TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP. HCM
KHOA CÔNG NGHỆ THÔNG TIN

BÀI TẬP THỰC HÀNH

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

BÀI THỰC HÀNH SỐ 1:
CÁC GIẢI THUẬT TÌM KIẾM
(Số tiết: 3)

Mục đích :

1. Ôn tập các thao tác trên mảng một chiều: nhập/ xuất mảng, phát sinh mảng, đọc/ ghi mảng vào file.
2. Thuật toán tìm kiếm tuyến tính.
3. Thuật toán tìm kiếm nhị phân.
4. So sánh thời gian chạy thực tế của hai thuật toán tìm tuyến tính và tìm nhị phân khi kích thước mảng rất lớn.

Vấn đề 1: Ôn tập các thao tác trên mảng một chiều

Cho mảng a có n phần tử số nguyên. Viết các hàm thực hiện các công việc:nhập/ xuất mảng, phát sinh mảng, đọc/ ghi mảng vào file.

```
voidNhapMang(int a[], int n)
{
   for(int i=0; i<n; i++)
   {
      printf("a[%d]=", i);
      scanf("%d", &a[i]);
   }
}
voidPhatSinhMang(int a[], int n)
{
   srand(time(NULL));
   for(int i=0; i<n; i++)
   {
      a[i] = rand();
   }
}
voidXuatMang(int a[], int n)
{
   printf("\n");
   for(int i=0; i<n; i++)
   {
      printf("%10d", a[i]);
   }
}
```

```
//Ghi mảng a có n phần tử vào file text
intGhiMangVaoFileText(char* filename, int a[], int n)
{
   FILE* f = fopen(filename, "w");
   if(!f)//Không mở file để ghi được
      return 0;
   for(int i=0; i<n; i++)
      fprintf(f ,"%d\t",a[i]);//Ghi từng phần tử
a[i] vào file, cách nhau một tab
   fclose(f);
   return 1;//Ghi file thành công trả về 1
}
//Đọc file text vào mảng a
intDocFileTextVaoMang(char* filename, int a[], int&n)
{
   FILE* f = fopen(filename, "r");
   if(!f)//Không mở file được
      return 0;
   int i=0;
   while(!feof(f))//Trong khi chưa hết file
   {
      fscanf(f,"%d",&a[i]); //Đọc từng PT vào mảng
      i++; //đếm số phần tử
   }
   n = i;
}
```

Vấn đề 2: Thuật toán tìm kiếm tuyến tính (tìm tuần tự).

Là một phương pháp tìm kiếm một phần tử cho trước trong một danh sách bằng cách duyệt lần lượt từng phần tử của danh sách đó cho đến lúc tìm thấy giá trị mong muốn hay đã duyệt qua toàn bộ danh sách.

```
intLinearSearch(int a[], int n, int x)
{
   int i=0;
   while(i<n && a[i]!=x)
      i++;
   if (i<n) return i;   //a[i] là phần tử có khoá x
   return -1;  // tìm hết mảng nhưng không có x
}
```

```
void main(){
   int a[100], n, x;
   printf("Nhap so phan tu cua mang");
   scanf("%d", &n);}
   NhapMang(a, n);`
   printf("Nhap khoa can tim");
   scanf("%d", &x);}
   int kq = LinearSearch(a, n, x);
   if(kq == -1)
          printf("Khong tim thay");
   else
          printf("Tim thay tai vi tri %d", kq);
}
```

Cải tiến cài đặt thuật toán tìm kiếm tuyến tính sử dụng phần tử "lính canh" để giảm bớt một phép so sánh ở vòng lặp while.

```
intLinearSearch_CaiTien(int a[], int n, int x)
{
   int i=0;
   a[n] = x;   // thêm lính canh vào cuối mảng
   while(i<n && a[i]!=x)
      i++;
   if (i<n) return i;    //a[i] là phần tử có khoá x
   return -1;  // tìm hết mảng nhưng không có x
}
```

Nhận xét: Tìm kiếm tuyến tính là một giải thuật rất đơn giản khi hiện thực. Giải thuật này tỏ ra khá hiệu quả khi cần tìm kiếm trên một danh sách đủ nhỏ hoặc một danh sách chưa sắp thứ tự đơn giản. Trong trường hợp cần tìm kiếm nhiều lần, dữ liệu thường được xử lý một lần trước khi tìm kiếm: có thể được sắp xếp theo thứ tự, hoặc được xây dựng theo một cấu trúc dữ liệu đặc trưng cho giải thuật hiệu quả hơn.

Vấn đề 3: Thuật toán tìm kiếm nhị phân.

Thuật toán tìm kiếm nhị phân dùng để tìm kiếm phần tử trong một danh sách đã được sắp xếp, ví dụ như trong một danh bạ điện thoại sắp xếp theo tên, có thể tìm kiếm số điện thoại của một người theo tên người đó.

```
// Hàm tìm kiếm nhị phân (Đệ quy)
intBinarySearch(inta[],intleft,intright,intx)
{
     if(left>right)return-1;
   intmid=(left+right)/2;
   if(x==a[mid])
      return mid;
   if(x<a[mid])
      returnBinarySearch(a,left,mid-1,x);
   if(x>a[mid])
      returnBinarySearch(a,mid+1,right,x);
}

// Phát sinh mảng tăng
voidPhatSinhMangTang(int a[], int n)
{
   srand(time(NULL));
   a[0] = rand()%50;
   for(int i=1; i<n; i++)
   {
      a[i] = a[i-1] + rand()%10;
   }
}
// Chương trình chính
void main(){
   int a[100], n, x;
   printf("Ban can phat sinh mang co bao nhieu PT?");
   scanf("%d", &n);}
   PhatSinhMangTang(a, n);`
   printf("Nhap khoa can tim");
   scanf("%d", &x);}
   int kq = BinarySearch(a, 0, n-1, x);
   if(kq == -1)
          printf("Khong tim thay");
   else
          printf("Tim thay tai vi tri %d", kq);
}
```

Vấn đề 4: So sánh thời gian chạy thực tế của hai thuật toán tìm tuyến tính và tìm nhị phân. Thử nghiệm với số phần tử mảng khoảng 100.000.

```
#include <time.h>
clock_t start, end;

start = clock();
// tìm kiếm tuyến tính
end = clock();
double t = end - start;
printf("Thoi gian tim kiem tuyen tinh la: %f", t);

start = clock();
// tìm kiếm nhị phân
end = clock();
double t = end - start;
printf("Thoi gian tim kiem nhi phan la: %f", t);
```

```
#include <chrono>
using namespace std::chrono;
void CompareLinearBinarySearch(int A[], int n, int k)
{
auto    start    =    steady_clock::now(),    stop    =
steady_clock::now();
    start = steady_clock::now();
    std::cout << LinearSearch(A,n,k) << endl;
    stop = steady_clock::now();

    std::chrono::duration<double> t1 = stop - start;

    // Binary Search
    start = steady_clock::now();
    std::cout << BinarySearch(A,0,n-1,k) << endl;
    stop = steady_clock::now();
    std::chrono::duration<double> t2 = stop - start;

    if(t1.count() < t2.count()) cout << "Linear faster
than Binary";
    else if(t2.count() < t1.count()) cout << "Binary
faster than Linear";
    else cout << "Linear equal to Binary";
}
```

BÀI TẬP VỀ NHÀ:*(bắt buộc– sinh viên nộp vào đầu buổi thực hành sau)*

Hãy viết lại hàm BinarySearch không dùng đệ quy.

BÀI TẬP LÀM THÊM:*(sinh viên có thể nộp bài vào đầu buổi thực hành sau để lấy điểm cộng)*

Áp dụng các thuật toán tìm kiếm để xây dựng chương trình tra từ điển Anh-Việt.

Ghi chú: Định nghĩa cấu trúc WORD trong từ điển bao gồm từ gốc tiếng Anh và nghĩa của từ (tiếng Việt không dấu).

```
struct WORD
{
      char Name[256];
      char Meaning[512];
};
```

Yêu cầu: Xây dựng các hàm thực hiện các chức năng sau

1. Nhập thông tin một từ (Word)
2. Xuất thông tin một từ (Word)
3. Sắp xếp theo Name tăng dần
4. Tìm kiếm từ theo Name
5. Ghi một từ (Word) vào tệp
6. Đọc một từ (Word) trong tệp
7. Cập nhật Meaning của một Word trong tệp
8. Xóa một từ khỏi tệp

BÀI THỰC HÀNH SỐ 2:
CÁC GIẢI THUẬT SẮP XẾP

(Số tiết: 3)

Mục đích :

1. Cài đặt các giải thuật sắp xếp: chọn trực tiếp, chèn trực tiếp, đổi chỗ trực tiếp, nổi bọt và quick sort.
2. So sánh thời gian chạy thực tế của các phương pháp sắp xếpkhi kích thước mảng lớn.

Vấn đề 1: Các giải thuật sắp xếp

Sắp xếp là quá trình xử lý một danh sách các phần tử (hoặc các mẫu tin) để đặt chúng theo một thứ tự thỏa mãn một tiêu chuẩn nào đó dựa trên nội dung thông tin lưu giữ tại mỗi phần tử.

Một số phương pháp sắp xếp thông dụng như: Chọn trực tiếp (Selection sort), chèn trực tiếp (Insertion sort), đổi chỗ trực tiếp (Interchange sort), nổi bọt (Bubble sort), shell sort, heap sort, quick sort, merge sort, radix sort.

Trong đó, các thuật toán như Interchange sort, Bubble sort, Insertion sort, Selection sort là những thuật toán đơn giản dễ cài đặt nhưng chi phí cao. Các thuật toán Shell sort, Heap sort, Quick sort, Merge sort phức tạp hơn nhưng hiệu suất cao hơn. Cả hai nhóm thuật toán trên đều có một điểm chung là đều được xây dựng dựa trên cơ sở việc so sánh giá trị của các phần tử trong mảng (hay so sánh các khóa tìm kiếm). Riêng phương pháp Radix sort đại diện cho một lớp các thuật toán sắp xếp khác hẳn các thuật toán trước. Lớp thuật toán này không dựa trên giá trị của các phần tử để sắp xếp.

1. Trong bài thực hành này, sinh viên cài đặt các thuật toán sắp xếp sau đây:
    1. Chọn trực tiếp
    2. Chèn trực tiếp
    3. Đổi chỗ trực tiếp*(làm ở nhà)*
    4. Nổi bọt*(làm ở nhà)*

5. Quicksort

2. Chạy thử các thuật toán trên với các mảng sau:

   Mảng 1: 10 3 7 4 2 8 5 12
   Mảng 2: 14 33 27 10 35 19 42 44

3. Ở mỗi thuật toán sắp xếp, hãy xuất mảng để theo dõi mỗi khi có sự thay đổi trong mảng. Gợi ý: gọi hàm xuất mảng ngay sau khi gọi hàm hoán vị.

Vấn đề 2: So sánh thời gian thực tế của các thuật toán sắp xếp.

