TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM
KHOA CÔNG NGHỆ THÔNG TIN
MÔN: **KĨ THUẬT LẬP TRÌNH**

HƯỚNG DẪN THỰC HÀNH

TUẦN 05

DANH SÁCH LIÊN KẾT ĐƠN

(SINGLE LINKED LIST)

TP.HCM, ngày 10 tháng 05 năm 2020

MỤC LỤC

1 Khái niệm danh sách liên kết đơn:

Danh sách liên kết là một cấu trúc dữ liệu đệ quy. Danh sách này có thể rỗng hoặc chứa một phần tử và bản than phần tử này chứa mối liên kết chỉ đến danh sách liên kết khác.

Danh sách liên kết có ưu điểm so với mảng tĩnh và mảng động ở điểm có thể mở rộng dễ dàng cũng như tránh hiện tượng phân mảnh bộ nhớ heap.

Xem khai báo sau:

```
typedef struct node* ref;
struct node{
  int key;
  ref next;
};
```

Các dòng mã có nghĩa một cấu trúc tên node gồm 2 trường, trường dữ liệu số nguyên tên là key, trường dữ liệu thứ 2 là địa chỉ trỏ tới một node khác.

2 Các thao tác trong danh sách liên kết đơn.

Trong cấu trúc dữ liệu này, ta có một số thao tác cơ bản liên quan bao gồm: xây dựng danh sách liên kết, duyệt danh sách, hủy danh sách, thêm phần tử vào cuối danh sách, thêm phần tử vào giữa danh sách, xóa phần tử đầu, xóa phần tử cuối, xóa phần tử giữa, đếm số phần tử, chèn phần tử vào vị trí xác định, xóa phần tử tại vị trí xác định. Ta sẽ lần lượt xem xét các thao tác này.

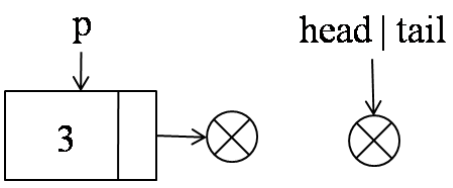
2.1 Thao tác xây dựng danh sách.

Ở thao tác này, giả sử ta có danh sách liên kết rỗng (head = tail = NULL), và ta cần thêm một node p nào đó vào danh sách rỗng này.

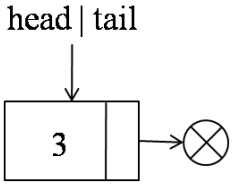
Dòng	
0	#include <stdio.h>
1	#include <stdlib.h>
2	typedef struct node* ref;
3	struct node{
4	int key;
5	ref next
6	};
7	
8	ref getNode(int k){
9	ref p;

```
10    p = (ref)malloc(sizeof(struct node));
11    if(p == NULL){
12       printf("Khong du bon ho\n");
13       exit(0);
14    }
15    p->key = k;
16    p->next = NULL;
17    return p;
18 }
19
20 void addFirst(ref& head, ref& tail, int k){
21    ref p = getNode(k);
22    if(head == NULL){
23       head = tail = NULL;
24    }
25    else{
26       p->next = head;
27       head = p;
28    }
29 }
30
31 void main(){
32    int k;
33    ref head = NULL, tail = NULL;
34    while(1){
35       printf("Nhap so nguyen(nhap so 0 de thoat): ");
36       scanf("%d", &k);
37       if(k == 0)break;
38       addFirst(head, tail, k);
39    }
40 }
```

Ta sẽ xem xét hình bên dưới để dễ hình dung. Lúc đầu danh sách là rỗng và ta có một node p (Hình 1a). Sau đó, p được thêm vào danh sách, đồng thời con trỏ head và tail cũng lần lượt được cập nhật lại thông tin.



Hình 1a



Hình 1b

2.2 Thao tác duyệt danh sách

Ở thao tác này, ta sẽ dùng vòng lặp để duyệt từ node đầu tiên (head) cho tới node cuối cùng (tail) trong danh sách.

