode vào giữa hai Node này ta chỉ cần thay đổi mối liên kết giữa là Node là xong. Cụ thể:

|  |  |
| --- | --- |
| 1  2 | pPre->pNext=pNode;  pNode->pNext=pIns; |



Tham số truyền vào ngoài cấu trúc dữ liệu của DSLK ta còn có biến pos (vị trí chèn), d (giá trị chèn).

|  |  |
| --- | --- |
| 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 | /\* chèn node vào giữa danh sách \*/  void InsertMid(SingleList &list, **int** pos, **int** d){    // Nếu pos < 0 hoặc pos lớn hơn kích thước của danh sách thì reuturn    if(pos < 0 || pos >= SizeOfList(list)){      cout<<"Không thể chèn Node!!!";      return;    }    // Nếu pos == 0 thì gọi hàm InsertFirst    if(pos == 0){      InsertFirst(list, d);    }    //Nếu pos == SizeOfList - 1 thì gọi hàm InsertLast    else if(pos == SizeOfList(list)-1){      InsertLast(list, d);    }    //Ngược lại thì thay đổi mối liên kết giữa các phần tử, cụ thể:    else{      Node \*pNode = CreateNode(d);      Node \*pIns = list.pHead;      Node \*pPre = NULL;  **int** i = 0;      //thực hiện vòng lặp tìm pPre và pIns      while(pIns != NULL){        if(i == pos)        break;        pPre = pIns;        pIns = pIns ->pNext;        i++;      }      //sau khi tìm được thì thay đổi con trỏ pNext      pPre ->pNext=pNode;      pNode->pNext=pIns;    }  } |

**\*Lưu ý:** Để viết được hàm trên cần có số lượng phần tử trong danh sách, vì vậy mình đã viết một hàm SizeOfList() để đếm số lượng phần tử:

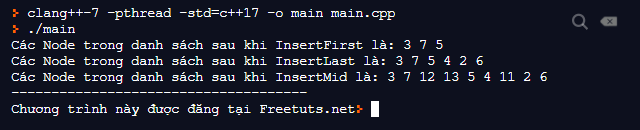
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | /\* Đếm số phần tử trong danh sách \*/  **int** SizeOfList(SingleList list)  {      Node \*pTmp=list.pHead;  **int** nSize=0;      while(pTmp!=NULL)      {          pTmp=pTmp->pNext;          nSize++;      }      return nSize;  } |

**4. Ví dụ chèn Node vào danh sách liên kết đơn**

Bây giờ chúng ta sẽ áp dụng tất cả các hàm đã viết thực hiện Insert dữ liệu vào danh sách liên kết đơn. Để hiển thị được kết quả chúng ta cần có một hàm PrintList(), mình đã viết ở bên dưới các bạn chú ý.

|  |  |
| --- | --- |
| 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  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160 | #include <iostream>  using namespace std;  /\* Khai báo giá trị data và con trỏ pNext trỏ tới phần tử kế tiếp \*/  struct Node  {  **int** data;// giá trị data của node    Node \*pNext;// con trỏ pNext  };    /\* 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 đầu và Node cuối \*/  void Initialize(SingleList &list)  {    list.pHead=list.pTail=NULL;// khởi tạo giá trị cho Node đầu và Node cuối là Null  }    /\* Đếm số phần tử trong danh sách \*/  **int** SizeOfList(SingleList list)  {      Node \*pTmp=list.pHead;  **int** nSize=0;      while(pTmp!=NULL)      {          pTmp=pTmp->pNext;          nSize++;      }      return nSize;  }    /\* tạo Node trong danh sách liên kết đơn \*/  Node \*CreateNode(**int** d)  {      Node \*pNode=new Node; //sử dụng pNode để tạo một Node mới      if(pNode!=NULL) // Nếu pNode != Null, tức là pNode có giá trị thì      {         pNode->data=d; // gán giá trị data cho d         pNode->pNext=NULL;// và cho con trỏ pNext trỏ tới giá trị Null      }      else // Nếu pNode == Null, tức là pNode không có giá trị thì xuất thông tin      {        cout<<"Error allocated memory";      }      return pNode;//trả về pNode  }    /\* chèn Node đầu danh sách \*/  void InsertFirst(SingleList &list,**int** d)  {    Node \*pNode=CreateNode(d);    if(list.pHead==NULL)    {      list.pHead=list.pTail=pNode;    }    else    {      pNode->pNext=list.pHead;      list.pHead=pNode;    }  }    /\* chèn node vào cuối danh sách \*/  void InsertLast(SingleList &list,**int** d)  {    Node \*pNode=CreateNode(d);    if(list.pTail==NULL)    {      list.pHead=list.pTail=pNode;    }    else    {      list.pTail->pNext=pNode;      list.pTail=pNode;    }  }    /\* chèn node vào giữa danh sách \*/  void InsertMid(SingleList &list, **int** pos, **int** d){    // Nếu pos < 0 hoặc pos lớn hơn kích thước của danh sách thì reuturn    if(pos < 0 || pos >= SizeOfList(list)){      cout<<"Không thể chèn Node!!!";      return;    }    // Nếu pos == 0 thì gọi hàm InsertFirst    if(pos == 0){      InsertFirst(list, d);    }    //Nếu pos == SizeOfList - 1 thì gọi hàm InsertLast    else if(pos == SizeOfList(list)-1){      InsertLast(list, d);    }    //Ngược lại thì thay đổi mối liên kết giữa các phần tử, cụ thể:    else{      Node \*pNode = CreateNode(d);      Node \*pIns = list.pHead;      Node \*pPre = NULL;  **int** i = 0;      //thực hiện vòng lặp tìm pPre và pIns      while(pIns != NULL){        if(i == pos)        break;        pPre = pIns;        pIns = pIns ->pNext;        i++;      }      //sau khi tìm được thì thay đổi con trỏ pNext      pPre ->pNext=pNode;      pNode->pNext=pIns;    }  }    /\* hàm xuất dữ liệu \*/  void PrintList(SingleList list)  {    Node \*pTmp=list.pHead;    if(pTmp==NULL)    {      cout<<"The list is empty!";      return;    }    while(pTmp!=NULL)    {      cout<<pTmp->data<<" ";    pTmp=pTmp->pNext;    }  }    **int** main() {    SingleList list;    Initialize(list);    //Thêm node đầu danh sách    InsertFirst(list, 5);    InsertFirst(list, 7);    InsertFirst(list, 3);    cout<<"Các Node trong danh sách sau khi InsertFirst là: ";    PrintList(list);    //Thêm node cuối danh sách    InsertLast(list, 4);    InsertLast(list, 2);    InsertLast(list, 6);    cout<<"\nCác Node trong danh sách sau khi InsertLast là: ";    PrintList(list);    //Thêm node giữa danh sách    InsertMid(list, 4, 11);    InsertMid(list, 2, 12);    InsertMid(list, 3, 13);    cout<<"\nCác Node trong danh sách sau khi InsertMid là: ";    PrintList(list);      cout<<"\n-------------------------------------\n";    cout<<"Chương trình này được đăng tại Freetuts.net";  } |

**Kết quả:**



Như vậy là chúng ta đã thực hiện xong các cách để thêm Node vào danh sách liên kết đơn. Ở bài tiếp theo mình sẽ hướng dẫn các bạn cách để hủy một Node trong DSLK đơn, hãy chú ý theo dõi nhé !!!

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

**Trong hướng dẫn này mình sẽ giới thiệu đến các bạn cách xóa Node trong danh sách liên kết đơn.**



Chúng ta sẽ cùng nhau tìm hiểu 3 trường hợp khi xóa một Node khỏi danh sách liên kết đơn:

* Xóa Node ở đầu danh sách liên kết đơn.
* Xóa Node ở cuối danh sách liên kết đơn.
* Xóa Node ở giữa danh sách liên kết đơn.