Bước 1: Phát sinh 5 mảng số nguyên, mỗi mảng có 50.000 phần tử. Lưu 5 mảng vừa tạo vào 5 tập tin mang1.int, mang2.int, mang3.int, mang4.int và mang5.int.

Bước 2: Chạy thử từng thuật toán sắp xếp cho 5 mảng trên và điền vàocác bảng thống kê sau:

Phương pháp chọn trực tiếp			
Mảng	**Thời gian chạy (mili giây)**	**Số lần so sánh (lệnh if)**	**Số lần hoán vị**
1			
2			
3			
4			
5			

Phương pháp chèn trực tiếp			
Mảng	**Thời gian chạy (mili giây)**	**Số lần so sánh (lệnh if)**	**Số lần hoán vị**
1			
2			
3			
4			
5			

Phương pháp đổi chỗ trực tiếp			
Mảng	**Thời gian chạy (mili giây)**	**Số lần so sánh (lệnh if)**	**Số lần hoán vị**
1			
2			
3			
4			
5			

Phương pháp nổi bọt			
Mảng	**Thời gian chạy (mili giây)**	**Số lần so sánh (lệnh if)**	**Số lần hoán vị**
1			
2			
3			
4			
5			

Phương pháp sắp xếp nhanh (quick sort)			
Mảng	**Thời gian chạy (mili giây)**	**Số lần so sánh (lệnh if)**	**Số lần hoán vị**
1			
2			
3			
4			
5			

BÀI TẬP VỀ NHÀ:*(bắt buộc– sinh viên nộp vào đầu buổi thực hành sau)*
Cho dãy số nguyên A như sau:

12 2 15 -3 8 5 1 -8 6 0 4 15

1. Sắp xếp dãy trên tăng dần.

2. Suy ra số lớn thứ 3 trong dãy.

3. Suy ra số lượng phần tử lớn nhất trong dãy.

4. Sắp xếp dãy trên theo thứ tự giá trị tuyệt đối tăng dần.

5. Sắp xếp dãy trên theo quy luật sau:

- Các số dương (nếu có) ở đầu mảng và có thứ tự giảm dần,

- Các số âm (nếu có) ở cuối mảng và có thứ tự tăng dần.

6. Sắp xếp dãy trên theo quy luật:

- Các số chẵn (nếu có) ở đầu mảng và có thứ tự tăng dần,
- Các số lẻ (nếu có) ở cuối mảng và có thứ tự giảm dần.

BÀI TẬP LÀM THÊM:*(sinh viên có thể nộp bài vào đầu buổi thực hành sau để lấy điểm cộng)*

Cho mảng một chiều quản lý thông tin của các sinh viên trongmột lớp học (tối đa 50 sinh viên). Mỗi sinh viên gồm các thông tin: MSSV, họ và tên, giới tính, địa chỉ và điểm trung bình. Viết chương trình thực hiện các yêu cầu sau:

1. Nhập các sinh viên vào danh sách.
2. In danh sách sinh viên.
3. Xuất thông tin của sinh viên có mã số x.
4. Sắp xếp danh sách sinh viên theo thứ tự tăng dần của điểm trung bình.
5. Sắp xếp danh sách sinh viên theo thứ tự tăng dần của họ và tên.
6. Ghi danh sách sinh viên vào FILE
7. Đọc danh sách sinh viên từ FILE
8. Sửa thông tin của một sinh viên có mã X
9. Xóa sinh viên có mã X
10. Xuất ra các sinh viên có điểm trung bình cao nhất

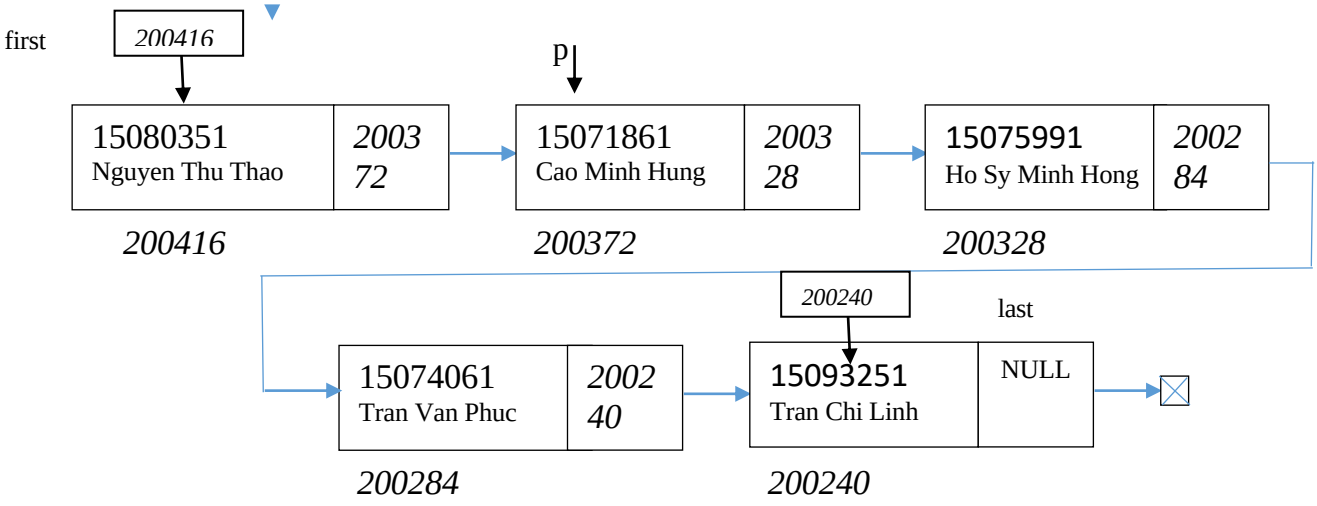
BÀI THỰC HÀNH SỐ 3:
DANH SÁCH LIÊN KẾT
(Số tiết: 3)

Mục đích :

1. Hiểu được các thành phần của danh sách liên kết.
2. Thực hiện được một số thao tác cơ bản trên danh sách liên kết đơn: Tạo danh sách, thêm một phần tử vào đầu/ cuối danh sách, duyệt, tìm kiếm trong danh sách.

Vấn đề 1: Hiểu được các thành phần của danh sách liên kết.

Danh sách liên kết dưới đây lưu thông tin của các sinh viên. Mỗi Node có hai trường dữ liệu (`data`) và trường liên kết (`link`). Trường `data` lưu thông tin của một sinh viên gồm có mã sinh viên và họ tên. Trường `link` lưu địa chỉ của của Node kế tiếp. Trong hình, dòng chữ in nghiêng phía dưới mỗi Node là địa chỉ của Node đó được lưu trong bộ nhớ.



1. Hãy cho biết nội dung của trường **link** của từng **Node**
2. Người ta dùng con trỏ **first** để quản lý đầu danh sách. Hãy cho biết giá trị của con trỏ **first** khi ta thực hiện câu lệnh
   `printf("%u", first);//kq':` *200416*
3. Tương tự, người ta cũng dùng con trỏ **last** để quản lý cuối danh sách. Hãy cho biết giá trị của con trỏ **last** khi ta thực hiện câu lệnh
   `printf("%u", last);//kq':200240`

4. Con trỏ p đang trỏ đến Node tương ứng như trong hình. Hãy cho biết kết quả khi thực hiện các câu lệnh sau:

```
printf("%u", p);// kq': 200372
printf("%d", p->data.MaSV);//kq':
printf("%u", p->link);
printf("%d", p->link->data);
printf("%u", p->link->link);
printf("%d", p->link->link->data);
```

Vấn đề 2: Thực hiện một số thao tác cơ bản trên danh sách liên kết đơn.

Danh sách liên kết đơn có mỗi phần tử chứa dữ liệu là một số nguyên. Hãy thực hiện các thao tác sau đây:

1. Định nghĩa cấu trúc**Node**và cấu trúc **List** (List gồm hai con trỏ trỏ đến đầu và cuối danh sách)

```
structNode
{
      int     data;
      Node *link;
};
```

```
struct                List
{
      Node *first, *last;
};
```

2. Xây dựng hàm tạo danh sách rỗng **void Init (List &L)**

```
voidInit(List&l)
{
   l.first=l.last=NULL;
}
```

3. Xây dựng hàm ***Node* GetNode (int x)***để tạo một **Node** mà trường **data** nhận giá trị x và trường **link** nhận giá trị NULL. Hàm trả về con trỏ TRỎ vào nút vừa tạo hoặc trả về NULL trong trường hợp không thành công.

```
Node *GetNode(intx)
{
   Node *p;
   P=newNode;
   if(p==NULL)
      returnNULL;
   p->data=x;
   p->link=NULL;
   returnp;
}
```

4. Xây dựng hàm **void AddFirst (List &L, Node* p)** để thêm một Node mới vào đầu danh sách được quản lý bởi L.

```
voidAddFirst(List&L, Node* new_ele)
{
   if(L.first == NULL)  //Danh sách rỗng
   {
      L.first = new_ele;
      L.last  = L.first;
   }
   else {
      new_ele->link = L.first;
      L.first = new_ele;
   }
}
```

5. Xây dựng hàm **void InsertFirst (List &L, int x)** để thêm một Node mới chứa dữ liệu x vào đầu danh sách được quản lý bởi L.

```
voidInsertFirst(List&L, int x)
{
   Node* new_ele = GetNode(x);
   if (new_ele == NULL)
      return;
   AddFirst(L, new_ele);
}
```

6. Xây dựng hàm **void CreateListFirst(List &L)** để tạo danh sách bằng cách thêm vào đầu danh sách. Việc nhập sẽ dừng khi người dùng nhập -1.

```
Void CreateListFirst (List &L)
     do
     {
          printf("\nBat dau nhap danh sach cac so
          nguyen, nhap -1 de ket thuc: \n");
          scanf("%d", &x);
          if (x == -1)
               break;
          InsertFirst(L, x);
     } while (x != -1);
     printf("\nDa nhap du lieu xong: \n");
}
```

7. Xuất danh sách liên kết đơn L.

```
voidPrinttList(List L)
{
   Node  *p;
   p = L.first;
   while (p!= NULL)
     {
      printf("%10d", p->data);
      p = p->link;
     }
}
```

8.Chương trình chính

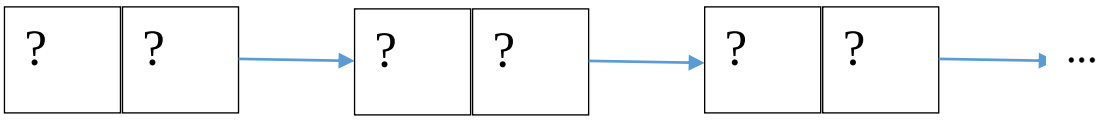
```
Void main()
{
   List  L;
   Init(L)
   CreateListFirst(L);
   PrintFirst(L);
}
```

9. Chạy chương trình, cho biết kết quả in ra màn hình khi người dùng nhập vào các dữ liệu sau:
   -1
   6 7 3 5 -1
   5 -4 7 -8 -1
   1 2 3 4 5 -1

   Hãy nhận xét về mối liên hệ giữa thứ tự nhập dữ liệu vào và thứ tự in dữ liệu ra màn hình.

