**BÀI TẬP THỰC HÀNH**

**CHƯƠNG 3 : DANH SÁCH LIÊN KẾT**

**Mục đích :**

1. Nắm vững nguyên lý dữ liệu động: Vùng nhớ Heap, biến động, địa chỉ, con trỏ
2. Tổ chức dữ liệu bằng danh sách liên kết : Cấu trúc NODE và LIST, tạo Node, tạo danh sách là tập hợp các Node.
3. Các thao tác trên danh sách liên kết : Tạo danh sách, duyệt danh sách, thêm một nút có dữ liệu x, xóa một số nút thỏa mãn điều kiện x, tìm kiếm một NODE trong LIST, sắp xếp theo khóa của một LIST.
4. Ứng dụng cấu trúc dữ liệu danh sách liên kết đơn vào bài toán tin học.

**Bài 1 : Tổ chức dữ liệu và tạo danh sách liên kết đơn**

***Yêu cầu:***

1. Định nghĩa một **Node** để lưu một số nguyên và lưu địa chỉ của **Node** cùng kiểu.
2. Xây dựng hàm ***Node\* GetNode (int x)*** thự hiện tạo một **Node** mà trường **Data** nhận giá trị x và trường **pNext** nhận giá trị NULL. Hàm trả về con trỏ TRỎ vào nút vừa tạo hoặc trả về NULL trong trường hợp không thành công.
3. Xây dựng hàm tạo danh sách rỗng **void Init (List &L)**
4. Xây dựng hàm **void AddHead (List &L, Node\* p)** hoặc **void AddFirst (List &L, int x)** để thêm 1 Node mới vào đầu danh sách được quản lý bởi L.
5. Xây dựng hàm **void AddTail (List &L, Node\* p)** hoặc **void AddLast (List &L, int x)** để thêm 1 Node mới vào đầu danh sách được quản lý bởi L.
6. Xây dựng hàm **void CreateListHead( List &L, int n)** để tạo danh sách gồm n Node bằng cách chèn đầu.
7. Xây dựng hàm **void CreateListTail ( List &L, int n)** để tạo danh sách gồm n Node bằng cách chèn đầu.
8. Xuất danh sách liên kết đơn L
9. Tạo Menu thực hiện các hàm trong chương trình quản lý danh sách liên kết đơn

**Ví dụ minh họa**

#include <conio.h>

#include <iostream.h>

#include <alloc.h>

#include <stdlib.h>

**//Định nghĩa Node và List**

struct Node

{ int Data;

Node \* pNext;

};

struct List

{Node \*pHead,\*pTail;

};

**// Hàm GetNode tạo một Node mới**

Node\* GetNode(int x)

{ Node \*p;

p=new Node;

if (p==NULL)

{ cout<<"Khong du bo nho";

return NULL;

}

p->Data=x;

p->pNext=NULL;

return p;

}

**//Hàm khởi tạo danh sách rỗng**

void Init(List &L)

{

l.pHead = l.pTail = NULL;

}

**/\*Hàm thêm 1 Node vào đầu danh sách với Input là danh sách L và con trỏ p trò đến Node đã được tạo bởi hàm GetNode**

**\*/**

void AddHead(List &L, Node\* p)

{ if (L.pHead == NULL) *// DS rỗng*

L.pHead = L.pTail = p;

else

{ p->pNext = L.pHead;

L.pHead = p;

}

}

**/\*Hàm thêm 1 Node có Data bằng x vào đầu danh sách với Input là danh sách L và giá trị x để tạo Node mới bằng hàm GetNode**

**\*/**

void AddFirst(List &L, int x)

{ Node \*p = GetNode(x);

if (L.pHead == NULL) *// DS rỗng*

L.pHead = L.pTail = p;

else

{ p->pNext = L.pHead;

L.pHead = p;

}

}

**/\*Hàm thêm 1 Node vào cuối danh sách với Input là danh sách L và con trỏ p trò đến Node đã được tạo bởi hàm GetNode**

**\*/**

void addTail(List &L, Node \*p)

{ if (l.pHead == NULL)

l.pHead = l.pTail = p;

else

{

l.pTail->pNext = p;

l.pTail = p ;