**Mục lục**

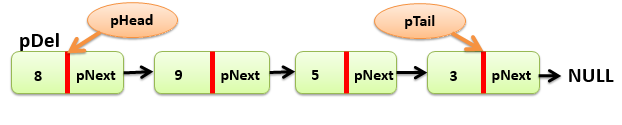
* [1. Xóa Node ở đầu danh sách liên kết đơn](https://freetuts.net/xoa-node-trong-danh-sach-lien-ket-don-2996.html#goto-h2-0)
* [2. Xóa Node ở cuối danh sách liên kết đơn.](https://freetuts.net/xoa-node-trong-danh-sach-lien-ket-don-2996.html#goto-h2-1)
* [3. Xóa Node ở giữa danh sách liên kết đơn.](https://freetuts.net/xoa-node-trong-danh-sach-lien-ket-don-2996.html#goto-h2-2)
* [4. Ví dụ xóa Node trong danh sách liên kết đơn](https://freetuts.net/xoa-node-trong-danh-sach-lien-ket-don-2996.html#goto-h2-3)

**1. Xóa Node ở đầu danh sách liên kết đơn**

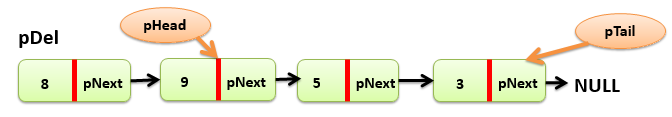
Trong trường hợp chúng ta muốn xóa một Node, mà Node đó lại nằm ở đầu danh sách. Đây là một trường hợp đặc biệt, các bạn hãy xem các bước thực hiện sau đây:

Giả sử chúng ta có một Node pDel là Node cần xóa và một danh sách liên kết đơn.

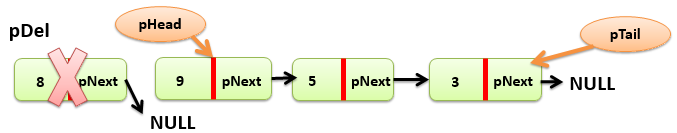
*Bài viết này được đăng tại [free tuts .net]*



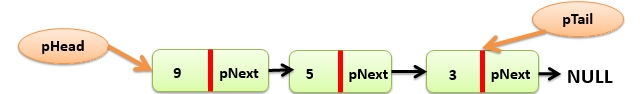
**Bước 1:** Vì Node cần xóa ở đầu danh sách, tức là ngay node pHead. Vì vậy chúng ta cần di chuyển pHead từ pDel sang Node kế tiếp: list.pHead = list.pHead -> pNext



**Bước 2:**Sau khi di chuyển pHead sang Node kế tiếp, chúng ta sẽ ngắt mối liên kết giữa pDel với Node phía sau nó: pDel -> pNext = Null.



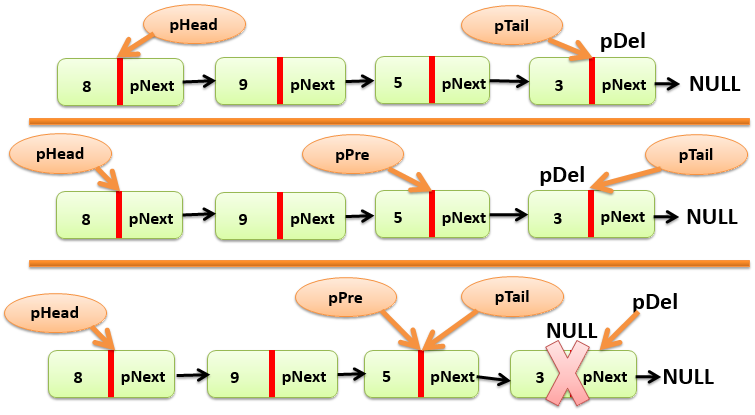
**Bước 3:** Bây giờ pDel không còn liên kết với bất kì Node nào trong danh sách nữa, chúng ta đã có thể xóa Node này. **delete**pDel



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | // Nếu pDel == list.pHead, tức là số cần xóa ở đầu danh sách        if(pDel == list.pHead){          list.pHead = list.pHead -> pNext;          pDel -> pNext = NULL;          delete pDel;          pDel = NULL;        } |

**2. Xóa Node ở cuối danh sách liên kết đơn.**

Trong trường hợp Node muốn xóa lại nằm ở cuối danh sách, tương tự như việc xóa ở đầu danh sách. Ta chỉ cần di chuyển pTail về Node trước đó (pPre) và thay đổi pNext = NULL.



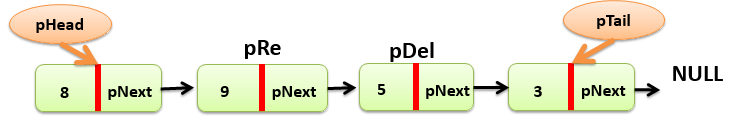
Sau khi di chuyển pTail về Node trước đó và ngắt mối liên kết giữa pPre với pDel, ta thực hiện xóa Node pDel: **delete**pDel

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | //Nếu pDel == list.pTail, tức là số cần xóa ở cuối danh sách        if(pDel -> pNext == NULL){          list.pTail = pPre;          pPre -> pNext = NULL;          delete pDel;          pDel = NULL;        } |

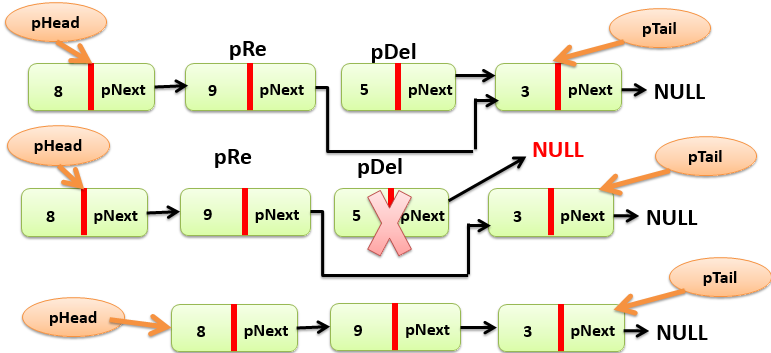
**3. Xóa Node ở giữa danh sách liên kết đơn.**

Và trường hợp cuối cùng, khi xóa Node mà Node đó không nằm đầu cũng không nằm cuối danh sách, ta thực hiện các bước như sau:

Khi ta muốn xóa một Node ở giữa danh sách, đầu tiên ta cần xác định Node cần xóa pDel và Node đứng trước nó pPre.



Sau khi xác định được pDel và pPre, ta thay đổi mối liên kết giữa pPre đến pTail (pPre -> pNext = pDel -> pNext) và cho pDel -> pNext == NULL. Các bạn có thể xem hướng mũi tên để biết được các bước thực hiện của nó.



