

LỜI NÓI ĐẦU

Bài giảng thực hành cấu trúc dữ liệu được viết để giảng dạy cho sinh viên chuyên ngành CNTT và sinh viên các ngành có liên quan đến tin học. Bài giảng này được thiết kế theo cấu trúc của bài giảng tự học nhằm khuyến khích sinh viên tự học và tự nghiên cứu. Với mỗi loại cấu trúc dữ liệu, bài giảng đều đề cập đến 3 dạng bài tập như sau:

Bài tập cơ sở: Ôn tập lại lý thuyết cơ sở đã học trong lý thuyết. Phần bài tập này sinh viên nhập vào và khảo sát kết quả như đã nêu trong bài giảng.

Bài tập nâng cao: Là phần bài tập có hướng dẫn. Sinh viên có thể tham khảo đến ý tưởng để thiết kế giải thuật cũng như có thể tham khảo đến chương trình con minh họa cụ thể.

Bài tập áp dụng: Là phần bài tập mà sinh viên phải tự tìm hiểu và vận dụng cấu trúc dữ liệu đã học vào thực tế để quản lý các vấn đề đặt ra. Phần bài tập này sẽ có giáo viên hướng dẫn và giải thích yêu cầu cụ thể cho sinh viên và sinh viên sẽ tự thực hiện đề tài của mình.

Yêu cầu đối với sinh viên trước khi học môn này:

Sinh viên phải nắm vững ngôn ngữ lập trình C.

Sinh viên phải học lý thuyết môn cấu trúc dữ liệu trước khi thực tập.

Yêu cầu sinh viên sau khi thực tập:

Sinh viên phải hiểu và thao tác được trên các kiểu dữ liệu trừu tượng.

Sinh viên phải vận dụng được kiểu dữ liệu trừu tượng vào thực tế.

Đề nghị phương pháp học:

Bước 1: Sinh viên tạo unit cho từng kiểu dữ liệu trừu tượng chứa các phép toán cơ bản của kiểu dữ liệu trừu tượng đó. (phần bài tập cơ sở).

Bước 2: Sinh viên tạo một chương trình để gọi các phép toán đã tạo trong unit để kiểm tra các phép toán đó đồng thời hiểu được cách sử dụng nó trên kiểu dữ liệu trừu tượng đó. (bài tập cơ sở)

Bước 3: Sinh viên đọc các yêu cầu bài tập, phân tích và tự đưa ra giải thuật để giải quyết yêu cầu. (bài tập nâng cao)

Bước 4: Sinh viên có thể tham khảo giải thuật bằng ngôn ngữ giả trong tài liệu để bổ sung kiến thức.

Bước 5: Sinh viên có thể tham khảo phần mã lệnh của chương trình con đã cài đặt.

Bước 6: Sinh viên lập một chương trình để gọi thực thi các chương trình con theo đúng yêu cầu đặt ra.

Bước 7: Sinh viên làm các bài tập áp dụng đã cho. (bài tập áp dụng)

Thực hành cấu trúc dữ liệu

Đây là phiên bản đầu tiên của bài giảng theo phương pháp tự học viết theo ngôn ngữ C nên không thể tránh khỏi sai sót. Rất mong nhận được ý kiến đóng góp của sinh viên và các bạn đọc để bài giảng ngày một được hoàn thiện hơn.

Nhóm viết bài giảng

Trương Thị Thanh Tuyền (trưởng nhóm)

Lâm Hoài Bảo

Phan Huy Cường

Các ý kiến đóng góp xin gửi theo địa chỉ email: ttttuyen@cit.ctu.edu.vn

CHƯƠNG I. CÁC KIỂU DỮ LIỆU TRỪU TƯỢNG CƠ BẢN

A. DANH SÁCH

I. Khái niệm

Các danh sách được đặc trưng bởi tính mềm dẻo, bởi vì chúng có thể phát triển dài ra hay thu ngắn lại, các phần tử (elements) có thể được truy cập, được chèn thêm vào, hay bị xóa bỏ tại bất cứ vị trí nào trong một danh sách. Các danh sách cũng có thể được nối lại với nhau hoặc tách ra thành các danh sách con. Các danh sách thường xuất hiện trong các ứng dụng như tìm kiếm thông tin, biên dịch ngôn ngữ lập trình (translation), và mô phỏng (simulation). Phần này chúng ta sẽ giới thiệu một số thao tác cơ bản trên danh sách, và ở cuối chương sẽ trình bày các cấu trúc danh sách hỗ trợ cho các thao tác một cách hiệu quả hơn.

Về mặt toán học, một danh sách là một tập hợp của 0 hay nhiều phần tử có cùng kiểu (mà ta gọi một cách tổng quát là kiểu *elementtype*). Ta thường trình bày danh sách như một dãy các phần tử cách nhau bởi dấu phẩy:

$$a_1, a_2, \dots, a_n$$

với $n \geq 0$, và mỗi a_i có kiểu *elementtype*. Số lượng n các phần tử được gọi là chiều dài (length) của danh sách. Nếu $n \geq 1$, ta nói rằng a_1 là phần tử đầu tiên (*first element*) và a_n là phần tử cuối cùng (*last element*). Nếu $n=0$, chúng ta có một danh sách rỗng (*empty list*) – không có phần tử nào.

Một tính chất (property) quan trọng của danh sách là các phần tử của nó có thứ tự tuyến tính (*linearly ordered*) dựa theo vị trí (position) của chúng trong danh sách. Ta nói a_i đứng trước a_{i+1} với $i=1, 2, \dots, n-1$, và a_i đứng sau a_{i-1} với $i=2, 3, \dots, n$. Ta cũng nói rằng a_i ở vị trí i (*position i*). Thật là tiện lợi nếu ta biết được vị trí theo sau phần tử cuối cùng trong danh sách. Hàm (function) $\text{END}(L)$ sẽ trả về vị trí theo sau vị trí n trong danh sách L có n phần tử. Chú ý rằng, vị trí $\text{END}(L)$ có một khoảng cách từ vị trí bắt đầu danh sách, và nó bị thay đổi khi danh sách phát triển dài ra hay thu ngắn lại, trong khi đó tất cả các vị trí khác có một khoảng cách cố định kể từ đầu danh sách.

Để thiết lập một kiểu dữ liệu trừu tượng (abstract data type - ADT) từ ý niệm toán học về danh sách, chúng ta phải định nghĩa một tập hợp (set) các phép toán (operation) làm việc trên những đối tượng kiểu *LIST*. Không có tập hợp các phép toán nào là thích hợp cho tất cả các ứng dụng (application). Ở đây, chúng tôi sẽ trình bày một tập hợp các phép toán cơ bản. Trong phần tiếp theo, chúng tôi sẽ cung cấp các thủ tục cài đặt cho các phép toán tiêu biểu này.

Để thuận tiện cho việc định nghĩa, ta giả sử L là một danh sách các phần tử có kiểu *elementtype*, x là một phần tử có kiểu đó, và p có “kiểu vị trí” - position. Chú ý rằng “position” là một kiểu khác và sự cài đặt của nó sẽ thay đổi tùy theo sự cài đặt khác nhau của danh sách. Dù vậy, ta cứ nghĩ về kiểu position này gần gũi như kiểu integer, trong thực tế, chúng có thể có kiểu khác.

INSERT_LIST(x, p, L). Xen x vào tại vị trí p trong danh sách L , di chuyển các phần tử tại vị trí p và các phần tử sau đó đến vị trí cao hơn kế tiếp. Tức là, nếu L là $a_1, a_2, \dots, a_{p-1}, a_p, \dots, a_n$ thì L sẽ trở thành $a_1, a_2, \dots, a_{p-1}, x, a_p, \dots, a_n$. Nếu p là $\text{END}(L)$, thì L trở thành a_1, a_2, \dots, a_n, x . Nếu L không có vị trí p , kết quả phép toán không xác định.

LOCATE(x, L). Hàm này trả về vị trí của x trên danh sách L . Nếu x xuất hiện hơn một lần, khi đó vị trí của lần xuất hiện đầu tiên được trả về. Nếu x không xuất hiện lần nào, khi đó $\text{END}(L)$ được trả về.

RETRIVE(p, L). Hàm này trả về phần tử ở vị trí p trên danh sách L . Kết quả trả về sẽ là không xác định nếu $p = \text{END}(L)$ hoặc nếu L không có vị trí p . Chú ý rằng các phần tử của danh sách phải có kiểu có thể được trả về bởi một hàm (function) nếu RETRIVE được dùng. Tuy nhiên, trong thực tế chúng ta cũng có thể thay đổi RETRIVE để trả về một con trỏ (pointer) chỉ đến một đối tượng có kiểu elementtype.

DELETE_LIST(p, L). Xóa bỏ phần tử ở vị trí p của danh sách L . Nếu L là a_1, a_2, \dots, a_n , khi đó L sẽ trở thành $a_1, a_2, \dots, a_{p-1}, a_{p+1}, \dots, a_n$. Kết quả là không xác định nếu L không có vị trí p hoặc nếu $p = \text{END}(L)$.

NEXT(p, L) và **PREVIOUS(p, L)** trả về vị trí (position) liền sau và trước vị trí p trên danh sách L . Nếu p là vị trí phần tử cuối cùng trong danh sách L thì $\text{NEXT}(p, L) = \text{END}(L)$. NEXT là không xác định nếu p là $\text{END}(L)$. PREVIOUS là không xác định nếu p là $\text{FIRST}(L)$. Cả hai hàm là không xác định nếu L không có vị trí p .

MAKENULL_LIST(L). Hàm này làm cho sách danh sách L trở thành danh sách rỗng và trả về vị trí $\text{END}(L)$.

FIRST(L). Hàm này trả về vị trí đầu tiên trên danh sách L . Nếu L là rỗng, vị trí được trả về là $\text{END}(L)$.

PRINT_LIST(L). In các phần tử của danh sách L theo đúng vị trí xuất hiện của chúng trong danh sách.

END_LIST(L). Trả về vị trí $n+1$, là vị trí sau vị trí phần tử cuối cùng của danh sách L .

II. Bài tập cơ sở

1. Danh sách đặc (cài đặt bằng mảng)

Yêu cầu: Tạo file với tên là **ALISTLIB.CPP** lưu trữ các phép toán cơ bản trên danh sách kiểu số nguyên.

```
// Khai báo danh sách đặc
```

```
#include <stdio.h>
```

```
#include <conio.h>
```

```
#define Maxlength 30
```

```
/*===== Khai bao danh sach dac =====*/
```

Thực hành cấu trúc dữ liệu

```
typedef int ElementType;
typedef int Position;
typedef struct {
    ElementType Elements[Maxlength];
    Position      Last;
}List;
/*===== Ket thuc khai bao =====*/
```

// Tạo danh sách rỗng

```
void MakeNull_List(List *L){
    L->Last = 0;
}
```

// Hàm kiểm tra danh sách rỗng

```
int Empty_List(List L){
    return (L.Last == 0);
}
```

// Hàm kiểm tra danh sách đầy

```
int Full_List(List L){
    return (L.Last == Maxlength)
}
```

// Hàm trả về vị trí phần tử đầu tiên trong danh sách

```
Position FirstList(List L){
    return 1;
}
```

// Hàm trả về vị trí sau phần tử cuối trong danh sách

```
Position EndList(List L){
    return L.Last+1;
}
```

```
}
```

// Hàm trả về vị trí phần tử kế tiếp phần tử P trong danh sách L

```
Position Next(Position P, List L){  
    return P+1;  
}
```

// Hàm trả về vị trí phần tử trước vị trí P trong danh sách L

```
Position Previous(Position P, List L){  
    return P-1;  
}
```

// Hàm trả về nội dung phần tử tại vị trí P trong danh sách L

```
ElementType Retrieve(Position P, List L){  
    return L.Elements[P-1];  
}
```

// Thêm phần tử có nội dung X vào tại vị trí P trong danh sách L

```
void Insert_List(ElementType X, Position P, List *L){  
    int i=0;  
    if(L->Last == Maxlength)  
        printf("\nDanh sach dday !!!");  
    else if ((P<1) || (P>L->Last+1))  
        printf("\nVi tri khong hop le !!!");  
    else {  
        for(i=L->Last ;i>=P ; i--)  
            L->Elements[i] = L->Elements[i-1];  
        L->Last++;  
        L->Elements[P-1] = X;  
    }  
}
```

// Xóa phần tử tại vị trí P trong danh sách L

```
void Delete_List(Position P, List *L){
    if((P > L->Last) || (P<1))
        printf("\nVi tri khong hop le !!!");
    else
        if(Empty_List(*L))
            printf("\nDanh sach rong !");
        else{
            Position i;
            for(i=P; i<L->Last; i++)
            {
                L->Elements[i-1] = L-> Elements[i];
            }
            L->Last--;
        }
}
```

// In danh sách L ra màn hình

```
void Print_List(List L){
    Position P;
    P = FirstList(L);
    printf("\nBat ddau in danh sach  ");
    while(P != EndList(L)){
        printf("\n%d",Retrieve(P,L));
        P = Next(P,L);
    }
    printf("\nKet thuc in danh sach\n ");
}
```

// Nhập danh sách từ bàn phím

```
void Read_List(List *L){
    int i,N;
```

Thực hành cấu trúc dữ liệu

```
ElementType X;
MakeNull_List(L);
printf("\nNhập vào số phần tử trong danh sách ");
scanf("%d", &N);fflush(stdin);
for(i=1; i<= N; i++){
    printf("\nPhần tử thứ %d là ",i);
    scanf("%d", &X);fflush(stdin);
    Insert_List(X,EndList(*L),L);
}
}
```

/* Hàm tìm vị trí phần tử đầu tiên có nội dung X trong danh sách L. Nếu không tìm thấy, hàm trả về vị trí EndList */

```
Position Locate(ElementType X,List L){
    Position P;
    int found = 0;
    P = FirstList(L);
    while ((P!=EndList(L)) && (found==0)){
        if(Retrieve(P,L) == X)
            found = 1;
        else P = Next(P, L);
    }
    return P;
}
```

Viết chương trình chính để kiểm tra tính đúng đắn của các phép toán cơ bản vừa thiết kế trên như các ví dụ sau:

Ví dụ 1: Tạo file chương trình tên **ALTEST1.CPP** để kiểm tra các chương trình con tạo danh sách rỗng, kiểm tra danh sách rỗng, thêm phần tử vào danh sách và in danh sách ra màn hình.

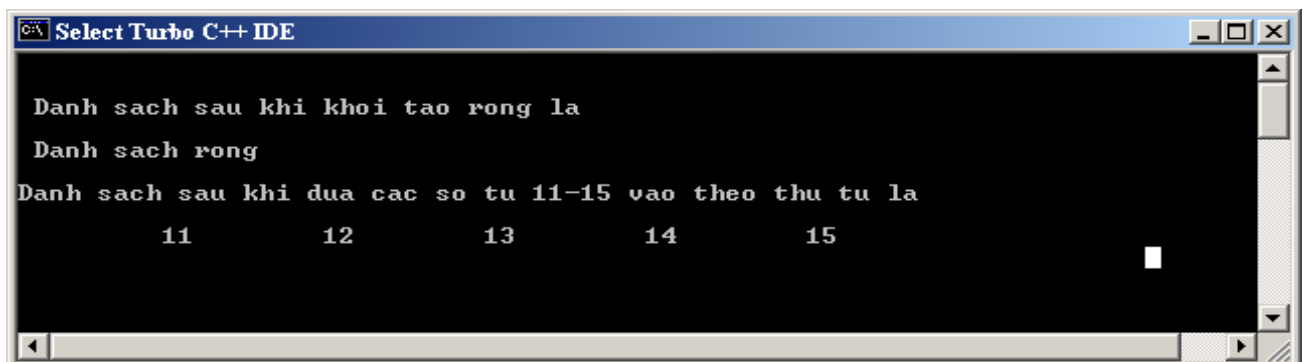
Nội dung chính của file có thể được viết như sau:

```
#include <stdio.h>
```


Thực hành cấu trúc dữ liệu

```
#include <conio.h>
#include <D:\ALISTLIB.CPP> /* Chỉ đúng đường dẫn và tên file thư viện
vừa tạo lưu trữ các phép toán trên*/
void main()
{
    clrscr();
    List L;
    MakeNull_List(&L); //Khoi tao danh sach ddac rong
    /* Doan lenh goi de kiem tra chuong trinh con tao rong va
    kiem tra rong */
    printf ("\n\n Danh sach sau khi khoi tao rong la ");
    if (Empty_List(L)) printf ("\n\n Danh sach rong ");
    else printf ("\n\nDanh sach khong rong");
    /* Xen 5 so (tu 11 den 15) vao danh sach theo dung thu tu
    nhap */
    /* Doan lenh kiem tra chuong trinh con xen va hien thi danh
    sach */
    for (int i=1; i<=5; i++)
        Insert_List (i+10,EndList(L),&L);
    printf ("\n\nDanh sach sau khi dua cac so tu 11-15 vao theo
    thu tu la \n\n ");
    Print_List(L);
    getch();
}
```

Khi chương trình trên khi thực thi sẽ có kết quả như sau:



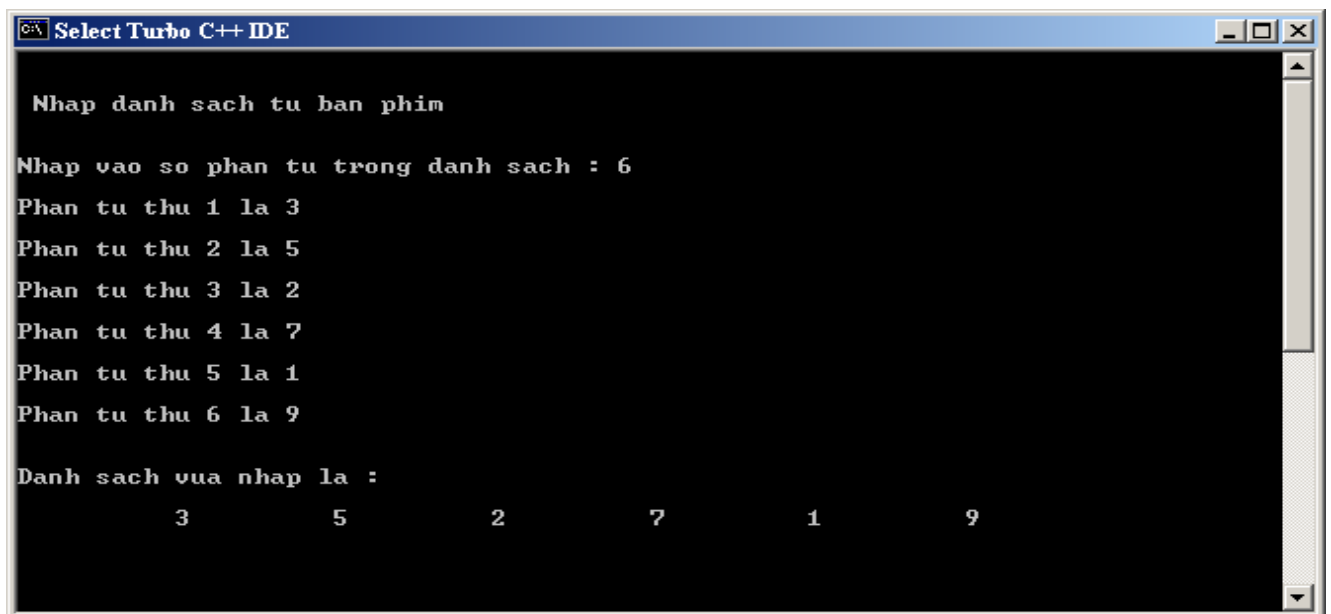
```
Select Turbo C++ IDE

Danh sach sau khi khoi tao rong la
Danh sach rong
Danh sach sau khi dua cac so tu 11-15 vao theo thu tu la
    11      12      13      14      15
```

Ví dụ 2: Viết chương trình chính thực hiện kiểm tra các chương trình con nhập danh sách từ bàn phím. Nội dung chương trình **ALTEST2.CPP** có thể được viết như sau:

```
#include <stdio.h>
#include <conio.h>
#include <D:\ALISTLIB.CPP> /* Chỉ đúng đường dẫn và tên file thư viện
vừa tạo lưu trữ các phép toán trên*/
void main()
{
    clrscr();
    List L;
    MakeNull_List(&L); //Khoi tao danh sach ddac rong
    printf("\n\n Nhap danh sach tu ban phim");
    Read_List(&L);      //Nhap danh sach tu ban phim
    printf("\n\nDanh sach vua nhap la :\n ");
    Print_List(L);
    getch();
}
```

Chương trình thực thi sẽ có kết quả như sau:



```
Select Turbo C++ IDE

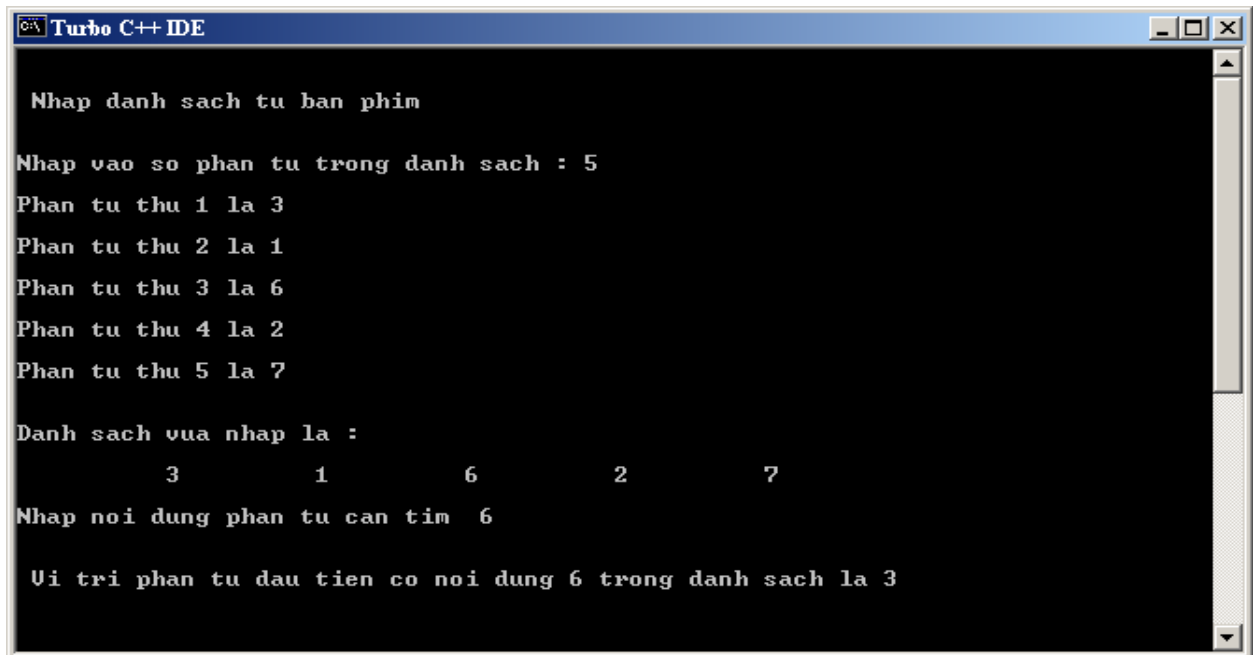
Nhap danh sach tu ban phim

Nhap vao so phan tu trong danh sach : 6
Phan tu thu 1 la 3
Phan tu thu 2 la 5
Phan tu thu 3 la 2
Phan tu thu 4 la 7
Phan tu thu 5 la 1
Phan tu thu 6 la 9

Danh sach vua nhap la :
      3      5      2      7      1      9
```

Ví dụ 3: Viết chương trình **ALTEST3.CPP** để kiểm tra tính đúng đắn của hàm Locate. Chương trình này có thể được viết như sau:

```
#include <stdio.h>
#include <conio.h>
#include <D:\ALISTLIB.CPP> /* Chỉ đúng đường dẫn và tên file thư viện
vừa tạo lưu trữ các phép toán trên */
void main()
{
    clrscr();
    List L;
    ElementType X;
    Position P;
    MakeNull_List(&L); //Khoi tao danh sach ddac rong
    printf("\n\n Nhap danh sach tu ban phim\n\n");
    Read_List(&L);      //Nhap danh sach tu ban phim
    printf("\n\nDanh sach vua nhap la :\n\n ");
    Print_List(L);
    printf("\n\nNhap noi dung phan tu can tim  ");
    scanf("%d",&X);
    P=Locate (X,L);
    // Tim vi tri phan tu dau tien co noi dung X
    // Doan lenhkiem tra ham Locate
    if (P==EndList(L))
        printf ("\n\nKhong ton tai phan tu co noi dung %d
                trong danh sach ",X);
    else
        printf("\n\n Vi tri phan tu dau tien co noi dung
                %d trong danh sach la %d ", X,P);
    getch();
}
```