Dòng	
1	void printList(ref head){
2	ref p;
3	if(head == NULL)
4	printf("\nDanh sach rong");
5	else
6	for(p = head; p; p = p->next)
7	printf("%d\n", p->key);
8	}
9	
10	void main(){
11	int k;
12	ref head = NULL, tail = NULL;
13	while(1){
14	printf("Nhap so nguyen (nhap so 0 de thoat): ");
15	scanf("%d", &k);
16	if(k == 0) break;
17	addFirst(head, tail, k)
18	}
19	printList(head);
20	}

2.3 Thao tác hủy danh sách liên kết đơn

Do danh sách liên kết đơn có sử dụng việc cấp phát vùng nhớ nên việc hủy nó không đơn giản. Nếu hủy nhầm có thể dẫn tới hiện tượng rò rỉ bộ nhớ. Phương pháp hủy đó là ta dùng một con trỏ p để giữ địa chỉ của head, sau đó cho head nhảy lên một bước (head = head->next), rồi mới hủy con trỏ p. Quá trình tiếp tục cho tới hết danh sách.

Dòng	
1	void destroyList(ref& head){
2	ref p;
3	while(head){
4	p = head;
5	head = head->next;
6	free(p);
7	}
8	}

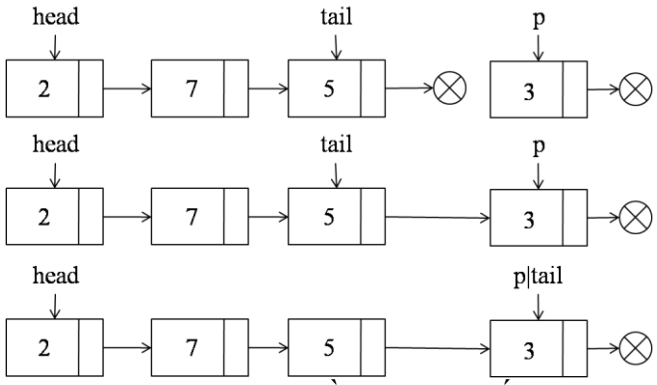
9	
10	void main(){
11	int k;
12	ref head = NULL, tail = NULL;
13	while(1){
14	printf(“Nhap so nguyen (nhap so 0 de thoat): ”);
15	scanf(“%d”, &k);
16	if(k == 0) break;
17	addFirst(head, tail, k)
18	}
19	printList(head);
20	destroyList(head);
21	}

2.4 Thao tác thêm phần tử vào cuối danh sách

Trong thao tác xây dựng danh sách liên kết đơn, ta đã có thao tác thêm phần tử vào đầu danh sách (addFirst). Giờ đây ta tiếp tục xây dựng hàm để đưa phần tử vào cuối danh sách liên kết đơn.

Dòng	
1	void addLast(ref& head, ref& tail, int k){
2	ref p = getNode(k);
3	if(head == NULL)
4	head = tail = NULL;
5	else{
6	tail->next = p;
7	tail = p;
8	}
9	}
10	void main(){
11	int k;
12	ref head = NULL, tail = NULL;
13	while(1){
14	printf(“Nhap so nguyen (nhap so 0 de thoat): ”);
15	scanf(“%d”, &k);
16	if(k == 0) break;
17	addFirst(head, tail, k)
18	}
19	printList(head);
20	destroyList(head);
21	}

Hình vẽ bên dưới minh họa việc thêm phần tử node p vào cuối danh sách, và sau cùng là cập nhật con trỏ tail về đúng vị trí sau khi thêm phần tử mới.



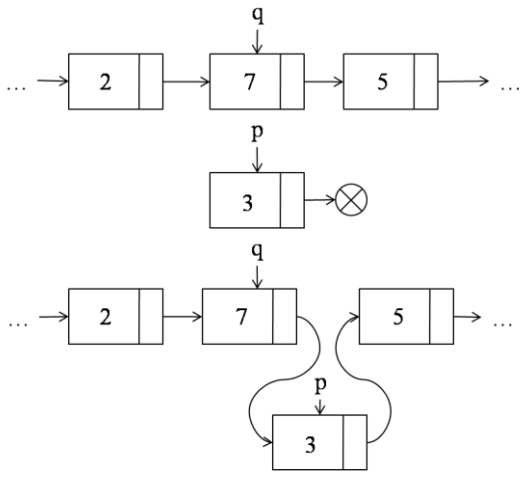
Hình 2: thao tác thêm phần tử vào cuối danh sách

2.5 Thêm phần tử vào sau một phần tử nào đó trong danh sách

Ở thao tác này, ta có nhu cầu thêm phần tử node p vào phía sau phần tử node q nào đó trong danh sách. Lúc này ta cần cho p->next giữ địa chỉ của q->next. Sau đó ta cập nhật lại q->next giữ địa chỉ của p.