Ta có thể xóa Node pDel khi đã ngắt mối liên kết giữa nó với các Node khác: **delete**pDel

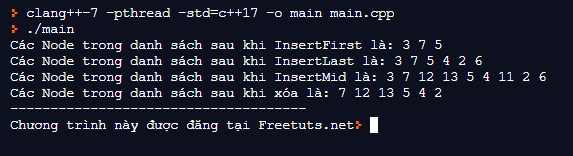
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | // và trường hợp cuối cùng số muốn xóa nằm ở giữa danh sách        else{          pPre -> pNext = pDel -> pNext;          pDel -> pNext = NULL;          delete pDel;          pDel = NULL;        } |

**4. Ví dụ xóa Node trong danh sách liên kết đơn**

Chúng ta sẽ sử dụng dữ liệu ở ví dụ trước để thực hiện xóa cho ví dụ này, vừa có thể ôn lại kiến thức cũ vừa áp dụng kiến thức mới.

|  |  |
| --- | --- |
| 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  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216 | #include <iostream>  using namespace std;  /\* Khai báo giá trị data và con trỏ pNext trỏ tới phần tử kế tiếp \*/  struct Node  {  **int** data;// giá trị data của node    Node \*pNext;// con trỏ pNext  };    /\* 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 đầu và Node cuối \*/  void Initialize(SingleList &list)  {    list.pHead=list.pTail=NULL;// khởi tạo giá trị cho Node đầu và Node cuối là Null  }    /\* Đếm số phần tử trong danh sách \*/  **int** SizeOfList(SingleList list)  {      Node \*pTmp=list.pHead;  **int** nSize=0;      while(pTmp!=NULL)      {          pTmp=pTmp->pNext;          nSize++;      }      return nSize;  }    /\* tạo Node trong danh sách liên kết đơn \*/  Node \*CreateNode(**int** d)  {      Node \*pNode=new Node; //sử dụng pNode để tạo một Node mới      if(pNode!=NULL) // Nếu pNode != Null, tức là pNode có giá trị thì      {         pNode->data=d; // gán giá trị data cho d         pNode->pNext=NULL;// và cho con trỏ pNext trỏ tới giá trị Null      }      else // Nếu pNode == Null, tức là pNode không có giá trị thì xuất thông tin      {        cout<<"Error allocated memory";      }      return pNode;//trả về pNode  }    /\* chèn Node đầu danh sách \*/  void InsertFirst(SingleList &list,**int** d)  {    Node \*pNode=CreateNode(d);    if(list.pHead==NULL)    {      list.pHead=list.pTail=pNode;    }    else    {      pNode->pNext=list.pHead;      list.pHead=pNode;    }  }    /\* chèn node vào cuối danh sách \*/  void InsertLast(SingleList &list,**int** d)  {    Node \*pNode=CreateNode(d);    if(list.pTail==NULL)    {      list.pHead=list.pTail=pNode;    }    else    {      list.pTail->pNext=pNode;      list.pTail=pNode;    }  }    /\* chèn node vào giữa danh sách \*/  void InsertMid(SingleList &list, **int** pos, **int** d){    // Nếu pos < 0 hoặc pos lớn hơn kích thước của danh sách thì reuturn    if(pos < 0 || pos >= SizeOfList(list)){      cout<<"Không thể chèn Node!!!";      return;    }    // Nếu pos == 0 thì gọi hàm InsertFirst    if(pos == 0){      InsertFirst(list, d);    }    //Nếu pos == SizeOfList - 1 thì gọi hàm InsertLast    else if(pos == SizeOfList(list)-1){      InsertLast(list, d);    }    //Ngược lại thì thay đổi mối liên kết giữa các phần tử, cụ thể:    else{      Node \*pNode = CreateNode(d);      Node \*pIns = list.pHead;      Node \*pPre = NULL;  **int** i = 0;      //thực hiện vòng lặp tìm pPre và pIns      while(pIns != NULL){        if(i == pos)        break;        pPre = pIns;        pIns = pIns ->pNext;        i++;      }      //sau khi tìm được thì thay đổi con trỏ pNext      pPre ->pNext=pNode;      pNode->pNext=pIns;    }  }    /\* xóa node khỏi danh sách liên kết \*/  void RemoveNode(SingleList &list, **int** d){    Node \*pDel = list.pHead; // tạo một node pDel để xóa    //Nếu pDel == Null thì danh sách rỗng    if(pDel == NULL){      cout<<"Danh sách rỗng!!";    }    //ngược lại thì xét điều kiện    else{      Node \*pPre = NULL;      //dùng vòng lặp while để tìm ra pDel và pPre (vị trí đứng trước pDel)      while(pDel != NULL){        if(pDel -> data == d){          break;        }        pPre = pDel;        pDel = pDel -> pNext;      }      //Nếu pDel == null tức là không tìm thấy số cần xóa      if(pDel == NULL){        cout<<"Không tìm thấy số cần xóa";      }      // Ngược lại tiếp tục xét điều kiện      else{        // Nếu pDel == list.pHead, tức là số cần xóa ở đầu danh sách        if(pDel == list.pHead){          list.pHead = list.pHead -> pNext;          pDel -> pNext = NULL;          delete pDel;          pDel = NULL;        }        //Nếu pDel == list.pTail, tức là số cần xóa ở cuối danh sách        else if(pDel -> pNext == NULL){          list.pTail = pPre;          pPre -> pNext = NULL;          delete pDel;          pDel = NULL;        }        // và trường hợp cuối cùng số muốn xóa nằm ở giữa danh sách        else{          pPre -> pNext = pDel -> pNext;          pDel -> pNext = NULL;          delete pDel;          pDel = NULL;        }      }    }  }    /\* hàm xuất dữ liệu \*/  void PrintList(SingleList list)  {    Node \*pTmp=list.pHead;    if(pTmp==NULL)    {      cout<<"The list is empty!";      return;    }    while(pTmp!=NULL)    {      cout<<pTmp->data<<" ";    pTmp=pTmp->pNext;    }  }    **int** main() {    SingleList list;    Initialize(list);    //Thêm node đầu danh sách    InsertFirst(list, 5);    InsertFirst(list, 7);    InsertFirst(list, 3);    cout<<"Các Node trong danh sách sau khi InsertFirst là: ";    PrintList(list);    //Thêm node cuối danh sách    InsertLast(list, 4);    InsertLast(list, 2);    InsertLast(list, 6);    cout<<"\nCác Node trong danh sách sau khi InsertLast là: ";    PrintList(list);    //Thêm node giữa danh sách    InsertMid(list, 4, 11);    InsertMid(list, 2, 12);    InsertMid(list, 3, 13);    cout<<"\nCác Node trong danh sách sau khi InsertMid là: ";    PrintList(list);    //Xóa node khỏi danh sách    RemoveNode(list, 3);    RemoveNode(list, 11);    RemoveNode(list, 6);    cout<<"\nCác Node trong danh sách sau khi xóa là: ";    PrintList(list);      cout<<"\n-------------------------------------\n";    cout<<"Chương trình này được đăng tại Freetuts.net";  } |

**Kết quả:**



Như vậy là chúng ta đã thực hiện xong các trường hợp xóa Node trong danh sách liên kết đơn. Ở bài tiếp theo chúng ta sẽ thực hiện tìm kiếm và sắp xếp các phần tử trong DSLK, các bạn hãy chú ý theo dõi nhé !!!

**Tìm kiếm và sắp xếp trong danh sách liên kết đơn**

**Trong hướng dẫn này mình sẽ giới thiệu các bạn cách tìm kiếm và sắp xếp trong danh sách liên kết đơn.**



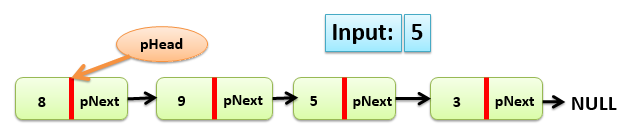
Chúng ta sẽ cùng nhau tìm hiều lần lượt cách tìm kiếm một giá trị index trong danh sách và thực hiện sắp xếp danh sách theo thứ tự tăng dần. Sắp xếp và tìm kiếm là hai thuật toán không thể thiếu trong các ngôn ngữ lập trình, bì vậy các bạn hãy luyện tập cho thành thạo nhé.