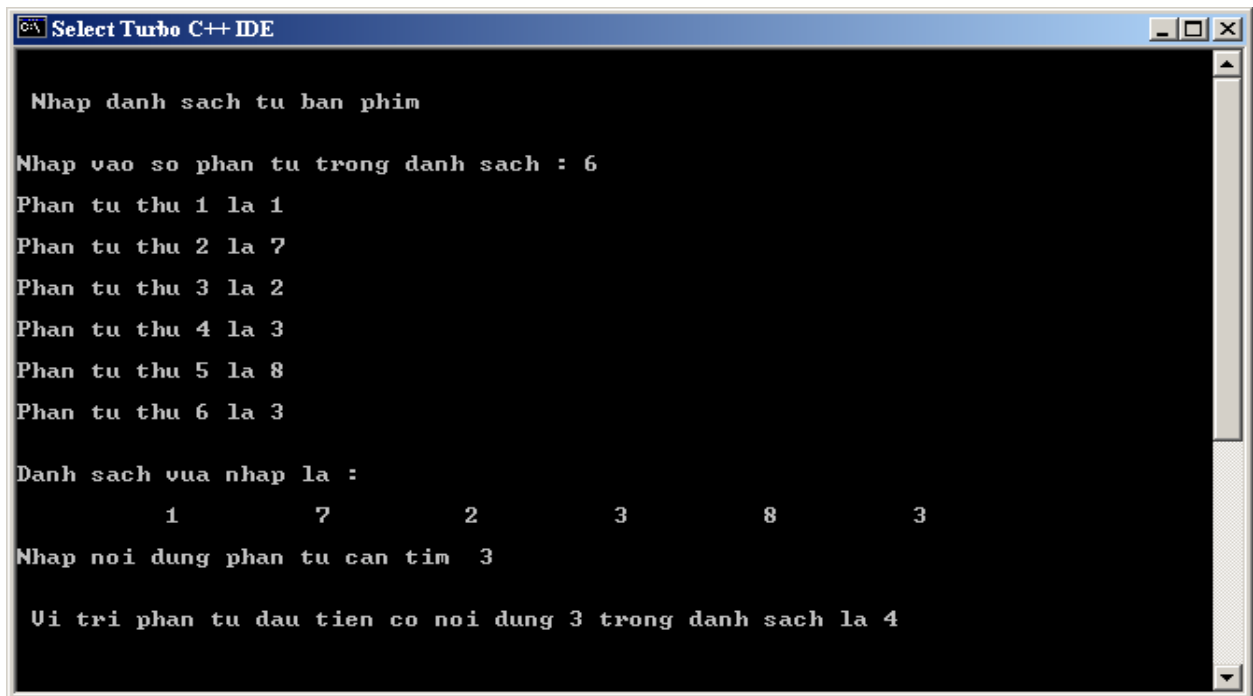
```
Turbo C++ IDE

Nhap danh sach tu ban phim

Nhap vao so phan tu trong danh sach : 5
Phan tu thu 1 la 3
Phan tu thu 2 la 1
Phan tu thu 3 la 6
Phan tu thu 4 la 2
Phan tu thu 5 la 7

Danh sach vua nhap la :
          3          1          6          2          7
Nhap noi dung phan tu can tim 6

Vi tri phan tu dau tien co noi dung 6 trong danh sach la 3
```



```
Select Turbo C++ IDE

Nhap danh sach tu ban phim

Nhap vao so phan tu trong danh sach : 6
Phan tu thu 1 la 1
Phan tu thu 2 la 7
Phan tu thu 3 la 2
Phan tu thu 4 la 3
Phan tu thu 5 la 8
Phan tu thu 6 la 3

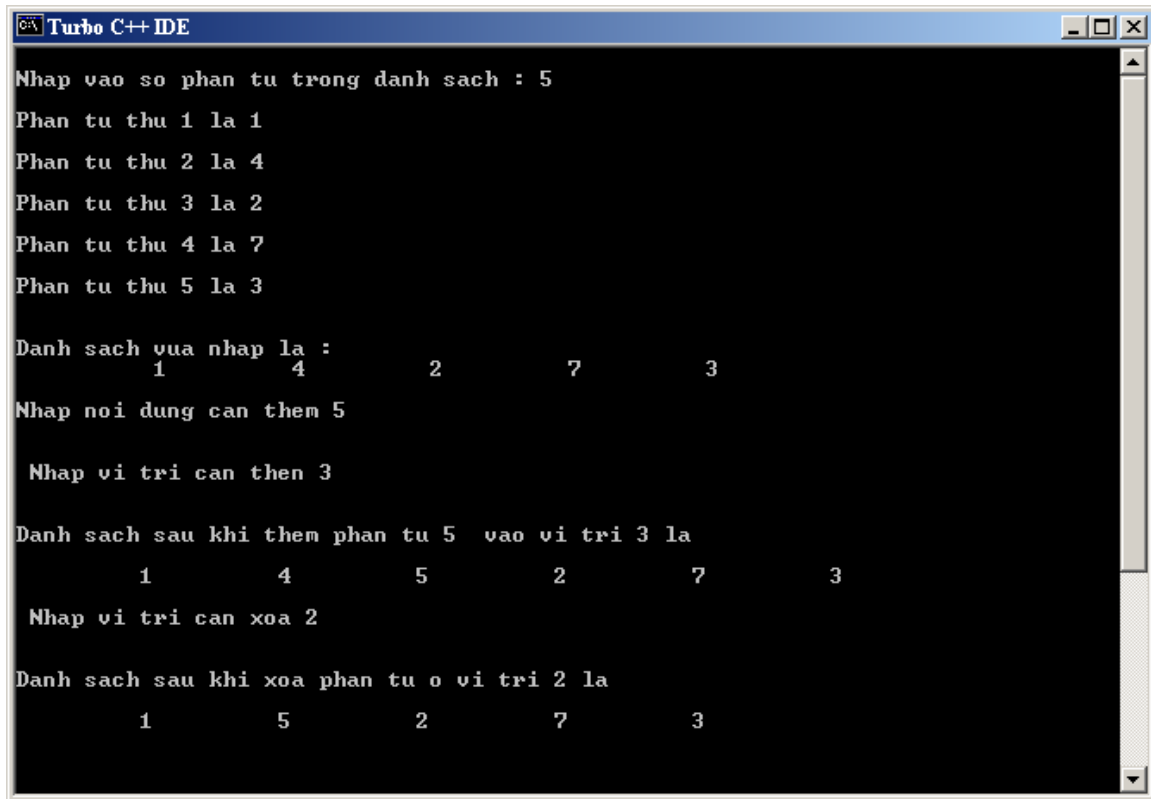
Danh sach vua nhap la :
          1          7          2          3          8          3
Nhap noi dung phan tu can tim 3

Vi tri phan tu dau tien co noi dung 3 trong danh sach la 4
```

Ví dụ 4: Viết chương trình **ALTEST4.CPP** để nhập một danh sách từ bàn phím. Sau đó thêm một phần tử vào vị trí cho trước. Kiểm tra tính đúng đắn của hàm xóa phần tử ra khỏi danh sách. Chương trình con này có thể được viết như sau:

```
void main()
```

```
{
    clrscr();
    List L;
    ElementType X;
    Position P;
    MakeNull_List(&L); //Khoi tao danh sach ddac rong
    Read_List(&L);      //Nhap danh sach tu ban phim
    printf("\n\nDanh sach vua nhap la :\n ");
    Print_List(L);
    printf("\n\nNhap noi dung can them ");
    scanf("%d",&X);
    printf ("\n\n Nhap vi tri can then ");
    scanf("%d", &P);
    Insert_List(X,P,&L);
    printf ("\n\nDanh sach sau khi them phan tu %d  vao vi tri
%d la \n\n",X,P);
    Print_List(L);
    printf("\n\n Nhap vi tri can xoa "); scanf("%d", &P);
    Delete_List(P,&L);
    printf("\n\nDanh sach sau khi xoa phan tu o vi tri %d la
\n\n",P);
    Print_List(L);
    getch();
}
```



```
Turbo C++ IDE
Nhap vao so phan tu trong danh sach : 5
Phan tu thu 1 la 1
Phan tu thu 2 la 4
Phan tu thu 3 la 2
Phan tu thu 4 la 7
Phan tu thu 5 la 3

Danh sach vua nhap la :
    1      4      2      7      3
Nhap noi dung can them 5

Nhap vi tri can them 3

Danh sach sau khi them phan tu 5 vao vi tri 3 la
    1      4      5      2      7      3
Nhap vi tri can xoa 2

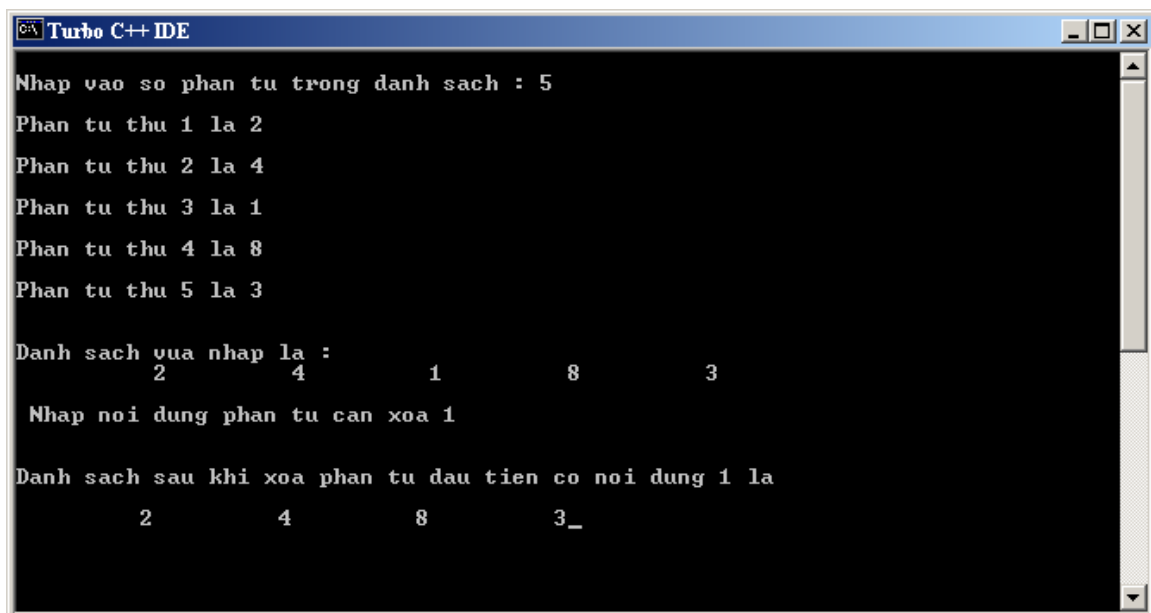
Danh sach sau khi xoa phan tu o vi tri 2 la
    1      5      2      7      3
```

Ví dụ 5: Chương trình **ALTEST5.CPP** để nhận vào một danh sách từ bàn phím. Sau đó xóa phần tử đầu tiên có nội dung X trong danh sách (với X nhập từ bàn phím) và hiển thị danh sách sau khi xóa. Nội dung chính của chương trình có thể được viết như sau:

```
void main()
{
    clrscr();
    List L;
    ElementType X;
    Position P;
    MakeNull_List(&L); //Khoi tao danh sach ddac rong
    Read_List(&L);      //Nhap danh sach tu ban phim
    printf("\n\nDanh sach vua nhap la :\n ");
    Print_List(L);
    printf ("\n\n Nhap noi dung phan tu can xoa ");
    scanf("%d", &X);
```

Thực hành cấu trúc dữ liệu

```
P=Locate(X,L);  
// Tìm vị trí phần tử có nội dung X cần xóa  
if (P== EndList(L))  
    printf("\n\n Không có phần tử %d trong dsach ",X);  
else    Delete_List(P,&L);  
printf ("\n\n Danh sách sau khi xóa phần tử đầu tiên có nội  
dung %d là \n\n",X);  
Print_List(L);  
getch();  
}
```



2. Danh sách liên kết đơn (cài đặt bằng con trỏ)

Yêu cầu: Tạo file với tên là **PLISTLIB.CPP** lưu trữ các phép toán cơ bản trên danh sách liên kết lưu trữ các số nguyên.

```
#include <stdio.h>  
#include <conio.h>  
#include <alloc.h>  
  
/* ==== Khai báo danh sách liên kết lưu trữ các số nguyên ==== */
```

Thực hành cấu trúc dữ liệu

```
typedef int ElementType;
typedef struct Node{
    ElementType Element;
    Node*      Next;
};
typedef Node*   Position;
typedef Position List;

/*== Chương trình con tạo danh sách rỗng ==*/
void MakeNull_List(List *Header){
    (*Header)      = (Node*)malloc(sizeof(Node));
    (*Header)->Next = NULL;
}

/*== Hàm kiểm tra danh sách rỗng. Hàm trả về giá trị 1 nếu danh sách rỗng, ngược lại hàm trả về giá trị 0 ==*/

int Empty_List(List L){
    return (L->Next == NULL);
}

/*== Chương trình con thêm phần tử có nội dung X vào vào vị trí P trong danh sách liên kết L ==*/
void Insert_List(ElementType X, Position P, List *L){
    Position T;
    T = (Node*)malloc(sizeof(Node));
    T->Element = X;
    T->Next    = P->Next;
    P->Next    = T;
}

/*== Chương trình con xóa phần tử tại vị trí P ra khỏi danh sách L ==*/
```



```
void Delete_List(Position P, List *L){
    Position T;
    if(P->Next != NULL){
        T->Next = P->Next;
        P->Next = P->Next->Next;
        free(T);
    }
    else printf("\nLoi !Danh sach rong khong the xoa");
}
```

/*== Hàm tìm vị trí phần tử đầu tiên có nội dung X trong danh sách. Nếu không có phần tử X thì hàm trả về EndList ==*/

```
Position Locate(ElementType X, List L){
    Position P;
    int found = 0;
    while ((P->Next != NULL) && (found ==0)){
        if(P->Next->Element == X)
            found=1;
        else P=P->Next;
    }
    return P;
}
```

/*== Hàm lấy nội dung phần tử tại vị trí P trong danh sách ==*/

```
ElementType Retrieve(Position P, List L){
    if(P->Next != NULL)
        return P->Next->Element;
}
```

```
Position First(List l)
{ return l; }
```

```
Position EndList(List l)
{ Position p;
  p=First(l);
  while(p->next!=NULL)
  { p=p->next;
  }
  return p;
}
```

Position

/*== Nhập danh sách L từ bàn phím ==*/

```
void Read_List(List *l)
{ int i,n,t;
  Elementtype x;
  Position p;
  p=first(*l);
  printf("So phan tu trong danh sach:");scanf("%d",&n);
  for(i=1;i<=n;i++)
  { printf("Phan tu thu %d:",i);scanf("%d",&x);
    insert_list(x,EndList(*l),l);
  }
}
```

/*== In danh sách ra màn hình ==*/

```
void Print_List(List L)
{ Position p;
  p=First(L);
  while(p!=EndList(L))
  { printf("%d",retrieve(p,L));
```

```
    p=p->next;
}
}
```

Yêu cầu: Viết các chương trình chính để kiểm tra tính đúng đắn của các phép toán cơ bản trên. (Ta có thể lấy các chương trình chính như các ví dụ trong cách cài đặt danh sách bằng mảng và chỉ cần đổi đường dẫn đến file chứa các phép toán cơ sở)

Ví dụ kiểm tra chương trình con tạo rỗng, hiển thị danh sách, thêm dữ liệu vào danh sách.

```
#include <stdio.h>
```

```
#include <conio.h>
```

```
#include <D:\PLISTLIB.CPP> /* Chỉ đúng đường dẫn và tên file thư viện  
vừa tạo lưu trữ các phép toán trên danh sách liên kết --- */
```

```
void main()
```

```
{
```

```
    clrscr();
```

```
    List L;
```

```
    MakeNull_List(&L); //Khởi tạo danh sách ddac rỗng
```

```
    /* Đoạn lệnh gọi để kiểm tra chương trình con tạo rỗng và  
    kiểm tra rỗng */
```

```
    printf ("\n\n Danh sách sau khi khởi tạo rỗng là ");
```

```
    if (Empty_List(L)) printf ("\n\n Danh sách rỗng ");
```

```
    else printf ("\n\nDanh sách không rỗng");
```

```
    /* Xen 5 số (từ 11 đến 15) vào danh sách theo đúng thứ tự  
    nhập */
```

```
    /* Đoạn lệnh kiểm tra chương trình con xen và hiển thị danh  
    sách */
```

```
    for (int i=1; i<=5; i++)
```

```
        Insert_List (i+10,EndList(L), &L);
```

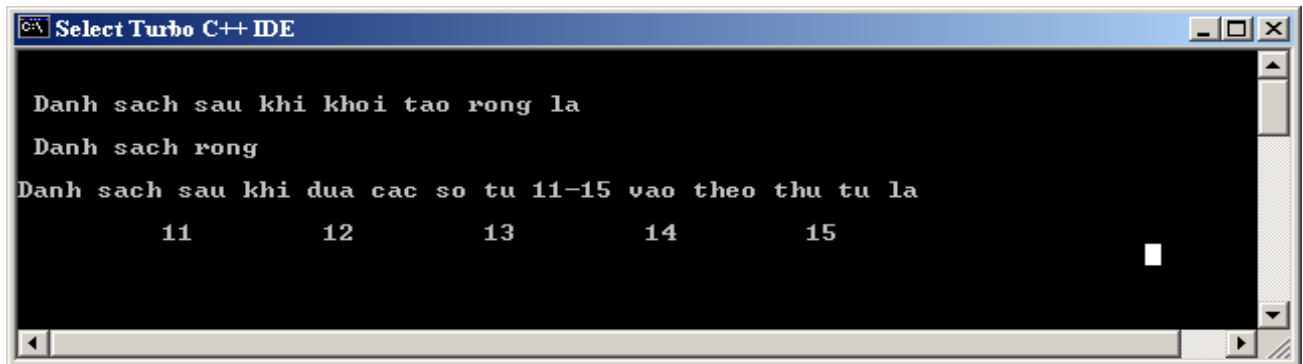
```
    printf ("\n\nDanh sách sau khi đưa các số từ 11-15 vào theo  
    thứ tự là \n\n ");
```

```
    Print_List(L);
```

```
    getch();
```

```
}
```

Kết quả trên màn hình khi thực thi chương trình như sau:



3. Danh sách liên kết kép (cài đặt bằng con trỏ hai chiều)

Yêu cầu: Tạo file với tên là **DLISTLIB.CPP** lưu trữ các phép toán cơ bản trên danh sách liên kết lưu trữ các số nguyên.

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
```

```
/* ==== Khai báo danh sách liên kết kép chứa các số nguyên ==== */
```

```
typedef int ElementType;
typedef struct Node{
    ElementType Element;
    Node*       Next;
    Node*       Prev;
};
typedef Node*   Position;
typedef Position List;
```

```
/*== Chương trình con tạo danh sách liên kết kép rỗng ==*/
```

```
void MakeNull_List(List *DL) {  
    *DL = NULL;  
}
```

/== **Hàm kiểm tra danh sách rỗng** == */*

```
int Empty_List(List DL) {  
    return (DL == NULL);  
}
```

/== **Chương trình con thêm phần tử có nội dung X vào tại vị trí P trong danh sách L** ==*/*

```
void Insert_List(ElementType X, Position P, List *DL)  
{  
    if(*DL == NULL)  
    { //Truong hop them vao DS rong  
        (*DL)=(Node*)malloc(sizeof(Node));  
        (*DL)->Element = X;  
        (*DL)->Prev = NULL;  
        (*DL)->Next = NULL;  
    }  
    else  
    {  
        if(P == EndList(*DL))  
        //Vi tri them cuoi cung trong danh sach  
        {  
            Position temp, R;  
            temp = (Node*)malloc(sizeof(Node));  
            temp->Element = X;  
            temp->Next = NULL;  
            R = *DL;  
            while (R->Next != NULL)
```

```
        R = R->Next;
    temp->Prev    = R;
    R->Next=temp;
}
else
{
    Position temp;
    temp = (Node*)malloc(sizeof(Node));
    temp->Element = X;
    temp->Next    = P;
    temp->Prev    = P->Prev;
    if(P->Prev != NULL)
// Kiem tra vi tri them co phai vi tri dau tien khong
        P->Prev->Next = temp;
    else
        (*DL) = temp;
    P->Prev = temp;
}
}
}

/*== Chương trình con xóa phần tử tại vị trí P ra khỏi danh sách L ==*/
void Delete_List(Position P,List *DL){
    if(*DL == NULL)
        printf("\nLoi !Danh sach rong khong the xoa");
    else
        if(P == *DL)
// Truong hop xoa phan tu dau tien trong DS kep
            (*DL) = (*DL)->Next;
        else
            { /*Truong hop phan tu can xoa khong phai phan tu dau
tien */
```

Thực hành cấu trúc dữ liệu

```
P->Prev->Next = P->Next;
if(P->Next != NULL)
    /*Kiem tra phan tu xoa co phai la phan tu cuoi khong
*/
    P->Next->Prev = P->Prev;
    free(P);
}
```

/*== Hàm lấy nội dung phần tử tại vị trí P trong danh sách L ==*/

```
ElementType Retrieve(Position P, List DL)
{
    return P->Element;
}
```

/*== Các hàm định vị trí trong danh sách liên kết kép ===*/

```
Position First(List DL)
{
    return DL;
}
```

```
Position EndList(List DL)
{
    return NULL;
}
```

```
Position Next(Position P, List DL)
{
    return P->Next;
}
```

```
Position Previous(Position P, List DL)
```

```
{  
    return P->Prev;  
}
```

Yêu cầu: Tương tự như các kiểu danh sách trước, học viên hãy viết các chương trình chính để kiểm tra tính đúng đắn của các phép toán cơ bản trên. (Ta có thể lấy các chương trình chính như các ví dụ trong cách cài đặt danh sách bằng mảng và chỉ cần đổi đường dẫn đến file chứa các phép toán cơ sở)

III. Bài tập nâng cao

Yêu cầu 1 : Cài đặt chương trình LIST1.CPP xác định số phần tử có nội dung x trong danh sách. Loại bỏ các phần tử trùng lặp:

Ý tưởng của giải thuật xác định số phần tử có nội dung x trong danh sách:

1. Khởi tạo biến đếm c=0.
2. Duyệt qua từng phần tử của danh sách.

Ứng với mỗi phần tử duyệt, nếu phát hiện phần tử nào có giá trị bằng x thì tăng biến đếm c lên 1 đơn vị.

Nội dung của hàm có thể được viết như sau:

```
int Quantity_X ( ElementType X, List L)  
{  
    Position P= FirstList(L);  
    int count=0;  
    while (P!= EndList(L))  
    { if (Retrieve (P,L)== X) count++;  
      P=Next (P,L);  
    }  
    return count;  
}
```

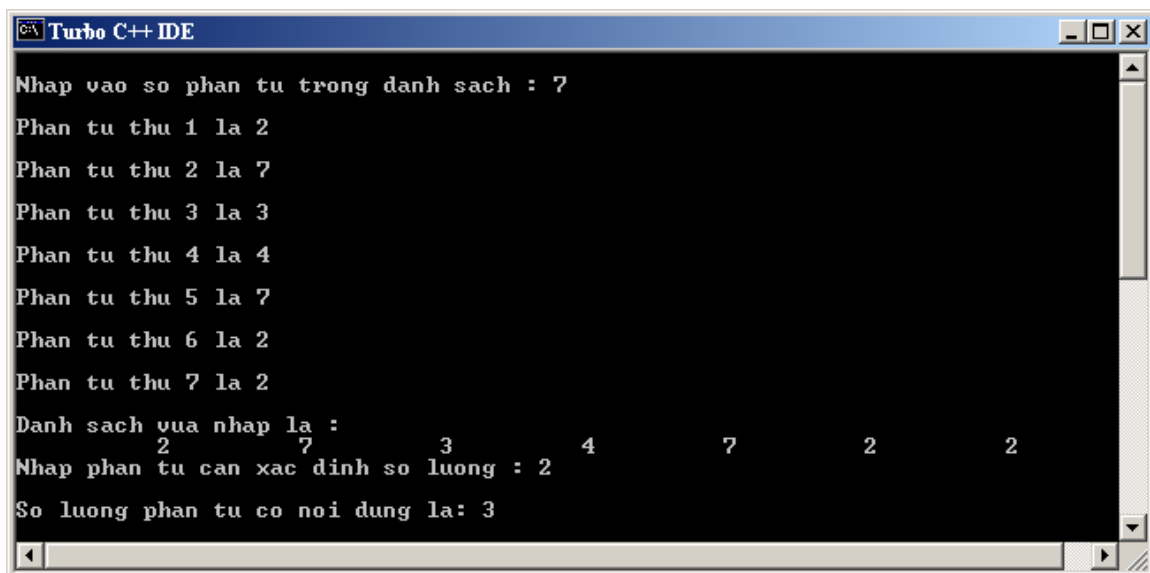
Chương trình chính có thể được thiết kế để kiểm tra tính đúng đắn của hàm như sau:

```
#include <stdio.h>  
#include <conio.h>
```



```
#include <D:\ALISTLIB.CPP> /* Co the chi duong dan den thu
vien chua file danh sach lien ket hay danh sach lien ket
kep*/

void main()
{
    clrscr();
    List L;
    ElementType X;
    Position P;
    MakeNull_List(&L); //Khoi tao danh sach ddac rong
    Read_List(&L);      //Nhan vao danh sach tu ban phim
    printf("\nDanh sach vua nhap la :\n ");
    Print_List(L);
    printf("\n\nNhap phan tu can xac dinh so luong : ");
    scanf("%d",&X);fflush(stdin);
    printf("\nSo luong phan tu co noi dung la: %d
\n",Quantity_X(X,L));
    getch()
}
```



The screenshot shows the Turbo C++ IDE window with the following output:

```
Nhap vao so phan tu trong danh sach : 7
Phan tu thu 1 la 2
Phan tu thu 2 la 7
Phan tu thu 3 la 3
Phan tu thu 4 la 4
Phan tu thu 5 la 7
Phan tu thu 6 la 2
Phan tu thu 7 la 2
Danh sach vua nhap la :
2       3       4       7       2       2
Nhap phan tu can xac dinh so luong : 2
So luong phan tu co noi dung la: 3
```

Ý tưởng của giải thuật loại bỏ các phần tử trùng lặp:

Duyệt qua từng phần tử ở vị trí p trong L

Ứng với mỗi phần tử đang xét ở vị trí p, ta duyệt qua các phần tử q từ vị trí ngay sau p.

Nếu phát hiện phần tử ở vị trí q có nội dung bằng phần tử ở vị trí p thì xóa phần tử ở vị trí q đi. Ngược lại, xét vị trí q kế tiếp. Sau khi đã xét hết tất cả các phần tử thì xét vị trí p kế tiếp. Quá trình sẽ kết thúc khi đã duyệt qua hết các phần tử theo vị trí p trong danh sách.

Chương trình con có thể thiết kế như sau:

```
void Delete_Same( List *L)
{
    Position p,q;
    p=FirstList(*L);
    while (p!=EndList(*L))
    { q=Next(p,*L);
        while (q!=EndList(*L))
            if (Retrieve(p,*L)==Retrieve(q,*L))
                Delete_List(q,L);
            else q=Next(q,*L);
        p=Next(p,*L);
    }
}
```

Chương trình chính như sau:

```
#include <stdio.h>
#include <conio.h>
#include <D:\ALISTLIB.CPP> /* Co the chi duong dan den thu
vien chua file danh sach lien ket hay dnah sach lien ket
kep*/
void main()
{
    clrscr();
    List L,L_Chan,L_Le;
    ElementType X;
    Position P;
```

```
MakeNull_List(&L); //Khoi tao danh sach ddac rong
Read_List(&L);      //Nhan vao danh sach tu ban phim
printf("\nDanh sach vua nhap la :\n ");
Print_List(L);
printf("\n\n Danh sach sau khi xa cac pha tu trung lap la
\n\n");
Delete_Same(&L);
Print_List(L);
getch()
}
```

Kết quả thực hiện:

```
Turbo C++ IDE
Nhap vao so phan tu trong danh sach : 8
Phan tu thu 1 la 2
Phan tu thu 2 la 4
Phan tu thu 3 la 2
Phan tu thu 4 la 2
Phan tu thu 5 la 6
Phan tu thu 6 la 2
Phan tu thu 7 la 6
Phan tu thu 8 la 4
Danh sach vua nhap la :      2      4      2      2      6      2      6
4
Danh sach sau khi xa cac pha tu trung lap la
      2      4      6
```

Yêu cầu 2: Cài đặt chương trình LIST3.CPP để sắp xếp danh sách theo thứ tự tăng. Thêm một phần tử vào danh sách có thứ tự tăng để được một danh sách mới vẫn có thứ tự tăng.

Ý tưởng sắp xếp danh sách theo thứ tự tăng

- Thiết kế hàm hoán đổi nội dung hai phần tử trong danh sách. (Hàm này tùy thuộc vào cách cài đặt danh sách mà ta thiết kế nó khác nhau vì phải truy xuất trực tiếp vào bên trong của cấu trúc)

Ví dụ: Hàm hoán chuyển nội dung trong danh sách cài đặt bằng mảng có thể thiết kế như sau:

```
void swap(Position a, Position b, List *L){
    ElementType temp;
    temp = L->Elements[a-1];
    L->Elements[a-1] = L->Elements[b-1];
    L->Elements[b-1] = temp;
}
```

- Thiết kế giải thuật sắp xếp chính (Có thể dùng chung cho tất cả các kiểu danh sách.)

