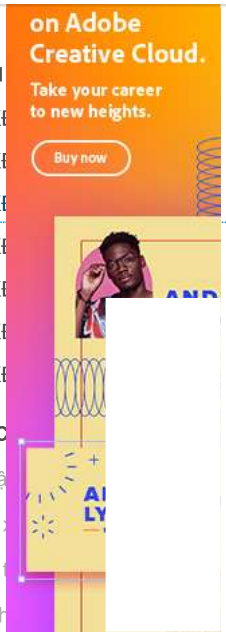


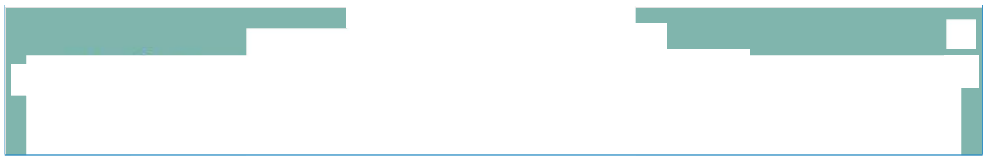







# Chèn Node vào danh sanh liên kết đơn

**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**

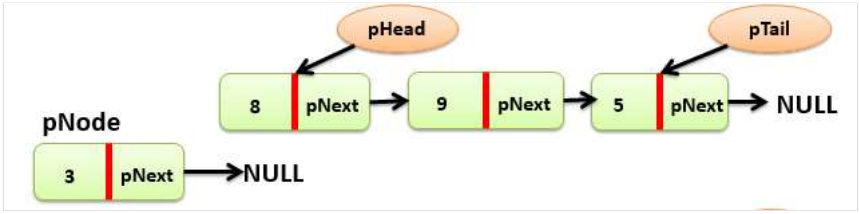


## 1. Chèn Node vào đầu danh sách liên kết đơn

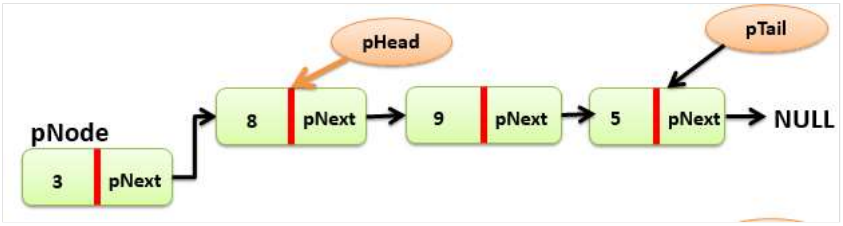
Trong phần này mình sẽ giới thiệu cách chèn Node vào đầu danh sách liên kết đơn.

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

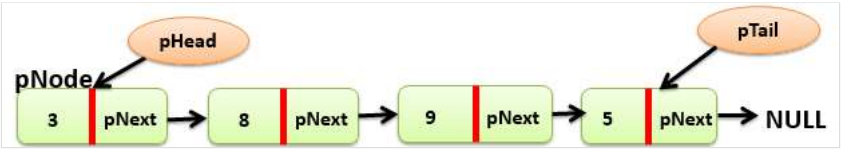
Để chèn được Node, việc đầu tiên các bạn phải có một danh sách liên kết và một Node đã được tạo sẵn.




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 .




```
/* 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;
  }
}
```



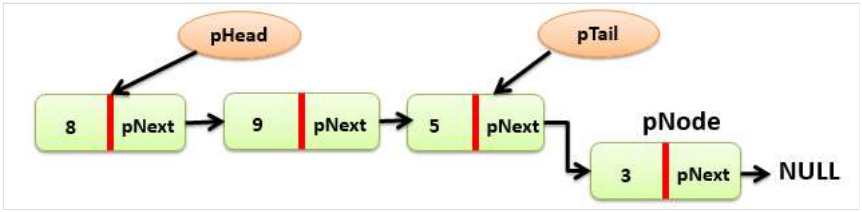
## 2. Chèn Node vào cuối danh sách liên kết đơn

Top

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.

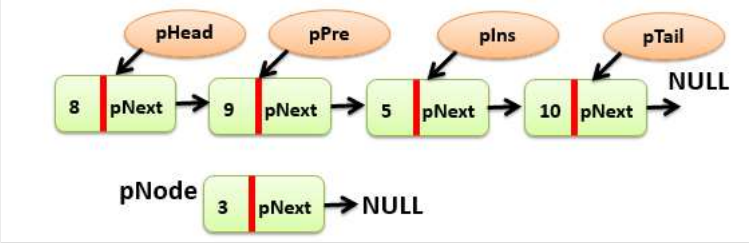



Vì pTail luôn luôn quản lý Node cuối cùng, vì vậy ta phải đưa pTail về phần tử cuối cùng: list.pTail=pNode;


```
/* 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;
  }
}
```

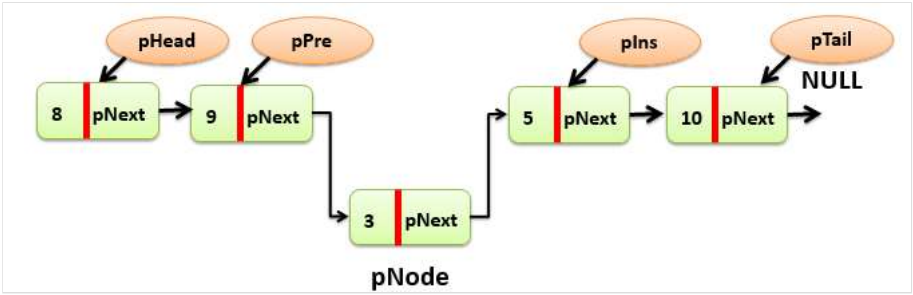
## 3. Chèn Node vào giữa danh sách liên kết đơn

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:

```
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).