10. Để biết được trong quá trình chạy chương trình (runtime) các Node được lưu tại địa chỉ nào trong bộ nhớ, hãy sửa lại hàm PrintList để xuất địa chỉ của từng Node.

    Nhập danh sách với các phần tử được nhập lần lượt là 10 9 50 55 90 -1. Hãy vẽ lại danh sách trên với đầy đủ các thông tin trường data và trường link.



11. Viết hàm trả về tổng của các phần tử có giá trị chẵn trong danh sách.
    long SumEvenNumber (List L);

12. Viết hàm tìm xem có phần tử có giá trị x trong danh sách hay không. Nếu có trả về con trỏ TRỎ đến Node tương ứng, không có thì trả về NULL.

    Node*Search(List L, int x);

BÀI TẬP VỀ NHÀ:*(bắt buộc– sinh viên nộp vào đầu buổi thực hành sau)*

Hãy viết tiếp chương trình để tạo danh sách bằng cách thêm vào cuối danh sách. Các nguyên mẫu hàm như sau:

void AddLast (List &L, Node* p);

void InsertLast (List &L, Node* p);

CreateListLast(List &L, int n);

BÀI TẬP LÀM THÊM:*(sinh viên có thể nộp bài vào đầu buổi thực hành sau để lấy điểm cộng)*

1. Sửa lại hàm **InsertFirst** để thêm một phần tử vào đầu danh sách sao cho danh sách không có hai số nguyên nào trùng nhau.
2. Viết hàm tách danh sách L thành hai danh sách khác nhau, trong đó danh sách L1 chứa các số nguyên lớn hơn x, danh sách L chứa các số còn lại.

void Separating_List(List &L, List &L1, int x);

3. Cho sẵn một danh sách liên kết đơn gồm các phần tử mang giá trị nguyên và một giá trị nguyên x. Hãy tách danh sách liên kết đã cho thành 2 danh sách liên kết: một danh sách gồm các phần tử có giá trị nhỏ hơn giá trị x và một danh sách gồm các phần tử có giá trị lớn hơn giá trị x. Giải quyết trong 2 trường hợp:

a. Danh sách liên kết ban đầu không cần tồn tại.

b. Danh sách liên kết ban đầu bắt buộc phải tồn tại.

BÀI THỰC HÀNH SỐ 4:
DANH SÁCH LIÊN KẾT (tt)

(Số tiết: 3)

Mục đích :

1. Áp dụng cấu trúc dữ liệu danh sách liên kết vào việc giải quyết một số bài toán đơn giản.
2. Sắp xếp các phần tử trong danh sách.

Vấn đề 1: Áp dụng cấu trúc dữ liệu danh sách liên kết vào việc giải quyết một số bài toán đơn giản. *[Bài toán cộng trừ hai đa thức]*.

Xét đa thức tổng quát: $P(x) = a_n x^n + a_{n-1}.x^{n-1} + a_1.x^2 + \ldots + a_0$

Cách đơn iản nhất để biểu diễn đa thức là dùng mảng lưu các hệ số của đa thức. Các phép toán như: cộng, trừ, nhân 2 đa thức,.. sẽ được thực hiện một cách dễ dàng nếu biểu diễn đa thức bằng mảng.

Giả sử ta có hai đa thức:

$P1(x) = 3x^7 + 5x^6 + x^5 + 2x^3 - 7x + 9$

$P2(x) = 2x^7 + 3x^5 - 5x^4 + 2x^3 + x - 8$

Cộng hai đa thức P1 và P2 ta được đa thức tổng:

$T(x) = 5x^7 + 5x^6 + 4x^5 - 5x^4 + 4x^3 - 6x + 1$

Có thể dùng hai mảng A, B và C để lưu hai đa thức P1, P2 và T như sau:

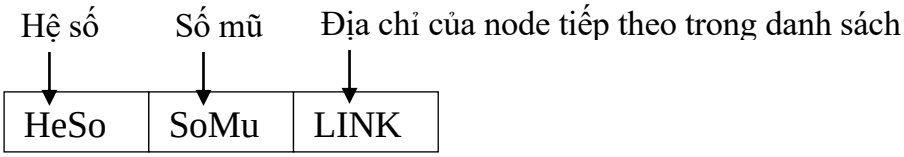
A:	3	5	1	0	2	0	-7	9
	0	1	2	3	4	5	6	7

B:	2	0	3	-5	2	0	1	-8
	0	1	2	3	4	5	6	7

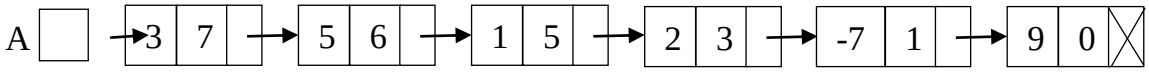
C:	5	5	4	5	4	0	-6	1
	0	1	2	3	4	5	6	7

Tuy nhiên, biểu diễn đa thức bằng mảng có một hạn chế lớn là: khi đa thức có nhiều hệ số bằng 0 thì việc biểu diễn bằng mảng gây lãng phí bộ nhớ. Ví dụ: đa thức $x^{2017} + 1$ đòi hỏi 1 mảng 2018 phần tử.

Để khắc phục các nhược điểm ở trên, ta có thể dùng danh sách liên kết đơn để biểu diễn đa thức. Mỗi node của DSLK sẽ chứa các thông tin sau:



Như vậy, với đa thức P1(x) ở trên thì danh sách biểu diễn nó có dạng:



Ở đây A là con trỏ lưu địa chỉ nút đầu tiên của danh sách.

1. Định nghĩa cấu trúc **Node**với mỗi phần tử của Node gồm có 3 thành phần: hệ số, số mũ và địa chỉ của Node tiếp theo.

```
Struct Node{
   float heSo;
   int soMu;
   Node *link;
};
Struct List
{
   Node *first, *last;
};
```

2. Hàm thêm một phần tử vào danh sách

```
//Khởi tạo danh sách rỗng
Void init(List &L)
{
   L.first = L.last = NULL;
}
//Tạo một node mới
Node *GetNode(float heSo, int soMu)
{
   Node *p;
   p = new Node;
   if(p == NULL)
      returnNULL;
   p->heSo = heSo;
   p->soMu = soMu;
   p->link = NULL;
   return p;
}
// Gắn node new_ele vào danh sách
void AddLast (List&L, Node *new_ele)
{
   if(L.first == NULL) //danh sách rỗng
   {
      L.first = L.last = new_ele;
   }
   else
   {
      L.last->link = new_ele;
      L.last = new_ele;
   }
}
//Thêm một node với dữ liệu là heSo và soMu vào danh sách
Void InsertLast (List&L, float heSo, int soMu)
{
   Node *new_ele=GetNode(heSo, soMu);
   if(new_ele==NULL)
      return;
   AddLast(L,new_ele);
}
```

3. Hàm nhập đa thức (hệ số và số mũ) bằng cách thêm vào cuối danh sách.

```
// Hàm nhập đa thức
void NhapDaThuc(List&L)
{
   float heSo;
   int soMu;
   printf("\nBat dau nhap da thuc (nhap he so 0 de
ket thuc): \n");
   do
   {
      printf("\nNhap he so: ");
      scanf("%f", &heSo);
      if (heSo == 0)
         break;
      printf("\nNhap so mu:");
      scanf("%d", &soMu);
      InsertLast(L, heSo, soMu);
   } while (heSo != 0);
   printf("\nDa nhap da thuc xong: \n");
}
```

4. Xuất danh sách biểu diễn đa thức

```
void XuatDanhSach(List L)
{
   Node  *p;
   p = L.first;
   printf("\n");
   while (p!= NULL)
   {
   printf("%.0f, %d   ", p->heSo, p->soMu);
   p = p->link;
   }
}
```

5. Cộng hai đa thức

```
//Cộng đa thức: d3 = d2 + d1
voidCongDaThuc(List d1, List d2, List&d3)
{
   init(d3);
   Node *p = d1.first, *q = d2.first;
   float tongHeSo;
   while(p&&q)
   {
      if(p->soMu == q->soMu)//Hai số mũ bằng nhau
      {
         tongHeSo = p->heSo + q->heSo;
         if(tongHeSo != 0)
            InsertLast(d3, tongHeSo, p->soMu);
         p = p->link;
         q = q->link;
      }
      else
      {
         if(p->soMu > q->soMu)
         {
            InsertLast(d3, p->heSo, p->soMu);
            p = p->link;
         }
         else
         {
            if(p->soMu < q->soMu)
            {
               InsertLast(d3, q->heSo, q->soMu);
               q = q->link;
            }
         }
      }
   }
   while(q) //bieu thuc d1 ket thuc truoc
   {
      InsertLast(d3, q->heSo, q->soMu);
      q = q->link;
   }
   while(p) //bieu thuc d2 ket thuc truoc
   {
      InsertLast(d3, p->heSo, p->soMu);
      p = p->link;
   }
}
```

6. Chương trình chính

```
int main()
{
   List d1, d2, d3;
   init(d1);
   init(d2);
   init(d3);
   NhapDaThuc(d1);
   printf("\nDanh sach bieu dien da thuc d1: ");
   XuatDanhSach(d1);
   NhapDaThuc(d2);
   printf("\nDanh sach bieu dien da thuc d2: ");
   XuatDanhSach(d2);
   CongDaThuc(d1, d2, d3);
   printf("\nDanh sach bieu dien đa thuc tong: ");
   XuatDanhSach(d3);
}
```

7. Chạy chương trình, nhập vào hai đa thức P1 và P2.

   $P1(x) = 3x^7 + 5x^6 + x^5 + 2x^3 - 7x + 9$

   $P2(x) = 2x^7 + 3x^5 - 5x^4 + 2x^3 + x - 8$

   Ghi kết quả xuất ra màn hình ở đây:

   .....

   .....

   .....

   .....

Tương tự, hãy ghi kết quả xuất ra màn hình khi nhập hai biểu thức P3 và P4.

$P3(x) = 5x^{15} - 3x^{12} - 2x^{10} + 5x^2$

$P4(x) = 2x^{30} + 6x^{21} - 4x^{10} + 5x - 2$

.....

.....

.....

.....

Vấn đề 2:Sắp xếp các phần tử trong danh sách.

Với chương trình ở trên, nếu ta nhập vào hai đa thức P5 và P6 (theo thứ tự các hệ số và số mũ như bên dưới) thì kết quả tính tổng hai đa thức đúng hay sai?

$P5(x) = 3x^7 - x^{10} + 2x^3 + 9x^5$

$P6(x) = 7x^5 + 6x^7 - 4x^{10} + x + 6$

Câu trả lời là SAI. Vì với phương pháp cộng hai đa thức như trên thì DSLK biểu diễn đa thức phải được sắp xếp giảm dần theo số mũ. Nếu người dùng nhập không vào không đúng thứ tự thì ta cần tiến hành sắp xếp DSLK giảm dần theo số mũ trước khi thực hiện cộng hai đa thức.