```
void Sort_List(List *L)
{
    Position P=FirstList(*L);
    while(P!=EndList(*L)) {
        Position Q=Next(P,*L);
        while(Q!=EndList(*L))
        {
            if((Retrieve(P,L)) > (Retrieve(Q,L)))
                //sap xep tang dan
                swap(P,Q,L);
            Q = Next(Q,*L);
        }
        P=Next(P,*L);
    }
}
```

Ý tưởng giải thuật thêm phần tử vào danh sách có thứ tự:

- Ta tìm vị trí hợp lệ để có thể xen X vào danh sách. Bắt đầu từ phần tử đầu tiên trong danh sách, so nội dung phần tử đang xét với nội dung phần tử x cần xen. Nếu nội dung phần tử $x < \text{nội dung phần tử đang xét}$ thì ta dừng lại và trả hàm về ngay vị trí đang đứng. Ngược lại, xét phần tử kế tiếp cho đến hết danh sách thì kết thúc.
- Gọi hàm xen X vào vị trí vừa tìm được.

```
Position OrderLocate(ElementType X, List L){
    Position P;
```

Thực hành cấu trúc dữ liệu

```
int found = 0;
P = FirstList(L);
while ((P!=EndList(L)) && (Retrieve(P,L)<X) &&
(found==0)) {
    if(Retrieve(P,L) == X)
    {
        found = 1;
        break;
    }
    else P = Next(P, L);
}
return P;
}
```

```
void Insert_OrderList(ElementType X,List *L){
    Position P = OrderLocate(X,*L);
    Insert_List(X,P,L);
}
```

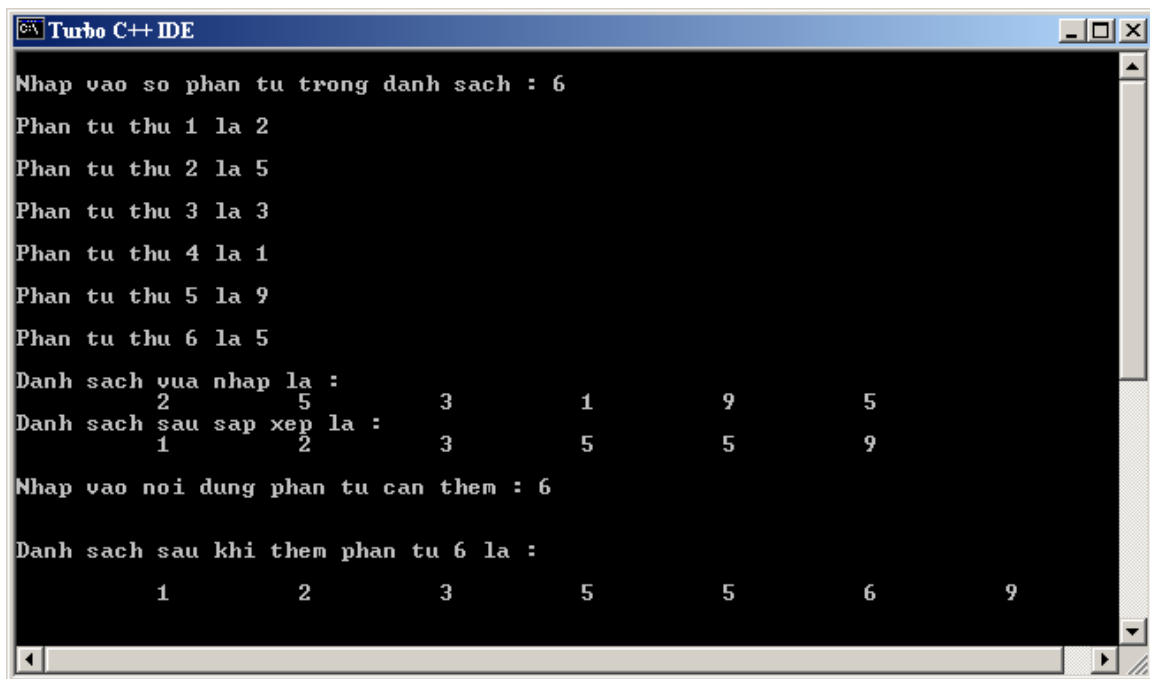
Chương trình chính có thể được thiết kế như sau để kiểm tra hai chương trình con vừa viết.

```
#include <stdio.h>
#include <conio.h>
#include <D:\ALISTLIB.CPP> /* Co the chi duong dan den cac
thu vien cua danh sach lien ket hay lien ket kep*/
void main()
{
    clrscr();
    List L;
    ElementType X;
    Position P;
    MakeNull_List(&L); //Khoi tao danh sach ddac rong
    Read_List(&L);      //Nhan vao danh sach tu ban phim
```

Thực hành cấu trúc dữ liệu

```
printf("\nDanh sach vua nhap la :\n ");
Print_List(L);
Sort_List(&L);
printf("\nDanh sach sau sap xep la :\n ");
Print_List(L);
printf("\n\nNhap vao noi dung phan tu can them : ");
scanf("%d",&X);fflush(stdin);
Insert_OrderList(X,&L);
//Them phan tu vao dung vi tri trong danh sach
printf("\nDanh sach sau khi them phan tu %d la :\n",X);
Print_List(L);
getch();
}
```

Kết quả thực thi chương trình như sau:



```
Turbo C++ IDE
Nhap vao so phan tu trong danh sach : 6
Phan tu thu 1 la 2
Phan tu thu 2 la 5
Phan tu thu 3 la 3
Phan tu thu 4 la 1
Phan tu thu 5 la 9
Phan tu thu 6 la 5
Danh sach vua nhap la :
2      5      3      1      9      5
Danh sach sau sap xep la :
1      2      3      5      5      9
Nhap vao noi dung phan tu can them : 6
Danh sach sau khi them phan tu 6 la :
1      2      3      5      5      6      9
```

Yêu cầu 3: Viết chương trình xác định nội dung phần tử lớn nhất và nhỏ nhất trong danh sách.

Ý tưởng:

Khởi tạo biến chạy P bắt đầu bằng phần tử đầu tiên và biến lưu giá trị Max (Min) tạm thời bằng nội dung phần tử đầu tiên đó. Vòng lặp xét qua hết tất cả các phần tử trong danh sách, so sánh nội dung phần tử đang xét với giá trị Max tạm (Min tạm) và đổi giá trị nếu Max tạm < nội dung phần tử đang xét. Sau khi xét qua hết tất cả các phần tử thì trả hàm về giá trị Max tạm này.

```
ElementType Max_Value(List L)
{
    Position P= FirstList (L);
    ElementType Max_temp= Retrieve(P,L);
    while (P!=EndList(L))
    { if (Max_temp<Retrieve(P,L)) Max_temp= Retrieve(P,L);
      P=Next(P,L);
    }
    return Max_temp;
}

ElementType Min_Value(List L)
{
    Position P= FirstList (L);
    ElementType Min_temp= Retrieve(P,L);
    while (P!=EndList(L))
    { if (Min_temp>Retrieve(P,L)) Min_temp= Retrieve(P,L);
      P=Next(P,L);
    }
    return Min_temp;
}
```

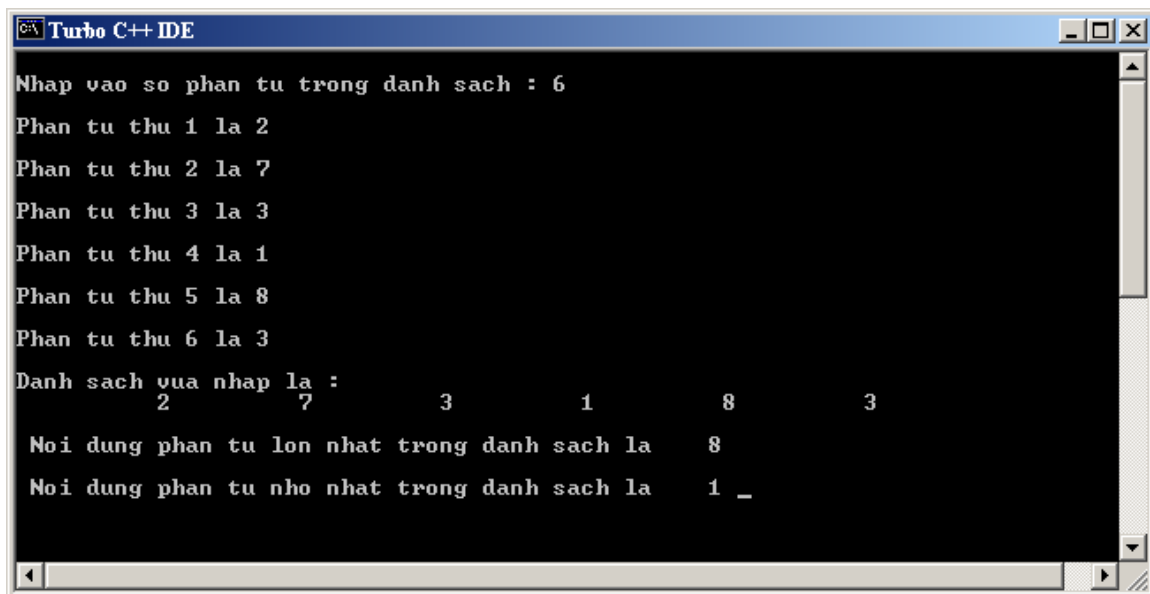
Chương trình chính như sau:

```
#include <stdio.h>
#include <conio.h>
#include <D:\ALISTLIB.CPP> /* Co the chi duong dan den cac
thu vien cua danh sach lien ket hay lien ket kep*/
void main()
{
```

Thực hành cấu trúc dữ liệu

```
clrscr();
List L,L_Chan,L_Le;
ElementType X;
Position P;
MakeNull_List(&L); //Khoi tao danh sach ddac rong
Read_List(&L);      //Nhan vao danh sach tu ban phim
printf("\nDanh sach vua nhap la :\n ");
Print_List(L);
printf("\n\n Noi dung phan tu lon nhat trong danh sach la
%d ",Max_Value(L));
printf("\n\n Noi dung phan tu nho nhat trong danh sach la
%d ",Min_Value(L));
getch()
}
```

Kết quả thực hiện:



```
Turbo C++ IDE
Nhap vao so phan tu trong danh sach : 6
Phan tu thu 1 la 2
Phan tu thu 2 la 7
Phan tu thu 3 la 3
Phan tu thu 4 la 1
Phan tu thu 5 la 8
Phan tu thu 6 la 3
Danh sach vua nhap la :
      2      7      3      1      8      3
Noi dung phan tu lon nhat trong danh sach la      8
Noi dung phan tu nho nhat trong danh sach la      1 _
```

Yêu cầu 4: Viết lại các chương trình con đếm số phần tử có nội dung X trong danh sách đã có thứ tự.

Ý tưởng: (Do danh sách đã có thứ tự nên các phần tử giống nhau sẽ nằm kế cạnh bên nhau)

- Khởi tạo P = vị trí phần tử đầu tiên có nội dung X trong danh sách. Biến đếm bằng 0

- Trong khi nội dung phần tử tại vị trí P vẫn còn bằng X thì tăng biến đếm lên 1. Xét phần tử kế tiếp phần tử P.
- Hàm trả về giá trị của biến đếm.

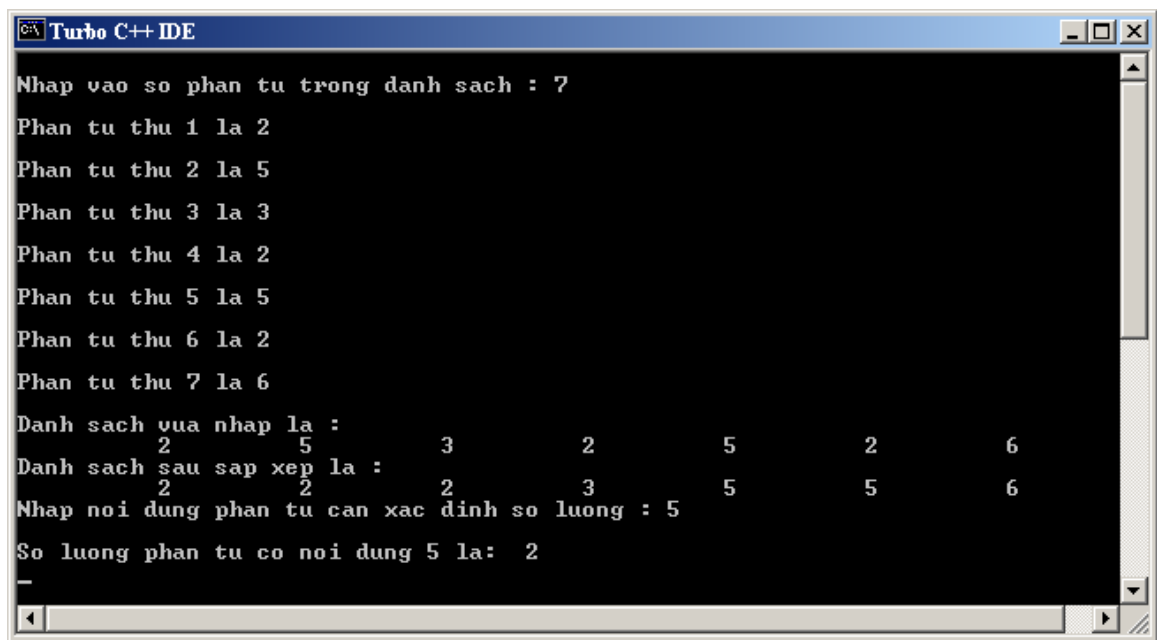
```
int Quantity_X_Order(ElementType X, List L){
    int cnt=0;
    ElementType temp;
    Position P=OrderLocate(X,L);
    if(P!=EndList(L)){
        while(Retrieve(P,L) == X ){
            P = Next(P,L);
            ++cnt;
        }
    }
    return cnt;
}
```

Chương trình chính để kiểm tra hàm có thể được viết như sau:

```
#include <stdio.h>
#include <conio.h>
#include <D:\ALISTLIB.CPP> /* Co the chi duong dan den cac
thu vien cua danh sach lien ket hay lien ket kep*/
void main()
{
    clrscr();
    List L;
    ElementType X;
    Position P;
    MakeNull_List(&L); //Khoi tao danh sach ddac rong
    Read_List(&L);      //Nhan vao danh sach tu ban phim
    printf("\nDanh sach vua nhap la :\n ");
    Print_List(L);
    Sort_List(&L);
```

Thực hành cấu trúc dữ liệu

```
printf("\nDanh sach sau sap xep la :\n ");
Print_List(L);
printf("\nNhap noi dung phan tu can xac dinh so luong : ");
scanf("%d",&X);fflush(stdin);
printf("\nSo luong phan tu co noi dung %d la:  %d \n",X,
Quantity_X_Order(X,L));
getch();
}
```



The screenshot shows the Turbo C++ IDE window with the following output:

```
Nhap vao so phan tu trong danh sach : 7
Phan tu thu 1 la 2
Phan tu thu 2 la 5
Phan tu thu 3 la 3
Phan tu thu 4 la 2
Phan tu thu 5 la 5
Phan tu thu 6 la 2
Phan tu thu 7 la 6
Danh sach vua nhap la :
2      5      3      2      5      2      6
Danh sach sau sap xep la :
2      2      2      3      5      5      6
Nhap noi dung phan tu can xac dinh so luong : 5
So luong phan tu co noi dung 5 la: 2
-
```

Yêu cầu 5: Tách danh sách ra thành hai danh sách một chứa số nguyên chẵn và một chứa các số nguyên lẻ.

Ý tưởng:

Duyệt qua tất cả các phần tử trong danh sách ban đầu và lấy nội dung ra kiểm tra. Nếu nó là giá trị chẵn ($\text{Retrieve}(P,L)/2 == 0$) thì xen nó vào danh sách chẵn, còn nếu nó là số lẻ thì xen nó vào danh sách chứa số lẻ.

```
void Split_List (List L1, List *L2, List *L3)
{
/* Tach danh sach L1 thanh hai danh sach chan L2 va le L3*/
```

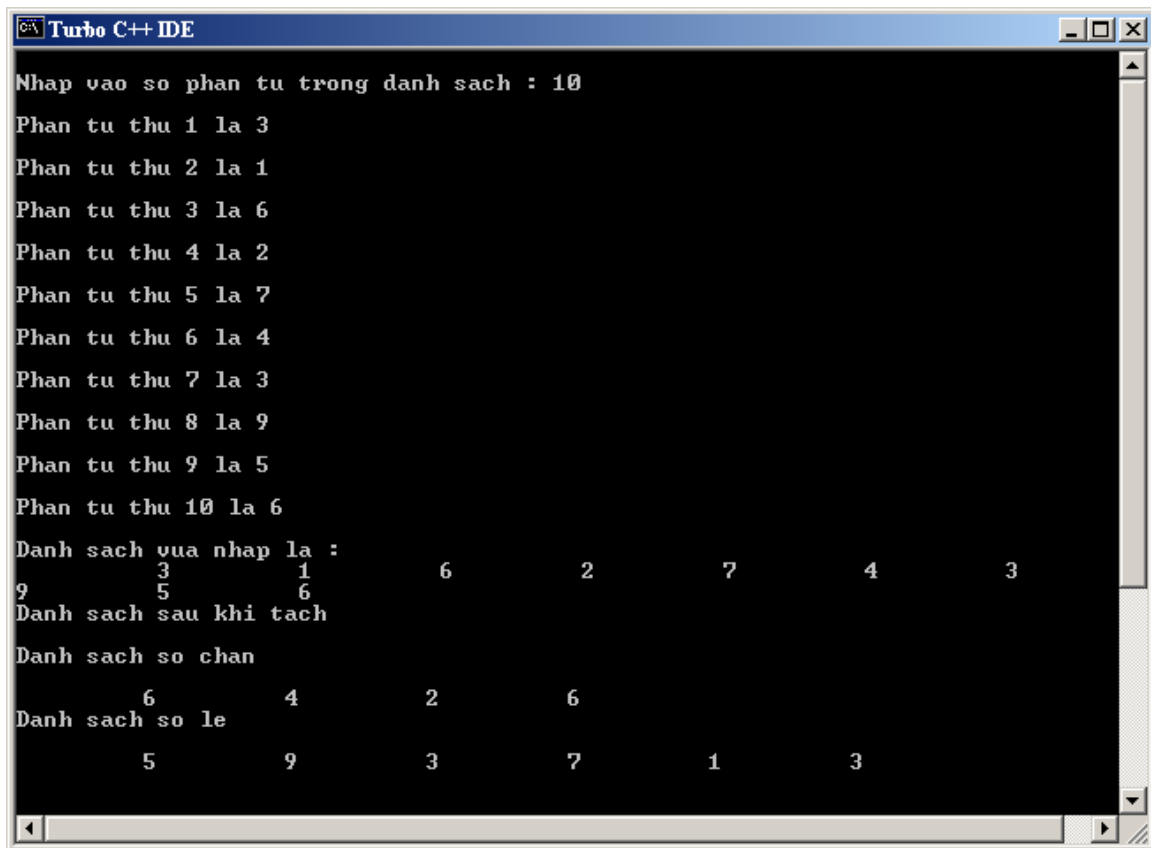
Thực hành cấu trúc dữ liệu

```
Position P=FirstList(L1);
MakeNull_List(L2);
MakeNull_List(L3);
while (P!=EndList(L1))
{if (Retrieve(P,L1) %2 ==0) // So chan
    Insert_List(Retrieve(P,L1),FirstList(*L2),L2);
else
    Insert_List(Retrieve(P,L1),FirstList(*L3),L3);
P=Next(P,L1);
}
}
```

Chương trình chính như sau:

```
void main()
{
    clrscr();
    List L,L_Chan,L_Le;
    ElementType X;
    Position P;
    MakeNull_List(&L); //Khoi tao danh sach ddac rong
    Read_List(&L);      //Nhan vao danh sach tu ban phim
    printf("\nDanh sach vua nhap la :\n ");
    Print_List(L);
    Split_List(L,&L_Chan,&L_Le);
    printf("\nDanh sach sau khi tach\n ");
    printf("\nDanh sach so chan\n\n");
    Print_List(L_Chan);
    printf("\nDanh sach so le\n\n");
    Print_List(L_Le);
    getch()
}
```

Kết quả chương trình như sau:



```
Turbo C++ IDE
Nhap vao so phan tu trong danh sach : 10
Phan tu thu 1 la 3
Phan tu thu 2 la 1
Phan tu thu 3 la 6
Phan tu thu 4 la 2
Phan tu thu 5 la 7
Phan tu thu 6 la 4
Phan tu thu 7 la 3
Phan tu thu 8 la 9
Phan tu thu 9 la 5
Phan tu thu 10 la 6
Danh sach vua nhap la :
          3      1      6      2      7      4      3
          5      6
Danh sach sau khi tach
Danh sach so chan
          6      4      2      6
Danh sach so le
          5      9      3      7      1      3
```

Yêu cầu 6: Viết chương trình con trộn hai danh sách đã có thứ tự tăng để được một danh sách mới vẫn có thứ tự tăng.

Ý tưởng:

Giải thuật:

Tạo rỗng danh sách L3

Đặt p chỉ đến phần tử đầu của danh sách L1.

Đặt q chỉ đến phần tử đầu trong danh sách L2.

Đặt t chỉ đến phần tử cuối trong danh sách L3 (Do danh sách L3 rỗng nên t=13)

while (p!= Endlist(L1)) and (q!= Endlist(L2))

{

 If retrieve(p)< retrieve(q) then

 {

 Xen nội dung p vào cuối danh sách L3.

 Xét phần tử kế tiếp phần tử p trong danh sách L1.

 Đặt t chỉ đến phần tử cuối trong danh sách L3.

 }

Ngược lại

{

 Xen nội dung q vào cuối danh sách L3.