**Mục lục**

* [1. Tìm kiếm trong danh sách liên kết đơn](https://freetuts.net/tim-kiem-va-sap-xep-trong-danh-sach-lien-ket-don-2997.html#goto-h2-0)
* [2. Sắp xếp trong danh sắp liên kết đơn](https://freetuts.net/tim-kiem-va-sap-xep-trong-danh-sach-lien-ket-don-2997.html#goto-h2-1)
* [3. Ví dụ tìm kiếm và sắp xếp trong danh sách liên kết đơn](https://freetuts.net/tim-kiem-va-sap-xep-trong-danh-sach-lien-ket-don-2997.html#goto-h2-2)

**1. Tìm kiếm trong danh sách liên kết đơn**

Trong phần này mình sẽ thực hiện tìm kiếm một giá trị index được nhập từ phím trong danh sách liên kết đơn. Đây là một thao tác đơn giản, chỉ cần ta duyệt từng phần tử trong danh sách.



Giả sử chúng ta có giá trị index = 5 là giá trị cần tìm trong danh sách. Lúc này ta cần khai báo một Node tạm pTmp để thay thế cho pHead duyệt danh sách.

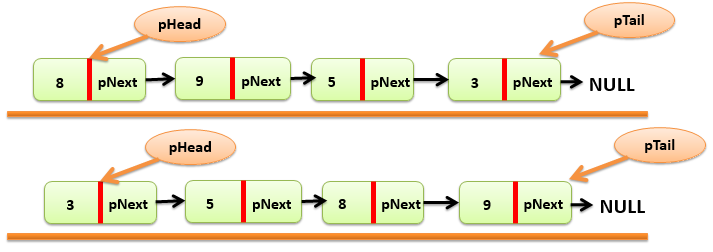
*Bài viết này được đăng tại [free tuts .net]*

Thực hiện vòng lặp while lặp từng phần tử trong danh sách với điều kiện pTmp != Null. Nếu pTmp -> data == index thì thoát khỏi vòng lặp và thông báo đã tìm thấy, ngược lại thì thông báo không tìm thấy.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | /\* tìm kiếm giá trị index trong danh sách \*/  Node \* SearchNode(SingleList list,**int** d)  {    Node \*pTmp=list.pHead; // khai báo biến tạm pTmp thay thế cho pHead để duyệt danh sách    //Thực hiện vòng lặp các phần tử trong danh sách với điều kiện pTmp != null    while(pTmp!=NULL)    {      if(pTmp->data==d) // nếu tìm thấy index thì thoát khỏi vòng lặp        break; // chưa tìm thấy thì tiếp tục duyệt phần tử kết tiếp      pTmp=pTmp->pNext;    }    return pTmp;  } |

**2. Sắp xếp trong danh sắp liên kết đơn**

Ở phần sắp xếp phần tử trong danh sách liên kết đơn, mình sẽ thực hiện sắp xếp bằng cách so sánh và thay đổi giá trị data chứ không thay đổi Node. Tức là chỉ so sánh các giá trị data rồi sắp xếp, các Node vẫn giữ nguyên không dịch chuyển.



Thao tác sắp xếp trong danh sách về cơ bản tương tự như các thuật toán sắp xếp khác, đơn giản chỉ là duyệt từng phần tử rồi so sánh với nhau, sau đó hoán đổi vị trí của chúng.

Đầu tiên ta có một vòng lặp For sử dụng biến pTmp để lặp từng phần tử trong danh sách, vòng lặp For thứ hai sử dụng biến pTmp2 để lặp từng phần tử trong danh sách.

Nếu pTmp > pTmp2 thì hoán đổi vị trí giữa chúng, nếu pTmp < pTmp2 thì tiếp tục so sách các phần tử tiếp theo, cứ như vậy cho đến hết danh sách.

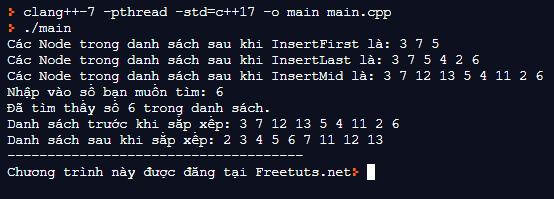
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | /\* sắp xếp trong danh sách liên kết đơn theo thứ tự tăng dần \*/  void SortList(SingleList &list)  {    // for loop thứ nhất   for(Node \*pTmp=list.pHead;pTmp!=NULL;pTmp=pTmp->pNext)   {     //for loop thứ hai    for(Node \*pTmp2=pTmp->pNext;pTmp2!=NULL;pTmp2=pTmp2->pNext)    {      if(pTmp->data>pTmp2->data) // nếu giá trị trước > giá trị sau thì hoán đổi hai vị trí       {  **int** tmp=pTmp->data;         pTmp->data=pTmp2->data;         pTmp2->data=tmp;       }    }   }  } |

**3. Ví dụ tìm kiếm và sắp xếp trong danh sách liên kết đơn**

Trong phần ví dụ này chúng ta sẽ sử dụng dữ liệu ở ví dụ trước để áp dụng cho việc tìm kiếm và sắp xếp. Các bạn lưu ý rằng muốn sử dụng được các thao tác mình đã hướng dẫn thì luôn luôn khái báo cấu trúc dữ liệu của DSLK đơn trước nhé.