Sắp xếp danh sách liên kết đơn có 2 phương pháp tiếp cận: Hoán vị dữ liệu của các phần tử trong danh sách, hoặc thay đổi các mối liên kiết. Trong trường hợp này, mỗi node của danh sách chỉ có hai thành phần là hệ số và số mũ, kích thước của dữ liệu nhỏ, vì thế ta có thể chọn cách sắp xếp DSLK bằng cách hoán vị dữ liệu của các phần tử trong danh sách. Hàm bên dưới thực hiện sắp xếp bằng phương pháp đổi chỗ trực tiếp.

Hàm sắp xếp danh sách bằng phương pháp đổi chỗ trực tiếp

```
void InterchangeSort ( List&L)
{
for ( Node* p=L.first ; p!=L.last ; p=p->link )
   for ( Node* q=p->link ; q!=NULL ; q=q->link )
   if ( p->soMu> q->soMu )
      {
          Swap1(p->heSo, q->heSo);//Hoán vị 2 số thực
        Swap2(p->soMu , q->soMu );//HV2 số nguyên
       }
}
```

BÀI TẬP VỀ NHÀ: *(bắt buộc – sinh viên nộp vào đầu buổi thực hành sau)*

Hãy xiết hàm xuất đa thức dưới dạng như sau: Giả sử có đa thức là $6x^7 - 3x^5 + 9x^2 - 1$ thì sẽ xuất dưới dạng 6*x^7 – 3*x^5 + 9*x^2 -1.

BÀI TẬP LÀM THÊM: *(sinh viên có thể nộp bài vào đầu buổi thực hành sau để lấy điểm cộng)*

Viết tiếp chương trình để thực hiện việc :

1. Trừ hai đa thức.
2. Nhân hai đa thức
3. Chia hai đa thức

BÀI THỰC HÀNH SỐ 5:
DANH SÁCH LIÊN KẾT (tt)

(Số tiết: 3)

Mục đích :

1. Áp dụng cấu trúc dữ liệu DSLK vào việc giải quyết một số bài toán quản lý đơn giản.
2. Thêm vào danh sách không có khóa trùng.
3. Xóa phần tử
4. Sắp xếp danh sách.
5. Thêm một phần tử vào danh sách có thứ tự.

Vấn đề 1: Áp dụng cấu trúc dữ liệu danh sách liên kết vào việc giải quyết bài toán quản lý đơn giản.

[Chương trình quản lý sinh viên].

Viết chương trình quản lý sinh viên của một trường đại học dùng danh sách liên kết đơn. Mỗi sinh viên có nhiều thông tin cần quản lý, tuy nhiên, trong bài tập này, để cho đơn giản ta chỉ quản lý các thông tin sau: Mã sinh viên, họ tên,giới tính, ngày tháng năm sinh, địa chỉ, lớp và khoa.

1. Khai báo cấu trúc dữ liệu cho bài toán.

```
struct SinhVien
{
   char maSV[8];
   char hoTen[50];
   int gioiTinh;
   Ngay ngaySinh;
   char diaChi[100];
   char lop[12];
   char khoa[7];
};
```

```
struct Ngay
{
   int ngay, thang, nam;
};
struct Node
{
   SinhVien data;
   Node *link;
};
struct List
{
   Node *first;
   Node *last;
};
```

2. Hàm **Node *GetNode(SinhVien x)** tạo một Node chứa dữ liệu sinh viên.

```
Node *GetNode(SinhVienx)
{
   Node *p;
   P=newNode;
   if(p==NULL)
      returnNULL;
   p->data=x;
   p->link=NULL;
   returnp;
}
```

3. Các hàm khởi tạo danh sách rỗng, thêm một node vào danh sách và thêm một node chứa thành phần dữ liệu x. (tương tự như Lab 4)

```
//Khởi tạo danh sách rỗng
void init(List &L);
// Gắn node new_ele vào danh sách
void AddLast (List &L, Node *new_ele);
//Thêm một node với dữ liệu là sinh viên x
void InsertLast (List &L, SinhVien x);
```

4. Việc nhập thông tin của một sinh viên có rất nhiều thao tác, vì thế, để tiện sử dụng ta viết hàm để nhập một sinh viên với nguyên mẫu hàm **int NhapSinhVien(SinhVien &x)**.

```
// Hàm nhập một sinh viên. Nhập thành công trả về 1,
nhập không thành công (MASV = 0) thì trả về 0
int NhapSinhVien(SinhVien &x)
{
   printf("Ma so sinh vien: ");
   fflush(stdin);
   gets(x.maSV);
   if(strcmp(x.maSV, "0") == 0)
      return 0;//Nhap MASV = 0 de dung
   printf("Ho va ten: ");
   fflush(stdin);
   gets(x.hoTen);
   printf("Gioi tinh: ");
   fflush(stdin);
   scanf("%d", &x.gioiTinh);
   printf("Ngay thang nam sinh: ");
   fflush(stdin);
   scanf("%d/%d/%d",&x.ngaySinh.ngay,
&x.ngaySinh.thang, &x.ngaySinh.nam);
   printf("Dia chi: ");
   fflush(stdin);
   gets(x.diaChi);
   printf("Lop: ");
   fflush(stdin);
   gets(x.lop);
   printf("Khoa: ");
   fflush(stdin);
   gets(x.khoa);
   return 1;
}
```

Hàm **void NhapDSSV(List &L)** thực hiện nhập danh sách sinh viên, nhập 0 để dừng.

```
void NhapDSSV(List&L)
{
   cout<<"\nBat dau nhap DSSV. Nhap MASV = 0 de dung\n";
   SinhVien x;
   int flag = NhapSinhVien(x)  ;
   while(flag)
   {
      InsertLast(L, x) ;
      flag = NhapSinhVien(x);
   }
   cout<<"\n Ket thuc nhap DSSV.";
}
```

5. Hàm**void XuatSinhVien(SinhVien s)** thực hiệnxuất một sinh viên.

```
void XuatSinhVien(SinhVien s)
{
   char gt[4];
   if(s.gioiTinh==0)
      strcpy(gt, "Nam");
   else
      strcpy(gt, "Nu");
   printf("\n%10s %20s %5d/%d/%d  %5s %40s %8s
%8s", s.maSV, s.hoTen, s.ngaySinh.ngay,
s.ngaySinh.thang, s.ngaySinh.nam, gt,
s.diaChi, s.khoa, s.lop);
}
```

6. Hàm**void XuatDSSV(List L)** thực hiệnxuất danh sách sinh viên.

```
voidXuatDSSV(ListL)
{
   Node *p = L.first;
   while(p)
   {
      XuatSinhVien(p->data);
      p = p->link;
   }
}
```

7. Chương trình chính

```
intmain()
{
   List L;
   init(L);
   NhapDSSV(L);
   XuatDSSV(L);
}
```

Vấn đề 2: Thêm vào danh sách không có khóa trùng

Mã số sinh viên là không được trùng nhau. Vì thế, khi nhập danh sách sinh viên ta cần kiểm tra mã trùng. Nếu mã sinh viên thêm vàođã có trong danh sách thì xuất thông báo và không thêm vào danh sách.

Hàm **int InsertFirst_KhongTrung(List &L, SinhVien x)**bên dưới thực hiện thêm phần tử x vào danh sách L. Nếu thêm thành công thì trả về 1, không thành công (đã tồn tại sinh viên có mã này) thì trả về 0.

Khi đó, trong hàm **void NhapDSSV(List &L)** thay vì gọi hàm **InsertFirst**thì ta sẽ gọi hàm **InsertFirst_KhongTrung**.

```
intInsertFirst_KhongTrung(List&L, SinhVien x)
{
   if(Search(L, x.maSV))
   {
      cout<<"Ma sinh vien trung";
      return 0;
   }
   else
   {
      InsertLast(L, x);
      return 1;
   }
}
```

Vấn đề 3:Xóa một phần tử trong danh sách.*[Xóa sinh viên có mã X trong danh sách]*

8. Thuật toán xóa phần tử có giá trị x trong danh sách

```
Input: danh sách, phần tử x cần xóa
Nếu phần tử cần xóa là phần tử đầu
   Gọi hàm xóa đầu danh sách (RemoveFirst)
Ngược lại
   Tìm q đứng trước phần tử cần xóa (TimXoa)
   Gọi hàm xóa phần tử sau q (RemoveAfter)
```

9. Thuật toán trên được viết chi tiết hơn bên dưới. Sinh viên tự code.

```
Input: danh sách, phần tử x cần xóa
int XoaX (LIST &L, char x[])
Nếu x=l.first->data
   RemoveFirst(l); re = 1
Ngược lại
   q = TimXoa(l, x)
   Nếu q==NULL
      Không tìm thấy phần tử cần xóa; re = 0
   Ngược lại
      RemoveAfter(l, q); re = 1
return re
```

```
//Xóa node đầu danh sách
int RemoveFirst(List&L)
{
   if(L.first == NULL) return 0;
   Node* p= L.first;
   L.first = p->pNext;
   if(L.first== NULL)  L.last = NULL;
   //Nếu danh sách sau khi xóa là rỗng
   Delete p;
   return 1;
}
// Xóa node sau node trỏ bởi q
int RemoveAfter(List&L, Node *q )
{
   if (q !=NULL&& q->link !=NULL)
   {
      Node*   p = q->link;
      q->link = p->link;
      if  (p == L.first)   L.last= q;
      delete p;
      return 1;
   }
   Else return 0;
}
```

10.Hàm **Node* TimXoa(LIST L, char x[ ])**trả về con trỏ lưu địa chỉ của node đứng trướcphần tử cần xóa.

```
Node* TimXoa(List L , char  x[])
{
   Node* p= L.first;
   while(p!= L.last &&strcmp(p->link->data.maSV,x)!=0)
      p=p->link;
   if(p!= L.last)
      return p;
   else
      returnNULL;
}
```

```
// Xóa node có mã sinh viên là x
Int RemoveX(List&L, char x[])
{
   int re;
```

```
    if(strcmp(L.first->data.maSV,x) ==0)
    {
       RemoveFirst(L);
       re = 1;
    }
    else
    {
       q = TimXoa(L, x)
       if(q==NULL)
       {
          cout<<"Không tìm thấy phần tử cần xóa";
          re = 0;
       }
       else
       {
          RemoveAfter(L, q) ;
          re = 1;
       }
    }
    return re;
}
```

11. Tạo danh sách có các phần tử được thêm vào cuối lần lượt theo thứ tự:

    6 3 5 10 25 7 1 2 9 15.

12. Xuất danh sách vừa tạo.
13. Tiếp tục thực hiện các thao tác dướiđây. Xuất lại danh sách sau mỗi lần thao tác.

- Xóa phần tử 10
- Xóa phần tử 6
- Xóa phần tử 15
- Xóa phần tử 10

Vấn đề 5: Sắp xếp danh sách *[Sắp xếp danh sách sinh viên theo tên sinh viên]*

Có hai hướng tiếp cận để sắp xếp danh sách liên kết. Phương án 1: Hoán vị nội dung các phần tử trong danh sách (thao tác trên vùng data). Phương án 2: Thay đổi các mối liên kết (thao tác trên vùng link).