```
        Xét phần tử kế tiếp phần tử q trong danh sách L2.  
        Đặt t chỉ đến phần tử cuối trong danh sách L3.  
    }  
}  
    Xen các phần tử còn lại của danh sách chưa kết thúc vào danh sách kết quả L3.  
}
```

```
void Merge( List L1, List L2, List *L3)  
{  
    Position P1,P2,Q1,Q2,Q3;  
    MakeNull_List(L3);  
    P1=FirstList(L1);  
    P2=FirstList(L2);  
    Q1=EndList(L1);  
    Q2=EndList(L2);  
    Q3=EndList(*L3);  
    while ((P1!=Q1) && (P2!=Q2))  
    { if (Retrieve(P1,L1)<Retrieve(P2,L2))  
        { Insert_List (Retrieve(P1,L1),Q3,L3);  
          P1=Next (P1,L1);  
        }  
      else  
        { Insert_List (Retrieve(P2,L2),Q3,L3);  
          P2= Next (P2,L2);  
        }  
      Q3=Next (Q3,*L3);  
    }  
    while (P1!=Q1)  
    { Insert_List (Retrieve(P1,L1),Q3,L3);  
      P1=Next (P1,L1);  
      Q3=Next (Q3,*L3);  
    }  
}
```

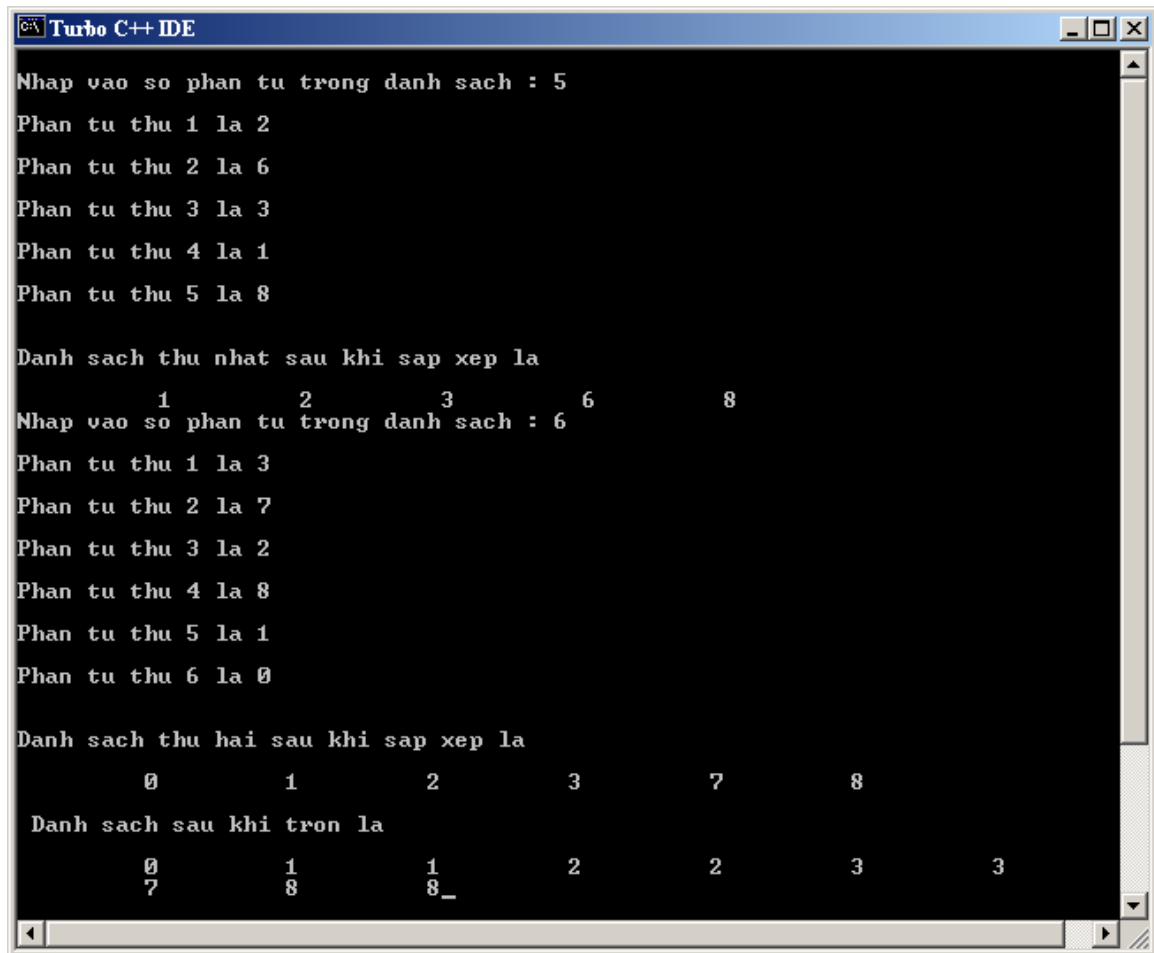
Thực hành cấu trúc dữ liệu

```
while (P2!=Q2)
    { Insert_List (Retrieve(P2,L2),Q3,L3);
      P2= Next (P2,L2);
      Q3=Next(Q3,*L3);
    }
}
```

Chương trình chính có thể thiết kế như sau:

```
void main()
{
    clrscr();
    List L,L1,L2;
    ElementType X;
    Position P;
    MakeNull_List(&L);
    Read_List(&L);
    Sort_List(&L);
    printf("\n\nDanh sach thu nhat sau khi sap xep la\n\n ");
    Print_List(L);
    MakeNull_List(&L1);
    Read_List(&L1);
    Sort_List(&L1);
    printf("\n\nDanh sach thu hai sau khi sap xep la\n\n");
    Print_List(L1);
    Merge(L,L1,&L2);
    printf ("\n\n Danh sach sau khi tron la \n\n");
    Print_List(L2);
    getch();
}
```

Kết quả thực hiện chương trình:



```
Turbo C++ IDE
Nhap vao so phan tu trong danh sach : 5
Phan tu thu 1 la 2
Phan tu thu 2 la 6
Phan tu thu 3 la 3
Phan tu thu 4 la 1
Phan tu thu 5 la 8

Danh sach thu nhat sau khi sap xep la
      1      2      3      6      8
Nhap vao so phan tu trong danh sach : 6
Phan tu thu 1 la 3
Phan tu thu 2 la 7
Phan tu thu 3 la 2
Phan tu thu 4 la 8
Phan tu thu 5 la 1
Phan tu thu 6 la 0

Danh sach thu hai sau khi sap xep la
      0      1      2      3      7      8
Danh sach sau khi tron la
      0      1      1      2      2      3      3
      7      8      8_
```

B. NGĂN XẾP

I. Khái niệm

A. GIỚI THIỆU LÝ THUYẾT

Ngăn xếp là một danh sách đặc biệt mà việc thêm phần tử hay xóa phần tử ra khỏi danh sách chỉ thực hiện tại một đầu, gọi là đỉnh của ngăn xếp.

Các phép toán trên ngăn xếp:

MAKENULL_STACK(S). Tạo ngăn xếp S rỗng.

TOP(S). Trả về phần tử trên đỉnh ngăn xếp S.

POP(S). Xóa phần tử trên đỉnh ngăn xếp. Đôi khi POP được dùng như 1 hàm trả về phần vừa bị lấy ra, tuy nhiên ta không dùng như thế ở đây.

PUSH(x, S). Chèn phần tử x lên đỉnh ngăn xếp S.

EMPTY_STACK(S). Trả về true nếu S là một ngăn xếp rỗng, trả về false trong trường hợp ngược lại.

FULL_STACK(S). Trả về true nếu ngăn xếp S đã đầy, trả về false trong trường hợp ngược lại. Chú ý phép toán này chỉ được cài đặt cho ngăn xếp được cài đặt bằng mảng.

II. Bài tập cơ sở

1. Cài đặt bằng danh sách

Tạo một file tên L_STKLIB.CPP để lưu trữ các phép toán cơ bản trên ngăn xếp với nội dung chính trong file như sau:

```
#include <D:\ALISTLIB.CPP> /* Chỉ duong dan den file chua
phép toán cơ bản trên danh sách */
/*==== Khai báo ngăn xếp cài đặt bằng danh sách ====*/
typedef List Stack;
/*==== Tạo ngăn xếp rỗng ====*/
void MakeNull_Stack(Stack *S)
{
    MakeNull_List(S);
}
/*==== Hàm kiểm tra ngăn xếp rỗng ====*/
int Empty_Stack(Stack S)
{
    return Empty_List(S);
}
/*==== Hàm kiểm tra ngăn xếp đầy ====*/
int Full_Stack(Stack S)
{
    return Full_List(S);
}
/*==== Hàm lấy nội dung phần tử tại đỉnh của ngăn xếp ====*/
ElementType Top(Stack S)
{
    return Retrieve(FirstList(S), S);
}
/*==== Thêm phần tử vào ngăn xếp ====*/
void Push(ElementType X, Stack *S)
{
    Insert_List(X, FirstList(*S), S);
}
```



```
/*=== Xóa phần tử ra khỏi ngăn xếp===*/  
void Pop(Stack *S)  
{  
    Delete_List(FirstList(*S), S);  
}
```

Yêu cầu: Sinh viên tự thiết kế các chương trình chính để kiểm tra các phép toán cơ bản của ngăn xếp.

2. Cài đặt bằng mảng

```
/*= Khai báo =*/  
#include <stdio.h>  
#define MaxLength 100  
typedef char ElementType;  
//typedef float ElementType;  
typedef struct  
{  
    ElementType Elements[MaxLength];  
    int Top_idx;  
}Stack;  
/*=== Tạo ngăn xếp rỗng ===*/  
void MakeNull_Stack(Stack *S)  
{  
    S->Top_idx = MaxLength;  
}  
/*=== Kiểm tra ngăn xếp rỗng ===*/  
int Empty_Stack(Stack S)  
{  
    return (S.Top_idx == MaxLength);  
}  
/*=== Kiểm tra ngăn xếp đầy ===*/  
int Full_Stack(Stack S)  
{  
    return (S.Top_idx == 0);
```

```
}
/*=== Lấy nội dung phần tử tại vị trí đỉnh của ngăn xếp ===*/
ElementType Top(Stack S)
{
    if(!Empty_Stack(S))
        return S.Elements[S.Top_idx];
    else{
        printf("\nLoi ! Stack rong");
        return NULL;
    }
}

/*=== Thêm phần tử vào ngăn xếp ===*/
void Push(ElementType X, Stack *S)
{
    if(Full_Stack(*S))
        printf("\nLoi ! Stack day khong the them");
    else{
        S->Top_idx = (S->Top_idx - 1);
        /* Giam Top 1 don vi (Cho Top chi dden phan tu ke)*/
        S->Elements[S->Top_idx] = X;
        /* Bo phan tu moi voi dau ngan xep*/
    }
}

/*=== Xóa phần tử ra khỏi ngăn xếp ===*/
void Pop(Stack *S)
{
    if(Empty_Stack(*S))
        printf("\nLoi ! Stack rong");
    else{
        S->Top_idx = (S->Top_idx + 1);
        /* Tang Top 1 don vi (Cho Top chi lu`i xuong phan tu sau)*/
    }
}
```

```
}  
}
```

3. Cài đặt bằng con trỏ (Xem như bài tập- Sinh viên tự tham khảo tài liệu)

III. Bài tập nâng cao

Viết chương trình chính để nhập vào một số thập phân và chuyển số thập phân đó thành số nhị phân bằng cách sử dụng cấu trúc ngăn xếp.

Ý tưởng:

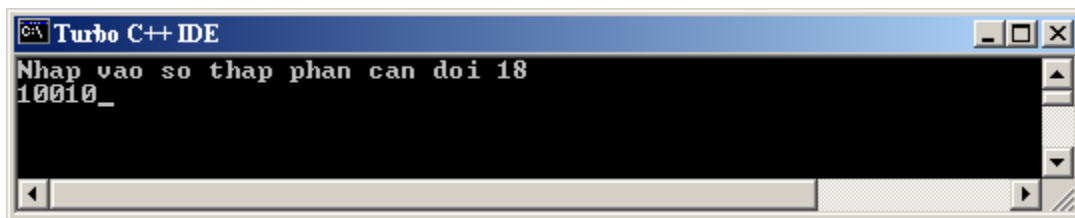
- Khởi tạo ngăn xếp rỗng
- While $n \neq 0$
 - { Lấy phần dư khi chia n cho 2 đưa vào ngăn xếp.
 - $n = \text{phần nguyên khi chia } n \text{ cho } 2$
- while ngăn xếp chưa rỗng
 - { Hiển thị nội dung phần tử tại đỉnh của ngăn xếp
 - Xóa phần tử tại đỉnh}

Chương trình chính có thể thực hiện như sau:

```
#include <D:\L_STKLIB.CPP> /* Chi duong dan den file chua  
phep toan co ban tren ngan xep*/  
  
void main()  
{  
    int n,m;  
    Stack S;  
    clrscr();  
    printf("Nhap vao so thap phan can doi ");  
    scanf("%d",&n);  
    m=n;  
    if (n==0) printf("\n\n so nhi phan la 0");  
    else  
    { MakeNull_Stack(&S);  
      // Day cac so du vao ngan xep  
      while (m!=0)
```

Thực hành cấu trúc dữ liệu

```
{ Push(m%2, &S);  
  m=m/2;  
}  
// Hien thi du lieu trong ngan xep ra man hinh  
while (!Empty_Stack(S))  
  { printf("%ld", Top(S));  
    Pop(&S);  
  }  
}  
getch();  
}
```



Yêu cầu 2: Viết chương trình kiểm tra một chuỗi dấu ngoặc đúng.

Ý tưởng:

- Nhập vào chuỗi dấu ngoặc mở và đóng.
- Khởi tạo ngăn xếp rỗng và biến finish = 0
- While (chuỗi chưa kết thúc) và not finish

{ If ký tự đang đọc là dấu ngoặc mở thì đưa nó vào ngăn xếp

Else ký tự là dấu ngoặc đóng thì xóa phần tử ra khỏi ngăn xếp. (Nếu ngăn xếp trước khi xóa rỗng thì chuỗi ngoặc không hợp lệ do thiếu dấu ngoặc mở. Khi đó đặt lại biến finish = 1 để thoát khỏi vòng lặp ngay)

}

Nếu chuỗi kết thúc mà ngăn xếp rỗng thì ta có chuỗi ngoặc đúng

Ngược lại, nếu chuỗi kết thúc mà ngăn xếp vẫn còn phần tử bên trong thì chuỗi ngoặc sai do thiếu dấu ngoặc đóng.

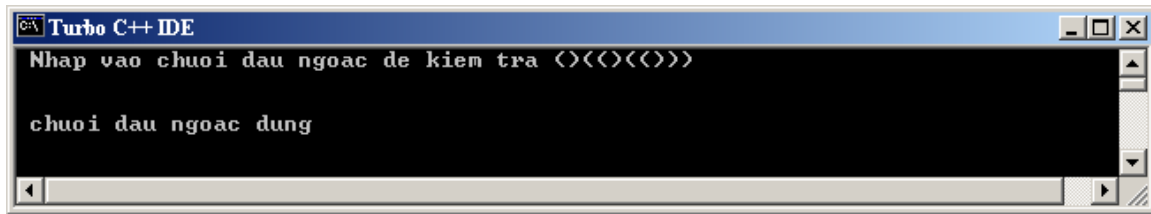
Chương trình có thể thực hiện như sau:

```
#include <D:\l_stklib.cpp> /* chỉ đường dẫn đến file các phép  
toán cơ bản trên ngăn xếp */  
#include <stdio.h>
```

Thực hành cấu trúc dữ liệu

```
#include <conio.h>
#include <string.h>
void main()
{
    char *chuoi;
    Stack S;
    MakeNull_Stack(&S);
    clrscr();
    printf(" Nhập vào chuỗi dấu ngoặc để kiểm tra ");
    fflush(stdin);
    scanf("%s",chuoi);
    fflush(stdin);
    int i=0;
    int finish =0;
    while ((i<strlen(chuoi))&& (!finish))
    { if (chuoi[i]=='(') Push(1,&S);
      else
        if (!Empty_Stack (S)) Pop(&S);
        else finish=1;
      i++;
    }
    if (finish)
        printf ("\n\n Chuỗi dấu ngoặc thiếu dấu ngoặc mở ");
    else
        if (Empty_Stack(S))
            printf ("\n\n chuỗi dấu ngoặc đúng ");
        else
            printf("\n\n Chuỗi dấu ngoặc thiếu dấu ngoặc đóng ");
    getch();
}
```

Kết quả khi thực thi chương trình:



C. HÀNG ĐỢI

I. Khái niệm

Cấu trúc hàng là một danh sách đặc biệt, việc thêm phần tử được thực hiện ở một đầu (gọi là đuôi của hàng) và việc xóa phần tử được thực hiện ở đầu còn lại (gọi là đầu hàng).

Các phép toán trên hàng đợi:

MAKENULL_QUEUE(Q). Tạo hàng Q rỗng.

EMPTY_QUEUE(Q). Trả về true nếu Q là một hàng đợi rỗng, trả về false trong trường hợp ngược lại.

FRONT(Q). Trả về phần tử đầu tiên của hàng Q.

ENQUEUE(x, Q). Xen x vào hàng Q, chú ý rằng phép thêm vào được thực hiện ở cuối hàng.

DEQUEUE(Q). Xóa phần tử tại đầu hàng Q.

FULL_QUEUE(Q). Trả về true nếu hàng Q đã đầy, trả về false trong trường hợp ngược lại. Chú ý phép toán này chỉ được cài đặt cho hàng đợi cài đặt bằng mảng.

II. Bài tập cơ sở

1. Cài đặt bằng mảng theo phương pháp tĩnh tiến

Tạo file tên **A_QUELIB.CPP** chứa các phép toán cơ bản trên cấu trúc hàng như sau:

```
//==Khai báo hàng ==  
#include<stdio.h>  
#include<conio.h>  
#define MaxLength 10  
typedef int ElementType;  
typedef struct  
{  
    ElementType Element[MaxLength];
```

```
    int Front, Rear;
} Queue;
//== Tạo hàng rỗng
void MakeNull_Queue(Queue *Q)
{
    Q->Front=-1;
    Q->Rear=-1;
}
//Kiểm tra hàng rỗng
int Empty_Queue(Queue Q)
{
    return Q.Front==-1;
}
//Kiểm tra hàng đầy
int Full_Queue(Queue Q)
{
    return (Q.Rear-Q.Front+1)==MaxLength;
}
// Thêm phần tử vào hàng
void EnQueue(ElementType X, Queue *Q)
{
    if (!Full_Queue(*Q))
    {
        if (Empty_Queue(*Q)) Q->Front=0;
        if (Q->Rear==MaxLength-1)
        {
            //Di chuyển tính tiền ra trước Front -1 vì tri
            for(int i=Q->Front; i<=Q->Rear; i++)
                Q->Element[i-Q->Front]=Q->Element[i];
            //Xác định vị trí Rear mới
            Q->Rear=MaxLength - Q->Front-1;
        }
    }
}
```

```
        Q->Front=0;
    }

    //Tang Rear de luu noi dung moi
    Q->Rear=Q->Rear+1;
    Q->Element[Q->Rear]=X;
}
else printf("Loi: Hang day!");
}

//Xóa phần tử ra khỏi hàng
void DeQueue(Queue *Q)
{
    if (!Empty_Queue(*Q))
    {
        Q->Front=Q->Front+1;
        if (Q->Front>Q->Rear) MakeNull_Queue(Q); //Dat lai
        hang rong
    }
    else printf("Loi: Hang rong!");
}

// Lấy nội dung phần tử tại vị trí đầu hàng
ElementType Front(Queue Q)
{
    return Q.Element[Q.Front];
}

// Nhập dữ liệu cho hàng
void ReadQueue(Queue *Q)
{
    MakeNull_Queue(Q);
    ElementType X;
    int n;
```


Thực hành cấu trúc dữ liệu

```
printf("\n\n Hang co bao nhieu phan tu ?");
scanf("%d",&n);
for(int i=1;i<=n;i++)
{
    printf("Phan tu thu %d: ",i);scanf("%d",&X);
    EnQueue(X,Q);
}
}

// Hiển thị hàng
/* Khi hiển thị hàng thì ta còn lại một hàng rỗng vì theo
nguyên tắc phải xóa phần tử đầu hàng ra để lấy được phần tử
mới */
void PrintQueue(Queue *Q)
{
    while (!Empty_Queue(*Q))
    {
        printf("%d ",Front(*Q));
        DeQueue(Q);
    }
    printf("\n");
}
```

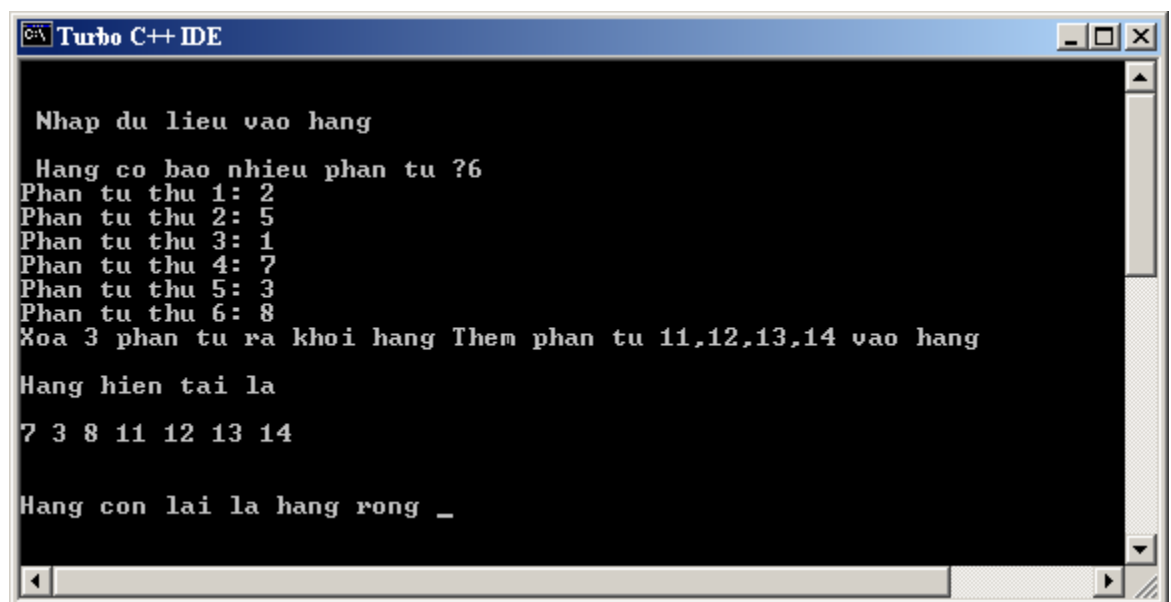
Yêu cầu: Viết chương trình chính để kiểm tra tính đúng đắn của các phép toán vừa thiết kế. Chương trình chính có thể được viết như sau:

```
#include <D:\A_QUELIB.CPP> /* chỉ đường dẫn đến file chứa các
phép toán cơ bản trên hàng vừa thiết kế */
void main()
{
    clrscr();
    Queue Q;
    printf("\n\n Nhap du lieu vao hang ");
```

Thực hành cấu trúc dữ liệu

```
ReadQueue (&Q) ;
printf("Xoa 3 phan tu ra khoi hang ");
DeQueue (&Q) ;
DeQueue (&Q) ;
DeQueue (&Q) ;
printf("Them phan tu 11,12,13,14 vao hang ");
EnQueue (11, &Q) ;
EnQueue (12, &Q) ;
EnQueue (13, &Q) ;
EnQueue (14, &Q) ;
printf("\n\nHang hien tai la \n\n");
PrintQueue (&Q) ;
if (Empty_Queue (Q))
    printf("\n\nHang con lai la hang rong ");
else
    printf("\n\nHang con lai la khac rong !!!! ");
getch();
}
```

Kết quả chương trình khi thực thi là:



```
Turbo C++ IDE

Nhap du lieu vao hang
Hang co bao nhieu phan tu ?6
Phan tu thu 1: 2
Phan tu thu 2: 5
Phan tu thu 3: 1
Phan tu thu 4: 7
Phan tu thu 5: 3
Phan tu thu 6: 8
Xoa 3 phan tu ra khoi hang Them phan tu 11,12,13,14 vao hang
Hang hien tai la
7 3 8 11 12 13 14
Hang con lai la hang rong _
```

2. Cài đặt hàng bằng mảng vòng

Tạo file tên **ACQUELIB.CPP** chứa các phép toán cơ bản trên hàng cài đặt bằng mảng vòng.

```
#include<stdio.h>
#include<conio.h>
// Khai báo hàng cài đặt bằng mảng vòng
#define MaxLength 10
typedef int ElementType;
typedef struct
{
    ElementType Elements[MaxLength];
    int Front, Rear;
} Queue;
// Tạo hàng rỗng
void MakeNull_Queue(Queue *Q)
{
    Q->Front=-1;
    Q->Rear=-1;
}
// Kiểm tra hàng rỗng
int Empty_Queue(Queue Q)
{
    return Q.Front== -1;
}
// Kiểm tra hàng đầy
int Full_Queue(Queue Q)
{
    return (Q.Rear-Q.Front+1) % MaxLength==0;
}
// Thêm phần tử vào hàng
```

Thực hành cấu trúc dữ liệu

```
void EnQueue(ElementType X, Queue *Q)
{
    if (!Full_Queue(*Q))
    {
        if (Empty_Queue(*Q)) Q->Front=0;
        Q->Rear=(Q->Rear+1) % MaxLength;
        Q->Elements[Q->Rear]=X;
    }
    else printf("Loi: Hang day!");
}

//Xóa phần tử ra khỏi hàng
void DeQueue(Queue *Q)
{
    if (!Empty_Queue(*Q))
    {
        if (Q->Front==Q->Rear) MakeNull_Queue(Q);
        else Q->Front=(Q->Front+1) % MaxLength;
    }
    else printf("Loi: Hang rong!");
}

// Lấy nội dung phần tử tại vị trí đầu hàng
ElementType Front(Queue Q)
{
    return Q.Elements[Q.Front];
}

// Tạo dữ liệu cho hàng
void ReadQueue(Queue *Q)
{
    MakeNull_Queue(Q);
    //ElementType X;
    for(int i=1;i<=10;i++)
```

Thực hành cấu trúc dữ liệu

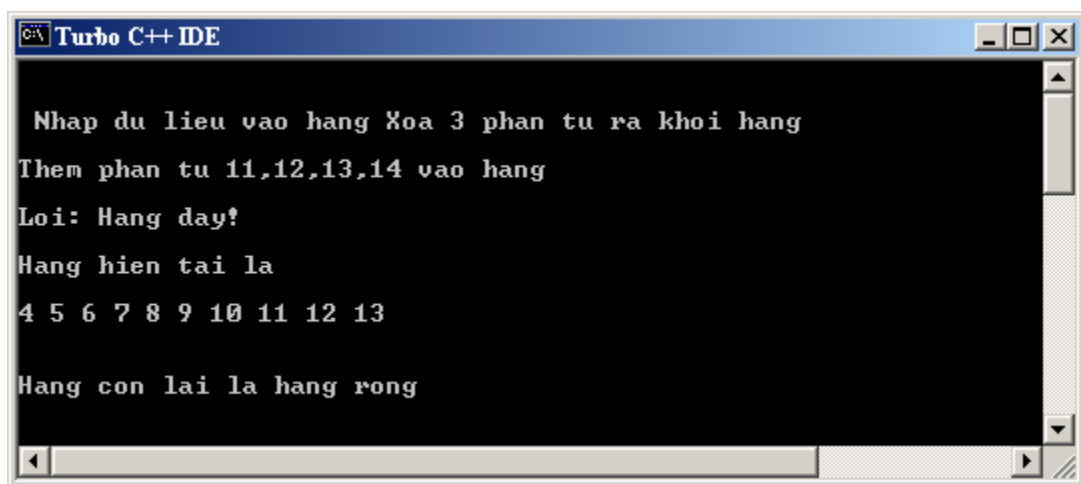
```
{
    //printf("Phan tu thu %d: ",i);scanf("%d",&X);
    EnQueue(i,Q);
}
}
// Hiển thị hàng ra màn hình
void PrintQueue(Queue *Q)
{
    while (!Empty_Queue(*Q))
    {
        printf("%d ",Front(*Q));
        DeQueue(Q);
    }
    printf("\n");
}
```

Yêu cầu: Hãy viết chương trình chính để kiểm tra tính đúng đắn của các phép toán vừa thiết kế. Chương trình chính có thể được viết như sau:

```
#include <D:\ACQUELIB.CPP> /* Duong dan chi den fiel chua cau
truc hang cai dat bang mang vong */
void main()
{
    clrscr();
    Queue Q;
    printf("\n\n Nhap du lieu vao hang ");
    ReadQueue(&Q);
    printf("Xoa 3 phan tu ra khoi hang ");
    DeQueue(&Q);
    DeQueue(&Q);
    DeQueue(&Q);
    printf("\n\nThem phan tu 11,12,13,14 vao hang \n\n");
```