}

}

**/\*Hàm thêm 1 Node có Data bằng x vào cuối danh sách với Input là danh sách L và giá trị x để tạo Node mới bằng hàm GetNode**

**\*/**

void AddLast(List &L, int x)

{ Node \* p = GetNode (x);

if (l.pHead == NULL)

l.pHead = l.pTail = p;

else

{ l.pTail->pNext = p;

l.pTail = p ;

}

}

**// Hàm tạo danh sách L gồm n nút bằng cách chèn đầu**

void CreateListHead (List &L, int &n)

{int x; Node \*p;

cout<<”Nhap so node n = “; cin>>n;//nhập từ bàn phím

srand (time (NULL));//<time.h>, nhập ngẫu nhiên

for (int i=1; i<=n; i++)

{ cout<<”x[“<<i<<”]=”; cin>>x;

//x = rand()%90 +10;

//Tạo ngẫu nhiên số nguyên dương có đúng 2 chữ số

p = GetNode (x);

AddHead (L,p); //AddFirst(L,x);

}

}

**// Hàm tạo danh sách L gồm n nút bằng cách chèn cuối**

void CreateListTail (List &L, int &n)

{int x; Node \*p;

cout<<”Nhap so node n = “; cin>>n;

srand (time (NULL));//<time.h>, nhập ngẫu nhiên

for (int i=1; i<=n; i++)

{ cout<<”x[“<<i<<”]=”; cin>>x;

//x = rand()%199 - 99;

// x ∈ [-99 .. 99]

p = GetNode (x);

AddTail (L,p); //AddLast(L,x);

}

}

**// Hàm xuất danh sách liên kết đơn L**

void PrintList (List L)

{ for(Node\* p = L.pHead; p!=NULL; p=p->pNext)

cout<<p->info<<" ";

}

**//Hàm tạo Menu**

void Menu()

{cout<<”\nChuong trinh quan ly danh sach lien ket don”;

cout<<”\n1. Tao DSLK don gom n node bang cach chen dau tu ban phím”;

cout<<”\n2. Tao DSLK don gom n node bang cach chen dau tu ban phím”;

cout<<”\n3. Tao DSLK don gom n node bang cach chen cuoi tu ban phím”;

cout<<”\n4. Tao DSLK don gom n node bang cach chen cuoi tu ban phím”;

//…

}

**//Hàm chính**

int main()

{int n,chon;

List L;

Init(L);

Menu();

do

{ cout<<”Chon chuc nang, chon 0 de thoat “;

switch (chon)

{ case 1 : CreateListHead (L,n); break;

case 2 : CreateListTail (L,n); break;

// …

}

}while (chon!=0);

return 0;

}

**Bài tập :**

Tạo một danh sách liên kết đơn gồm n node với data là các số nguyên được tạo ngẫu nhiên sao cho khi xuất danh sách sẽ tạo nên dãy các số nguyên thỏa mãn yêu cầu sau :

1. Tăng dần (Giảm dần)
2. Xen kẻ chẳn lẻ (Âm dương)
3. Các số nguyên trong khoảng [a , b] với a, b là 2 số nguyên nhập từ bàn phím
4. Chỉ bao gồm các số nguyên tố, …

**Bài 2 : Chèn 1 node vào giữa danh sách liên kết. Tìm kiếm trong DSLK**

**// Hàm chèn 1 Node được trỏ bởi con trỏ p vào sau nút được trỏ với con trỏ q trong DSLK**

void AddAfter (List &L, Node \*q, Node\*p)

{ if (q!=NULL)

{ p->pNext = q->pNext;

q->pNext = p;

if(q == l.pTail)

l.pTail = p;

}

}

**//Hàm tìm kiếm node có khóa x trong DSLK**

Node\* Search (List L, int x)

{ Node\* p = L.pHead;

while (p!=NULL && p->data!=x) p=p->pNext;

return p;

}

**//Hàm tìm kiếm phần tử lớn nhất trong DSLK**

Node\* MaxNode (List L)

{ if (L.pHead ==NULL) return NULL;