Do việc thực hiện hoán vị nội dung của các phần tử đòi hỏi sử dụng thêm vùng nhớ trung gian, nên nó chỉ thích hợp với các danh sách có thành phần dữ liệu kích thước nhỏ. Khi kích thước của trường dữ liệu lớn, việc hoán vị giá trị của hai phân tử sẽ chiếm chi phí đáng kể.

Thay vì hoán đổi giá trị, ta có thể tìm cách thay đổi trình tự mối liên kết của các phần tử sao cho tạo lập nên được thứ tự mong muốn. Phương pháp này chỉ thao tác trên các mối liên kết. Kích thước của trường liên kết không phụ thuộc vào dữ liệu lưu trong danh sách, nó bằng kích thước của một con trỏ.

Trong bài toán này, trường dữ liệu lưu thông tin của một sinh viên, kích thước của một sinh viên là 193 bytes. Vì kích thước của trường dữ liệu không lớn nên có thể sắp xếp bằng cách hoán vị nội dung các phần tử.Ta có thể chọn một trong các thuật toán sắp xếp trên mảng đã học như interchange sort, selection sort, insertion sort hoặc bubble sort.

```
voidInterchangeSortList(List&L)
{
   for(Node *i=L.first; i!=L.last; i=i->link)
   for(Node *j=i->link; j!=NULL; j=j->link)
      if(strcmp(i->data.hoten, j->data.hoten)>0)
         swap(i->data, j->data);//Hoán vị hai
sinh viên
}
```

Đây chỉ là ứng dụng demo nên chỉ lưu trữ một vài thông tin của sinh viên. Trong thực tế thì mỗi sinh viên cần được lưu trữ rất nhiều thông tin, vì thế kích thước của trường data là rất lớn. Khi đó, ta nên sắp xếp bằng cách thay đổi các mối liên kết. Dưới đây là thuật toán Quick Sort sắp xếp danh sách liên kết bằng cách thay đổi các mối liên kết.

Thuật toán Quick Sort:

Input: Danh sáchL

Output: Danh sách L đã được sắp tăng dần

Bước 1: Nếu danh sách có ít hơn 2 phần tử
Dừng; *// danh sách đã có thứ tự*
Bước 2: Chọn X là phần tử đầu danh sách L làm ngưỡng. Trích X ra khỏi L.
Bước 3: Tách danh sách L ra làm 2 danh sách L_1 (gồm các phần tử nhỏ hơn hay bằng X) và L_2 (gồm các phần tử lớn hơn X).
Bước 4: Sắp xếp Quick Sort (L_1).
Bước 5: Sắp xếp Quick Sort (L_2).
Bước 6: Nối L_1, X, và L_2 lại theo trình tự ta có danh sách L đã được sắp xếp.

14. Hàm **voidSListAppend(LIST &L, LIST &L2)**thực hiện nối danh sách L2 vào sau danh sách L, kết quả là danh sách L đã thay đổi.

```
void  SListAppend(LIST&L, LIST&L2)
{
   if (L2.first == NULL) return;
   if (L.first == NULL)
      L = L2;
   else {
      L.last->link = L2.first;
      L.last = L2.last;
   }
   Init(L2);
}
```

Cài đặt thuật toán Quick Sort:

```
void  ListQSort(List&L) {
   Node *X, *p;
   List L1, L2;
   if (L.first == L.last) return;
   Init(L1);  Init(L2);
   X = L.first; L.first=x->link;
   while (L.first != NULL) {
      p = L.first;
      if (p->data <= X->data) AddFirst(L1, p);
      else     AddFirst(L2, p);
   }
   ListQSort(L1);ListQSort(L2);
   ListAppend(L, L1);
   AddLast(L, X);
   ListAppend(L, L2);
}
```

```
void QuickSort(list &l)
{
    node *p, *x;
    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;
        l.pHead = p->pNext;
        p->pNext = NULL;
        if (p->data <= x->data)
            addHead(l1, p);
        else
            addHead(l2, p);
    }
    QuickSort(l1);
```

```
    QuickSort(l2);
if (l1.pHead != NULL)
{
     l.pHead = l1.pHead;
     l1.pTail->pNext = x;
   }
 else
       l.pHead = x;
  x->pNext = l2.pHead;
  if (l2.pHead != NULL)
       l.pTail = l2.pTail;
  else
        l.pTail = x;
}
```

BÀI TẬP VỀ NHÀ: *(bắt buộc – sinh viên nộp vào đầu buổi thực hành sau)*

Vấn đề 5: Thêm vào danh sách có thứ tự.

Danh sách sinh viên L đã được sắp xếp tăng dần theo mã sinh viên. Hãy viết hàm thêm một sinh viên mới vào danh sách sau cho sau khi thêm danh sách L vẫn còn là một danh sách có thứ tự.

Hướng dẫn:

Để thêm vào danh sách có thứ tự ta phải tìm vị trí thích hợp để thêm. Nếu phần tử cần thêm `x`bé hơn phần tử đầu danh sách thì thực hiện thêm `x`vào đầu danh sách. Ngược lại, sẽ tìm vị trí p (p là con trỏ) thỏa điều kiện, dữ liệu tại p bé hơn `x`, và dữ liệu tại node sau p lớn hơn `x`. Khi đó ta sẽ tiến hành thêm `x` sau p.

15. Hàm **Node* TimThem(List L , int x)** thực hiện tìm vị trí để thêm.

```
Node* TimThem(List  L , int  x)
{
      Node* p=L.first;
      while( p!=L.last && p->link->data<x)
         p=p->link;
      return p;
}
```

16. Hàm **ThemCoThuTu(LIST &L, int x)**

```
voidThemCoThuTu(LIST&L, int x)
{
   if(x<L.first->data) //Phần tử cần thêm bé hơn PT đầu
         InsertFirst(L, x);
     else
     {
      NODE* p= TimThem(L, x);//Tìm vị trí thêm
      InsertAfter(L, p, x); //Thêm x vào sau node có
                                    địa chỉ p
     }
}
```

17. **Yêu cầu bổ sung:** Vì mã sinh viên là không được trùng nhau. Vì thế khi thêm vào danh sách cần phải kiểm tra, nếu trùng mã thì không thêm. Hàm **void ThemCoThuTu(LIST &L, int x)** được sửa lại thành hàm **void ThemCoThuTu_KhongTrungMa(LIST &L, int x)** như dưới đây. Nếu thêm thành công thì trả về 1, không thành công thì trả về 0.

```
intThemCoThuTu_KhongTrungMa(LIST&L, int x) {
   if(x == L.first->data) return 0;
   if(x<L.first->data) {
      InsertFirst(L, x)
      return 1;
   }
   NODE* p= TimThem(L, x);
   if(p->link->data!=x)//khong co trung
   {
      InsertAfter(L, p, x)
      return 1;
   }
   return 0;
}
```

BÀI TẬP LÀM THÊM:*(sinh viên có thể nộp bài vào đầu buổi thực hành sau để lấy điểm cộng)*

Thông tin của một quyển sách trong thư việc gồm các thông tin:

- Tên sách (chuỗi)
- Tác giả (chuỗi, tối đa 5 tác giả)
- Nhà xuất bản (chuỗi)
- Năm xuất bản (số nguyên)

a. Hãy tạo danh sách liên kết (đơn hoặc kép) chứa thông tin các quyển sách có trong thư viện (được nhập từ bàn phím).

b. Cho biết số lượng các quyển sách của một tác giả bất kỳ (nhập từ bàn phím).

c. Trong năm YYYY (nhập từ bàn phím), nhà xuất bản ABC (nhập từ bàn phím) đã phát hành những quyển sách nào.

d. Xóa tất cả các quyển sách của tác giả X (X: tham số của hàm)

BÀI THỰC HÀNH SỐ 6:
NGĂN XẾP (STACK)
(Số tiết: 3)

Mục đích :

1. Cài đặt các thao tác cơ bản trên stack dùng danh sách liên kết.
2. Ứng dụng stacktrong những bài toán đơn giản.

Vấn đề 1: Cài đặt các thao tác cơ bản trên stack dùng danh sách liên kết.

Hai thao tác chính trên stack gồm có: thêm và lấy phần tử ra khỏi stack. Đối với stack dùng danh sách liên kết thì thêm phần tử vào stack chính là thao tác thêm phần tửvào đầu danh sách liên kết. Lấy phần tử ra khỏi stack chính là thao tác lấy phần tử ở đầu danh sách ra khỏi danh sách liên kết. Ta cũng cần một thao tác hổ trợ là kiểm tra danh sách rỗng.

(Lưu ý: ta cũng có thể thêm phần tử vào cuối danh sách liên kết, khi đó,để lấy phần tử ra khỏi danh sách, ta thực hiện lấy phần tử ở cuối danh sách liên kết.)

```
struct Node
{
     intdata;  //Giả sử stack chứa các số nguyên
     NODE *link;
};
struct stack  {
   Node *top;
};

//Khởi tạo stack
voidInit(stack&s)
{
   s.top=NULL;
}

// Kiểm tra stack rỗng
intEmpty(stack s)
{
      return s.top == NULL ? 1 : 0; // stack rỗng
}
```

```
//Thêm một phần tử  x vào stack S
void  Push(stack&s, float x)
{
   node *p;
   p=new node;
   if(p!=NULL)
   {
         p->data=x;
         p->link=s.top;
         s.top=p;
   }
}

//Trích thông tin và huỷ phần tử ở đỉnh stack S
float Pop(stack&s)
{  float x;
   if(!Empty(s))
   {
      node*p= s.top;
      s.top=p->link;
      x = p->data;
      delete(p);
      returnx;
   }
}
```

Vấn đề 2:Ứng dụng stack trong những bài toán đơn giản.
Bài toán: Tính giá trị của biểu thức dạng hậu tố.

Xét một biểu thức số học với các phép toán cộng, trừ, nhân, chia, lũy thừa, ...

Ví dụ:biểu thức a + (b - c) * d+ e. Biểu thức như trênđược viết theo ký pháp trung tố, nghĩa là toán tử (dấu phép toán) đặt giữa hai toán hạng.Với ký pháp trung tố, để phân biệt toán hạng ứng với toán tử nào ta phải dùng các cặp dấu ngoặc đơn, và phải chấp nhận một thứ tự ưu tiên giữa các phép toán. Các phép toán cùng thứ tự ưu tiên thì sẽ thực hiện theo trình tự từ trái sang phải. Thứ tự ưu tiên như sau:

1. Phép lũy thừa
2. Phép nhân, chia
3. Phép cộng, trừ

Cách trình bày biểu thức theo ký pháp trung tố là tự nhiên với con người nhưng lại "khó chịu" đối với máy tính, vì nó không thể hiện một cách tường minh quá trình tính toán để đưa ra giá trị của biểu thức.Để đơn giản hóa quá trình tính toán này, ta phải biến đổi lại biểu thức thông thường về dạng ký pháp Ba Lan, gồm có có hai dạng là tiền tố (prefix) và hậu tố (postfix). Đó là một cách viết biểu thức đại số rất thuận lợi cho việc thực hiện các phép toán. Đặc điểm cơ bản của cách viết này là không cần dùng đến các dấu ngoặc và luôn thực hiện từ trái sang phải.