```
EnQueue(11, &Q);
EnQueue(12, &Q);
EnQueue(13, &Q);
EnQueue(14, &Q);
printf("\n\nHang hien tai la \n\n");
PrintQueue(&Q);
if (Empty_Queue(Q))
    printf("\n\nHang con lai la hang rong ");
else
    printf("\n\nHang con lai la khac rong !!!! ");
getch();
}
```

Kết quả khi thực thi chương trình là:

A screenshot of the Turbo C++ IDE window. The title bar reads "Turbo C++ IDE". The main text area has a black background with white text. The output of the program is as follows:
Nhap du lieu vao hang Xoa 3 phan tu ra khoi hang
Them phan tu 11,12,13,14 vao hang
Loi: Hang day!
Hang hien tai la
4 5 6 7 8 9 10 11 12 13
Hang con lai la hang rong

3. Cài đặt bằng con trỏ

Tạo một file tên **P_QUELIB.CPP** để lưu các phép toán và cách khai báo hàng cài đặt bằng con trỏ.

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
// Khai báo hàng
typedef int ElementType;
```

```
typedef struct Node
{
    ElementType Data;
    Node* Next;
};
typedef Node* Position;
typedef struct
{
    Position Front, Rear;
} Queue;
//Tạo hàng rỗng
void MakeNullQueue(Queue *Q)
{
    Q->Front=(Node*)malloc(sizeof(Node));
    Q->Front->Next=NULL;
    Q->Rear=Q->Front;
}
// Kiểm tra hàng rỗng
int EmptyQueue(Queue Q)
{
    return (Q.Front==Q.Rear);
}
// Thêm phần tử vào hàng
void EnQueue(ElementType X, Queue *Q)
{
    Q->Rear->Next=(Node*)malloc(sizeof(Node));
    Q->Rear=Q->Rear->Next;
    //Dat gia tri vao cho Rear
    Q->Rear->Data=X;
    Q->Rear->Next=NULL;
}
```

//Xóa phần tử ra khỏi hàng

```
void DeQueue(Queue *Q)
{
    Position T;
    T=Q->Front;
    Q->Front=Q->Front->Next;
    free(T);
}
```

//Lấy nội dung phần tử tại vị trí đầu hàng

```
ElementType Front(Queue Q)
{
    return Q.Front->Next->Data;
}
```

// Nhập dữ liệu cho hàng

```
void ReadQueue(Queue *Q)
{
    MakeNullQueue(Q);
    int i,N,X;
    printf("So phan tu N = "); scanf("%d",&N);
    for(i=1;i<=N;i++)
    {
        printf("Phan tu thu %d: ",i); scanf("%d",&X);
        EnQueue(X,Q);
    }
}
```

// Hiển thị hàng ra màn hình

```
void PrintQueue(Queue *Q)
{
    while (!EmptyQueue(*Q))
    {
        printf("%d ",Front(*Q));
    }
}
```

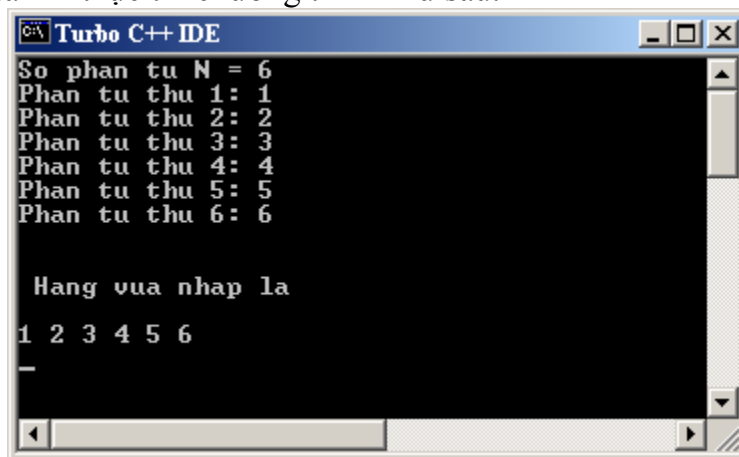


```
        DeQueue(Q) ;  
    }  
    printf("\n");  
}
```

Yêu cầu: Viết chương trình chính để kiểm tra tính đúng đắn của các phép toán trên hàng.

```
#include <D:\P_QUELIB.CPP>  
void main()  
{  
    clrscr();  
    Queue Q;  
    ReadQueue(&Q) ;  
    printf("\n\n Hang vua nhap la \n\n");  
    PrintQueue(&Q) ;  
    getch();  
}
```

Kết quả khi thực thi chương trình như sau:



III. Bài tập nâng cao

Yêu cầu: Cài đặt chương trình **QUEUE1.CPP** có sử dụng hàng đợi để đảo ngược nội dung một ngăn xếp S nhưng vẫn giữ nguyên giá trị các phần tử của S.

Ví dụ: Trước khi đảo ngược S: 5 3 9 10

Sau khi đảo ngược S: 10 9 3 5

Ý tưởng giải thuật đảo ngược:

Sử dụng một ngăn xếp S và một hàng đợi Q. Ở đây S, Q được cài bằng kiểu con trỏ, bạn có thể chọn cách cài đặt khác.

Ta theo các bước:

1. Khởi tạo hàng đợi Q rỗng.
2. Lấy các phần tử từ S đưa vào Q theo đúng nguyên tắc làm việc của hàng đợi và ngăn xếp.
3. Lấy các phần tử từ hàng đợi Q vừa thu được đưa trở lại ngăn xếp S cũng theo đúng nguyên tắc làm việc của hàng đợi và ngăn xếp.

Chương trình con có thể được viết như sau:

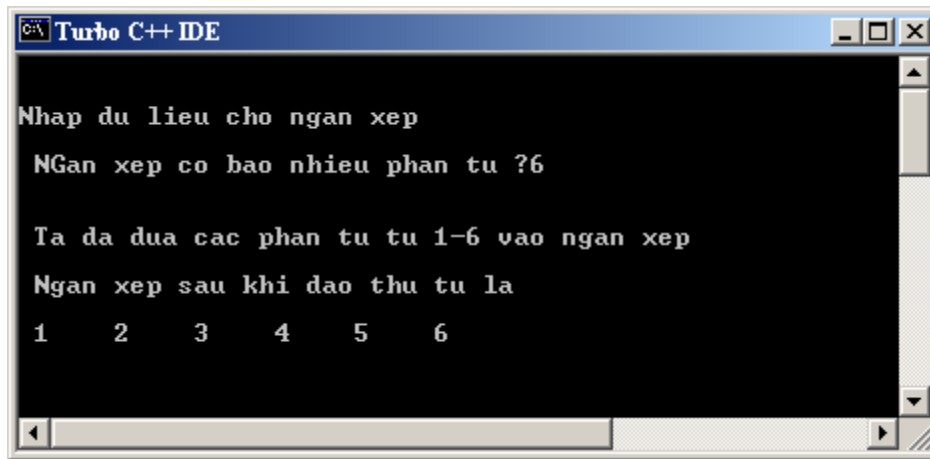
```
void ReverseStack( Stack *S)
{
    Queue Q;
    MakeNull_Queue(&Q);
    // Day du lieu trong stack vao queue
    while (!Empty_Stack (*S))
    {
        EnQueue (Top(*S), &Q);
        Pop(S);
    }
    // Bo nguoc cac phan tu trong que ve stack
    while (!Empty_Queue(Q))
    {
        Push (Front(Q), S);
        DeQueue (&Q);
    }
}
```

Chương trình chính thực hiện như sau:

```
#include <D:\A_STKLIB.CPP> // Thu vien cua ngan xep
#include <D:\A_QUELIB.CPP> // Thu vien cua hang
void Read_Stack( Stack *S)
```

Thực hành cấu trúc dữ liệu

```
{ int n,i;
ElementType X;
printf("\n\n NGan xep co bao nhieu phan tu ?");
scanf ("%d",&n);
MakeNull_Stack (S);
for (i=1;i<=n;i++)
    Push(i,S);
printf("\n\n Dua cac phan tu tu 1-%d vao ngan xep ",n);
}
void Print_Stack(Stack *S)
{
    while (!Empty_Stack(*S))
        { printf(" %d  ",Top(*S));
          Pop (S);
        }
}
void main()
{ Stack S;
  Queue Q;
  ElementType X;
  int n;
  clrscr();
  printf("\n\nNhap du lieu cho ngan xep ");
  Read_Stack(&S);
  printf("\n\n Ngan xep sau khi dao thu tu la \n\n");
  ReverseStack (&S);
  Print_Stack(&S);
  getch();
}
```



```
Turbo C++ IDE

Nhap du lieu cho ngan xep
  NGan xep co bao nhieu phan tu ?6

Ta da dua cac phan tu tu 1-6 vao ngan xep
Ngan xep sau khi dao thu tu la
1    2    3    4    5    6
```

CHƯƠNG II. CẤU TRÚC CÂY

A. CÂY TỔNG QUÁT

I. Khái niệm

1. Định nghĩa

Cây là một tập hợp các phần tử gọi là nút (nodes) trong đó có một nút được phân biệt gọi là nút gốc (root). Trên tập hợp các nút này có một quan hệ, gọi là mối quan hệ cha - con (parenthood), để xác định hệ thống cấu trúc trên các nút. Mỗi nút, trừ nút gốc, có duy nhất một nút cha. Một nút có thể có nhiều nút con hoặc không có nút con nào. Mỗi nút biểu diễn một phần tử trong tập hợp đang xét và nó có thể có một kiểu nào đó bất kỳ, thường ta biểu diễn nút bằng một kí tự, một chuỗi hoặc một số ghi trong vòng tròn. Mối quan hệ cha con được biểu diễn theo qui ước nút cha ở dòng trên nút con ở dòng dưới và được nối bởi một đoạn thẳng

Cây là một tập hợp hữu hạn các nút và một tập hợp hữu hạn các cạnh nối các cặp nút lại với nhau và không tạo thành chu trình.

2. Các phép toán cơ bản trên cây:

➤ Hàm **PARENT(n,T)** cho nút cha của nút n trên cây T, nếu n là nút gốc thì hàm cho giá trị NULL. Trong cài đặt cụ thể thì NULL là một giá trị nào đó do ta chọn, nó phụ thuộc vào cấu trúc dữ liệu mà ta dùng để cài đặt cây.

➤ Hàm **LEFTMOST_CHILD(n,T)** cho nút con trái nhất của nút n trên cây T, nếu n là lá thì hàm cho giá trị NULL.

➤ Hàm **RIGHT_SIBLING(n,T)** cho nút anh em ruột phải nút n trên cây T, nếu n không có anh em ruột phải thì hàm cho giá trị NULL.

➤ Hàm **LABEL_NODE(n,T)** cho nhãn tại nút n của cây T.

➤ Hàm **ROOT(T)** trả ra nút gốc của cây T. Nếu Cây T rỗng thì hàm trả về NULL.

➤ Hàm **CREATEi(v,T1,T2,...,Ti)**, với $i=0..n$, thủ tục tạo cây mới có nút gốc là n được gán nhãn v và có i cây con T1,...,Ti. Nếu $n=0$ thì thủ tục tạo cây mới chỉ gồm có 1 nút đơn độc là n có nhãn v. Chẳng hạn, giả sử ta có hai cây con T1 và T2, ta muốn thiết lập cây mới với nút gốc có nhãn là v thì lời gọi thủ tục sẽ là **CREATE2(v,T1,T2)**.

II. Bài tập cơ sở

1. Cài đặt cây bằng mảng:

Tạo file tên **ATREELIB.CPP** để lưu trữ các phép toán cơ bản trên cây tổng quát với nội dung như sau:

```
/*== Khai báo cây ==*/  
  
#include <conio.h>  
  
#include <stdio.h>  
  
#include <stdlib.h>
```

Thực hành cấu trúc dữ liệu

```
#define MaxLength 30
#define NIL -1      // Không tồn tại nút
typedef char DataType;
typedef int Node;
typedef struct{
    Node Parent[MaxLength]; // Lưu trữ cha của nút
    int MaxNode;            // Lưu trữ số nút hiện tại trên cây
    DataType Label[MaxLength]; // Lưu trữ nhãn của nút
}Tree;
/*=== Tạo cây rỗng ===*/
void MakeNull_Tree(Tree *T){
    T->MaxNode = 0;
}
/*=== Kiểm tra cây rỗng ===*/
int Empty_Tree(Tree T){
    return (T.MaxNode) == 0;
}
/*=== Kiểm tra cây đầy ===*/
int Full_Tree(Tree T)
{
    return (T.MaxNode == MaxLength);
}
/*=== Hàm xác định cha của nút n trên cây T ===*/
Node Parent(Node n,Tree T)
{
    if( (Empty_Tree(T)) || (n>T.MaxNode-1) )
        return NIL;
    else return T.Parent[n];
}
/*=== Hàm xác định nhãn của nút n trên cây T ===*/
DataType Label(Node n,Tree T)
```

```
{
    if( (Empty_Tree(T)) || (n>T.MaxNode-1) ){
        printf("Loi ! Khong tim thay nhan !");
        return '*';
    }
    else return T.Label[n];
}

/*== Hàm trả về nút gốc trong cây T ==*/
Node Root(Tree T)
{
    if (!Empty_Tree(T))
        return 0;
    else return NIL;
}

/*== Hàm trả về nút con trái nhất của nút n trong cây T ==*/
Node LeftMostChild(Node n,Tree T)
{
    Node i=n+1;
    int found=0;
    if( (n<0) || (Empty_Tree(T)) || (n>=T.MaxNode-1) )
        return NIL;
    else
    {
        while ( (i<T.MaxNode) && (found==0) )
            if(T.Parent[i] == n) found=1;
            else i++;
        if(found == 0) return NIL;
        else return i;
    }
}

/*= Hàm trả về nút anh em ruột phải của nút n trên cây T =*/
```

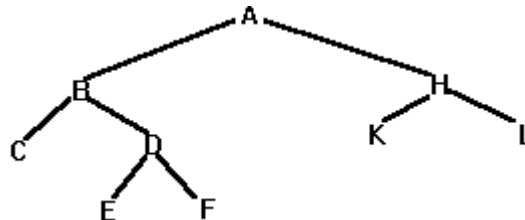
Thực hành cấu trúc dữ liệu

```
Node RightSibling(Node n, Tree T)
{
    Node i=n+1;
    int found=0;
    if(n<0) return NIL;
    while( (i<=T.MaxNode-1)&&(found==0) )
        if(T.Parent[n] == T.Parent[i])
            found = 1;
        else i++;
    if(found) return i;
    else return NIL;
}
```

Yêu cầu viết chương trình chính kiểm tra tính đúng đắn của các phép toán vừa thiết kế

Tạo file tên **ATRTEST1.CPP** để kiểm tra các phép toán tìm con trái nhất, anh em ruột phải, lấy nhãn của nút trên cây.

Để thuận tiện, ta gán trị cho cây T như sau:



Hình II.1 Cây tổng quát

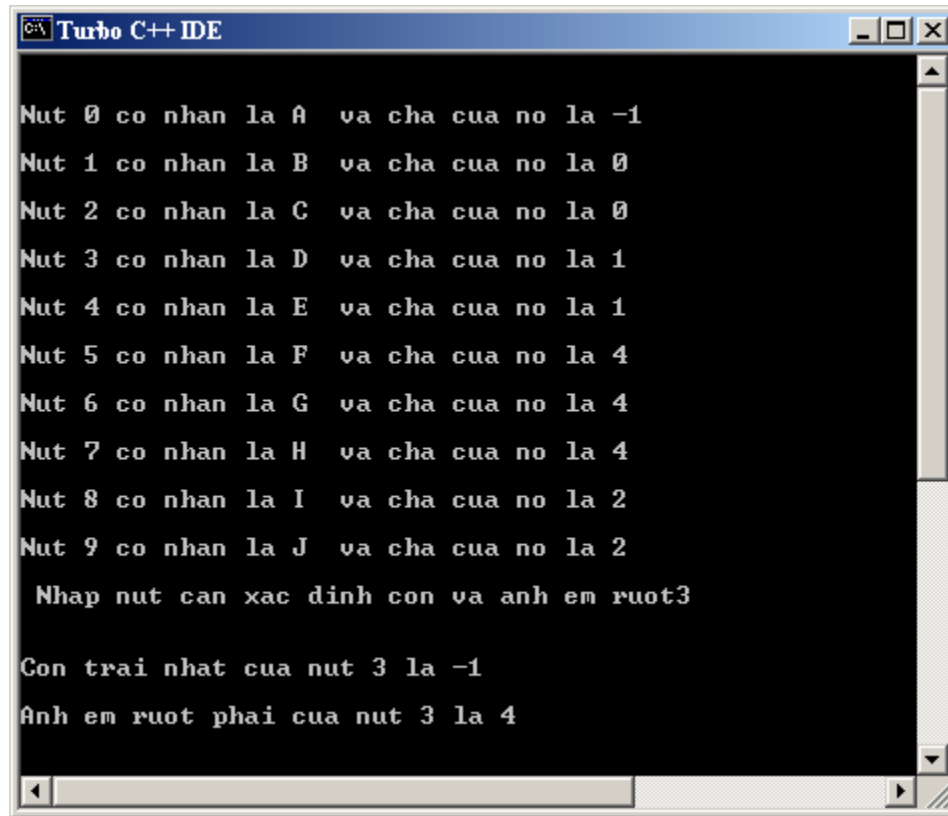
Chương trình có thể được thiết kế như sau:

```
#include <D:\TREELIB.CPP> /* duong dan den thu vien chua cac  
phép toán cơ bản trên cây */  
void main()  
{  
    clrscr();  
    Tree T;  
    // Doan lệnh gán trị cho cây T như hình vẽ  
    T.MaxNode = 10;
```


Thực hành cấu trúc dữ liệu

```
T.Parent[0]=-1;T.Label[0]='A';
T.Parent[1]=0; T.Label[1]='B';
T.Parent[2]=0; T.Label[2]='C';
T.Parent[3]=1; T.Label[3]='D';
T.Parent[4]=1; T.Label[4]='E';
T.Parent[5]=4; T.Label[5]='F';
T.Parent[6]=4; T.Label[6]='G';
T.Parent[7]=4; T.Label[7]='H';
T.Parent[8]=2; T.Label[8]='I';
T.Parent[9]=2; T.Label[9]='J';
for (int i=0; i<T.MaxNode ;i++)
printf("\n\nNut %d co nhan la %c va cha cua no la %d ",i,
T.Label[i],T.Parent[i]);
int j;
printf("\n\nNhap nut can xac dinh con va anh em ruot");
scanf("%d",&j);
printf("\n\nCon trai nhat cua nut %d la
%d",j,LeftMostChild(j,T));
printf("\n\nAnh em ruot phai cua nut %d la
%d",j,RightSibling (j,T));
getch();
}
```

Kết quả thực thi chương trình như sau:



```
Turbo C++ IDE

Nut 0 co nhan la A va cha cua no la -1
Nut 1 co nhan la B va cha cua no la 0
Nut 2 co nhan la C va cha cua no la 0
Nut 3 co nhan la D va cha cua no la 1
Nut 4 co nhan la E va cha cua no la 1
Nut 5 co nhan la F va cha cua no la 4
Nut 6 co nhan la G va cha cua no la 4
Nut 7 co nhan la H va cha cua no la 4
Nut 8 co nhan la I va cha cua no la 2
Nut 9 co nhan la J va cha cua no la 2

Nhap nut can xac dinh con va anh em ruot3

Con trai nhut cua nut 3 la -1
Anh em ruot phai cua nut 3 la 4
```

III. Bài tập nâng cao

Yêu cầu 1: Tạo file **AT_TEST1.CPP** để duyệt cây theo phương pháp duyệt tiền tự, trung tự, hậu tự. Viết chương trình chính kiểm tra tính đúng đắn của các phép duyệt cây trên.

Ý tưởng của các phép duyệt cây:

```
void PREORDER(node n)
{
    liệt kê nút n;
    for (mỗi nút con c của nút n theo thứ tự từ trái sang phải)
        PREORDER(c);
}
```

```
void INORDER(node n)
{
    if (n là nút lá)
        {(liệt kê nút n)}
    else {
        INORDER( con trái nhất của nút n);
        liệt kê nút n;
    }
}
```

```
        for (mỗi nút con c của nút n, trừ nút con trái nhất, theo thứ tự từ trái sang phải)
            INORDER(c);
    }
}
```

```
void POSORDER(node c)
{
    if (n là nút lá)
        {liệt kê nút n}
    else {
        for (mỗi nút con c của nút n theo thứ tự từ trái sang phải)
            POSORDER(c);
        liệt kê nút n;
    }
}
```

Các chương trình con có thể viết cụ thể như sau:

```
/*=== Duyet tien tu cay T ===*/
void PreOrder(Node n, Tree T)
{
    Node i;
    printf(" %c",Label(n,T)); //Xu ly nut
    i = LeftMostChild(n,T);
    while(i!=NIL)
    {
        PreOrder(i,T);
        i = RightSibling(i,T);
    }
}

/*=== Duyet trung tu cay T ===*/
void InOrder(Node n, Tree T)
{
    Node i = LeftMostChild(n,T);
    if(i!=NIL) InOrder(i,T);
    printf(" %c",Label(n,T)); //Xu ly Nut
    i = RightSibling(i,T);
}
```

Thực hành cấu trúc dữ liệu

```
while(i!=NIL)
{
    InOrder(i,T);
    i = RightSibling(i,T);
}
}
/*=== Duyệt hậu tu cay T ===*/
void PostOrder(Node n, Tree T)
{
    Node i;
    i = LeftMostChild(n,T);
    while(i!=NIL)
    {
        PostOrder(i,T);
        i = RightSibling(i,T);
    }
    printf(" %c",Label(n,T));
}
```

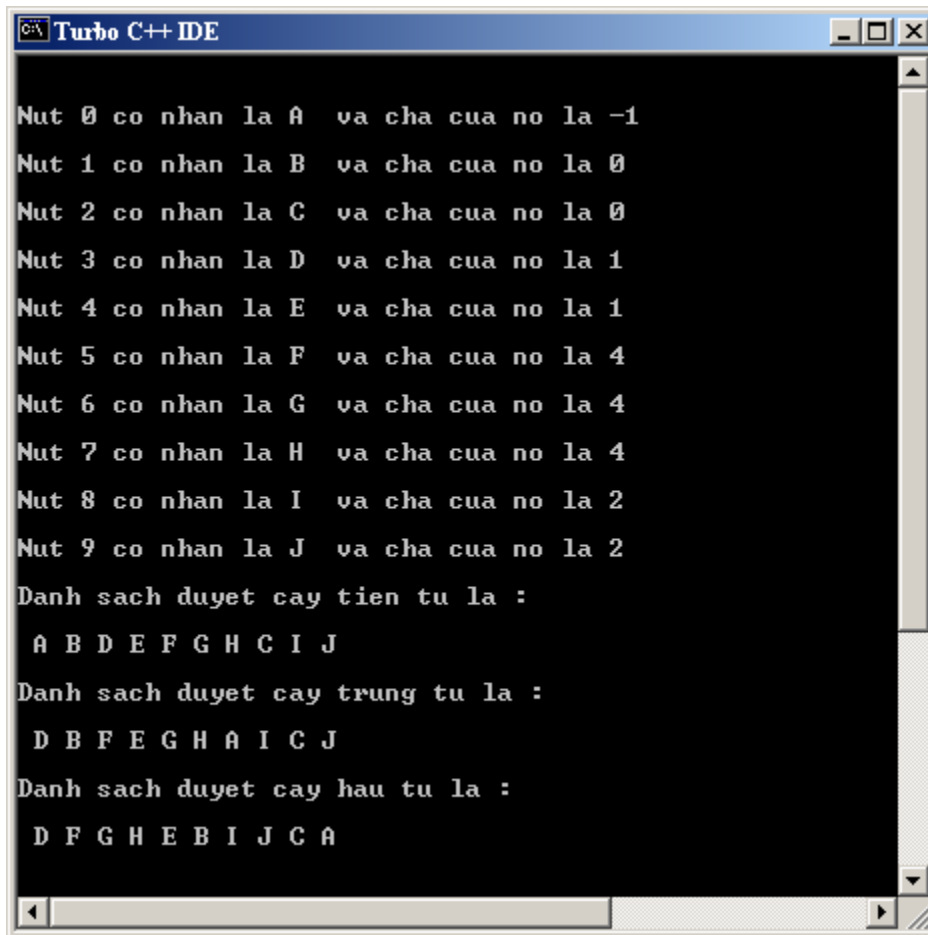
Chương trình chính có thể được viết như sau: (Vấn dùng cây như hình II.1)

```
#include <D:\TREELIB.CPP> /* Duong dan chi toi thu vien chua
phep toan tren cay tong quat */
void main()
{
    clrscr();
    Tree T;
    T.MaxNode = 10;
    T.Parent[0]=-1;T.Label[0]='A';
    T.Parent[1]=0; T.Label[1]='B';
    T.Parent[2]=0; T.Label[2]='C';
```

Thực hành cấu trúc dữ liệu

```
T.Parent[3]=1; T.Label[3]='D';
T.Parent[4]=1; T.Label[4]='E';
T.Parent[5]=4; T.Label[5]='F';
T.Parent[6]=4; T.Label[6]='G';
T.Parent[7]=4; T.Label[7]='H';
T.Parent[8]=2; T.Label[8]='I';
T.Parent[9]=2; T.Label[9]='J';
for (int i=0; i<T.MaxNode ;i++)
printf("\n\nNut %d co nhan la %c va cha cua no la %d ",i,
T.Label[i],T.Parent[i]);
printf("\n\nDanh sach duyet cay tien tu la : \n\n");
PreOrder(Root(T),T);
printf("\n\nDanh sach duyet cay trung tu la : \n\n");
InOrder(Root(T),T);
printf("\n\nDanh sach duyet cay hau tu la : \n\n");
PostOrder(Root(T),T);
getch();
}
```

Kết quả thực thi chương trình như sau:



```
Turbo C++ IDE

Nut 0 co nhan la A va cha cua no la -1
Nut 1 co nhan la B va cha cua no la 0
Nut 2 co nhan la C va cha cua no la 0
Nut 3 co nhan la D va cha cua no la 1
Nut 4 co nhan la E va cha cua no la 1
Nut 5 co nhan la F va cha cua no la 4
Nut 6 co nhan la G va cha cua no la 4
Nut 7 co nhan la H va cha cua no la 4
Nut 8 co nhan la I va cha cua no la 2
Nut 9 co nhan la J va cha cua no la 2
Danh sach duyet cay tien tu la :
A B D E F G H C I J
Danh sach duyet cay trung tu la :
D B F E G H A I C J
Danh sach duyet cay hau tu la :
D F G H E B I J C A
```

Yêu cầu 2: Tạo file tên **AT_TEST2.CPP** để nhập dữ liệu cho cây tổng quát từ bàn phím và thiết kế chương trình chính kiểm tra nó.