|  |  |
| --- | --- |
| 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  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212 | #include <iostream>  using namespace std;  /\* Khai báo giá trị data và con trỏ pNext trỏ tới phần tử kế tiếp \*/  struct Node  {  **int** data;// giá trị data của node    Node \*pNext;// con trỏ pNext  };    /\* 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 đầu và Node cuối \*/  void Initialize(SingleList &list)  {    list.pHead=list.pTail=NULL;// khởi tạo giá trị cho Node đầu và Node cuối là Null  }    /\* Đếm số phần tử trong danh sách \*/  **int** SizeOfList(SingleList list)  {      Node \*pTmp=list.pHead;  **int** nSize=0;      while(pTmp!=NULL)      {          pTmp=pTmp->pNext;          nSize++;      }      return nSize;  }    /\* tạo Node trong danh sách liên kết đơn \*/  Node \*CreateNode(**int** d)  {      Node \*pNode=new Node; //sử dụng pNode để tạo một Node mới      if(pNode!=NULL) // Nếu pNode != Null, tức là pNode có giá trị thì      {         pNode->data=d; // gán giá trị data cho d         pNode->pNext=NULL;// và cho con trỏ pNext trỏ tới giá trị Null      }      else // Nếu pNode == Null, tức là pNode không có giá trị thì xuất thông tin      {        cout<<"Error allocated memory";      }      return pNode;//trả về pNode  }    /\* chèn Node đầu danh sách \*/  void InsertFirst(SingleList &list,**int** d)  {    Node \*pNode=CreateNode(d);    if(list.pHead==NULL)    {      list.pHead=list.pTail=pNode;    }    else    {      pNode->pNext=list.pHead;      list.pHead=pNode;    }  }    /\* chèn node vào cuối danh sách \*/  void InsertLast(SingleList &list,**int** d)  {    Node \*pNode=CreateNode(d);    if(list.pTail==NULL)    {      list.pHead=list.pTail=pNode;    }    else    {      list.pTail->pNext=pNode;      list.pTail=pNode;    }  }    /\* chèn node vào giữa danh sách \*/  void InsertMid(SingleList &list, **int** pos, **int** d){    // Nếu pos < 0 hoặc pos lớn hơn kích thước của danh sách thì reuturn    if(pos < 0 || pos >= SizeOfList(list)){      cout<<"Không thể chèn Node!!!";      return;    }    // Nếu pos == 0 thì gọi hàm InsertFirst    if(pos == 0){      InsertFirst(list, d);    }    //Nếu pos == SizeOfList - 1 thì gọi hàm InsertLast    else if(pos == SizeOfList(list)-1){      InsertLast(list, d);    }    //Ngược lại thì thay đổi mối liên kết giữa các phần tử, cụ thể:    else{      Node \*pNode = CreateNode(d);      Node \*pIns = list.pHead;      Node \*pPre = NULL;  **int** i = 0;      //thực hiện vòng lặp tìm pPre và pIns      while(pIns != NULL){        if(i == pos)        break;        pPre = pIns;        pIns = pIns ->pNext;        i++;      }      //sau khi tìm được thì thay đổi con trỏ pNext      pPre ->pNext=pNode;      pNode->pNext=pIns;    }  }  /\* tìm kiếm giá trị index trong danh sách \*/  Node \* SearchNode(SingleList list,**int** d)  {    Node \*pTmp=list.pHead; // khai báo biến tạm pTmp thay thế cho pHead để duyệt danh sách    //Thực hiện vòng lặp các phần tử trong danh sách với điều kiện pTmp != null    while(pTmp!=NULL)    {      if(pTmp->data==d) // nếu tìm thấy index thì thoát khỏi vòng lặp        break; // chưa tìm thấy thì tiếp tục duyệt phần tử kết tiếp      pTmp=pTmp->pNext;    }    return pTmp;  }    /\* sắp xếp trong danh sách liên kết đơn theo thứ tự tăng dần \*/  void SortList(SingleList &list)  {    // for loop thứ nhất   for(Node \*pTmp=list.pHead;pTmp!=NULL;pTmp=pTmp->pNext)   {     //for loop thứ hai    for(Node \*pTmp2=pTmp->pNext;pTmp2!=NULL;pTmp2=pTmp2->pNext)    {      if(pTmp->data>pTmp2->data) // nếu giá trị trước > giá trị sau thì hoán đổi hai vị trí       {  **int** tmp=pTmp->data;         pTmp->data=pTmp2->data;         pTmp2->data=tmp;       }    }   }  }      /\* hàm xuất dữ liệu \*/  void PrintList(SingleList list)  {    Node \*pTmp=list.pHead;    if(pTmp==NULL)    {      cout<<"The list is empty!";      return;    }    while(pTmp!=NULL)    {      cout<<pTmp->data<<" ";    pTmp=pTmp->pNext;    }  }    **int** main() {    SingleList list;    Initialize(list);    //Thêm node đầu danh sách    InsertFirst(list, 5);    InsertFirst(list, 7);    InsertFirst(list, 3);    cout<<"Các Node trong danh sách sau khi InsertFirst là: ";    PrintList(list);    //Thêm node cuối danh sách    InsertLast(list, 4);    InsertLast(list, 2);    InsertLast(list, 6);    cout<<"\nCác Node trong danh sách sau khi InsertLast là: ";    PrintList(list);    //Thêm node giữa danh sách    InsertMid(list, 4, 11);    InsertMid(list, 2, 12);    InsertMid(list, 3, 13);    cout<<"\nCác Node trong danh sách sau khi InsertMid là: ";    PrintList(list);  //Tìm kiếm giá trị index trong danh sách  **int** index;    cout<<"\nNhập vào số bạn muốn tìm: ";    cin>>index;    Node \*pFound = SearchNode(list, index);    if(pFound == NULL){      cout<<"Không tìm thấy số "<<index<<" trong danh sách";      }    else{      cout<<"Đã tìm thấy số "<<index<<" trong danh sách.";    }  //Sắp xếp các phần tử trong danh sách    cout<<"\nDanh sách trước khi sắp xếp: ";    PrintList(list);    SortList(list);    cout<<"\nDanh sách sau khi sắp xếp: ";    PrintList(list);        cout<<"\n-------------------------------------\n";    cout<<"Chương trình này được đăng tại Freetuts.net";  } |

**Kết quả:**



Như vậy là chúng ta đã thực hiện xong chương trình tìm kiếm và sắp xếp các phần tử trong danh sách liên kết đơn. Cũng như kết thúc hướng dẫn thao tác tìm kiếm và sắp xếp. Chúc các bạn thực hiện thành công !!!

**Quản lý sinh viên sử dụng danh sách liên kết đơn**

**Trong hướng dẫn này mình sẽ thực hiện một chương trình quản lý sinh viên sử dụng danh sách liên kết đơn.**



Chúng ta sẽ quản lý sinh viên với các thông tin cần thiết và các thao tác thêm, xóa và sắp xếp sinh viên.

**Mục lục**

* [Đề bài: Xây dựng chương trình quản lý sinh viên bằng DSLK đơn](https://freetuts.net/quan-ly-sinh-vien-su-dung-danh-sach-lien-ket-don-2999.html#goto-h2-0)
* [Chương trình quản lý sinh viên sử dụng DSLK đơn](https://freetuts.net/quan-ly-sinh-vien-su-dung-danh-sach-lien-ket-don-2999.html#goto-h2-1)

**Đề bài: Xây dựng chương trình quản lý sinh viên bằng DSLK đơn**

Cho một sinh viên có cấu trúc: mã (int), tên (char \*). Dùng danh sách liên kết đơn với con trỏ phead để thao tác:

* Khởi tạo list dạng con trỏ
* Thêm node vào cuối danh sách
* Sắp xếp theo mã
* Xóa node

**Chương trình quản lý sinh viên sử dụng DSLK đơn**

Chúng ta sẽ lần lượt tạo cấu trúc sinh viên, cấu trúc danh sách liên kết đơn và các thao tác liên quan.

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

Đầu tiên chúng ta cần tạo một cấu trúc sinh viên với mã số sinh viên ***ma***và tên sinh viên***ten****.*

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | //tao cau truc sinh vien  struct SinhVien  {  **int** ma;  **char** ten[150];  }; |

Tiếp đến tạo cấu trúc dữ liệu của danh sách liên kết đơn với giá trị data và con trỏ pNext. Khởi tạo giá trị cho pHead và pTail bằng NULL.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | //tao cau truc danh sach lien ket don  struct Node  {      SinhVien \*data;      Node \*pNext;  };  struct SingleList  {      Node \*pHead;  };  //khoi tao danh sach lien ket don  void Initialize(SingleList \*&list)  {      list=new SingleList;      list->pHead=NULL;  } |

Tạo một hàm NhapSinhVien() sử dụng cấu trúc SinhVien để nhập các thông tin của sinh viên như: MSSV và tên sinh viên

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | SinhVien \*NhapSinhVien()  {      SinhVien \*sv=new SinhVien;      cout<<"Nhap MSSV:";      cin>>sv->ma;      cin.ignore();      cout<<"Nhap ho va ten:";  **gets**(sv->ten);      return sv;  } |

Bây giờ chúng ta bắt đầu tạo Node với các thông tin của cấu trúc SinhVien, sau đó thêm Node vào cuối danh sách.

|  |  |
| --- | --- |
| 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 | //tao node sinh vien  Node \*CreateNode(SinhVien \*sv)  {      Node \*pNode=new Node;      if(pNode!=NULL)      {          pNode->data=sv;          pNode->pNext=NULL;      }      else      {          cout<<"cap phat bo nho that bai!!!";      }      return pNode;  }  //them node vao cuoi danh sach  void InsertLast(SingleList \*&list,SinhVien \*sv)  {      Node \*pNode=CreateNode(sv);      if(list->pHead==NULL)      {          list->pHead=pNode;      }      else      {          Node \*pTmp=list->pHead;            while(pTmp->pNext!=NULL)          {              pTmp=pTmp->pNext;          }          pTmp->pNext=pNode;      }  } |

Sau khi thêm Node vào danh sách ta thực hiện các thao tác theo yêu cầu của đề bài. Đầu tiên là việc sắp xếp các sinh viên theo MSSV.