Ta có thể biến đổi biểu thức dạng trung tố sang tiền tố hoặc hậu tố. Ví dụ:

Dạng trung tố	**Dạng tiền tố**	**Dạng hậu tố**
A + B	+ A B	A B +
A / B	/ A B	A B /
(A + B) * C	* + A B C	A B + C*
(A + B) / (C – D)	/ + A B – C D	A B + C D - /
A + B / C – D	- + A / B C D	A B C / + D -

Để hiểu rõ cách chuyển biểu thức sang các dạn khác nhau, sinh viên hãy thực hiện chuyển các biểu thức trung tố dưới đây sang dạng tiền tố và hậu tố:

1. A + B – C
2. A * (B – C)
3. A + B * C / D
4. A – B – (C + D) / E
5. A + (B - C) * D+ E

Cách tính giá trị của một biểu thức dạng hậu tố

Xét biểu thức dạng hậu tố sau đây: 128 + 4 - 2 4 * /

Nếu đọc biểu thức dang hậu tố từ trái qua phải ta sẽ thấy khi một toán tử xuất hiện thì hai toán hạng vừa đọc sát nó sẽ được kết hợp với toán tử này để tạo thành toán hạng mới ứng với toán tử sẽ được đọc sau nó, và cứ như vậy.

Với biểu thức trên, các bước thực hiện lần lượt như sau:

Khi đọc phép: +, thực hiện 12 + 8 = 20

Khi đọc phép: -, thực hiện 20 – 4 = 16

Khi đọc phép: *, thực hiện 2 * 4 = 8

Khi đọc phép: /, thực hiện 16 / 8 = 2

Nhận xét: Trước khi đọc tới toán tử thì giá trị của các toán hạng phải được lưu lại để chờ thực hiện phép tính. Hai toán hạng được đọc sau thì lại kết hợp với toán tử đọc trước, điều đó cũng có nghĩa là hai giá trị được lưu lại sau thì phải lấy ra trước để tính toán. Điều này trùng khớp với cơ chế "last in fist out" của stack. Vì thế, để tính giá trị của biểu thức hậu tố người ta cần dùng một stack để lưu các giá trị của toán hạng.

Cách thực hiện tính giá trị của biểu thức hậu tố có thể được tóm tắt như dưới đây. Lưu ý, quy ước trình bày biểu thức là: biểu thức là một mảng ký tự, trong đó các toán tử và các toán hạng được viết cách nhau bằng một ký tự khoảng trắng.

Tính giá trị của biểu thức dạng hậu tố:

Đọc từng "từ" của biểu thức hậu tố từ trái sang phải (các "từ" cách nhau bằng một khoảng trắng). "Từ" đọc được gọi là X.
Nếu X là toán hạng thì đưa X vào stack.
Nếu X là toán tử thì:
- Lần lượt lấy từ stack ra hai giá trị a và b *(a lấy trước, b lấy sau)*.
- Tính: kq = b X a *(với X là phép toán)*.
- Đưa kq vào stack.

Quá trình trên được tiếp tục cho tới khi kết thúc biểu thức. Lúc đó giá trị còn trong stack chính là giá trị của biểu thức.

Câu hỏi: Tại bước tính: kq = b X a có thể thay bằng: kq = b X a được

Ví dụ, ta có biểu thức hậu tố 5 17 3 + * 6 - . Biểu thức này được lưu trong một mảng ký tự như sau:

5		1	7		3		+		*		6		-		#
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...

Hướng dẫn cài đặt

1. Khai báo stack và cài đặt các thao tác trên stack

```
struct NODE
{
     Float data;  //các toán hạng kiểu số thực
     NODE *link;
};

struct stack  {
   Node *top;
};

//Khởi tạo stack
voidInit(stack&s);

// Kiểm tra stack rỗng
intEmpty(stack s);

// Thêm một phần tử vào stack
Void Push(stack &s, float x);

// Lấy một phần tử ra khỏi stack
float Pop(stack &s);
```

2. Hàm tính giá trị biểu thức hậu tố.

   Đầu vào là stack s và biểu thức cần tính giá trị dưới dạng một chuỗi ký tự. Đầu ra là giá trị của biểu thức.

```
floatTinhBieuThuc(stack&s, char bieuThuc[])
{
   float kq;
   char t[10] ;
   int i=0;
   do
   {
      DocTu(bieuThuc, t, i);//Trong chuỗi bieuThuc, đọc
                  một từ bắt đầu từ vị trí i, kết quả là từ t
      if(LaToanTu(t)) //Nếu t là một toán tử
      {
         char toanTu = t[0]; //Toán tử chỉ có 1 ký tự
         float toanHang1 = Pop(s);
         float toanHang2 = Pop(s);
         kq = TinhToan(toanHang2, toanHang1, toanTu);
   //thực hiện phép tính
         Push(s, kq); //tính xong đưa kq vào stack
      }
      else//t là toán hạng
      {
         float toanHang = atof(t); //chuyển thành số thực
         Push(s, toanHang); //đưa toán hạng vào stack
      }
      i++;
   }while(bieuThuc[i]!='#'); //Giả sử quy ước '#' là ký tự
kết thúc biểu thức
   returnPop(s);
}
```

Các hàm hỗ trợ cho hàm tính biểu thức:

2. Hàm đọc một "từ".*("Từ" ở đâychính là một toán tử hay một toán hạng trong biểu thức, viết cách nhau bằng một khoảng trắng.)*

   Đầu vào là chuỗi `s` (tương ứng trong bài toán chính là biểu thức cần tính giá trị), `vt` là vị trí bắt đầu đọc, sẽ đọc từ vị trí `vt` đến khi gặp một khoảng trắng. Giá trị trả về là mảng ký tự "`tu`" từ đọc được.

```
void DocTu(char s[], char tu[], int&vt)
{

   //Khởi tạo từ ban đầu chỉ chứa các khoảng trắng
   for(int i = 0; i<strlen(tu); i++)
      tu[i]= ' ';
   int i = 0;
   while(s[vt] != ' ')//Trong khi chưa gặp khoảng trắng
   {
      tu[i] = s[vt];
      vt++;
      i++;
   }
}
```

3. Hàm kiểm tra một chuỗi xem đó có phải là chuỗi chứa toán tử hay không. Hàm trả về 1 nếu chuỗi s là toán tử, ngược lại trả về 0.

```
intLaToanTu(char s[])
{
   char c = s[0];//Chỉ cần kiểm tra phần tử đầu của chuỗi
   if((c == '+') || (c == '-') || (c == '*') ||
(c == '/'))
      return 1;
   return 0;
}
```

4. Hàm**float TinhToan(float toanHang1, float toanHang2, char toanTu)**sẽ thực hiện phép toán tương ứng `(toanTu)` cho hai toán hạng `toanHang1` và `toanHang2`.Ví dụ, `toanHang1` là 5, `toanHang2` là 7, `toanTu` là trừ thì sẽ thực hiện 5 – 7 = -2. Kết quả trả về là 12.

```
floatTinhToan(float toanHang1, float toanHang2,
char toanTu)
{
   float kq;
   switch (toanTu)
   {
      case '+': kq = toanHang1 + toanHang2; break;
      case '-': kq = toanHang1 - toanHang2; break;
      case '*': kq = toanHang1 * toanHang2; break;
      case '/': kq = toanHang1 / toanHang2;
   }
   return kq;
}
```

3. Viết hàm main trong đó nhập một biểu thức dạng hậu tố rồi tính giá trị biểu thức.

```
void main()
{
   stack s;
   Init(s);
   char bieuThuc[100] = "";
   cout<<"Nhap bieu thuc dang hau to\n";
   fflush(stdin);
   gets(bieuThuc);
   float kq;
   kq = TinhBieuThuc(s, bieuThuc);
   cout<<"\nGia tri cua bieu thuc la: "<<kq;
}
```

4. Thực hiện chương trình với đầu vào là các chuỗi sau:
   - 5 17 + 20 + 3 + #
   - 5 3 2 * + 6 – 1 + #
   - 5 6 7 * 8 / + 6 - #

BÀI TẬP VỀ NHÀ: *(bắt buộc – sinh viên nộp vào đầu buổi thực hành sau)*

Chương trình trên chỉ thực hiện với biểu thức có các phép toán cộng, trừ, nhân, chia. Bạn hãy hiệu chỉnh lại chương trình để có thể thực hiện thêm phép toán lũy thừa.

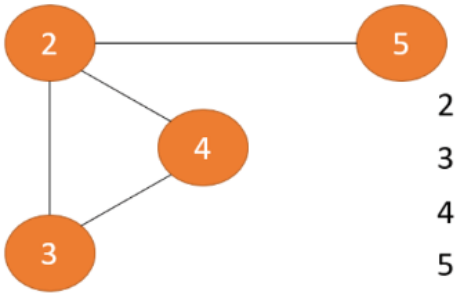
BÀI TẬP LÀM THÊM:*(sinh viên có thể nộp bài vào đầu buổi thực hành sau để lấy điểm cộng)*

Bài 1: Viết chương trình để chuyển đổi biểu thức có dạng trung tố sang dạng hậu tố.

Bài 2: Tìm đường trong mê cung (thực hiện loang theo chiều sâu DFS <sử dụng stack>).

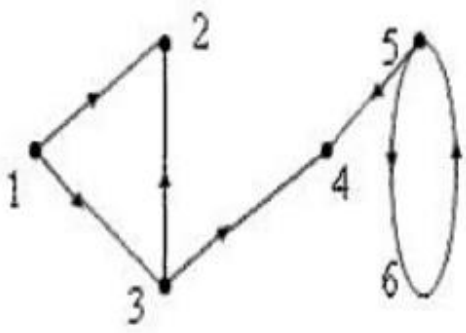
Bài toán: cho ma trận mxn, mỗi phần tử là số 0 hoặc 1.

Giá trị 1 : có thể đi tới và giá trị 0 : không thể đi tới được.



	2	3	4	5
2	0	1	1	1
3	1	0	1	0
4	1	1	0	0
5	1	0	0	0

Đồ thị vô hướng



Đồ thị có hướng G_1

	1	2	3	4	5	6
1	0	1	1	0	0	0
2	0	0	0	0	0	0
3	0	1	0	1	0	0
4	0	0	0	0	0	0
5	0	0	0	1	0	1
6	0	0	0	0	1	0

Ma trận kề của G_1

Câu hỏi:

Từ ô ban đầu có tọa độ (x1, y1) có thể đi tới ô (x2, y2) không?

Biết rằng từ 1 ô (x,y) chỉ có thể đi qua ô có chung cạnh với ô đang đứng và mang giá trị là 1, ngược lại không có đường đi.

BÀI THỰC HÀNH SỐ 7:
HÀNG ĐỢI (QUEUE)
(Số tiết: 3)

Mục đích :

1. Cài đặt các thao tác cơ bản trên queue dùng cấu trúc liên kết.
2. Ứng dụng queuetrong những bài toán đơn giản.

Vấn đề 1: Cài đặt các thao tác cơ bản trên queuedùng danh sách liên kết.