Ý tưởng:

- Nhập số nút trong cây và lưu vào trường MaxNode.
- Ứng với từng nút ta nhập nhãn và cha của nó.

Chương trình con có thể được viết cụ thể như sau:

```
void ReadTree(Tree *T)
{
    MakeNull_Tree(T);
    int i=1,n=0;
    //Nhập vào tổng số nút của cây cho đến khi hợp lệ
    do{
        printf("\nNhập vào tổng số nút của cây : ");
        fflush(stdin);scanf("%d",&n);
```

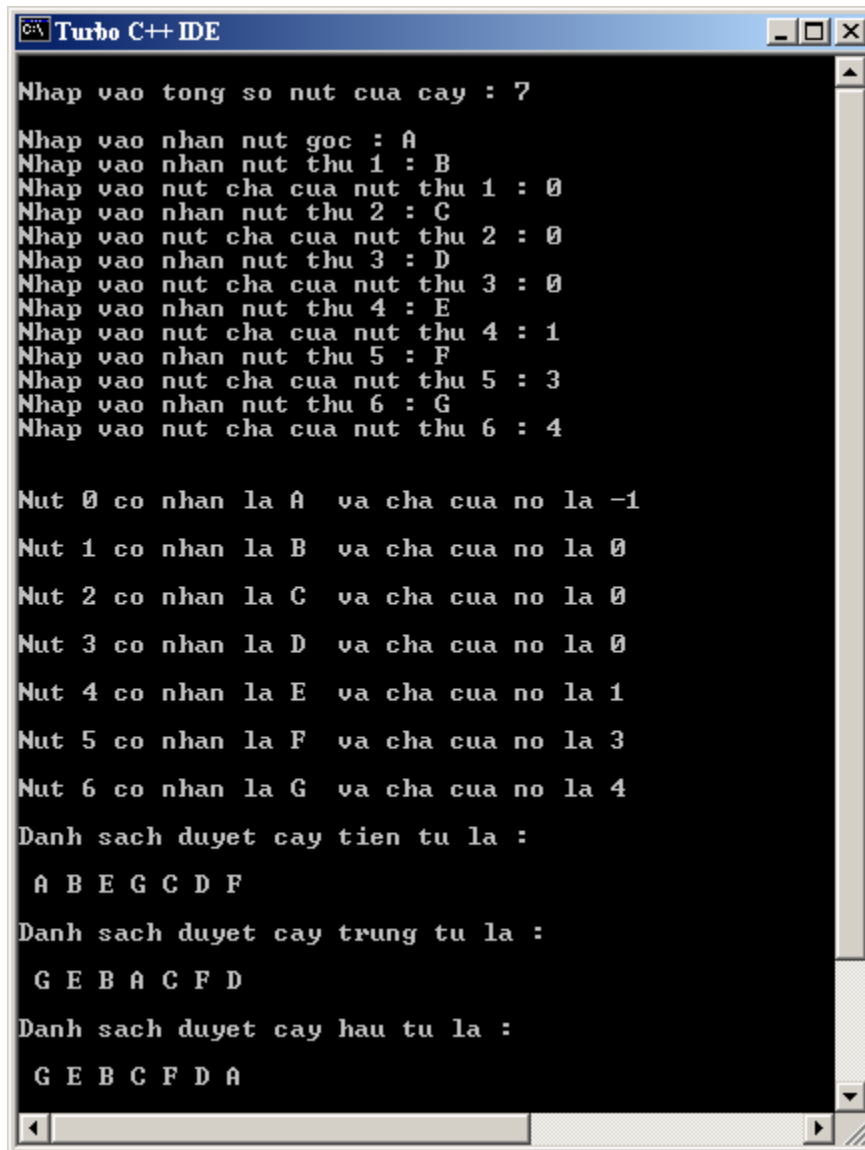
```
        }while( (n<1) || (n>MaxLength) );
T->MaxNode = n;
//Nhap vao nut goc
printf("\nNhap vao nhan nut goc : ");
fflush(stdin);scanf("%c",&(*T).Label[0]);
T->Parent[0] = -1;
//Nhap cac nut con lai
for(i=1;i<n;i++)
{
    printf("Nhap vao nhan nut thu %d : ",i);
    fflush(stdin);scanf("%c",&(*T).Label[i]);
    printf("Nhap vao nut cha cua nut thu %d : ",i);
    scanf("%d",&(*T).Parent[i]);
}
}
```

Chương trình chính có thể thiết kế như sau:

```
#include <D:\TREELIB.CPP>
void main()
{
    clrscr();
    Tree T;
    ReadTree(&T);
    for (int i=0; i<T.MaxNode ;i++)
    printf("\n\nNut %d co nhan la %c   va cha cua no la %d ",i,
    T.Label[i],T.Parent[i]);
    printf("\n\nDanh sach duyet cay tien tu la :  \n\n");
    PreOrder(Root(T),T);
    printf("\n\nDanh sach duyet cay trung tu la : \n\n");
    InOrder(Root(T),T);
    printf("\n\nDanh sach duyet cay hau tu la :  \n\n");
    PostOrder(Root(T),T);
}
```

```
getch();  
}
```

Kết quả chương trình khi thực thi là:



The screenshot shows the Turbo C++ IDE window with the following output:

```
Nhap vao tong so nut cua cay : 7  
Nhap vao nhan nut goc : A  
Nhap vao nhan nut thu 1 : B  
Nhap vao nut cha cua nut thu 1 : 0  
Nhap vao nhan nut thu 2 : C  
Nhap vao nut cha cua nut thu 2 : 0  
Nhap vao nhan nut thu 3 : D  
Nhap vao nut cha cua nut thu 3 : 0  
Nhap vao nhan nut thu 4 : E  
Nhap vao nut cha cua nut thu 4 : 1  
Nhap vao nhan nut thu 5 : F  
Nhap vao nut cha cua nut thu 5 : 3  
Nhap vao nhan nut thu 6 : G  
Nhap vao nut cha cua nut thu 6 : 4  
  
Nut 0 co nhan la A va cha cua no la -1  
Nut 1 co nhan la B va cha cua no la 0  
Nut 2 co nhan la C va cha cua no la 0  
Nut 3 co nhan la D va cha cua no la 0  
Nut 4 co nhan la E va cha cua no la 1  
Nut 5 co nhan la F va cha cua no la 3  
Nut 6 co nhan la G va cha cua no la 4  
  
Danh sach duyet cay tien tu la :  
A B E G C D F  
Danh sach duyet cay trung tu la :  
G E B A C F D  
Danh sach duyet cay hau tu la :  
G E B C F D A
```

Yêu cầu 3: Tạo file **AT_TEST3.CPP** để thiết kế hàm xác định bậc của nút n trong cây T. Dựa vào hàm vừa tạo để xác định bậc của cây.

Ý tưởng xác định bậc của nút:

- Khởi tạo kq=0 ;
- i= con trái nhất của nút n.
- Trong khi nút vẫn còn con chưa xét (i!=NIL)


```
{  
    Xét con kế tiếp của nút n  
    Tăng kq lên 1; (bậc tăng 1);  
}  
- Return kq
```

Ý tưởng xác định bậc của cây (dựa trên hàm bậc của nút):

- Khởi tạo bậc của cây tạm thời là bậc của nút gốc.
- Duyệt qua tất cả các nút trên cây, nếu bậc của nút đang xét lớn hơn bậc tạm thời của cây thì ta đổi giá trị của bậc tại thời của cây bằng bậc của nút đang xét.
- Sau khi duyệt hết các nút, giá trị bậc tạm thời của cây chính là bậc của cây.

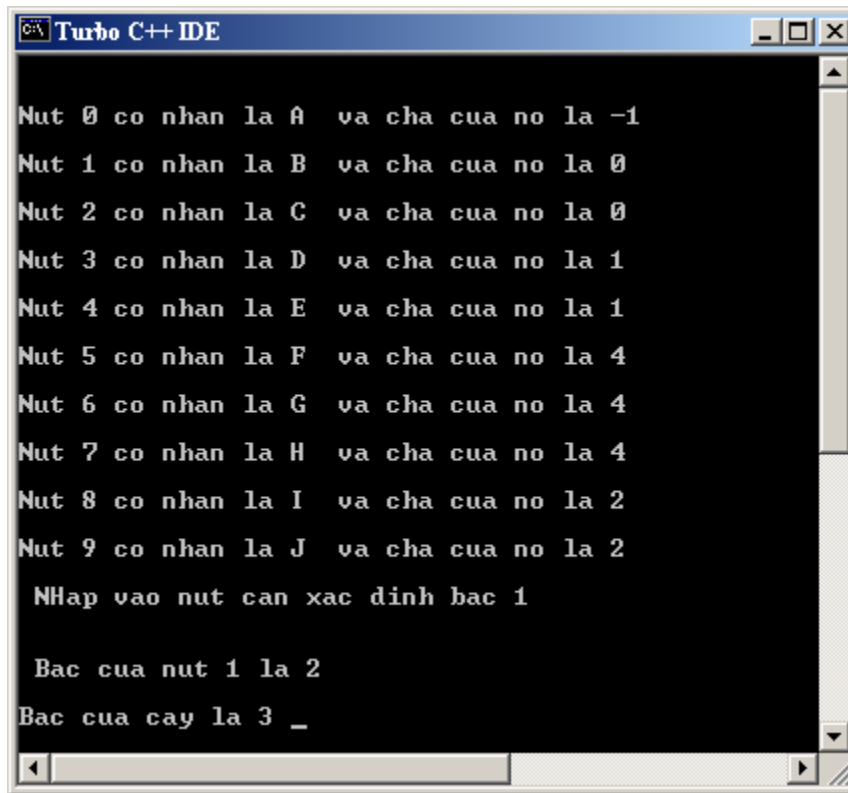
Chương trình con được viết cụ thể như sau:

```
int Nb_Child_Node (Node N, Tree T)  
{  
    int kq=0;  
    Node i;  
    i=LeftMostChild(N,T);  
    while (i!=NIL)  
    { kq++;  
      i=RightSibling (i,T);  
    }  
    return kq;  
}
```

Chương trình chính được thiết kế như sau:

```
void main()  
{  
    clrscr();  
    Tree T;
```

```
T.MaxNode = 10;
T.Parent[0]=-1;T.Label[0]='A';
T.Parent[1]=0; T.Label[1]='B';
T.Parent[2]=0; T.Label[2]='C';
T.Parent[3]=1; T.Label[3]='D';
T.Parent[4]=1; T.Label[4]='E';
T.Parent[5]=4; T.Label[5]='F';
T.Parent[6]=4; T.Label[6]='G';
T.Parent[7]=4; T.Label[7]='H';
T.Parent[8]=2; T.Label[8]='I';
T.Parent[9]=2; T.Label[9]='J';
for (int i=0; i<T.MaxNode ;i++)
printf("\n\nNut %d co nhan la %c va cha cua no la %d ",i,
T.Label[i],T.Parent[i]);
printf("\n\n NHap vao nut can xac dinh bac ");
scanf("%d",&i);
printf("\n\n Bac cua nut %d la %d ",i,Nb_Child_Node(i,T));
int Nb_child =Nb_Child_Node(Root(T),T);
for (i=1; i<T.MaxNode;i++)
    if (Nb_child <Nb_Child_Node(i,T))
        Nb_child = Nb_Child_Node(i,T);
printf("\n\nBac cua cay la %d ", Nb_child);
getch();
}
```



```
Turbo C++ IDE

Nut 0 co nhan la A va cha cua no la -1
Nut 1 co nhan la B va cha cua no la 0
Nut 2 co nhan la C va cha cua no la 0
Nut 3 co nhan la D va cha cua no la 1
Nut 4 co nhan la E va cha cua no la 1
Nut 5 co nhan la F va cha cua no la 4
Nut 6 co nhan la G va cha cua no la 4
Nut 7 co nhan la H va cha cua no la 4
Nut 8 co nhan la I va cha cua no la 2
Nut 9 co nhan la J va cha cua no la 2

NHap vao nut can xac dinh bac 1

Bac cua nut 1 la 2
Bac cua cay la 3 _
```

Yêu cầu 4: Tạo file tên **AT_TEST4.CPP** để viết hàm xác định chiều cao của cây và kiểm tra tính đúng đắn của hàm này.

Ý tưởng

```
int Height(Tree T)
{ if (n là nút lá )
    chiều cao =0
else
{
    kq:=0;
    i:=con trái nhất của nút n
    while vẫn còn tồn tại con chưa xét của nút n
    {
        kq:=max(kq,chiều cao của nút (i,T));
        i:=Con kế tiếp của nút i đang xét;
    }
    Chiều cao:=kq+1;
}
```

```
}// while  
    return chiều cao  
}
```

B. CÂY NHỊ PHÂN

I. Khái niệm

Trong phần này sinh viên làm quen với các thao tác cơ bản trên cây nhị phân, cây tìm kiếm nhị phân. Phần lớn các thao tác đã được cài đặt sẵn, sinh viên có nhiệm vụ đọc, hiểu và sử dụng được chúng vào chương trình của mình. Tùy theo yêu cầu cụ thể của bài tập, sinh viên tự sửa đổi mã nguồn của các thư viện được cung cấp cho phù hợp.

II. Bài tập cơ sở

Tạo file tên TTREELIB.CPP để lưu trữ các phép toán cơ bản trên cây nhị phân.

```
include <conio.h>  
#include <stdio.h>  
#include <alloc.h>  
//Khai báo cây nhị phân  
typedef char TData;  
typedef struct TNode{  
    TData  Data;  
    TNode* left;  
    TNode* right;  
};  
typedef TNode* TTree;  
/*=== Tạo cây rỗng ===*/  
void MakeNull_Tree(TTree *T)  
{  
    (*T)=NULL;  
}  
/*=== Kiểm tra cây rỗng ===*/  
int EmptyTree(TTree T)  
{  
    return T==NULL;
```

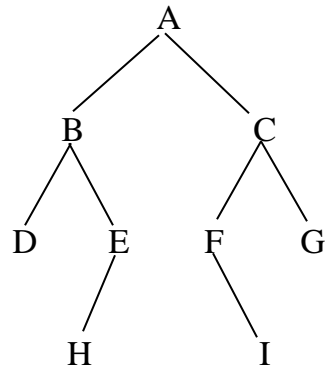
```
}  
  
/*=== Xác định con trái của nút T ===*/  
TTree LeftChild(TTree T)  
{  
    if(T != NULL)  
        return T->left;  
    else    return NULL;  
}  
  
/*=== Xác định con phải ===*/  
TTree RightChild(TTree T)  
{  
    if(T != NULL)  
        return T->right;  
    else    return NULL;  
}  
  
/*=== Kiểm tra xem nút T có phải là nút lá hay không ===*/  
int isLeaf(TTree T)  
{  
    if((T != NULL) && (T->left == NULL) && (T->right == NULL))  
        return 1;  
    else    return NULL;  
}  
  
/*=== Tạo cây từ hai cây con cho trước ===*/  
TTree Create2(TData v, TTree left, TTree right)  
{  
    TTree N; // Khai báo 1 cây mới  
    N = (TNode*)malloc(sizeof(TNode));  
    //Cấp phát vùng nhớ cho nút N  
    N->Data = v; // Nhan của nút N là v  
    N->left = left; //Con trái của N là cây left  
    N->right = right; //Con phải của N là cây right
```

```
    return N;
}
//Duyệt tiền tự
void NLR(TTree T)
{
    if(!EmptyTree(T))
    {
        printf(" %c",T->Data); //Xu ly nut
        NLR(LeftChild(T)); //Duyet tien tu con trai
        NLR(RightChild(T)); //Duyet tien tu con phai
    }
}
//Duyệt trung tự
void LNR(TTree T)
{
    if(!EmptyTree(T))
    {
        LNR(LeftChild(T)); //Duyet trung tu con trai
        printf(" %c",T->Data); //Xu ly nut
        LNR(RightChild(T)); //Duyet trung tu con phai
    }
}
//Duyệt hậu tự
void LRN(TTree T)
{
    if(!EmptyTree(T))
    {
        LRN(LeftChild(T)); //Duyet hau tu con trai
        LRN(RightChild(T)); //Duyet hau tu con phai
        printf(" %c",T->Data); //Xu ly nut
    }
}
```

```
}
```

Yêu cầu: Tạo file tên **TTEST.CPP** để kiểm tra các phép toán trên cấu trúc cây nhị phân vừa thiết kế.

Nhập cây nhị phân có dạng như sau:



Hình II.2 Cây nhị phân

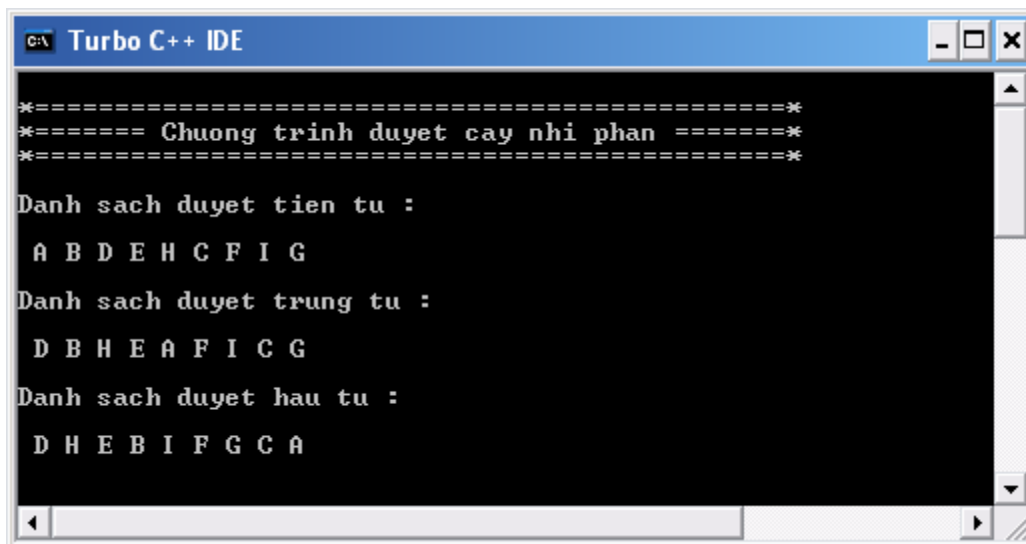
Chương trình chính có thể được viết như sau:

```
#include <D:\TTREELIB.CPP> /* duong dan chi den thu vien chua phep toan  
co ban tren cay nhi phan */  
#include <stdlib.h>  
void main()  
{  
    clrscr();  
    TTree T,T1,T2;  
  
    printf("\n*=====  
;    printf("\n*=====  
=====");  
  
    printf("\n*=====  
");  
    T1=  
Create2('B',Create2('D',NULL,NULL),Create2('E',Create2('H',NU  
LL,NULL),NULL));  
    T2=  
Create2('C',Create2('F',NULL,Create2('I',NULL,NULL)),Create2(  
'G',NULL,NULL));
```

Thực hành cấu trúc dữ liệu

```
T = Create2('A',T1,T2);
printf("\nDanh sach duyet tien tu : \n");
NLR(T);
printf("\n\nDanh sach duyet trung tu : \n");
LNR(T);
printf("\n\nDanh sach duyet hau tu : \n");
LRN(T);
getch();
}
```

Kết quả khi thực thi chương trình như sau:



```
*****
***** Chuong trinh duyet cay nhi phan *****
*****
Danh sach duyet tien tu :
A B D E H C F I G
Danh sach duyet trung tu :
D B H E A F I C G
Danh sach duyet hau tu :
D H E B I F G C A
```

III. Bài tập nâng cao

Yêu cầu 1: Viết hàm xác định số nút trên cây.

Ý tưởng thực hiện:

- Nếu cây rỗng thì số nút bằng 0
- Ngược lại, số nút trên cây bằng số nút của cây con trái cộng với số nút của cây con phải và cộng 1. (do đứng tại nút đang xét)

Hàm có thể được viết cụ thể như sau:

```
/*=== Xác định số nút trên cây ===*/
int nb_nodes(TTree T)
{
```


Thực hành cấu trúc dữ liệu

```
if (EmptyTree (T) )
    return 0;
else    return nb_nodes (T->left) +nb_nodes (T->right) +1;
}
```

Yêu cầu 2: Viết hàm xác định chiều cao của cây.

Ý tưởng:

Nếu cây rỗng thì chiều cao bằng 0;

Ngược lại, nếu cây có 1 nút thì chiều cao bằng 0;

Ngược lại, chiều cao bằng 1 + chiều cao lớn nhất của cây con trái và cây con phải.

Hàm có thể được thiết kế cụ thể như sau:

// Hàm xác định giá trị lớn nhất trong 2 giá trị số nguyên

```
int max(int value1, int value2)
{
    return ( (value1 > value2) ? value1 : value2);
}
```

// Hàm xác định chiều cao của cây

```
int TreeHeight(TTree T)
{
    int height=0;
    if (!EmptyTree(T))
    {
        if( isLeaf(T)) //
            height=0;
        else
            height
            =
            max(TreeHeight(LeftChild(T)), TreeHeight(RightChild(T)))+1;
    }
    return height;
}
```

Yêu cầu 3: Viết hàm xác định số nút lá trên cây

Ý tưởng:

- Khởi tạo số nút là =0
- Nếu nút gốc là nút lá thì tăng số nút lá trong cây lên 1;
- Ngược lại, số nút lá bằng số nút lá trong cây con trái cộng với số nút lá trong cây con phải.

Hàm có thể được viết cụ thể như sau:

```
int nb_leaf(TTree T)
{
    int leaf=0;
    if(!EmptyTree(T))
    {
        if(isLeaf(T))
            leaf++;
        else
            leaf = nb_leaf(LeftChild(T))+nb_leaf(RightChild(T));
    }
    return leaf;
}
```

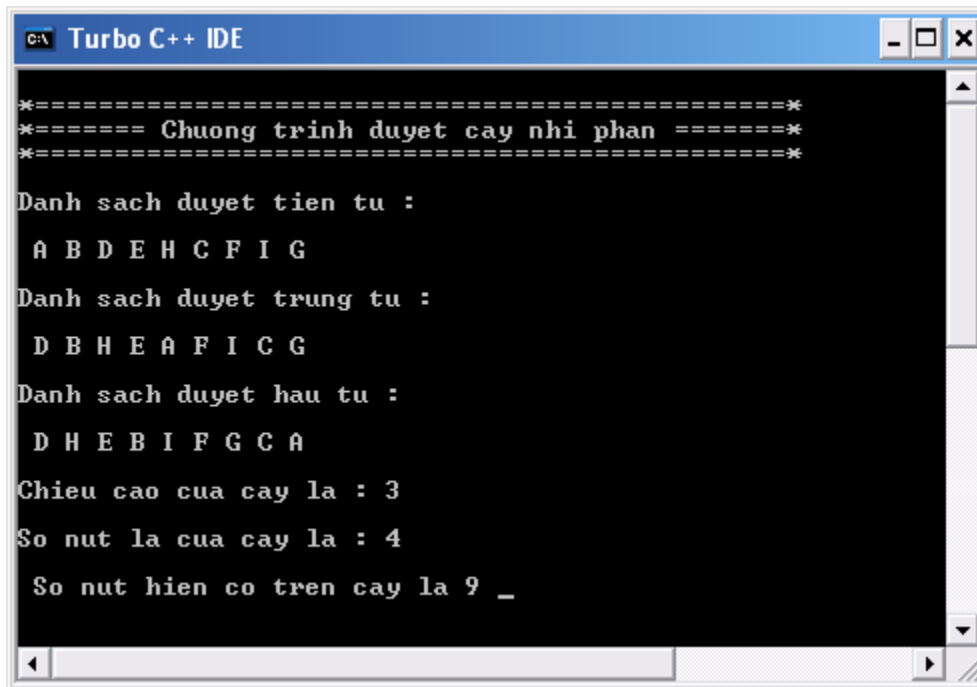
Viết đoạn chương trình chính để kiểm tra các hàm vừa thiết kế như sau:

```
#include <D:\TTREELIB.CPP> /* Duong dan den thu vien chua cac
pheap toan co ban tren cay nhi phan */
#include <stdlib.h>
void main()
{
    clrscr();
    TTree T,T1,T2;

    printf("\n*=====*\n");
    ;
    printf("\n*=====  Chương trình duyệt cây nhi phan
=====*\n");
    ;
```

```
printf("\n*=====*\n");
T1=
Create2('B',Create2('D',NULL,NULL),Create2('E',Create2('H',NULL,NULL),NULL));
T2=
Create2('C',Create2('F',NULL,Create2('I',NULL,NULL)),Create2('G',NULL,NULL));
T = Create2('A',T1,T2);
printf("\nDanh sach duyet tien tu : \n");
NLR(T);
printf("\n\nDanh sach duyet trung tu : \n");
LNR(T);
printf("\n\nDanh sach duyet hau tu : \n");
LRN(T);
printf("\n\nChieu cao cua cay la : %d",TreeHeight(T));
printf("\n\nSo nut la cua cay la : %d",nb_leaf(T));
printf ("\n\n So nut hien co tren cay la %d ", nb_nodes(T));
getch();
}
```

Kết quả thực thi chương trình như sau:



```
C:\ Turbo C++ IDE

=====
===== Chương trình duyệt cây nhị phân =====
=====

Danh sách duyệt tiền tu :
  A B D E H C F I G
Danh sách duyệt trung tu :
  D B H E A F I C G
Danh sách duyệt hậu tu :
  D H E B I F G C A
Chiều cao của cây là : 3
Số nút lá của cây là : 4
Số nút hiện có trên cây là 9 _
```

C. CÂY TÌM KIẾM NHỊ PHÂN

I. Khái niệm

Cây tìm kiếm nhị phân (TKNP) là cây nhị phân mà khoá tại mỗi nút cây lớn hơn khoá của tất cả các nút thuộc cây con bên trái và nhỏ hơn khoá của tất cả các nút thuộc cây con bên phải.

Các phép toán cơ bản trên cây tìm kiếm nhị phân cũng giống như một cây nhị phân. Trong đó, đặc biệt cây tìm kiếm nhị phân có thêm ba phép toán chính tương ứng với đặc trưng của cây là tìm nút có nhãn cho trước, thêm một nút có nhãn X vào cây và cây và xóa một nút có nhãn cho trước ra khỏi cây.