```
/* 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:

```
/* 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( bên dưới các bạn chú ý.



```
#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";
}
```

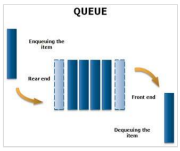
**Kết quả:**

```
clang++-7 -pthread -std=c++17 -o main main.cpp
./main
Các Node trong danh sách sau khi InsertFirst là: 3 7 5
Các Node trong danh sách sau khi InsertLast là: 3 7 5 4 2 6
Các Node trong danh sách sau khi InsertMid là: 3 7 12 13 5 4 11 2 6
-----------------------------------
Chương trình này được đăng tại Freetuts.net
```

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 !!!

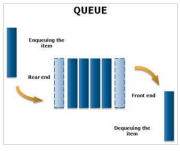
« BÀI TRƯỚC BÀI TIẾP »

## Cùng chuyên mục:



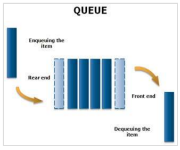

**Tìm các số chẵn lẻ bằng Queue và Stack**

Để làm được bài này các bạn cần có kiến thức về cấu trúc Queue…



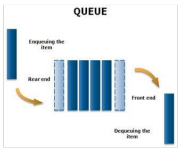

**Cài đặt hàng đợi Queue bằng mảng một chiều**

Chúng ta sẽ cùng nhau tìm hiểu về cách cài đặt hàng đợi Queue bằng…




**Cài đặt hàng đợi Queue bằng danh sách liên kết**

Chúng ta sẽ cùng nhau tìm hiểu về cách khởi tạo cấu trúc dữ liệu…




**Hàng đợi Queue là gì? Cấu trúc dữ liệu và các cách cài đặt Queue**

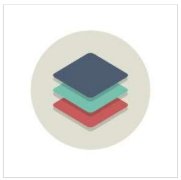
Trong hướng dẫn này mình sẽ giới thiệu các bạn một (

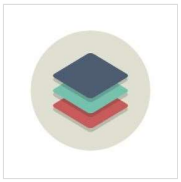


**Bài tập kiểm tra số nguyên tố bằng Stack**

Chúng ta sẽ cùng nhau tạo một cấu trúc Stack với danh sách liên kết…

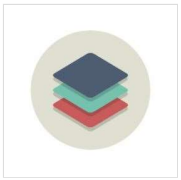
Top





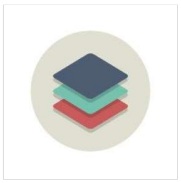
**Bài tập chuyển đổi cơ số bằng Stack**

Trong hướng dẫn này mình sẽ thực hiện giải một bài toán chuyển đổi cơ…



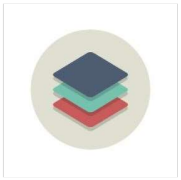
**Cài đặt Stack bằng mảng một chiều**

Chúng ta sẽ lần lượt thực hiện tạo các hàm cơ bản cho Stack như:…



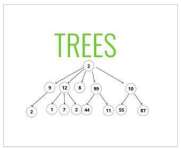
**Cài đặt Stack bằng danh sách liên kết**

Chúng ta sẽ thực hiện lần lượt các thao tác trong Stack sử dụng danh…



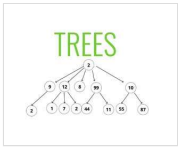
**Ngăn xếp Stack là gì? Cấu trúc và cơ chế hoạt động ra sao?**

Trong hướng dẫn này mình sẽ giới thiệu các bạn một cấu trúc lưu trữ…




**Xóa Node khỏi cây đỏ đen**

Chúng ta sẽ cùng nhau tìm hiểu về cách xóa một Node khỏi cây đỏ…




**Thêm Node mới vào cây đỏ đen**


**Cây đỏ đen là gì? Cấu trúc của Red-Black Tree**

Trong hướng dẫn này mình sẽ giới thiệu các bạn một cấu trúc dữ liệu…

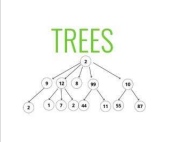

**Xóa Node khỏi cây nhị phân tìm kiếm**

Chúng ta sẽ cùng nhau thực hiện xóa Node có 1 con, Node có 2…




**Tìm Node MAX và MIN trong cây nhị phân tìm ki**

Chúng ta sẽ thực hiện một vài cách tìm ra giá trị MAX

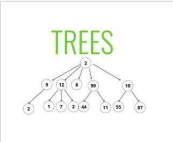



**Xuất Node con và lá trong cây nhị phân tìm kiếm**

Trong hướng dẫn này mình sẽ giới thiệu các bạn cách xuất các Node con…

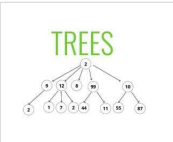

**Tìm kiếm Node trên cây nhị phân tìm kiếm**

Trong hướng dẫn này mình sẽ giới thiệu các bạn cách tìm kiếm một Node…



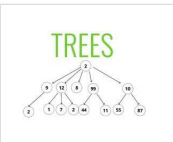

**Duyệt cây nhị phân tìm kiếm**

Chúng ta sẽ tìm hiểu lần lượt 6 cách duyệt cây nhị phân tìm kiếm:




**Thêm Node vào cây nhị phân tìm kiếm**

Trong hướng dẫn này mình sẽ giới thiệu các bạn về cấu trúc dữ liệu…




**Cấu trúc cây nhị phân là gì? Hoạt động ra sao?**

Trong bài này mình sẽ giới thiệu các bạn một trong các cấu trúc dữ…


**Gộp hai danh sách liên kết đôi**

Chúng ta sẽ cùng nhau tìm hiểu về cách nối hai danh sách liên kết…