CHƯƠNG II

CẤU TRÚC DỮ LIỆU DANH SÁCH

I. TỔNG QUAN VỀ DANH SÁCH

1. Định nghĩa

Danh sách (*List*) là dãy các phần tử

$$a_1, ..., a_n \ (n \geq 1)$$

với tính chất: nếu biết được a_i sẽ biết a_{i+1}.

Danh sách được sắp xếp tuyến tính theo vị trí của chúng.

Số các phần tử n gọi là *độ dài* của danh sách. Nếu $n = 0$, thì ta có danh sách *rỗng*.

◊ *Ví dụ*

- Danh sách nhân viên trong cơ quan.
- Danh sách sinh viên của lớp.
- Danh sách các môn học.
- Danh sách các số tự nhiên không quá 100.

2. Các phép toán trên danh sách

Để tạo lập, cập nhật, khai thác danh sách, người ta định nghĩa các phép toán sau.

a. Phép duyệt danh sách

Duyệt danh sách là duyệt qua tất cả phần tử danh sách thoả mãn <*điều kiện*> nào đó để thực hiện công việc <*xử lý*> (chẳng hạn như hiển thị phần tử).

◊ *Ví dụ*: Liệt kê danh sách nữ nhân viên trong cơ quan.

b. Phép tìm kiếm

Tìm kiếm là thao tác tìm phần tử trong danh sách thoả mãn điều kiện nào đó. Kết quả phép tìm kiếm có thể là *tìm thấy* hoặc *không tìm thấy*. Sau khi tìm thấy phần tử cần tìm ta có thể thực hiện các phép toán đối với phần tử đó như sửa đổi hay xoá.

◊ *Ví dụ*: Trong danh sách nhân viên trong cơ quan, tìm nhân viên có tên là "Trần Công Minh".

c. Thêm phần tử vào danh sách

Đây là thao tác thêm phần tử mới vào danh sách. Phần tử có thể được thêm vào cuối danh sách, đầu danh sách hoặc chèn vào giữa danh sách. Nếu danh sách đã đầy, tức là số phần tử của danh sách đã bằng số cực đại cho phép thì phép toán thêm không

thực hiện được nữa. Nếu danh sách chưa đầy thì có thể thêm phần tử mới vào danh sách và độ dài (số phần tử) của danh sách tăng thêm 1.

◊ *Ví dụ*: Trong danh sách nhân viên trong cơ quan, thêm nhân viên mới hợp đồng có tên là "Trần Công Minh".

d. Loại bỏ phần tử khỏi danh sách

Đây là thao tác loại bỏ phần tử khỏi danh sách. Trước khi loại bỏ phần tử, nếu chưa xác định được phần tử loại bỏ, thì phải thực hiện phép tìm kiếm. Nếu không tìm thấy thì không thể thực hiện phép loại bỏ. Nếu tìm thấy thì có thể thực hiện phép loại bỏ và độ dài (số phần tử) của danh sách giảm đi 1.

Phần tử loại bỏ có thể ở cuối danh sách, đầu danh sách hoặc giữa danh sách.

◊ *Ví dụ*. Trong danh sách nhân viên trong cơ quan, loại bỏ nhân viên cắt hợp đồng có tên là "Trần Công Minh".

e. Sửa đổi phần tử trong danh sách

Đây là thao tác hiệu chỉnh phần tử trong danh sách. Trước khi hiệu chỉnh phần tử, nếu chưa xác định được phần tử hiệu chỉnh, thì phải thực hiện phép tìm kiếm. Nếu không tìm thấy thì không thể thực hiện phép hiệu chỉnh. Nếu tìm thấy thì có thể thực hiện phép hiệu chỉnh. Số phần tử của danh sách không thay đổi

Phần tử hiệu chỉnh có thể ở cuối danh sách, đầu danh sách hoặc giữa danh sách.

◊ *Ví dụ*. Trong danh sách nhân viên trong cơ quan, hiệu chỉnh mức lương cho nhân viên có tên là "Trần Công Minh".

f. Sắp xếp thứ tự danh sách

Đây là thao tác sắp xếp lại thứ tự các phần tử trong danh sách theo một *khoá* nào đó. Thứ tự phần tử có thể thay đổi, nhưng số phần tử vẫn giữ nguyên.

◊ *Ví dụ*: Sắp xếp danh sách nhân viên trong cơ quan theo họ tên hoặc theo mức lương.

g. Tách một danh sách thành nhiều danh sách

Đây là thao tác tách một phần hoặc lấy tất cả phần tử của một danh sách đưa sang danh sách khác.

◊ *Ví dụ*. Từ danh sách nhân viên trong cơ quan tách riêng danh sách nữ nhân viên.

h. Ghép nhiều danh sách thành danh sách mới

Đây là thao tác lấy tất cả phần tử từ các danh sách khác nhau tạo thành một danh sách mới. Các phần tử của cùng một danh sách con vẫn xếp cạnh nhau.