II. Bài tập cơ sở

Tạo file tên **BSTLIB.CPP** lưu trữ các phép toán cơ bản trên cây tìm kiếm nhị phân.

Cách khai báo cây tìm kiếm nhị phân và các phép toán cơ bản trên cây như tạo rỗng, duyệt cây, kiểm tra rỗng, xác định số nút, chiều cao (trừ hàm tạo cây)... đều giống như cây nhị phân cài đặt bằng con trỏ. Ta hầu như chỉ cần cài đặt thêm một vài phép toán cơ bản trên cây tìm kiếm nhị phân như tìm nút, thêm nút vào cây và xóa nút ra khỏi cây.

Các phép toán này được viết cụ thể như sau:

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

//Khai báo cây tìm kiếm nhị phân
```

```
typedef int KeyType;
typedef struct Node
{
    KeyType Key;
    Node* left;
    Node* right;
};
typedef Node* TTree;
/*=== Tạo cây tìm kiếm ===*/
void MakeNull_Tree(TTree *T)
{
    (*T)=NULL;
}
/*=== Kiểm tra cây rỗng ===*/
int EmptyTree(TTree T)
{
    return T==NULL;
}
/*=== Xác định con trái của nút ===*/
TTree LeftChild(TTree T)
{
    if(T != NULL)
        return T->left;
    else    return NULL;
}
/*=== Xác định con phải của nút ===*/
TTree RightChild(TTree T)
{
    if(T != NULL)
        return T->right;
    else    return NULL;
```

Thực hành cấu trúc dữ liệu

```
}
/*=== Kiểm tra nút lá trong cây ===*/
int isLeaf(TTree T)
{
    if((T != NULL) && (T->left == NULL) && (T->right == NULL))
        return 1;
    else return NULL;
}
/*=== Duyệt cây nhị phân ===*/
//Duyệt tiền tự
void NLR(TTree T)
{
    if(!EmptyTree(T))
    {
        printf(" %d",T->Key); //Xu ly nut
        NLR(LeftChild(T)); //Duyệt tiền tu con trai
        NLR(RightChild(T)); //Duyệt tiền tu con phải
    }
}
//Duyệt trung tự
void LNR(TTree T)
{
    if(!EmptyTree(T))
    {
        LNR(LeftChild(T)); //Duyệt trung tu con trai
        printf(" %d",T->Key); //Xu ly nut
        LNR(RightChild(T)); //Duyệt trung tu con phải
    }
}
//Duyệt hậu tự
void LRN(TTree T)
```

```
{
    if (!EmptyTree(T))
    {
        LRN(LeftChild(T)); //Duyet hau tu con trai
        LRN(RightChild(T)); //Duyet hau tu con phai
        printf(" %d", T->Key); //Xu ly nut
    }
}

/*Hàm trả về vị trí của nút có khoa K cho trước. Nếu không
tìm thấy, hàm trả về NULL*/
Tree Search(KeyType K, TTree T)
{
    KeyType temp;
    temp = T->Key;
    if (T != NULL) //Kiem tra cay rong
        if (K == temp) //tim thay khoa
            return T;
        else
            if (K < temp) // Hy vong K nam ben trai
                return Search(K, LeftChild(T));
            else // Hy vong K nam ben phai
                return Search(K, RightChild(T));
    else return NULL;
}

//Thêm nút vào cây
void InsertNode(KeyType X, TTree *T)
{
    if ((*T) == NULL)
    {
        (*T) = (Node*)malloc(sizeof(Node));
        (*T)->Key = X;
    }
}
```

```
    (*T)->left = NULL;
    (*T)->right = NULL;
}
else
    if ((*T)->Key == X)
        printf("Da ton tai khoa X");
    else
        if ((*T)->Key > X)
            InsertNode(X, &(*T)->left);
        else
            InsertNode(X, &(*T)->right);
}
```

//Xoá một nút trong cây tìm kiếm nhị phân

KeyType DeleteMin(TTree *T)

```
{
    KeyType k;
    if ((*T)->left == NULL)
    {
        k = (*T)->Key;
        (*T) = (*T)->right;
        return k;
    }
    else return DeleteMin(&(*T)->left);
}
```

void DeleteNode(KeyType X, TTree *T)

```
{
    if ((*T) != NULL) //Kiem tra cay khac rong
        if (X < (*T)->Key) //Hy vong X nam ben trai cua nut
            DeleteNode(X, &(*T)->left);
}
```



```
else
    if(X > (*T)->Key) //Hy vong X nam ben phai cua nut
DeleteNode(X, &(*T)->right);
else // Tim thay khoa X tren cay
    if(((*T)->left==NULL) && ((*T)->right==NULL))
        //X la nut la
        (*T)=NULL; // Xoa nut X
    else // X co it nhat mot con
        if((*T)->left==NULL) //Chac chan co con phai
            (*T) = (*T)->right;
        else
            if((*T)->right==NULL) //Chac chan co con trai
                (*T) = (*T)->left;
            else // X co hai con
                (*T)->Key = DeleteMin(&(*T)->right);
}
```

Tạo file tên **BSTTEST2.CPP** để viết chương trình chính kiểm tra tính đúng đắn của các phép toán vừa tạo.

Chương trình chính có thể được thiết kế như sau:

```
#include <D:\BSTLIB.CPP>
/* Thu vien luu tru cay tim kiem nhi phan */
#include <stdlib.h>
void main()
{
    clrscr();
    TTree T;
    MakeNull_Tree(&T);
    printf("\n\n Them cac nut co cac gia tri 50, 20, 30, 60, 55
theo thu tu vao cay \n\n");
    InsertNode(50, &T);
    InsertNode(20, &T);
    InsertNode(30, &T);
```

Thực hành cấu trúc dữ liệu

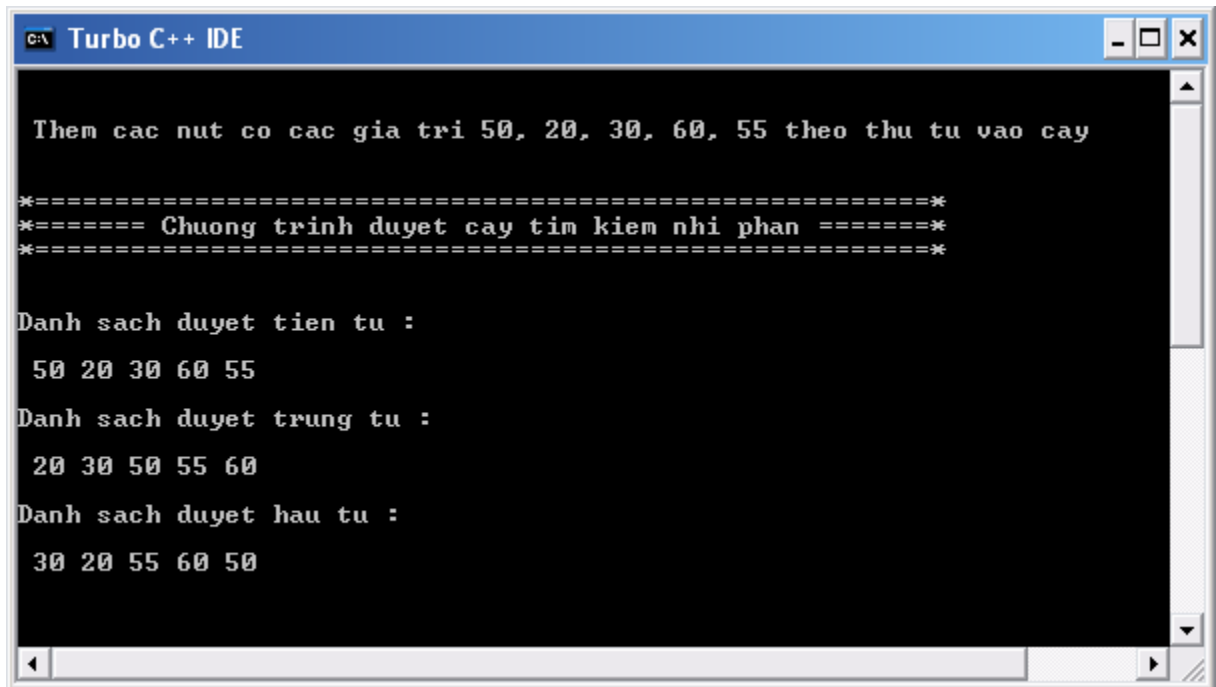
```
    InsertNode(60, &T);
    InsertNode(55, &T);
printf("\n*=====
=====*\n");

    printf("\n*===== Chương trình duyệt cây tìm kiếm nhị phân
=====*\n");

printf("\n*=====
=====*\n");

    printf("\n\nDanh sach duyệt tiền tu : \n\n");
    NLR(T);
    printf("\n\nDanh sach duyệt trung tu : \n\n");
    LNR(T);
    printf("\n\nDanh sach duyệt hậu tu : \n\n");
    LRN(T);
getch();
}
```

Kết quả thực hiện chương trình trên màn hình là:



```
C:\ Turbo C++ IDE

Them cac nut co cac gia tri 50, 20, 30, 60, 55 theo thu tu vao cay

*=====*
*===== Chương trình duyệt cây tìm kiếm nhị phân =====*
*=====*

Danh sach duyệt tiền tu :
50 20 30 60 55
Danh sach duyệt trung tu :
20 30 50 55 60
Danh sach duyệt hậu tu :
30 20 55 60 50
```

Ví dụ 2: Tạo chương trình chính (**BSTTEST2.CPP**) để kiểm tra các hàm DeleteNode, tìm vị trí nút, tính số nút, chiều cao, số nút lá.

III. Bài tập nâng cao

Yêu cầu: Viết hàm nhập dữ liệu cây từ bàn phím.

Ý tưởng:

- Khởi tạo cây tìm kiếm nhị phân rỗng.
- Nhập số nút trong cây.
- Cho vòng lặp ứng với từng nút thì nhập vào khoá cho nút đó và xen nó vào trong cây tìm kiếm nhị phân.

```
void ReadTree(TTree *T)
{
    int i,n=0;
    KeyType X;
    do{
        printf("\nNhap vao so nut ");
        fflush(stdin);scanf("%d",&n);
    }while(n==0);
    printf("\nNhap vao nut goc ");
    fflush(stdin);scanf("%d",&X);
    InsertNode(X,T);
    for(i=1;i<n;i++)
    {
        printf("\nNhap vao nut thu %d ",i);
        fflush(stdin);scanf("%d",&X);
        InsertNode(X,T);
    }
}
```

CHƯƠNG III. CẤU TRÚC TẬP HỢP

A. TẬP HỢP

I. Khái niệm

Tập hợp là một khái niệm cơ bản trong toán học. Tập hợp được dùng để mô hình hoá hay biểu diễn một nhóm bất kỳ các đối tượng trong thế giới thực cho nên nó đóng vai trò rất quan trọng trong mô hình hoá cũng như trong thiết kế các giải thuật.

Khái niệm tập hợp cũng như trong toán học, đó là sự tập hợp các thành viên (members) hoặc phần tử (elements). Tất cả các phần tử của tập hợp là khác nhau. Tập hợp có thể có thứ tự hoặc không có thứ tự, tức là, có thể có quan hệ thứ tự xác định trên các phần tử của tập hợp hoặc không. Tuy nhiên, trong chương này, chúng ta giả sử rằng các phần tử của tập hợp có thứ tự tuyến tính, tức là, trên tập hợp S có quan hệ $<$ và $=$ thoả mãn hai tính chất:

Với mọi $a, b \in S$ thì $a < b$ hoặc $b < a$ hoặc $a = b$

Với mọi $a, b, c \in S$, $a < b$ và $b < c$ thì $a < c$

Cũng như các kiểu dữ liệu trừu tượng khác, các phép toán kết hợp với mô hình tập hợp sẽ tạo thành một kiểu dữ liệu trừu tượng là rất đa dạng. Tùy theo nhu cầu của các ứng dụng mà các phép toán khác nhau sẽ được định nghĩa trên tập hợp. Ở đây ta đề cập đến một số phép toán thường gặp nhất như sau

UNION(A,B,C): Hợp của hai tập $C = A \cup B$.

INTERSECTION(A,B,C): Giao của hai tập $C = A \cap B$.

Thủ tục **DIFFERENCE(A,B,C):** Hiệu của hai tập A và B và trả ra kết quả là tập hợp $C = A \setminus B$

Hàm **MEMBER(x,A)** Kiểm tra xem x có phải là thành viên của tập hợp hay không? Nếu $x \in A$ thì hàm cho kết quả là 1 (đúng), ngược lại cho kết quả 0 (sai).

Thủ tục **MAKENULLSET(A)** tạo tập hợp A tập rỗng

Thủ tục **INSERTSET(x,A)** thêm x vào tập hợp A

Thủ tục **DELETESSET(x,A)** xoá x khỏi tập hợp A

Thủ tục **ASSIGN(A,B)** gán A cho B (tức là $B=A$)

Hàm **MIN(A)** cho phần tử bé nhất trong tập A

Hàm **EQUAL(A,B)** cho kết quả TRUE nếu $A=B$ ngược lại cho kết quả FALSE

II. Bài tập cơ sở

1. Tập hợp cài đặt bằng vecto bit

Tạo file tên **SETLIB.CPP** để lưu trữ cách khai báo và các phép toán cơ bản trên tập hợp cài đặt bằng vecto bit.

```
// Khai báo tập hợp cài đặt bằng vecto bit
```

```
#include <conio.h>
```

Thực hành cấu trúc dữ liệu

```
#include <stdio.h>
const maxlength =100;

typedef int SET[maxlength];
// Khởi tạo tập hợp rỗng
void makenull(SET a)
{
    int i;
    for(i=0;i<maxlength;i++)
        a[i]=0;
}
// Nhập tập hợp các số nguyên từ bàn phím
void nhap ( SET a)
{
    int n,i,x;
    makenull(a);
    printf("Cho biet so phan tu trong tap hop "); scanf("%d",&n);
    for(i=0;i<n;i++)
    {   printf ("\nNhap vao noi dung phan tu thu %d ",i);
        scanf("%d",&x);
        a[x]=1;
    }
}
// Hiển thị tập hợp ra màn hình
void hienthi (SET a)
{ int i;
  for (i=0;i<maxlength;i++)
    if (a[i]==1) printf("%d  ",i);
}
// Tìm hiệu của hai tập hợp
void SET_difference (SET a,SET b, SET c)
```

Thực hành cấu trúc dữ liệu

```
{ int i;
  for (i=0;i<maxlength;i++)
    if ((a[i]==1)&&(b[i]==0)) c[i]=1;
    else c[i]=0;
}

// Tìm hợp của hai tập hợp
void SET_union (SET a,SET b,SET c)
{ int i;
  for (i=0;i<maxlength;i++)
    if ((a[i]==1)|| (b[i]==1)) c[i]=1;
    else c[i]=0;
}

// Tìm giao của hai tập hợp
void SET_intersection (SET a,SET b, SET c)
{ int i;
  for (i=0;i<maxlength;i++)
    if ((a[i]==1)&&(b[i]==1)) c[i]=1;
    else c[i]=0;
}
```

Yêu cầu: Viết chương trình chính tên TEST1.CPP để kiểm tra tính đúng đắn của các chương trình con tạo rỗng, nhập và hiển thị tập hợp vừa tạo.

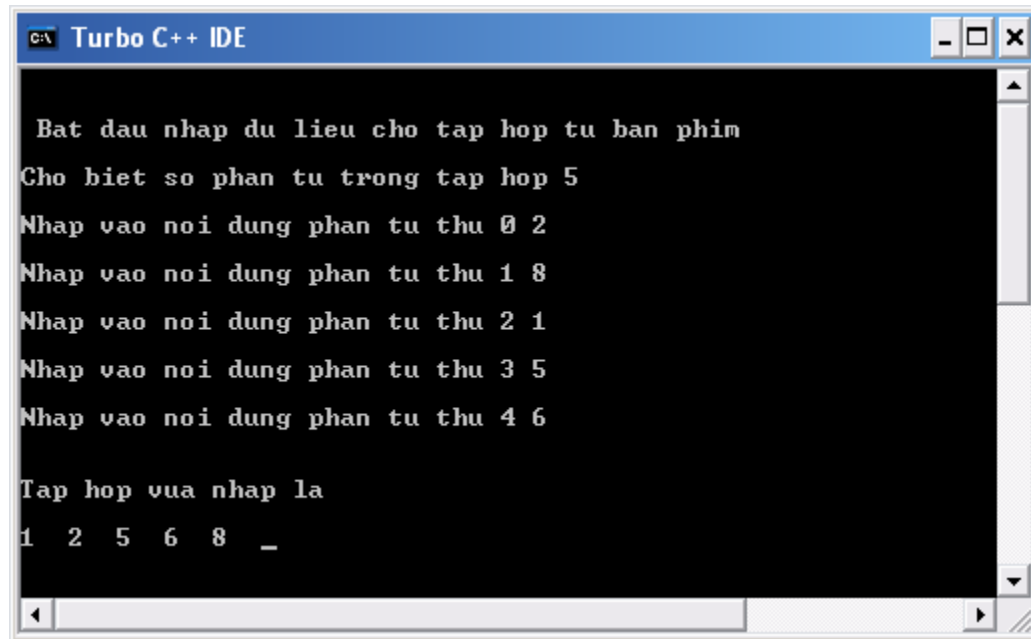
Chương trình chính có thể được thiết kế như sau:

```
#include <D:\SETLIB.CPP> /* Đường dẫn đến thư viện lưu trữ  
các phép toán cơ bản trên tập hợp */
void main()
{
  SET a;
  clrscr();
  printf("\n\n Bat dau nhap du lieu cho tap hop tu ban phim\n\n");
  nhap(a);
  printf("\n\nTap hop vua nhap la\n\n");
}
```

Thực hành cấu trúc dữ liệu

```
    hienthi(a);  
    getch();  
}
```

Kết quả chương trình thực thi như sau:

A screenshot of the Turbo C++ IDE window. The title bar reads "C:\ Turbo C++ IDE". The main text area has a black background with white text. The output of the program is as follows:
Bat dau nhap du lieu cho tap hop tu ban phim
Cho biet so phan tu trong tap hop 5
Nhap vao noi dung phan tu thu 0 2
Nhap vao noi dung phan tu thu 1 8
Nhap vao noi dung phan tu thu 2 1
Nhap vao noi dung phan tu thu 3 5
Nhap vao noi dung phan tu thu 4 6

Tap hop vua nhap la
1 2 5 6 8 _
The IDE shows standard window controls (minimize, maximize, close) in the top right corner and a scroll bar on the right side of the text area.

Yêu cầu 2: Viết chương trình tên **TEST2.CPP** để kiểm tra phép toán hợp, giao, hiệu hai tập hợp.

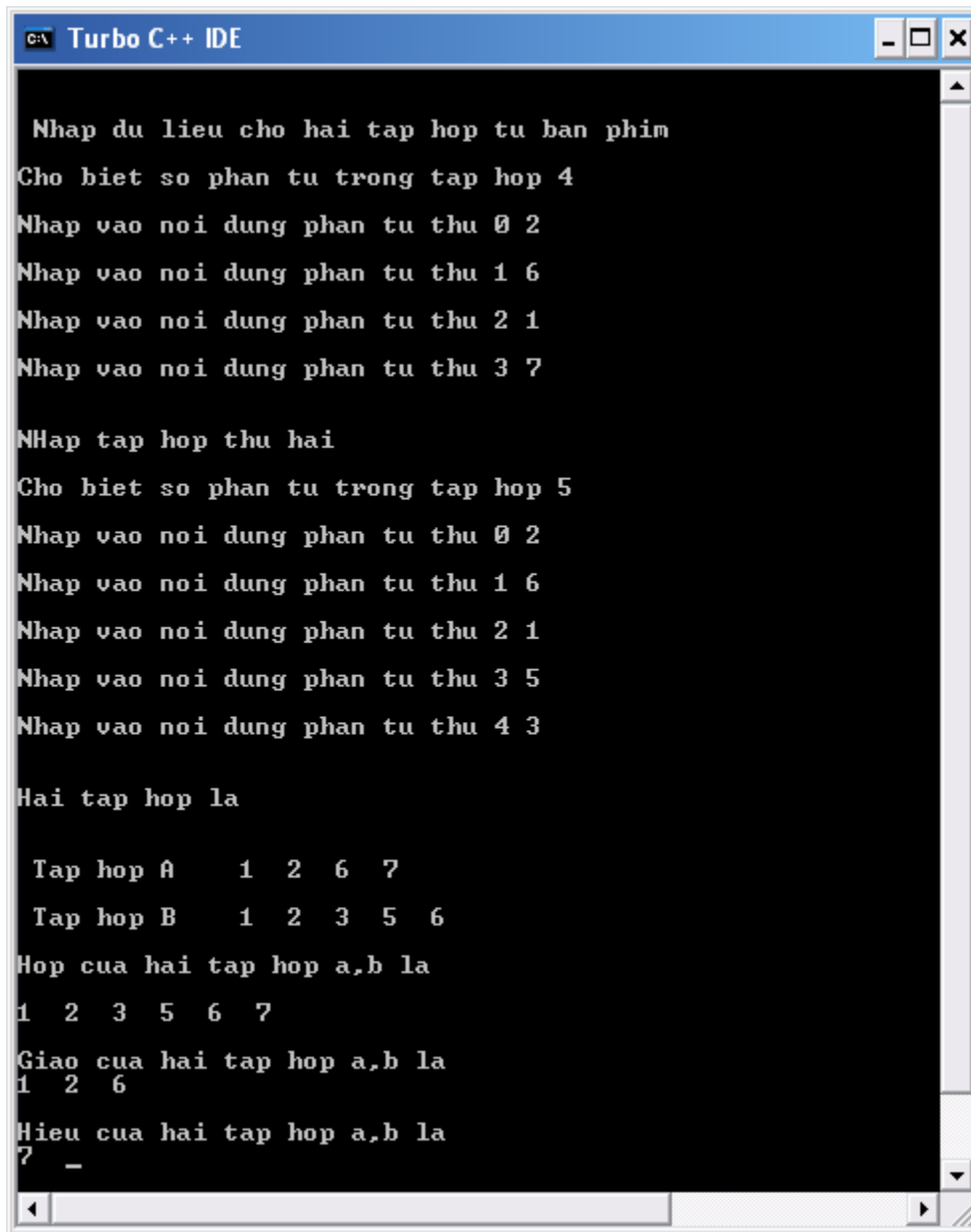
Chương trình chính có thể được thiết kế như sau:

```
#include <D:\SETLIB.CPP>  
void main()  
{  
    SET a,b,c;  
    clrscr();  
    printf("\n\n Nhap du lieu cho hai tap hop tu ban phim  
\n\n");  
    nhap(a);  
    printf("\n\nNhap tap hop thu hai \n\n");  
    nhap(b);  
    printf("\n\nHai tap hop la\n\n");
```

Thực hành cấu trúc dữ liệu

```
printf ("\n Tap hop A      ");
hienthi(a);
printf("\n\n Tap hop B      ");
hienthi(b);
SET_union(a,b,c);
printf("\n\nHop cua hai tap hop a,b la\n\n");
hienthi(c);
SET_intersection (a,b,c);
printf("\n\nGiao cua hai tap hop a,b la\n");
hienthi(c);
SET_difference(a,b,c);
printf("\n\nHieu cua hai tap hop a,b la\n");
hienthi(c);
getch();
}
```

Kết quả chương trình khi thực thi là:



```
C:\ Turbo C++ IDE

Nhap du lieu cho hai tap hop tu ban phim
Cho biet so phan tu trong tap hop 4
Nhap vao noi dung phan tu thu 0 2
Nhap vao noi dung phan tu thu 1 6
Nhap vao noi dung phan tu thu 2 1
Nhap vao noi dung phan tu thu 3 7

Nhap tap hop thu hai
Cho biet so phan tu trong tap hop 5
Nhap vao noi dung phan tu thu 0 2
Nhap vao noi dung phan tu thu 1 6
Nhap vao noi dung phan tu thu 2 1
Nhap vao noi dung phan tu thu 3 5
Nhap vao noi dung phan tu thu 4 3

Hai tap hop la

Tap hop A    1  2  6  7
Tap hop B    1  2  3  5  6
Hop cua hai tap hop a,b la
1  2  3  5  6  7
Giao cua hai tap hop a,b la
1  2  6
Hieu cua hai tap hop a,b la
7  -
```

2. Tập hợp cài đặt bằng con trỏ

Tạo file tên **P_SETLIB.CPP** để lưu trữ các phép toán cơ bản trên tập hợp cài đặt bằng con trỏ với nội dung trong file như sau:

```
// Khai báo tập hợp cài đặt bằng con trỏ
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
```

Thực hành cấu trúc dữ liệu

```
typedef int ElementType;
typedef struct Node
{
    ElementType Data;
    Node * Next;
};
typedef Node * Position;
typedef Position SET;
// Tạo tập hợp rỗng
void MakeNullSET(SET *L)
{
    (*L)=(Node*)malloc(sizeof(Node));
    (*L)->Next= NULL;
}
// Hàm kiểm tra tập hợp rỗng
int EmptySET(SET L)
{    return (L->Next==NULL);    }
// Thêm phần tử vào tập hợp.
void InsertSET(ElementType X, SET L)
{
    Position T,P;
    int finish;
    P=L;
    finish=0;
    while ((P->Next!=NULL)&&(finish==0))
        if (P->Next->Data<=X)
            P=P->Next;
        else finish=1;
    // P đang lưu trữ vị trí để xen phần tử X vào
    T=(Node*)malloc(sizeof(Node));
    T->Data=X;
```

```
T->Next=P->Next;
P->Next=T;
}
// Xoá phần tử ra khỏi tập hợp
void DeleteSET(ElementType X, SET L)
{
    Position T,P=L;
    int finish=0;
    while ((P->Next!=NULL)&& (finish==0))
        if (P->Next->Data<=X) P=P->Next;
        else finish=1;
    if ((finish==1) && (P->Next->Data==X))
    { T=P->Next;
      P->Next=T->Next;
      free(T);
    }
}

// Các hàm trả về vị trí đặc biệt trong tập hợp
Position First(SET L)
{ return L; }

Position EndSET(SET L)
{ Position P;
  P=First(L);
  while (P->Next!=NULL) P=P->Next;
  return P;
}