Hai thao tác chính trên queue gồm có: thêm và lấy phần tử ra khỏi queue. Đối với queuedùng danh sách liên kết thì thêm phần tử vào queuechính là thao tác thêm phần tử vào cuối danh sách liên kết. Lấy phần tử ra khỏi queuechính là thao tác lấy phần tử ở đầu danh sách ra khỏi danh sách liên kết. Ta cũng cần một thao tác hỗ trợ là kiểm tra danh sách rỗng.

(Lưu ý: ta cũng có thể thêm phần tử vào đầu danh sách liên kết, khi đó, để lấy phần tử ra khỏi danh sách, ta thực hiện lấy phần tử ở cuối danh sách liên kết.)

```
struct Node
{
     intdata;  //Giả sử queue chứa các số nguyên
     NODE *link;
};
structqueue{
   Node*front, *rear;
};

//Khởi tạo queue
voidInit(queue&q)
{
   q.front =NULL;
}

// Kiểm tra queue rỗng
intEmpty(queueq)
{
      returnq.front == NULL ? 1 : 0; // queuerỗng
}
```

```
//Thêm một phần tử  x vào queueS
void  EnQueue(queue&q, int x)
{
   node *p;
   p=new node;
   p->data = x;
   p->pNext=NULL;
   if (isEmpty(q))
   {
      q.front=p;
      q.rear=p;
   }
   else
   {
      q.rear->pNext=p;
      q.rear = p;
   }
}

// Trich thong tin va huy phan tu o dinh Q
intDeQueue(Queue&q)
{  if (isEmpty(q))
   {
      printf("Queue rong \n");
      return 1;
   }
   Node *p = q.front;
   q.front = q.front->pNext;
   if (isEmpty(q))
      q.rear = NULL;
   int x = p->data;
   p->pNext = NULL;
   delete p;
   return x;
}

// Xem thong tin phan tu dau Q
intFront(Queue q)
{
   if (isEmpty(q)) return 1;
   return q.front->data;
```

Chương trình chính

```
// Chuong trinh chinh
Int main()
{
   int k;
   Init(q);
   printf("\nNhap cac phan tu vao Queue (-1 de ket
thuc): ");
   scanf("%d", &k);
   while (k != -1)
   {
     EnQueue(q,k);
     scanf("%d", &k);
   }
   printf("\n\nLay cac phan tu ra khoi Queue: ");
   while (!isEmpty(q))
      {
         k=DeQueue(q);
         printf("%d ", k);
      }
}
```

Vấn đề 2: Ứng dụng Queue trong những bài toán đơn giản.

Bài toán: Lập trình Queue giải quyết bài toán quản lý kho (nhập trước – xuất trước) như sau:

1. Nhập một mặt hàng
2. Xuất một mặt hàng
3. Xem mặt hàng chuẩn bị xuất
4. Xem mặt hàng mới nhập
5. Xem các mặt hàng có trong kho
6. Xuất toàn bộ kho hàng
7. Kết thúc chương trình

CHỨC NĂNG BẠN CHỌN :[1..7]:

```
// Khai bao cau truc mathang
#define max 100
typedef struct mathang
{
    int mamh;
   char tenmh[12];
};


struct Queue
```

```
{
    int n;
    int front, rear;
    mathang list[max];
};

Void Init (Queue &q)
{
      q.front = 0; q.rear= max - 1;
      q.n = 0;
}

Int Empty(Queue q)
{
    if (q.n == 0)
         return 1;
     return 0;
}
// kiem tra tinh day
int isFull(Queue q)
{
  if (q.n == max)   return 1;
  return 0;
}
//Them mot phan tu vao Queue
int EnQueue(Queue &q, mathang x)
{
   if (isFull(q))
      return 0; // không thêm du?c vì Queue d?y
   q.rear =(q.rear + 1) % max;
   q.list[q.rear] = x;
   q.n++;
   return 1;
}
// Xoa mot mat hang
mathang DeQueue(Queue &q)
{
   if (!Empty(q))
   {
      mathang x = q.list[q.front];
      q.front = (q.front + 1) % max;
      q.n--;
      return x;
   }
}
//Nhap mot mat hang
int NhapMatHang(mathang &x)
```

```
{
   printf("\n Nhap ma hang");
   scanf("%d", &x.mamh);
   if(x.mamh == 0)
      return 0;
   printf("\n Nhap ten hang");
   fflush(stdin);
   gets(x.tenmh);
}
//Nhap danh sach cac mat hang vao Queue
void NhapDSMatHang(Queue &q)
{
   mathang x;
   printf("\n Nhap MaHang = 0 de dung\n");

   int flag = NhapMatHang(x);
   while(flag)
   {
      EnQueue(q, x);;
      flag = NhapMatHang(x);
   }
   printf("\n Ket thuc nhap DS mat hang.");
}
//Xuat thong tin mat hang
void XuatMatHang(mathang x)
{
   printf("\n Ma hang : %d", x.mamh);
   printf("\n Ten hang %s", x.tenmh);
}
// Xuat Queue
void XuatQueue(Queue q)
{
   mathang x;
   int i = q.n;
   if(Empty(q))
   {
      printf("\nKho khong con hang");
      return;
   }
      // vong lap in cac nut tu front den nut ke cuoi
   while(i != 0)
   {
      x = q.list[q.front];
      XuatMatHang(x);
      q.front = (q.front + 1) % max;
      i--;
   }
```

```
}
// Xuat thong tin mat hang chuan bi xuat
void   Output_front(Queue q)
{
   if (!Empty(q))
   {
      mathang x = q.list[q.front];
      XuatMatHang(x);
   }
}
// Xuat thong tin mat hang vua moi nhap
void   Output_rear(Queue q)
{
   if (!Empty(q))
   {
      mathang x = q.list[q.rear];
      XuatMatHang(x);
   }
}
```

Chương trình chính

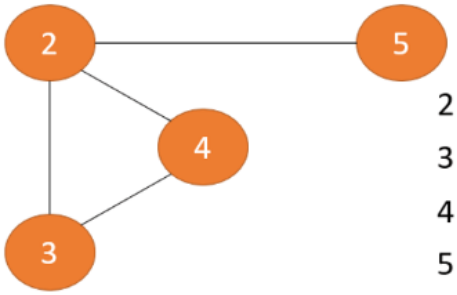
```
// chuong trinh chinh
int main()
{
   Queue q;
   mathang x;
   Init(q);
   NhapDSMatHang(q);s
   XuatQueue(q);
   printf("\n Mat hang sap xuat: \n");
   Output_front(q);
   printf("\n Mat hang moi nhap: \n");
   Output_rear(q);
   printf("\n Xoa phan tu khoi Queue: \n");
   DeQueue(q);
   printf("\n Ket qua Queue sau khi xoa: \n");
   XuatQueue(q);
}
```

BÀI TẬP LÀM THÊM:*(sinh viên có thể nộp bài vào đầu buổi thực hành sau để lấy điểm cộng)*

Bài 1: Tìm đường trong mê cung (thực hiện loang theo chiều rộng BFS <sử dụng queue>.

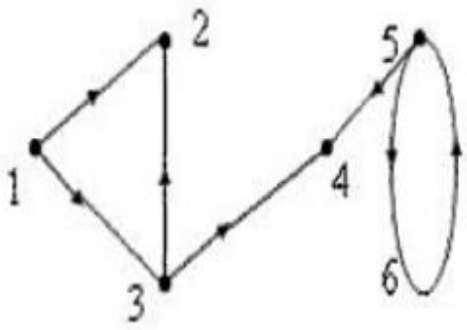
Bài toán: cho ma trận mxn, mỗi phần tử là số 0 hoặc 1.

Giá trị 1 : có thể đi tới và giá trị 0 : không thể đi tới được.



	2	3	4	5
2	0	1	1	1
3	1	0	1	0
4	1	1	0	0
5	1	0	0	0

Đồ thị vô hướng



Đồ thị có hướng G_1

	1	2	3	4	5	6
1	0	1	1	0	0	0
2	0	0	0	0	0	0
3	0	1	0	1	0	0
4	0	0	0	0	0	0
5	0	0	0	1	0	1
6	0	0	0	0	1	0

Ma trận kề của G_1

Câu hỏi:

Từ ô ban đầu có tọa độ (x1, y1) có thể đi tới ô (x2, y2) không?

Biết rằng từ 1 ô (x,y) chỉ có thể đi qua ô có chung cạnh với ô đang đứng và mang giá trị là 1, ngược lại không có đường đi.

BÀI THỰC HÀNH SỐ 8:
CÂY NHỊ PHÂN TÌM KIẾM (CNPTK)
(Số tiết: 3)

Mục đích :

Hoàn tất bài thực hành này, sinh viên có thể:

1. Hiểu được các thành phần của cây nhị phân tìm kiếm.
2. Thành thạo các thao tác trên cây nhị phân tìm kiếm: tạo cây, thêm phần tử, xóa phần tử, cập nhật, duyệt cây, đếm số các nút, tình chiều cao, tìm kiếm, sắp xếp cây nhị phân tìm kiếm.
3. Áp dụng cấu trúc dữ liệu cây nhị phân tìm kiếm vào việc giải quyết một số bài toán.

Vấn đề 1: Khai báo và cài đặt các thao tác cơ bản trên tree dùng cấu trúc liên kết.

```
struct Node
{
    Node* pLeft;
    Node* pRight;
    int idata;
};
typedef Node* Tree;
// Khoi tao
void Init(Tree &T)
{
   T = NULL;
}
Node* TaoNode(int X)
{
   Node* p = new Node;
   if (p == NULL)
       return NULL;
    p->pLeft = NULL;
    p->pRight = NULL;
    p->idata=X;
    return p;
}
void ThemNodeVaoCay(Node* p, Tree &T)
{
    if (T == NULL)//nếu cây rỗng
        T = p;
    else //cây khác rỗng
    {
       if (p->idata < T->idata)
           ThemNodeVaoCay(p,T->pLeft);
       else if (p->idata > T->idata)
           ThemNodeVaoCay(p,T->pRight);
```

```
        else
            return;
    }
}
void Nhap(Tree &c)
{
   int chon = 0;
    do
    {
      int x;
      printf("\nNhap x: ");
      scanf("%d",&x);
      Node* p = TaoNode(x);
      ThemNodeVaoCay(p,c);
      printf("Muon nhap thong tin tiep ko? 1: co,
                0: ko ~~>");
      scanf("%d",&chon);
    }while(chon);
}
// Viết hàm xuất các giá trị trong cây
void Xuat(Tree T)
{
   if (T!=NULL)
    {
       if (T->pLeft != NULL)
          Xuat(T->pLeft);
       printf("%4d", T->idata);
       if (T->pRight != NULL)
          Xuat(T->pRight);
    }
}
```

Cài đặt chương trình chính

```
Int main()
{
     Tree T = NULL;
     Nhap(T);
     printf("Xuat cay LNR (Tang dan): ");
     Xuat(T);
     return 0;
}
```

Yêu cầu: Bổ sung chương trình mẫu

1. Hàm tính và trả về tổng giá trị các node trên cây nhị phân gồm các giá trị nguyên

   Gợi ý: tham khảo hàm NLR để viết hàm SumTree

2. Hàm tìm giá trị nguyên lớn nhất và nhỏ nhất trong số các phần tử nguyên trên cây nhị phân tìm kiếm gồm các giá trị nguyên
   Gợi ý: dựa vào tính chất của cây nhị phân
3. Hàm tính và trả về số lượng các node của cây nhị phân gồm các giá trị nguyên
   Gợi ý: tham khảo hàm NLR để viết hàm CountNode
4. Hàm tính và trả về số lượng các node lá trên cây.
   Gợi ý: tham khảo hàm duyệt cây nhị phân NLR

Vấn đề 2: Ứng dụng tree trong những bài toán đơn giản.