Ở bài tìm kiếm và sắp xếp trong danh sách liên kết đơn mình đã giới thiệu các bạn thao tác sắp xếp. Dựa vào đó ta chỉ cần biến đổi một chút sẽ có ngay hàm sắp xếp SortList() theo MSSV.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | void SortList(SingleList \*&list)  {      for(Node \*pTmp=list->pHead;pTmp!=NULL;pTmp=pTmp->pNext)      {          for(Node \*pTmp2=pTmp->pNext;pTmp2!=NULL;pTmp2=pTmp2->pNext)          {              SinhVien \*svTmp=pTmp->data;              SinhVien \*svTmp2=pTmp2->data;              if(svTmp2->ma<svTmp->ma)              {  **int** ma=svTmp->ma;  **char** ten[150];  **strcpy**(ten,svTmp->ten);                    svTmp->ma=svTmp2->ma;  **strcpy**(svTmp->ten,svTmp2->ten);                  svTmp2->ma=ma;  **strcpy**(svTmp2->ten,ten);              }          }      }  } |

Tương tự như hàm sắp xếp, để xóa một sinh viên dựa vào tên ta thực hiện vòng lặp while lặp từng phần tử trong danh sách. Nếu phần tử đó trùng với phần tử được nhập vào từ bàn phím ta thực hiện delete phần tử đó ra khỏi danh sách.

|  |  |
| --- | --- |
| 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 | void RemoveNode(SingleList \*&list,**int** ma)  {      Node \*pDel=list->pHead;      if(pDel==NULL)      {          cout<<"Danh sach rong!";      }      else      {          Node \*pPre=NULL;          while(pDel!=NULL)          {              SinhVien \*sv=pDel->data;              if(sv->ma==ma)                  break;              pPre=pDel;              pDel=pDel->pNext;          }          if(pDel==NULL)          {              cout<<"khong tim thay MSSV: "<<ma;          }          else          {              if(pDel==list->pHead)              {                  list->pHead=list->pHead->pNext;                  pDel->pNext=NULL;                  delete pDel;                  pDel=NULL;              }              else              {                  pPre->pNext=pDel->pNext;                  pDel->pNext=NULL;                  delete pDel;                  pDel=NULL;              }          }      }  } |

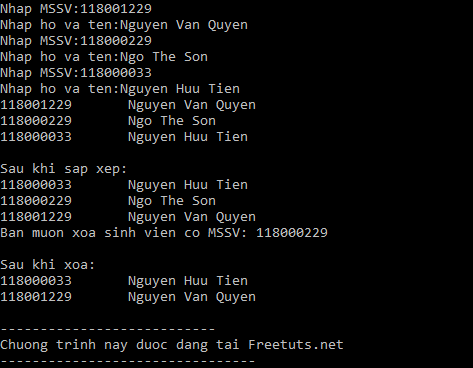
Sau khi thực hiện tạo các thao tác, ta chỉ cần tạo hàm main() và gọi các thao tác đó ra để sử dụng.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | **int** main(**int** argc, **char**\*\* argv) {      SingleList \*list;      Initialize(list);      SinhVien \*teo=NhapSinhVien();      InsertLast(list,teo);      SinhVien \*ty=NhapSinhVien();      InsertLast(list,ty);      SinhVien \*bin=NhapSinhVien();      InsertLast(list,bin);      PrintList(list);      SortList(list);      cout<<"\nSau khi sap xep:\n";      PrintList(list);      cout<<"\Ban muon xoa sinh vien co MSSV: ";  **int** ma;      cin>>ma;      RemoveNode(list,ma);      cout<<"\nSau khi xoa:\n";      PrintList(list);  } |

**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  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175 | #include <iostream>  #include <string.h>  #include <stdio.h>  using namespace std;  //tao cau truc sinh vien  struct SinhVien  {  **int** ma;  **char** ten[150];  };  //tao cau truc danh sach lien ket don  struct Node  {      SinhVien \*data;      Node \*pNext;  };  struct SingleList  {      Node \*pHead;  };  //khoi tao danh sach lien ket don  void Initialize(SingleList \*&list)  {      list=new SingleList;      list->pHead=NULL;  }  //nhap thong tin sinh vien  SinhVien \*NhapSinhVien()  {      SinhVien \*sv=new SinhVien;      cout<<"Nhap MSSV:";      cin>>sv->ma;      cin.ignore();      cout<<"Nhap ho va ten:";  **gets**(sv->ten);      return sv;  }  //tao node sinh vien  Node \*CreateNode(SinhVien \*sv)  {      Node \*pNode=new Node;      if(pNode!=NULL)      {          pNode->data=sv;          pNode->pNext=NULL;      }      else      {          cout<<"cap phat bo nho that bai!!!";      }      return pNode;  }  //them node vao cuoi danh sach  void InsertLast(SingleList \*&list,SinhVien \*sv)  {      Node \*pNode=CreateNode(sv);      if(list->pHead==NULL)      {          list->pHead=pNode;      }      else      {          Node \*pTmp=list->pHead;            while(pTmp->pNext!=NULL)          {              pTmp=pTmp->pNext;          }          pTmp->pNext=pNode;      }  }  //hien thi danh sach  void PrintList(SingleList \*list)  {      Node \*pTmp=list->pHead;      if(pTmp==NULL)      {          cout<<"Danh sach rong";          return;      }      while(pTmp!=NULL)      {          SinhVien \*sv=pTmp->data;          cout<<sv->ma<<"\t"<<sv->ten<<"\n";          pTmp=pTmp->pNext;      }  }  //sap xep  void SortList(SingleList \*&list)  {      for(Node \*pTmp=list->pHead;pTmp!=NULL;pTmp=pTmp->pNext)      {          for(Node \*pTmp2=pTmp->pNext;pTmp2!=NULL;pTmp2=pTmp2->pNext)          {              SinhVien \*svTmp=pTmp->data;              SinhVien \*svTmp2=pTmp2->data;              if(svTmp2->ma<svTmp->ma)              {  **int** ma=svTmp->ma;  **char** ten[150];  **strcpy**(ten,svTmp->ten);                    svTmp->ma=svTmp2->ma;  **strcpy**(svTmp->ten,svTmp2->ten);                  svTmp2->ma=ma;  **strcpy**(svTmp2->ten,ten);              }          }      }  }  //xoa  void RemoveNode(SingleList \*&list,**int** ma)  {      Node \*pDel=list->pHead;      if(pDel==NULL)      {          cout<<"Danh sach rong!";      }      else      {          Node \*pPre=NULL;          while(pDel!=NULL)          {              SinhVien \*sv=pDel->data;              if(sv->ma==ma)                  break;              pPre=pDel;              pDel=pDel->pNext;          }          if(pDel==NULL)          {              cout<<"khong tim thay MSSV: "<<ma;          }          else          {              if(pDel==list->pHead)              {                  list->pHead=list->pHead->pNext;                  pDel->pNext=NULL;                  delete pDel;                  pDel=NULL;              }              else              {                  pPre->pNext=pDel->pNext;                  pDel->pNext=NULL;                  delete pDel;                  pDel=NULL;              }          }      }  }  **int** main(**int** argc, **char**\*\* argv) {      SingleList \*list;      Initialize(list);      SinhVien \*teo=NhapSinhVien();      InsertLast(list,teo);      SinhVien \*ty=NhapSinhVien();      InsertLast(list,ty);      SinhVien \*bin=NhapSinhVien();      InsertLast(list,bin);      PrintList(list);      SortList(list);      cout<<"\nSau khi sap xep:\n";      PrintList(list);      cout<<"\Ban muon xoa sinh vien co MSSV: ";  **int** ma;      cin>>ma;      RemoveNode(list,ma);      cout<<"\nSau khi xoa:\n";      PrintList(list);      cout<<"\n---------------------------\n";    cout<<"Chuong trinh nay duoc dang tai Freetuts.net";  } |

**Kết quả:**



Như vậy là chúng ta đã thực hiện xong chương trình quản lý sinh viên sử dụng danh sách liên kết đơn. Hãy luyện tập thật nhiều nhé, chúc các bạn thực hiện thành công!!!