Node\* Max = L.pHead, p;

for(p=L.pHead->pNext; p!=NULL; p=p->pNext)

if (p->Data>Max->Data) Max=p;

return Max;

}

**// Hàm tính tổng các khóa của các node trong DSLK**

int SumList (List L)

{ int S=0;

for(Node \*p=L.pHead; p!=NULL; p=p->pNext)

S+=p->Data;

return S;

}

**// Hàm đếm số node trong DSLK**

int CountList (List L)

{ int d=0;

for(Node \*p=L.pHead; p!=NULL; p=p->pNext,d++);

return d;

}

**Bài tập**

1. Viết hàm chèn một Node mới có khóa là x vào **sau** Node thỏa tính chất sau trong DSLK
   1. Node có khóa lớn nhất (nhỏ nhất) .
   2. Node có khóa lá số chẳn (lẻ/nguyên tố/ …) đầu tiên.
   3. Node có khóa lá số chẳn (lẻ/nguyên tố/ …) lớn nhất.
2. Viết hàm chèn một Node mới có khóa là x vào **trước** Node thỏa các tính chất trong DSLK như câu 1
3. Viết hàm tính tổng khóa các node có khóa là chẳn/lẻ/số nguyên tố/ …
4. Viết hàm đếm số node có khóa là chẳn/lẻ/số nguyên tố/ …
5. Viết hàm chèn một nút mới có khóa là x vào sau node thứ i trong DSLK.

**Bài 3 : Xóa node trong DSLK**

**// Hàm xóa node đầu DSLK**

int removeHead (List &L)

{ if (L.pHead == NULL) return 0;

Node\* p=L.pHead;

L.pHead = p->pNext;

if (L.pHead == NULL) L.pTail=NULL; //Nếu danh sách rỗng

delete p;

return 1;

}

**// Hàm xóa node sau nút được trỏ bởi con trỏ q**

int removeAfter (List &L, Node \*q )

{ if (q !=NULL && q->pNext !=NULL)

{ Node\* p = q->pNext;

q->pNext = p->pNext;

if (p==L.pTail) L.pTail = q;

delete p;

return 1;

}

else return 0;

}

**// Xóa node có khóa k**

int removeNode (List &L, int k)

{ Node \*p = l.pHead;

Node \*q = NULL;

while (p != NULL)

{ if (p->data == k) break;

q = p;

p = p->pNext;

}

if (p == NULL) { cout<<“Không tìm thấy k”; return 0;}

else if (q == NULL)

// thực hiện xóa phần tử đầu ds là p

else

// thực hiện xóa phần tử p sau q

}

**Bài tập**

1. Viết hàm xóa node **sau** node có tính chất sau :
   1. Lớn nhất/Nhỏ nhất
   2. Là số chẳn/lẻ/Nguyên tố/ … đầu tiên trong danh sách
   3. Là số chẳn/lẻ/Nguyên tố/ … lớn nhất trong danh sách
2. Viết hàm xóa node **trước** node có tính chất như câu 1
3. Viết hàm xóa node thỏa tính chất câu 1

**Bài 4 : Sắp xếp DSLK**

**// Sắp xếp DSLK bằng phương pháp chèn trực tiếp**

void ListInterChangeSort ( List &L )

{ for ( Node\* p=L.pHead ; p!=L.pTail ; p=p->pNext )

for ( Node\* q=p->pNext ; q!=NULL ; q=q->pNext )

if ( p->data > q->data )

Swap( p->data , q->data );

}

**// Sắp xếp DSLK bằng phương pháp chọn trực tiếp**

void ListSelectionSort (List &L)

{ for ( Node\* p = L.pHead ; p != L.pTail ; p = p->pNext)

{ Node\* min = p;

for ( Node\* q = p->pNext ; q != NULL ; q = q->pNext)

if ( min->data > q->data ) min = q ;

Swap (min->data, p->data);

}

}

**// Sắp xếp DSLK bằng phương pháp QuickSort**

**//Nối 2 danh sách**

void SListAppend(List &L, List &L2)

{ if (L2.pHead == NULL) return;

if (L.pHead == NULL) L = L2;

else

{L.pTail->pNext = L2.pHead;