Dòng	
1	void insertAfter(ref q, int k){
2	ref p;
3	p = getNode(k);
4	p->next = q->next;
5	q->next = p;
6	}

Xem hình vẽ để dễ hình dung hơn các thao tác trong đoạn mã



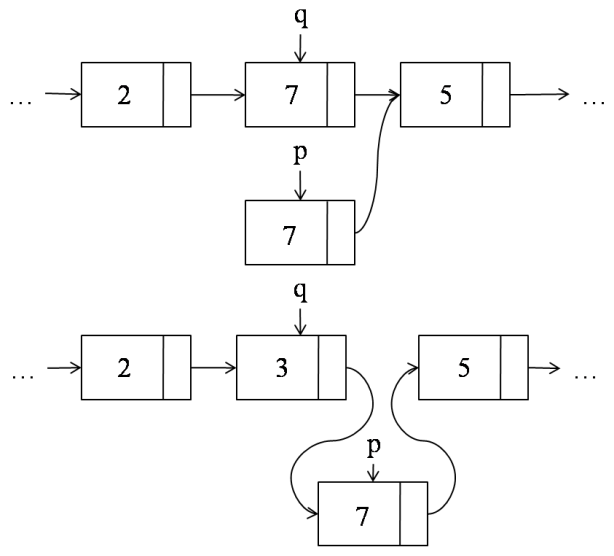
Hình 3: Thêm phần tử p vào sau phần tử q trong danh sách

2.6 Thêm phần tử vào trước một phần tử nào đó trong danh sách

Ở thao tác này, ta có nhu cầu thêm phần tử node p vào phía trước phần tử node q nào đó trong danh sách. Ở thao tác này ta sử dụng kĩ thuật sao chép **nội dung** (*p = *q) của node q vào node p, sau đó ta chuyển q->next giữ địa chỉ của node p này, và cuối cùng ta sửa nội dung key của node q thành nội dung key của node p. (Lưu ý: nếu phần tử q giữ địa chỉ phần tử cuối của danh sách thì hàm này sẽ gây ra lỗi gián tiếp do chưa cập nhật địa chỉ cho con trỏ tail)

Dòng	
1	void insertBefore(ref q, int k){
2	ref p;
3	p = (ref)malloc(sizeof(struct node));
4	if(p == NULL){
5	printf("Loi khong du bo nho\n");
6	exit(0);
7	}
8	*p = *q;
9	q->next = p;
10	q->key = k;
11	}

Xem hình vẽ để dễ hình dung quá trình chuyển đổi và kết nối.



Hình 4: Thêm phần p vào trước phần tử q trong danh sách

2.7 Thao tác xóa phần tử đầu danh sách liên kết

Vì thao tác liên quan tới con trỏ, nên ta không thể đơn giản gọi free(head) để xóa phần tử đầu danh sách. Ta cần cập nhật head = head->next trước khi xóa phần tử đầu này.

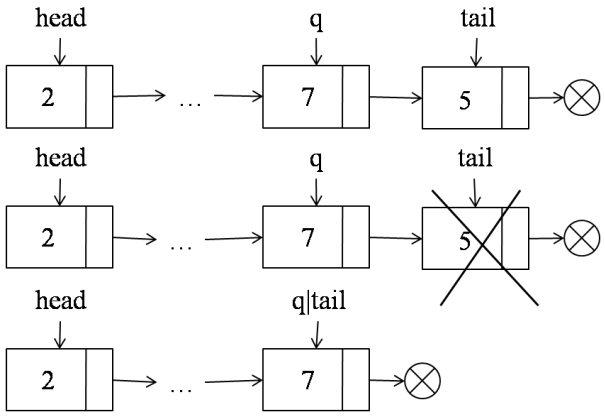
Dòng	
1	void deleteBegin(ref& head, ref& tail){
2	ref q;
3	if(head == tail){
4	free(head);
5	head = tail = NULL;
6	}
7	else{
8	q = head;
9	head = head->next;
10	free(q);
11	}
12	}

2.8 Thao tác xóa phần tử cuối danh sách liên kết

Tương tự như phần tử đầu danh sách, ta cũng không thể dùng free(tail) mà cần kiểm tra trước khi thực hiện xóa phần tử cuối vì nếu không thì ta sẽ mất địa chỉ tail của danh sách.