Bài toán: Viết chương trình theo các yêu cầu sau:

1. Nhập dữ liệu cho cây và kiểm tra nếu masv trùng thì thông báo trùng và nhập lại masv khác.
2. Duyệt và xuất dữ liệu của cây theo thức tự LNR ra màn hình.
3. Sắp xếp cây theo trường dữ liệu.
4. Đếm số nút lá của cây
5. Tính chiều cao của cây
6. Chèn một Node vào cây.
7. Tìm kiếm một Node có giá trị được nhập vào từ bàn phím.
8. Xóa một Node.

Hướng dẫn cài đặt:

```
struct item {
   char masv[20];
   char name[20];
   double diem;
};
struct Node {
   item key; //truong key cua du lieu
   Node *pLeft, *pRight; //con trai va con phai
};
Typedef Node *Tree;  //cay
void NhapSV(item &sv);
void XuatSV(item sv);
Node* TaoNode(item sv);
void ThemNodeVaoCay(Tree &T, item sv);
void TaoCay(Tree &T);
void XuatTree(Tree T);
```

BÀI TẬP LÀM THÊM:*(sinh viên có thể nộp bài vào đầu buổi thực hành sau để lấy điểm cộng)*

Bài 1: Sử dụng cây nhị phân tìm kiếm để giải bài toán (thống kê) số lượng ký tự có trong văn bản (không dấu)

1. Xây dựng cây cho biết nỗi ký tự có trong văn bản xuất hiện mấy lần
2. Nhập vào 1 ký tự. Kiểm tra ký tự đó xuất hiện bao nhiêu lần trong văn bản

Bài 2: Sử dụng cây nhị phân tìm kiếm để giải bài toán:

1. Đếm có bao nhiêu giá trị phân biệt trong dãy số cho trước
2. Với mỗi giá trị phân biệt, cho biết số lượng phần tử

BÀI THỰC HÀNH SỐ 9
CÂY NHỊ PHÂN TÌM KIẾM CÂN BẰNG(AVL)
(Số tiết: 3)

Mục đích :

Hoàn tất bài thực hành này, sinh viên có thể:

1. Hiểu được các thành phần của cây nhị phân tìm kiếm cân bằng.
2. Thành thạo các thao tác trên cây nhị phân tìm kiếm cân bằng: cân bằng, tạo cây, thêm phần tử, xóa phần tử, cập nhật, duyệt cây, đếm số các nút, tình chiều cao, tìm kiếm, sắp xếp cây nhị phân tìm kiếm cân bằng.
3. Áp dụng cấu trúc dữ liệu cây nhị phân tìm kiếm cân bằng vào việc giải quyết một số bài toán.

Vấn đề 1: Khai báo vàcài đặt các thao tác cơ bản trên tree dùng cấu trúc liên kết.

- bal: thuộc tính cân bằng:
  - 0: cân bằng
  - 1: lệch trái
  - 2: lệch phải

```
Struct AVLNode
{
   int bal;
   Node* pLeft;
   Node* pRight;
   int key;
};

Typedef AVLNode* TREE;
// Tạo node chứa số x
AVLNode*  CreateNode(int X)
{
   AVLNode * p = new AVLNode;
   if (p == NULL)
      return NULL;
   p->pLeft = NULL;
   p->pRight = NULL;
   p->key=X;
   p->bal = 0;
   return p;
}
//Thêm một node vào cây
void InsertNode(TREE &tree, int x)
{
   int res;

   if (tree == NULL)//nếu cây rỗng
```

```
        tree = CreateNode(x);
        if(tree == NULL)return -1;// ko tao duoc node
        return 2;//them node thanh cong, chieu cao cay tang
   else //cây khác rỗng
    {
      if (tree->key == x) return 0; // x da co tren cay

         ThemNodeVaoCay(p,c->pLeft);
      else if (tree->key > x)
         {
           InsertNode(tree->pLeft,x);
            if(res < 2){
            res = InsertNode(tree->pLeft,x);
            if(res < 2)return res;
            switch(tree->bal){
            case 0:
                   tree->bal = 1;
                   return 2;
            case 1:
                   LeftBalance(tree);
                   return 1;
            case 2:
                  tree->bal = 0;
                  return 1;
         }
      else
         res = InsertNode(tree->pRight,x);
         if(res<2) return res;
         switch(tree->bal){
         case 0:
                tree->bal=2;
                return 2;
         case 1:
                tree->bal = 0;
                return 1;
         case 2:
                RightBalance(tree);
                return 1;
         }
    }
  }
}
//
Void CreateTree(TREE &tree)
{
```

```
    int chon = 0;
     do
     {
        int x;
        printf("\nNhap x: ");
        scanf("%d",&x);
        InsertTree(tree,x);
        printf("Muon nhap thong tin tiep ko? 1: co,
                  0: ko ~~>");
        scanf("%d",&chon);
     }while(chon);
}
// Viết hàm xuất các giá trị trong cây
void  Traverse(TREE tree)
{
   if (tree!=NULL)
     {
       if (tree ->pLeft != NULL)
          Traverse(c->pLeft);
       printf("Data: %4d", tree->key);
       if (tree ->pRight != NULL)
          Traverse(c->pRight);
     }
}
// Xoay trai cay
void LeftRotate(TREE &P)
{
  AVLNODE *Q;
  Q = P->pRight;
  P->pRight = Q->pLeft;
  Q->pLeft = P;
  P = Q;
}
// Xoay phai cay
void RightRotate(TREE &P)
{
  //Ghi chú: sinh viên tự code cho hàm này
}
// Can bang lai cay khi cay lech trai
void LeftBalance(TREE &P)
{
  switch(P->pLeft->bal){
  case 1: //mất cân bằng trái trái
    RightRotate(P);
    P->bal = 0;
    P->pRight->bal = 0;
```

```
    break;
  case 2: //Ghi chú: cho biết đây là trường hợp mất cân
bằng nào?
    LeftRotate(P->pLeft);
    RightRotate(P);
    switch(P->bal){
    case 0:
      P->pLeft->bal= 0;
      P->pRight->bal= 0;
      break;
    case 1:
      P->pLeft->bal= 0;
      P->pRight->bal= 2;
      break;
    case 2:
      P->pLeft->bal= 1;
      P->pRight->bal= 0;
      break;
    }
    P->bal = 0;
    break;
  }
}
 // Can bang khi cay lech phai
void RightBalance(TREE &P)
{
  switch(P->pRight->bal){
  case 1:
    RightRotate(P->pRight);
    LeftRotate(P);
    switch(P->bal){
    case 0:
      P->pLeft->bal= 0;
      P->pRight->bal= 0;
      break;
    case 1:
      P->pLeft->bal= 1;
      P->pRight->bal= 0;
      break;
    case 2:
      P->pLeft->bal= 0;
      P->pRight->bal= 2;
      break;
    }
    P->bal = 0;
    break;
  case 2:
    LeftRotate(P);
```

```
    P->bal = 0;
    P->pLeft->bal = 0;
    break;
  }
}
// Xoa toan bo node tren cay
void RemoveAll(AVLNODE* &t)
{
    if (t!=NULL){
    RemoveAll(t->pLeft);
    RemoveAll (t->pRight);
    delete t;
  }
}

int main(int argc, _TCHAR* argv[])
{
  AVLNODE *tree;
  //Ghi chu: Tại sao lại phải thực hiện phép gán phía dưới?
  tree = NULL;
  CreateTree(tree);
  printf("\nCay AVL vua tao: \n");
  Traverse(tree);
  RemoveAll(tree);
  return 0;
}
```

Yêu cầu:

1. Biên dịch đoạn chương trình nêu trên.
2. Cho biết kết quả in ra màn hình khi người dùng nhập vào các dữ liệu sau:

10	30	35	32	20	8
30	40	50	-10	-5	

3. Nhận xét trình tự các node được xuất ra màn hình? Giải thích tại sao lại in ra được trình tự như nhận xét?
4. Sinh viên hoàn tất hàm RightRotate trong source code.

Gợi ý: RightRotate tương tự hàm LeftRotate.

5. Biên dịch lại chương trình sau khi hoàn thành câu 3 và cho biết kết quả in ra màn hình khi người dùng nhập vào các dữ liệu sau:

50	20	30	10	-5	7	15	35	57	65	55

6. Vẽ hình cây AVL được tạo ra từ phần nhập liệu ở câu 5.
7. Hãy ghi chú các thông tin bằng cách trả lời các câu hỏi ứng với các dòng lệnh có yêu cầu ghi chú

(//Ghi chú) trong các hàm InsertNode, BalanceLeft, BalanceRight, main.

8. Sinh viên cài đặt lại các hàm dùng cho cây nhị phân và cây NPTK để áp dụng cho cây AVL.

Áp dụng – Nâng cao

1. Sinh viên tự cài đặt thêm chức năng cho phép người dùng nhập vào khóa x và kiểm tra xem khóa x có nằm trong cây AVL hay không.

Cho dãy A như sau:

1	3	5	7	9	12	15	17	21	23	25	27

a. Tạo cây AVL từ dãy A. Cho biết số phép so sánh cần thực hiện để tìm phần tử 21 trên cây AVL vừa tạo.

b. Tạo cây nhị phân tìm kiếm từ dãy A dùng lại đoạn code tạo cây của bài thực hành trước). Cho biết số phép so sánh cần thực hiện để tìm phần tử 21 trên cây nhị phân tìm kiếm vừa tạo.

c. So sánh 2 kết quả trên và rút ra nhận xét?

2. Cài đặt chương trình đọc các số nguyên từ tập tin input.txt (không biết trước số lượng số nguyên trên tập tin) và tạo cây AVL từ dữ liệu đọc được

3. Cài đặt cây cân bằng AVL trong đó mỗi node trên cây lưu thông tin sinh viên.

4. Tự tìm hiểu và cài đặt chức năng xóa một node ra khỏi cây AVL.

BÀI TẬP THÊM

1. Viết chương trình cho phép tạo, tra cứu và sửa chữa từ điển Anh-Việt (sinh viên liên hệ với GVLT để chép file từ điển Anh-Việt)

2. Cài đặt lại các bài tập thêm của cây NPTK bằng cách dùng cây AVL