L.pTail = L2.pTail;

}

Init(L2);

}

**// QuickSort**

void SListQSort(List &L)

{ Node \*X, \*p;

List L1, L2;

if (L.pHead == L.pTail) return;

Init(L1); Init(L2);

X = L.pHead; L.pHead=X->pNext;

while (L.pHead!= NULL)

{ p = L.pHead;

if (p->data <= X->data) AddHead(l1, p);

else AddHead(l2, p);

}

SListQSort(l1); SListQSort(l2);

SListAppend(l, l1);

AddTail(l, X);

SListAppend(l, l2);

}

**Bài tập**

1. **Sắp xếp danh sách bằng phương pháp MergeSort**
2. **Sắp xếp danh sách bằng phương pháp RadixSort**

**Bài tập tổng hợp về DSLK đơn**

**//Tách một danh sách thành hai danh sách chẳn và lẻ**

void SplitListEvenOdd(List l, List &L1, List &L2)

{for (Node \*p=l.pHead;p!=NULL;p=p->pnext)

if (p->Data%2!=0)

AddTail(l1,GetNode(p->info));

else

AddTail(l2,GetNode(p->info));

}

**//Đào ngược danh sách**

void ReverseList(List &L)

{ List LR;Init(LR);

Node \*p;

while (L.pHead!=NULL)

{ p=L.pHead;

L.pHead = p->pNext;

AddHead(LR,p);

}

L=LR;

}

**Bài tập lớn :**

**BT1.**Viết chương trình thực hiện các yêu cầu sau:

a.Khai báo cấu trúc dữ liệu của một danh sách liên kết đơn các tỉnh. Biết rằng thông tin của mỗi tỉnh bao gồm: tên tỉnh, diện tích, dân số

b.Cài đặt các thao tác cơ bản cho danh sách liên kết đơn các tỉnh (thêm, sửa , xóa, duyệt).

c.Tính tổng diện tích của tất cả các tỉnh trong danh sách liên kết

d.Tìm địa chỉ của node chứa tính có diện tích lớn nhất trong danh sách liên kết.

e.Tìm một tỉnh có dân số lớn nhất.

f.Sắp xếp danh sách tăng dần theo diện tích.

**BT2.**Viết chương trình thực hiện các yêu cầu sau:

a.Khai báo cấu trúc dữ liệu của một danh sách liên kết đơn để lưu tọa độ các đỉnh của một đa giác lồi trong mặt phẳng OXY.

b.Tính chu vi của đa giác.

c.Tính diện tích của đa giác.

**BT3.**Cho một danh sách liên kết, mỗi nút chứa một số nguyên.

a.Thêm một phần tử có giá trị x vào đầu danh sách

b.Chỉ giữ lại một giá trị trong số các giá trị giống nhau.

c.Kiểm tra xem danh sách có được sắp xếp tăng dần hay không?

d.Đảo ngược danh sách.

e.Sắp xếp các số chẵn trong danh sách theo thứ tự tăng, sắp xếp các số lẻ theo thứ tự giảm dần, các số 0 giữ nguyên vị trí..

**BT4.**Cộng hai đa thức (mỗi node có 3 thành phần: hệ số khác 0 của một số hạng, số mũ tương ứng và mốc nối tới node tiếp theo).

**BT5**. Số nguyên lớn là số nguyên có chứa vài chục chữ số trở lên. Sử dụng danh sách liên kết đơn với mỗi node chứa các chữ số của số nguyên lớn. Hãy khai báo cấu trúc dữ liệu danh sách liên kết trên và viết các hàm cộng, trừ, nhân chia 2 số nguyên lớn trong trường hợp mỗi node có khóa là một số nguyên chứa

1. Một chữ số của số nguyên lớn
2. Hai chữ số (kiểu char), bốn chữ số (kiểu int) hay chín chữ số (kiểu long) của số nguyên lớn.