Dòng	
1	void deleteEnd(ref& head, ref& tail){
2	ref q;
3	if(head == tail){
4	free(head);
5	head = tail = NULL;
6	}
7	else{
8	for(q=head; q->next != tail; q = q->next);
9	free(tail);
10	tail = q;
11	q->next = NULL;
12	}
13	}

Xem hình vẽ để dễ hình dung thao tác xóa phần tử cuối danh sách.



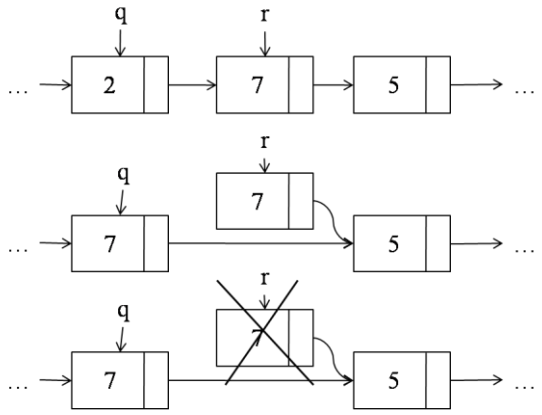
Hình 5: Thao tác xóa phần tử cuối trong danh sách liên kết

2.9 Thao tác xóa phần tử giữa trong danh sách liên kết

Thao tác này sẽ xóa phần tử bất kì nào khác head và tail. Lưu ý trong đoạn mã bên dưới sẽ gây ra **lỗi gián tiếp** nếu node q cần xóa nằm kế node tail.

Dòng	
1	void deleteMiddle(ref q){
2	ref r;
3	r = q->next;
4	*q = *r;
5	free(r);
6	}

Hình bên dưới minh họa cho thao tác xóa phần tử giữa trong danh sách



Hình 6: Thao tác xóa phần tử giữa trong danh sách liên kết

2.10 Thao tác đếm số phần tử trong danh sách liên kết

Ở thao tác này, ta chỉ cần duyệt danh sách từ đầu tới cuối, sau đó tăng biến đếm lên từng bước.

Dòng	
1	int length(ref head){
2	ref p;
3	int count = 0;
4	for(p = head; p; p = p->next) count++;
5	return count;
6	}

2.11 Chèn phần tử tại một vị trí xác định trong danh sách liên kết

Ở thao tác này, ta sẽ sử dụng lại các hàm xây dựng bên trên để thực hiện chèn một phần tử vào vị trí xác định trong danh sách liên kết.

Dòng	
1	void insertAt(ref& head, ref& tail, int pos, int k){
2	int n, i;
3	ref q;
4	n = length(head);
5	if((pos < 0) || (pos > n)){
6	printf("Vi tri chen khong phu hop\n");
7	return;
8	}
9	if(pos == 0) addFirst(head, tail, k);
10	else{
11	if(pos == n) addLast(head, tail, k);
12	else{
13	for(i = 0, q = head; i < pos; i++, q = q->next);
14	insertBefore(q, k);
15	if(tail->next)tail = tail->next;
16	}
17	}
18	}

2.12 Xóa phần tử tại một vị trí xác định trong danh sách liên kết

Ở thao tác này, ta sẽ sử dụng lại các hàm xây dựng bên trên để thực hiện xóa một phần tử tại vị trí xác định trong danh sách liên kết.

Dòng	
1	void deleteAt(ref& head, ref& tail, int pos){
2	int n, i;
3	ref q;
4	n = length(head);
5	if((pos < 0) || (pos >= n)){

```
6       printf("Vi tri xoa khong phu hop\n");
7       return;
8     }
9     if(pos == 0) deleteBegin(head, tail);
10    else{
11      if(pos == n – 1) deleteEnd(head, tail);
12      else{
13        for(i = 0, q = head; i < pos; i++, q = q->next);
14        if(q->next == tail) tail = q;
15        deleteMiddle(q);
16      }
17    }
18  }
```

3 Bài tập.

Xem kĩ các ví dụ trên, sinh viên hãy xây dựng danh sách liên kết các PHANSO gồm các hàm chính có chức năng như sau:

- Xây dựng danh sách các phân số

- Xuất danh sách các phân số

- Thêm một phân số vào cuối danh sách

- Chèn một phân số tại một vị trí xác định trong danh sách

- Xóa một phân số tại một vị trí xác định trong danh sách

- Hủy toàn bộ danh sách phân số

Sinh viên tự thiết kế đầu vào và đầu ra, cũng như tất cả các hàm phụ trợ, sau đó viết hàm main() **minh họa các hàm chính** vừa xây dựng.