Position Next(Position P, SET L)
{ return P->Next; }
```

// Hàm lấy nội dung phần tử tại vị trí P trong tập hợp

```
ElementType Retrieve(Position P, SET L)
{    return P->Next->Data; }
```

// Hàm kiểm tra xem X có thuộc tập hợp hay không

// Neu X la phan tu trong tap hop thi tra ve 1 va
// Ham tra ve 0 neu X khong la phan tu trong tap hop

```
int Member(ElementType X, SET L)
{
    Position P;
    int Found = 0;
    P = First(L);
    while ((P != EndSET(L)) && (Found == 0))
        if (Retrieve(P,L) == X) Found = 1;
        else P = Next(P, L);
    return Found;
}
```

// Nhập dữ liệu cho tập hợp từ bàn phím

```
void ReadSET(SET *L)
{
    MakeNullSET(L);
    int i,N,X;
    printf("So phan tu N = "); scanf("%d",&N);
    for(i=1;i<=N;i++)
    {
        printf("Phan tu thu %d: ",i); scanf("%d",&X);
        if (Member(X, *L)==0)
            InsertSET(X, *L);
        else
            printf ("\nPhan tu %d da ton tai trong tap hop ",X);
    }
}
```

```
}
```

```
// Hiển thị tập hợp ra màn hình
```

```
void PrintSET(SET L)
{
    Position P;
    P = First(L);
    while (P != EndSET(L))
    {
        printf("%d ", Retrieve(P, L));
        P = Next(P, L);
    }
    printf("\n");
}
```

```
//Hợp của hai tập hợp
```

```
void UnionSET(SET A, SET B, SET *C)
{
    Position p;
    MakeNullSET(C);
    p=First(A);
    while (p!=EndSET(A))
    { InsertSET (Retrieve(p,A), *C);
      p=Next(p,A);
    }
    p=First(B);
    while (p!=EndSET (B))
    { if (Member(Retrieve(p,B), *C)==0)
        InsertSET (Retrieve(p,B), *C);
      p=Next(p,B);
    }
}
```

// Giao của hai tập hợp

```
void IntersectionSET(SET A, SET B, SET *C)
{
    Position p;
    MakeNullSET(C);
    p=First(A);
    while (p!=EndSET(A))
    { if (Member(Retrieve(p,A),B)==1)
        InsertSET (Retrieve(p,A),*C);
        p=Next(p,A);
    }
}
```

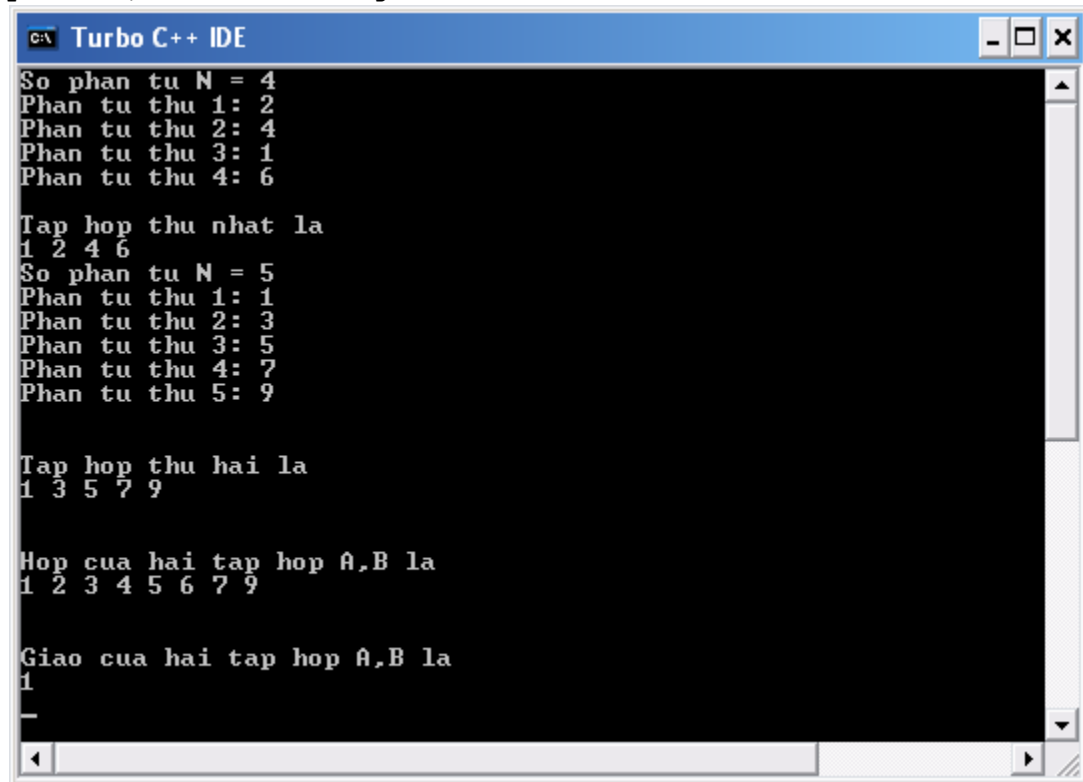
Yêu cầu: Viết chương trình chính kiểm tra tính đúng đắn của các phép toán trên. Chương trình chính có thể được thiết kế như sau:

```
#include <D:\P_SETLIB.CPP> /* Thu viện lưu trữ các phép toán
cơ bản trên tập hợp cài đặt bằng con trỏ */
void main()
{
    clrscr();
    SET A,B,C;
    ReadSET(&A);
    printf("\nTap hop thu nhat la \n");
    PrintSET(A);
    ReadSET(&B);
    printf("\nTap hop thu hai la \n");
    PrintSET(B);
    UnionSET(A,B,&C);
    printf("\nHop cua hai tap hop A,B la \n");
    PrintSET(C);
    IntersectionSET(A,B,&C);
    printf("\nGiao cua hai tap hop A,B la \n");
```

Thực hành cấu trúc dữ liệu

```
PrintSET(C);  
getch();  
}
```

Kết quả thực thi chương trình là:



```
C:\ Turbo C++ IDE  
So phan tu N = 4  
Phan tu thu 1: 2  
Phan tu thu 2: 4  
Phan tu thu 3: 1  
Phan tu thu 4: 6  
  
Tap hop thu nhat la  
1 2 4 6  
So phan tu N = 5  
Phan tu thu 1: 1  
Phan tu thu 2: 3  
Phan tu thu 3: 5  
Phan tu thu 4: 7  
Phan tu thu 5: 9  
  
Tap hop thu hai la  
1 3 5 7 9  
  
Hop cua hai tap hop A,B la  
1 2 3 4 5 6 7 9  
  
Giao cua hai tap hop A,B la  
1  
-
```

B. TỰ ĐIỂN

I. Khái niệm

Từ điển là một kiểu dữ liệu trừu tượng tập hợp đặc biệt, trong đó chúng ta chỉ quan tâm đến các phép toán InsertSet, DeleteSet, Member và MakeNullSet. Sở dĩ chúng ta nghiên cứu từ điển là do trong nhiều ứng dụng không sử dụng đến các phép toán hợp, giao, hiệu của hai tập hợp. Ngược lại ta cần một cấu trúc làm sao cho việc tìm kiếm, thêm và bớt phần tử có phần hiệu quả nhất gọi là từ điển. Chúng ta cũng chấp nhận MakeNullSet như là phép khởi tạo cấu trúc.

II. Bài tập cơ sở

Yêu cầu: Tạo file tên **DICLIB.CPP** để lưu trữ cách khai báo và các phép toán cơ bản trên từ điển. File có nội dung chính như sau:

```
// Khai báo từ điển  
#include<stdio.h>
```

Thực hành cấu trúc dữ liệu

```
#include<conio.h>
#define MaxLength 100
typedef int ElementType;
typedef int Position;
typedef struct
{
    ElementType Data[MaxLength];
    Position Last;
} SET;

// Tạo tự điểm rỗng
void MakeNullSET(SET *L)
{
    (*L).Last=0;
}

// Hàm kiểm tra rỗng
int EmptySET(SET L)
{
    return L.Last==0;
}

//Kiểm tra đầy
int FullSET(SET L)
{
    return L.Last==MaxLength;
}

// In tự diễn ra màn hình
void PrintSET(SET L)
{
    Position P=1;
```


Thực hành cấu trúc dữ liệu

```
while (P <= L.Last)
{
    printf("%d ", L.Data[P]);
    P++;
}
printf("\n");
}
```

/* Hàm kiểm tra X có phải là thành viên trong tập điển hay không? Nếu X thuộc tập điển thì hàm trả về 1, ngược lại hàm trả về 0*/

```
int Member(ElementType X, SET L)
{
    Position P=1;
    int Found = 0;
    while ((P <= (L.Last)) && (Found == 0))
        if ((L.Data[P]) == X) Found = 1;
        else P++;
    return Found;
}
```

// Thêm phần tử vào tập điển

```
void InsertSET(ElementType X, SET *L)
{
    if (FullSET(*L))
        printf("Tập hop day");
    else
        if (Member(X, *L)==0)
        {
            (*L).Last++;
            (*L).Data[(*L).Last]=X;
        }
}
```

```
    }
    else
        printf ("\nPhan tu da ton tai trong tu dien");
}

// Xoá phần tử ra khỏi tự diễn
void DeleteSET(ElementType X, SET *L)
{
    if (EmptySET(*L))
        printf("Tap hop rong!");
    else
    {
        Position Q=1;
        while ((Q<=(*L).Last)&& ((*L).Data[Q]!=X)) Q++;
        if ((*L).Data[Q]==X)
        { for (int i=Q;i<(*L).Last;i++)
            (*L).Data[i]=(*L).Data[i+1];
          (*L).Last--;
        }
    }
}

// Nhập tự diễn từ bàn phím
void ReadSET(SET *L)
{
    int i,N;
    ElementType X;
    MakeNullSET(L);
    printf("So phan tu tap hop N= ");scanf("%d",&N);
    for(i=1;i<=N;i++)
    {
        printf("Phan tu thu %d: ",i);scanf("%d",&X);
        InsertSET(X,L);
    }
}
```

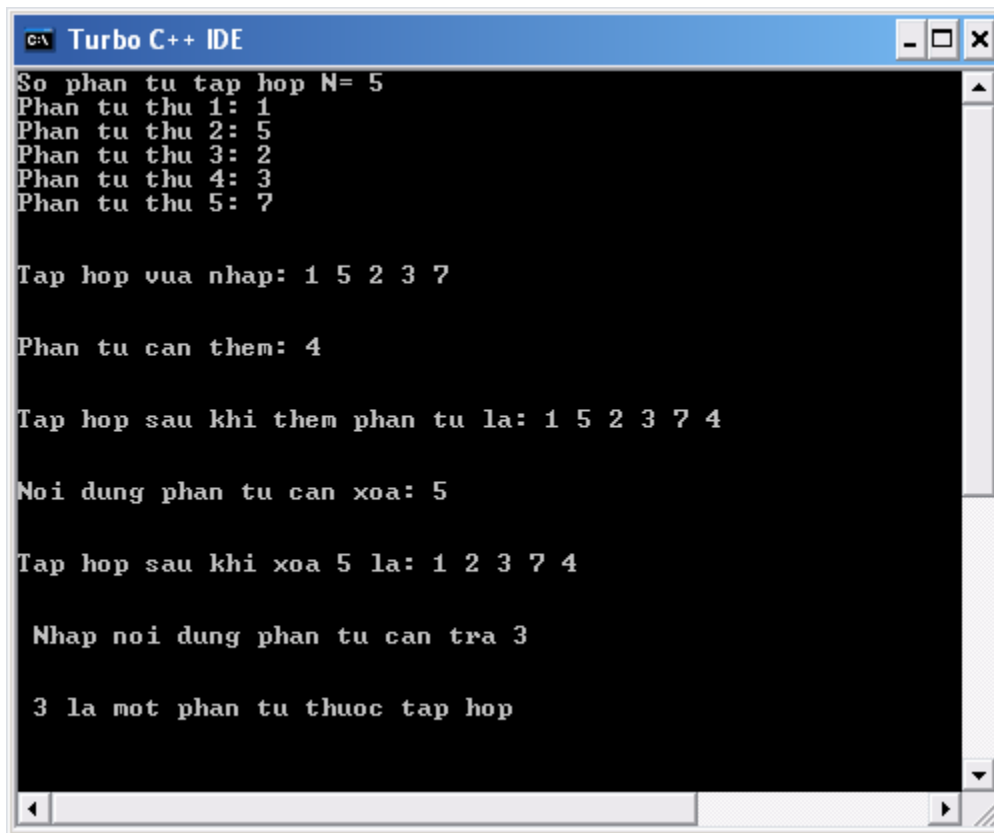
```
}  
}
```

Yêu cầu: Viết đoạn chương trình chính để kiểm tra tính đúng đắn của các phép toán vừa tạo. Chương trình chính có thể được thiết kế như sau:

```
#include <D:\DICLIB.CPP> /* Duong dan chi den thu vien chua  
phep toan co ban*/
```

```
void main()  
{  
    SET L;  
    ElementType X;  
    Position P;  
    ReadSET(&L);  
    printf("\n\nTap hop vua nhap: ");  
    PrintSET(L);  
    printf("\n\nPhan tu can them: ");scanf("%d",&X);  
    InsertSET(X,&L);  
    printf("\n\nTap hop sau khi them phan tu la: ");  
    PrintSET(L);  
    printf("\n\nNoi dung phan tu can xoa: ");scanf("%d",&X);  
    DeleteSET(X,&L);  
    printf("\n\nTap hop sau khi xoa %d la: ",X);  
    PrintSET(L);  
    printf("\n\nNhap noi dung phan tu can tra ");  
    scanf("%d",&X);  
    if (Member(X,L))  
        printf("\n\n %d la mot phan tu thuoc tap hop ",X);  
    else  
        printf("\n\n Khong ton tai phan tu co noi dung %d",X);  
    getch();  
}
```

Kết quả thực thi chương trình là:



```
C:\ Turbo C++ IDE
So phan tu tap hop N= 5
Phan tu thu 1: 1
Phan tu thu 2: 5
Phan tu thu 3: 2
Phan tu thu 4: 3
Phan tu thu 5: 7

Tap hop vua nhap: 1 5 2 3 7

Phan tu can them: 4

Tap hop sau khi them phan tu la: 1 5 2 3 7 4

Noi dung phan tu can xoa: 5

Tap hop sau khi xoa 5 la: 1 2 3 7 4

Nhap noi dung phan tu can tra 3

3 la mot phan tu thuoc tap hop
```

C. BẢNG BĂM

I. Khái niệm

Việc cài đặt tự điển bằng mảng đòi hỏi tốn n phép so sánh để xác định xem một phần tử có thuộc từ điển n phần tử hay không thông qua hàm Member. Trên từ điển, việc tìm kiếm một phần tử được xác định bằng hàm Member sẽ thường xuyên được sử dụng. Do đó, nếu hàm Member thực hiện không hiệu quả sẽ làm giảm đi ý nghĩa của từ điển (vì nói đến từ điển là phải tìm kiếm nhanh chóng). Mặt khác hàm Member còn được gọi từ trong thủ tục InsertSet và nó cũng dẫn đến là thủ tục này cũng không hiệu quả. Kỹ thuật băm cho phép chúng ta khắc phục nhược điểm trên

Băm mở là một mảng một chiều có B phần tử có chỉ số từ 0 đến B-1. Mỗi phần tử là một con trỏ, trỏ tới một danh sách liên kết mà dữ liệu sẽ của từ điển sẽ được lưu trong các danh sách liên kết này. Mỗi danh sách được gọi là một Bucket (một danh sách có chứa các phần tử có cùng giá trị hàm băm).

Bảng băm đóng lưu giữ các phần tử của từ điển ngay trong mảng chứ không dùng mảng làm các chỉ điểm đầu của các danh sách liên kết. Bucket thứ i chứa phần tử có giá trị băm là i, nhưng vì có thể có nhiều phần tử có cùng giá trị băm nên ta sẽ gặp trường hợp sau: ta muốn đưa vào bucket i một phần tử x nhưng bucket này đã bị chiếm bởi một phần tử y nào

đó (đụng độ). Như vậy khi thiết kế một bảng băm đóng ta phải có cách để giải quyết sự đụng độ này.

Cách giải quyết đụng độ đó gọi là **chiến lược băm lại** (rehash strategy). Chiến lược băm lại là chọn tuần tự các vị trí h_1, \dots, h_k theo một cách nào đó cho tới khi gặp một vị trí trống để đặt x vào. Dãy h_1, \dots, h_k gọi là dãy các phép thử. Một chiến lược đơn giản là **băm lại tuyến tính**, trong đó dãy các phép thử có dạng :

$$h_i(x) = (h(x) + i) \% B$$

II. Bài tập cơ sở

1. Bảng băm mở

Yêu cầu: tạo file tên **OHASHLIB.CPP** để lưu trữ khai báo bảng băm mở và các phép toán cơ bản trên đó. Nội dung chính của file có thể được viết như sau:

```
// Khai báo bảng băm đóng
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
#define B 10
typedef int ElementType;
typedef struct Node
{
    ElementType Data;
    Node* Next;
};
typedef Node* Position;
typedef Position Dictionary[B];

//Thiết kế hàm băm H(X)=X % B
int H(ElementType X)
{
    return X%B;
}

// Tạo bảng băm rỗng
void MakeNullSet(Dictionary *D)
```

```
{
    for(int i=0;i<B;i++)
        (*D)[i]=NULL;
}

// Kiểm tra bảng băm rỗng
int EmptySet(Dictionary D)
{
    int IsEmpty=1;
    int i=0;
    while ((i<B) && (IsEmpty))
        if (D[i]==NULL) IsEmpty=0;
        else i++;
    return IsEmpty;
}

// Kiểm tra thành viên trong bảng băm
int Member(ElementType X, Dictionary D)
{
    Position P;
    int Found=0;
    //Tìm o muc H(X)
    P=D[H(X)];
    //Duyet tren ds thu H(X)
    while((P!=NULL) && (!Found))
        if (P->Data==X) Found=1;
        else P=P->Next;
    return Found;
}

//Thêm phần tử vào bảng băm
void InsertSet(ElementType X, Dictionary *D)
{

```

```
int Bucket;
Position P;
if (!Member(X, *D))
{
    Bucket=H(X);
    P=(*D)[Bucket];
    //Cap phat o nho moi cho *D[Bucket]
    (*D)[Bucket] = (Node*)malloc(sizeof(Node));
    (*D)[Bucket]->Data=X;
    (*D)[Bucket]->Next=P;
}
}
```

// Nhập dữ liệu cho bảng băm từ bàn phím

```
void ReadSet(Dictionary *D)
{
    int i,X,N;
    MakeNullSet(D);
    printf("So phan tu N= ");scanf("%d",&N);
    for(i=1;i<=N;i++)
    {
        printf("Phan tu thu %d= ",i);scanf("%d",&X);
        InsertSet(X,D);
    }
}
```

//In nội dung trong bảng băm ra màn hình

```
void PrintSet(Dictionary D)
{
    Position P;
    for(int i=0;i<B;i++)
    {
```

```
P=D[i];
printf("Danh sach trong bucket thu %d: ",i);
while(P!=NULL)
{
    printf("%5d",P->Data);
    P=P->Next;
}
printf("\n");
}
printf("\n\n");
}
```

//Xoá phần tử ra khỏi bảng băm

```
void DeleteSet(ElementType X, Dictionary *D)
{
    int Bucket, Done;
    Position P,Q;
    Bucket=H(X);
    // Neu danh sach ton tai
    if ((*D)[Bucket]!=NULL)
    {
        // X o dau danh sach
        if ((*D)[Bucket]->Data==X)
        {
            Q=(*D)[Bucket];
            (*D)[Bucket]=(*D)[Bucket]->Next;
            free(Q);
        }
        else // Tim X
        {
            Done=0;
            P=(*D)[Bucket];
```



```
        while ((P->Next!=NULL) && (!Done))
            if (P->Next->Data==X) Done=1;
            else P=P->Next;

        // Neu tim thay
        if (Done)
        {
            //Xoa P->Next
            Q=P->Next;
            P->Next=Q->Next;
            free(Q);
        }
    }
}
```

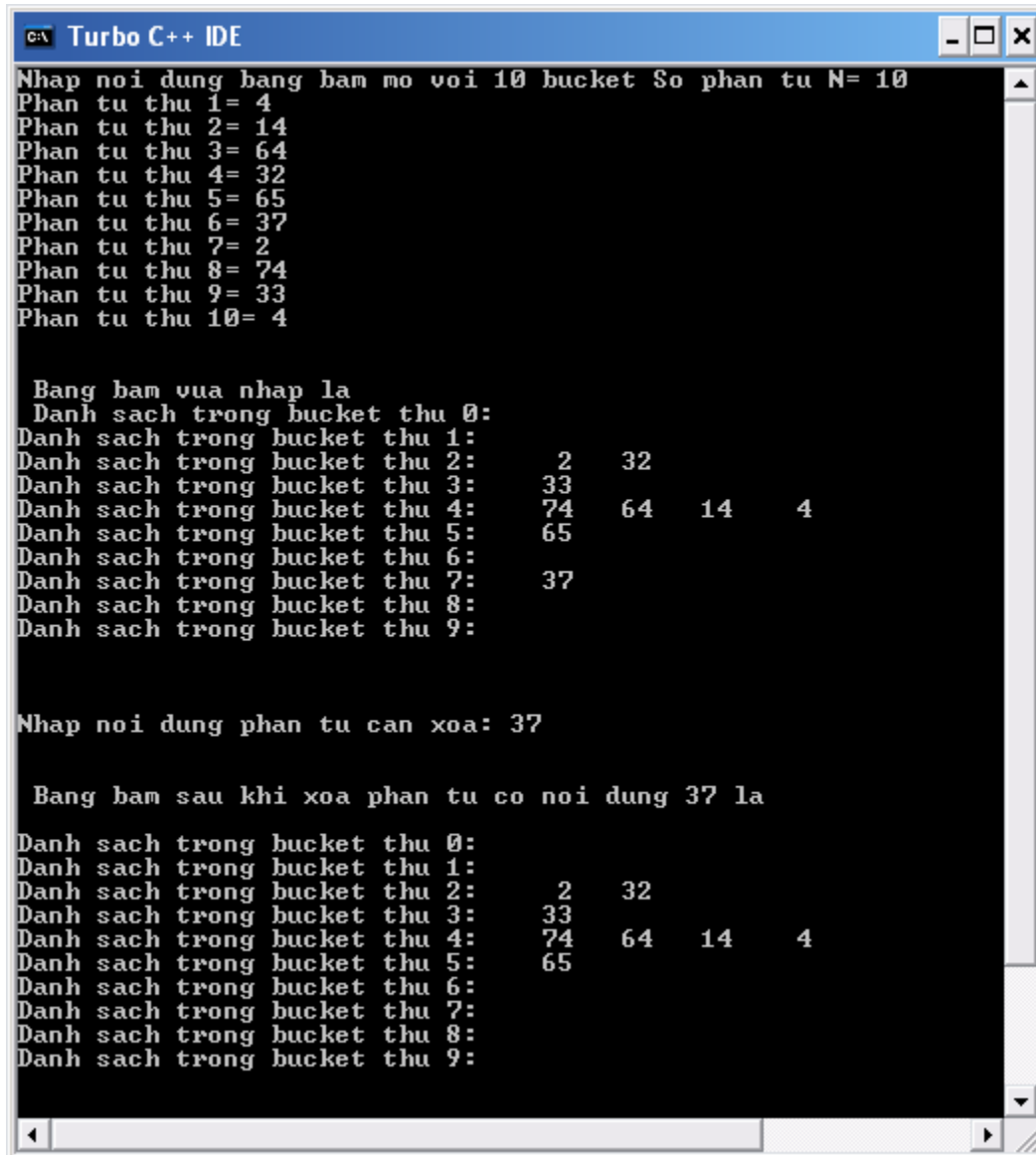
Yêu cầu: Thiết kế một chương trình chính để kiểm tra tính đúng đắn của các phép toán như sau:

```
#include <D:\OHASHLIB.CPP> /*Duong dan den thu vien chua cac
phep toan tren bang bam mo */
int main()
{
    clrscr();
    Dictionary D;
    ElementType X;
    printf("Nhap noi dung bang bam mo voi %d bucket ",B);
    ReadSet(&D);
    printf("\n\n Bang bam vua nhap la \n ");
    PrintSet(D);
    printf("\nNhap noi dung phan tu can xoa: ");
    scanf("%d",&X);
    DeleteSet(X,&D);
```

Thực hành cấu trúc dữ liệu

```
printf("\n\n Bang bam sau khi xoa phan tu co noi dung %d  
la \n\n",X);  
PrintSet(D);  
getch();  
return 0;  
}
```

Kết quả khi thực thi chương trình là:



The screenshot shows the Turbo C++ IDE window with the following output:

```
Nhap noi dung bang bam mo voi 10 bucket So phan tu N= 10  
Phan tu thu 1= 4  
Phan tu thu 2= 14  
Phan tu thu 3= 64  
Phan tu thu 4= 32  
Phan tu thu 5= 65  
Phan tu thu 6= 37  
Phan tu thu 7= 2  
Phan tu thu 8= 74  
Phan tu thu 9= 33  
Phan tu thu 10= 4  
  
Bang bam vua nhap la  
Danh sach trong bucket thu 0:  
Danh sach trong bucket thu 1:  
Danh sach trong bucket thu 2:      2      32  
Danh sach trong bucket thu 3:      33  
Danh sach trong bucket thu 4:      74      64      14      4  
Danh sach trong bucket thu 5:      65  
Danh sach trong bucket thu 6:  
Danh sach trong bucket thu 7:      37  
Danh sach trong bucket thu 8:  
Danh sach trong bucket thu 9:  
  
Nhap noi dung phan tu can xoa: 37  
  
Bang bam sau khi xoa phan tu co noi dung 37 la  
Danh sach trong bucket thu 0:  
Danh sach trong bucket thu 1:  
Danh sach trong bucket thu 2:      2      32  
Danh sach trong bucket thu 3:      33  
Danh sach trong bucket thu 4:      74      64      14      4  
Danh sach trong bucket thu 5:      65  
Danh sach trong bucket thu 6:  
Danh sach trong bucket thu 7:  
Danh sach trong bucket thu 8:  
Danh sach trong bucket thu 9:
```

2. Bảng băm đóng (Xem như bài tập tự giải)