**BT6**. Danh sách liên kết đơn với mỗi node chứa một phân số (tử và mẫu là các số nguyên). Hãy khai báo cầu trúc dữ liệu một node và danh sách trên và viết các hàm thực hiện các yêu cầu sau:

1. Nhập và xuất danh sách
2. Tìm node chứa phân số nhỏ nhất
3. Kiểm tra các node chứa các phân số tối giản ?
4. Xuất phân số tổng danh sách trên.
5. Kiểm tra danh sách có sắp xếp thứ tự ?
6. Sắp xếp danh sách tăng dần.

**BT7**. Danh sách liên kết đơn với mỗi node chứa một điểm trong mặt phẳng (hoành độ x và tung độ y là các số nguyên). Hãy khai báo cấu trúc dữ liệu một node và danh sách trên và viết các hàm thực hiện các yêu cầu sau:

1. Nhập và xuất danh sách
2. Tìm node gần gốc tọa độ O(0,0) nhất
3. Tìm 2 node chứa 2 điểm gần/xa nhất trong các điểm trên.
4. Sắp xếp danh sách tăng dần của khoảng cách các node đến gốc tọa độ O(0,0).

**Đề thi tổng hợp**

**Đề 1 :**

Cho danh sách liên kết đơn bao gồm các **Node** lưu thông tin của các thí sinh dự thi như sau :

* Số báo danh (**SBD**) : kiểu số nguyên
* Họ và tên thí sinh (**HTTS**)  : 30 ký tự
* Điểm môn 1 (**M1**), môn 2 (**M2**), môn 3 (**M3**) : kiểu số thực

1. (1 điểm) Khai báo cấu trúc dữ liệu **Node** và danh sách liên kết L chỉ được quản lý bằng con trỏ đầu **pHead.**
2. (1.5 điểm) Cho số thực X, viết hàm đếm xem trong danh sách có bao nhiêu thí sinh có tổng điểm thi không nhỏ hơn X theo prototype : **int CountTS (List L, float X)** ;
3. (1.5 điểm) Cho số nguyên S, viết hàm xuất tổng điểm của thí sinh có số báo danh là S nếu có (nếu không có xuất -1) theo prototype : **float SumS (List L, int S)** ;

**Đề 2 :**

Cho mảng 1 chiều a gồm những phần tử có khoá là số nguyên và danh sách liên kết đơn L bao gồm các phần tử **Node**, mỗi **Node** có cấu trúc vùng dữ liệu **key** có kiểu số nguyên và vùng liên kết **pNext** trỏ đến Node kế, danh sách L được quản lý chỉ bằng con trỏ đầu danh sách **pHead** và con trỏ cuối danh sách **pTail**. Hãy thực hiện các yêu cầu sau bằng ngôn ngữ lập trình C++

1. (1 điểm) Khai báo cấu trúc dữ liệu mảng a, Node vá danh sách liên kết L. Khởi tạo danh sách L rỗng.
2. (1 điểm) Viết hàm tạo ngẫu nhiên mảng a gồm n phần tử có khoá gồm các số nguyên nằm trong khoảng [-99 .. 99] theo prototype : **void CreateArr (int a [ ], int n)**;
3. (1 điểm) Viết hàm sắp xếp **giảm dần** mảng a bằng một phương pháp sắp xếp đã học (nêu rõ tên phương pháp) theo prototype : **void Sort (int a [ ], int n)**;
4. (2 điểm) Từ danh sách liên kết L rỗng, viết thêm vào danh sách liên kết L n **Node** với vùng dữ liệu **key** có thứ tự **giảm dần** được lấy từ khoá của n phần tử mảng a đã được sắp xếp **giảm dần** ở cầu c theo prototype : **void CreateList (List &L, int a[ ], int n)**;
5. (2 điểm) Cho số nguyên **X** và danh sách **L** gồm các **Node** với vùng dữ liệu **key** đã có thứ tự **giảm dần**, viết hàm kiểm tra xem trong **L** có **Node** nào có vùng dữ liệu **key** có giá trị bằng **X** hay không ? (***Có*** *trả về* ***1****,* ***không có*** *trả về* ***0***), nếu chưa có hãy tạo một **Node** mới có **key** bằng **X** và vào chèn vào danh sách **L** sao cho sau khi chèn danh sách **L** cũng có thứ tự **giảm dần** theo prototype : **int InsOrdedList (List &L, int X)**;