**Bài tập thực hành với danh sách liên kết đơn**

**Trong hướng dẫn này mình sẽ giải một số bài tập đơn giản liên quan đến danh sách liên kết đơn.**



Phần này giúp các bạn ôn lại bài và biết cách áp dụng kiến thức đã học vào bài tập thực tế.

Chúng ta sẽ thực hiện giải hai bài tập:

* Bài tập về danh sách liên kết đơn kiểu cơ sở
* Bài tập về danh sách liên kết đơn kiểu cấu trúc

Qua hai bài tập này các bạn có thể quản lý được danh sách liên kết và thao tác với nó một cách thành thạo.

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

**Mục lục**

* [1. Bài tập về danh sách liên kết đơn kiểu cơ sở](https://freetuts.net/bai-tap-danh-sach-lien-ket-don-2998.html#goto-h2-0)
* [Bài tập về danh sách liên kết đơn kiểu cấu trúc](https://freetuts.net/bai-tap-danh-sach-lien-ket-don-2998.html#goto-h2-1)

**1. Bài tập về danh sách liên kết đơn kiểu cơ sở**

Trong bài tập này chúng ta sẽ thực hiện giải một bài toán như sau:

s(x,n) = x1 + x2 + x3 + ... + xn

Xây dựng danh sách liên kết đơn có pHead, pTail. Nhập x, n tạo thành danh sách liên kết( *Mỗi nút có 2 giá trị x và i, i chạy từ 1 -> n*), dùng con trỏ để khai báo cho danh sách liên kết.

Viết hàm xuất ra tổng các phần tử trong danh sách liên kết.

**Gợi ý:**

Để giải bài toán này việc đầu tiên ta cần cấu trúc dữ liệu của danh sách liên kết đơn. Giá trị data là x và i, mối liên kết là pNext. Khởi tạo cho pHead và pTail bằng NULL.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | /\* Tạo cấu trúc dữ liệu cho danh sách liên kết đơn \*/  struct Node  {  **int** x;  **int** i;      Node \*pNext;  };  struct SingleList  {      Node \*pHead;      Node \*pTail;  };  /\* Khởi tạo cho pHead và pTail \*/  void Initialize(SingleList \*&list)  {      list=new SingleList;      list->pHead=list->pTail=NULL;  } |

Sau đó viết một hàm tạo Node dựa vào cấu trúc dữ liệu vừa viết.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | /\* Tạo Node \*/  Node \*CreateNode(**int** x,**int** i)  {      Node \*pNode=new Node;      if(pNode==NULL)      {          cout<<"Loi cap phat bo nho";  **exit**(0);      }      pNode->i=i;      pNode->x=x;      pNode->pNext=NULL;      return pNode;  } |

Trong bài tập này ta cần thêm Node vào cuối danh sách, vì vậy chúng ta cần tạo một hàm InsertLast() với tham số truyền vào là x và i.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | /\* insertlast \*/  void InsertLast(SingleList \*&list,**int** x,**int** i)  {      Node \*pNode=CreateNode(x,i);      if(list->pTail==NULL)          list->pHead=list->pTail=pNode;      else      {          list->pTail->pNext=pNode;          list->pTail=pNode;      }  } |

Một hàm PrintList() để in danh sách liên kết.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | /\*printlist\*/  void PrintList(SingleList \*list)  {      Node \*pTmp=list->pHead;      while(pTmp!=NULL)      {          if(pTmp->pNext!=NULL)              cout<<pTmp->x<<"^"<<pTmp->i<<"+";          else              cout<<pTmp->x<<"^"<<pTmp->i;          pTmp=pTmp->pNext;      }  } |

Và cuối cùng là hàm tính tổng SumOfList(), dùng để tính tổng các phần tử trong danh sách liên kết đơn.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | /\* Hàm tính tổng \*/  **double** SumOfList(SingleList \*list)  {  **double** sum=0;      for(Node \*pTmp=list->pHead;pTmp!=NULL;pTmp=pTmp->pNext)      {  **double** value=**pow**(pTmp->x,pTmp->i);          sum+=value;      }      return sum;  } |

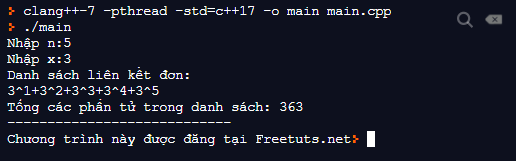
Để hiển thị và kiểm tra kết quả ta cần có hàm main() để làm điều này.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | **int** main(**int** argc, **char**\*\* argv) {      SingleList \*list;      Initialize(list);  **int** n,x;      cout<<"Nhập n:";      cin>>n;      cout<<"Nhập x:";      cin>>x;      for(**int** i=1;i<=n;i++)      {          InsertLast(list,x,i);      }      cout<<"Danh sách liên kết đơn: \n";      PrintList(list);  **double** sum=SumOfList(list);      cout<<"\nTổng các phần tử trong danh sách: "<<sum;  } |

**Full code:**

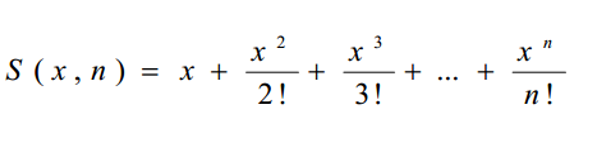
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92 | #include <iostream>  #include <math.h>  using namespace std;  /\* Tạo cấu trúc dữ liệu cho danh sách liên kết đơn \*/  struct Node  {  **int** x;  **int** i;      Node \*pNext;  };  struct SingleList  {      Node \*pHead;      Node \*pTail;  };  /\* Khởi tạo cho pHead và pTail \*/  void Initialize(SingleList \*&list)  {      list=new SingleList;      list->pHead=list->pTail=NULL;  }  /\* Tạo Node \*/  Node \*CreateNode(**int** x,**int** i)  {      Node \*pNode=new Node;      if(pNode==NULL)      {          cout<<"Loi cap phat bo nho";  **exit**(0);      }      pNode->i=i;      pNode->x=x;      pNode->pNext=NULL;      return pNode;  }  /\* insertlast \*/  void InsertLast(SingleList \*&list,**int** x,**int** i)  {      Node \*pNode=CreateNode(x,i);      if(list->pTail==NULL)          list->pHead=list->pTail=pNode;      else      {          list->pTail->pNext=pNode;          list->pTail=pNode;      }  }  /\*printlist\*/  void PrintList(SingleList \*list)  {      Node \*pTmp=list->pHead;      while(pTmp!=NULL)      {          if(pTmp->pNext!=NULL)              cout<<pTmp->x<<"^"<<pTmp->i<<"+";          else              cout<<pTmp->x<<"^"<<pTmp->i;          pTmp=pTmp->pNext;      }  }  /\* Hàm tính tổng \*/  **double** SumOfList(SingleList \*list)  {  **double** sum=0;      for(Node \*pTmp=list->pHead;pTmp!=NULL;pTmp=pTmp->pNext)      {  **double** value=**pow**(pTmp->x,pTmp->i);          sum+=value;      }      return sum;  }    **int** main(**int** argc, **char**\*\* argv) {      SingleList \*list;      Initialize(list);  **int** n,x;      cout<<"Nhập n:";      cin>>n;      cout<<"Nhập x:";      cin>>x;      for(**int** i=1;i<=n;i++)      {          InsertLast(list,x,i);      }      cout<<"Danh sách liên kết đơn: \n";      PrintList(list);  **double** sum=SumOfList(list);      cout<<"\nTổng các phần tử trong danh sách: "<<sum;      cout<<"\n----------------------------\n";    cout<<"Chương trình này được đăng tại Freetuts.net";  } |

**Kết quả:**



**Bài tập về danh sách liên kết đơn kiểu cấu trúc**

Trong bài tập này ta thực hiện một bài toán như sau:



* Tạo một cấu trúc phân số a/b.
* Xây dựng danh sách liên kết đơn có pHead, pTail. Dùng vòng for để thêm các phân số vào danh sách.
* Xuất danh sách theo dãy số.
* Xuất kết quả cuối cùng.

**Gợi ý:**

Tương tự như bài 1, ta thực hiện tạo cấu trúc dữ liệu cho danh sách liên kết đơn, một cấu trúc **PhanSo.**Sau đó, khởi tạo giá trị cho pHead và pTail bằng NULL.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | /\* Tạo cấu trúc cho phân số \*/  struct PhanSo  {  **int** tu;  **int** mau;  };  /\* Tạo cấu trúc dữ liệu cho danh sách liên kết đơn \*/  struct Node  {      PhanSo \*data;      Node \*pNext;  };  struct SingleList  {      Node \*pHead;      Node \*pTail;  };  /\* Khởi tạo cho pHead và pTail \*/  void Initialize(SingleList \*&list)  {      list=new SingleList;      list->pHead=list->pTail=NULL;  } |

Tiếp đến viết một hàm tạo Node với cấu trúc dữ liệu ở trên.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | /\* Tạo node \*/  Node \*CreateNode(**int** tu,**int** mau)  {      Node \*pNode=new Node;      if(pNode==NULL)      {          cout<<"Loi bo nho";  **exit**(0);      }      PhanSo \*ps=new PhanSo;      ps->mau=mau;      ps->tu=tu;      pNode->data=ps;      pNode->pNext=NULL;      return pNode;  } |

Để thêm dữ liệu cho danh sách ta cần một hàm InsertLast() và để xuất dữ liệu ta cần một hàm PrintList().

|  |  |
| --- | --- |
| 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 | /\* Insertlast \*/  void InsertLast(SingleList \*&list,**int** tu,**int** mau)  {      Node \*pNode=CreateNode(tu,mau);      if(list->pTail==NULL)      {          list->pHead=list->pTail=pNode;      }      else      {          list->pTail->pNext=pNode;          list->pTail=pNode;      }  }  /\* printlist \*/  void PrintList(SingleList \*list)  {      Node \*pTmp=list->pHead;      while(pTmp!=NULL)      {          PhanSo \*ps=pTmp->data;          cout<<ps->tu<<"/"<<ps->mau<<"+";          pTmp=pTmp->pNext;      }  } |

Cuối cùng là một hàm tính giai thừa GiaiThua() và hàm tính tổng SumOfList().

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | /\* hàm tính giai thừa \*/  **int** GiaiThua(**int** n)  {      if(n<=1)          return 1;      return n\*GiaiThua(n-1);  }  /\* hàm tính tổng \*/  **double** SumOfList(SingleList \*list)  {  **double** sum=0;      Node \*pTmp=list->pHead;      while(pTmp!=NULL)      {          PhanSo \*ps=pTmp->data;          sum=sum+(ps->tu\*1.0/ps->mau);          pTmp=pTmp->pNext;      }      return sum;  } |

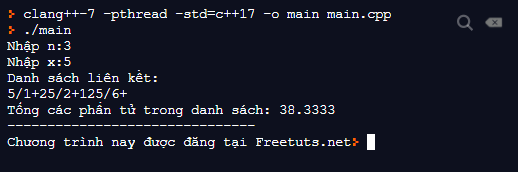
Để hiển thị và kiểm tra kết quả ta cần một hàm main().

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | **int** main(**int** argc, **char**\*\* argv) {      SingleList \*list;      Initialize(list);  **int** n,x;      cout<<"Nhập n:";      cin>>n;      cout<<"Nhập x:";      cin>>x;      for(**int** i=1;i<=n;i++)      {  **int** tu=(**int**) **pow**(x,i);  **int** mau=GiaiThua(i);          InsertLast(list,tu,mau);      }      cout<<"Danh sách liên kết: \n";      PrintList(list);  **double** sum=SumOfList(list);      cout<<"\nTổng các phần tử trong danh sách: "<<sum;  } |

**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>  #include <math.h>  using namespace std;  /\* Tạo cấu trúc cho phân số \*/  struct PhanSo  {  **int** tu;  **int** mau;  };  /\* Tạo cấu trúc dữ liệu cho danh sách liên kết đơn \*/  struct Node  {      PhanSo \*data;      Node \*pNext;  };  struct SingleList  {      Node \*pHead;      Node \*pTail;  };  /\* Khởi tạo cho pHead và pTail \*/  void Initialize(SingleList \*&list)  {      list=new SingleList;      list->pHead=list->pTail=NULL;  }  /\* Tạo node \*/  Node \*CreateNode(**int** tu,**int** mau)  {      Node \*pNode=new Node;      if(pNode==NULL)      {          cout<<"Loi bo nho";  **exit**(0);      }      PhanSo \*ps=new PhanSo;      ps->mau=mau;      ps->tu=tu;      pNode->data=ps;      pNode->pNext=NULL;      return pNode;  }  /\* Insertlast \*/  void InsertLast(SingleList \*&list,**int** tu,**int** mau)  {      Node \*pNode=CreateNode(tu,mau);      if(list->pTail==NULL)      {          list->pHead=list->pTail=pNode;      }      else      {          list->pTail->pNext=pNode;          list->pTail=pNode;      }  }  /\* printlist \*/  void PrintList(SingleList \*list)  {      Node \*pTmp=list->pHead;      while(pTmp!=NULL)      {          PhanSo \*ps=pTmp->data;          cout<<ps->tu<<"/"<<ps->mau<<"+";          pTmp=pTmp->pNext;      }  }  /\* hàm tính giai thừa \*/  **int** GiaiThua(**int** n)  {      if(n<=1)          return 1;      return n\*GiaiThua(n-1);  }  /\* hàm tính tổng \*/  **double** SumOfList(SingleList \*list)  {  **double** sum=0;      Node \*pTmp=list->pHead;      while(pTmp!=NULL)      {          PhanSo \*ps=pTmp->data;          sum=sum+(ps->tu\*1.0/ps->mau);          pTmp=pTmp->pNext;      }      return sum;  }    **int** main(**int** argc, **char**\*\* argv) {      SingleList \*list;      Initialize(list);  **int** n,x;      cout<<"Nhập n:";      cin>>n;      cout<<"Nhập x:";      cin>>x;      for(**int** i=1;i<=n;i++)      {  **int** tu=(**int**) **pow**(x,i);  **int** mau=GiaiThua(i);          InsertLast(list,tu,mau);      }      cout<<"Danh sách liên kết: \n";      PrintList(list);  **double** sum=SumOfList(list);      cout<<"\nTổng các phần tử trong danh sách: "<<sum;      cout<<"\n-------------------------------\n";    cout<<"Chương trình nay được đăng tại Freetuts.net";  } |

**Kết quả:**



Như vậy là chúng ta đã thực hiện xong hai bài tập áp dụng các thao tác trong danh sách liên kết đơn, các bạn hãy luyện tập thật nhiều các bài tập khác nữa nhé. Chúc các bạn thực hiện thành công!!!