◊ *Ví dụ*. Từ danh sách nhân viên của các phòng ghép lại thành danh sách nhân viên của toàn cơ quan.

i. Trộn nhiều danh sách thành danh sách mới

Đây là thao tác lấy tất cả phần tử từ các danh sách khác nhau tạo thành một danh sách mới theo một cấu trúc nào đó. Các phần tử của cùng một danh sách con có thể sẽ bị xáo trộn.

◊ *Ví dụ*. Từ danh sách nhân viên của các phòng đã được sắp xếp theo họ tên trộn lại thành danh sách nhân viên của toàn cơ quan sắp xếp theo họ tên.

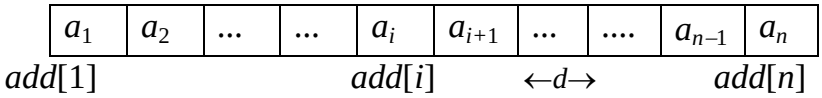
II. DANH SÁCH ĐẶC

1. Định nghĩa và khai báo

a. Định nghĩa

Danh sách đặc (*condensed list*) hay còn gọi là *danh sách kề* (*contiguous list*) là danh sách mà các phần tử được lưu trữ kế tiếp nhau trong bộ nhớ, phần tử thứ i+1 đứng ngay sau phần tử thứ i.

Hình sau mô tả danh sách đặc, trong đó các phần tử có kích thước bằng nhau và xếp kế tiếp nhau.



Nếu các phần tử có độ dài d byte như nhau và ký hiệu địa chỉ phần tử i là *add*[i], thì ta sử dụng công thức tính địa chỉ như sau:

$$add[i] = add[1] + (i-1)*d$$

b. Khai báo

◊ Trong PASCAL:

```
Const   ElemNo = <số phần tử tối đa của danh sách>;
Type
    InfoType = <kiểu dữ liệu của các trường không khóa>;
    KeyType = <kiểu dữ liệu của trường khóa>;
    Element =  record
                    key : KeyType; {khoá}
                    info: InfoType; {dữ liệu}
               end;
    ListType = array[1 .. ElemNo] of Element;
Var List: ListType;
    ListNum: integer; {số phần tử của danh sách,  không được vượt quá ElemNo}
```

◊ Trong C:

```
#define  ElemNo  <số phần tử tối đa của danh sách>
typedef  <kiểu dữ liệu của các trường không khóa > InfoType;
typedef  <kiểu dữ liệu của các trường khóa > KeyType;
struct  Element  {
                    KeyType   key; //khoá
                    InfoType  info; // dữ liệu
                };
struct Element List[ElemNo+1];
int ListNum; // số phần tử của danh sách,  không được vượt quá  ElemNo
```

2. Các phép toán

a. Khởi tạo danh sách

Ban đầu danh sách rỗng, nên ta khởi tạo đặt độ dài danh sách *ListNum* = 0.

Trong PASCAL	Trong C
procedure Initialize; *begin* *ListNum* := 0; *end;*	*void Initialize*() { *ListNum* = 0; }

b. Duyệt danh sách

Duyệt danh sách thoả mãn *<điều kiện>* để *<xử lý>*

Trong PASCAL	Trong C
procedure Traverse; *var i: integer; {biến cục bộ}* *begin* *for i* := 1 *to ListNum do* *if <điều kiện> then* *<xử lý List*[*i*]*>* *end;*	*void Traverse*() { *int i; //biến cục bộ* *for* (*i* = 1; *i* <= *ListNum; i*++) *if <điều kiện>* { *<xử lý List*[*i*]*>* } }

c. Tìm kiếm tuyến tính

Giả sử ta phải tìm phần tử có trường *Key* = *KeyValue*.

Hàm *Search* trả về chỉ số của phần tử tìm thấy đầu tiên, hoặc trả về giá trị –1 nếu không tìm thấy.

• *Trường hợp danh sách không thứ tự*

◊ Trong PASCAL:

```
function Search(KeyValue: KeyType): integer;
var vitri: integer;
begin
    vitri := 1;
    while (vitri <= ListNum) and (List[vitri].key <> KeyValue) do
            vitri := vitri + 1;
    if (vitri <= ListNum) then
        search := vitri
    else
        search := -1;{không tìm thấy}
end;
```

◊ Trong C:

```
int Search(KeyType KeyValue)
{
    int vitri;
    vitri = 1;
    while ((vitri <= ListNum) && (List[vitri].Key != KeyValue)) vitri++;
    if (vitri == ListNum+1) return -1; //không tìm thấy
    else  return vitri; //tìm thấy
}
```

• *Trường hợp danh sách có thứ tự*

◊ Trong PASCAL:

```
function Search(KeyValue: KeyType): integer;
var found: boolean;
    vitri : integer;
begin
    found := false;
    vitri := 1;
    while (vitri <= ListNum) and (not found) do
        if  List[vitri].key = KeyValue then
            found := true
        else
            if  (List[vitri].key > KeyValue) then
                vitri := ListNum + 1
            else
                vitri := vitri + 1;
     if found then
        search := vitri
    else
        search := -1; {không tìm thấy}
end;
```

◊ Trong C:

```
int Search(KeyType KeyValue)
{
    int vitri;
    vitri = 1;
    while ((vitri <= ListNum) && (List[vitri].Key < KeyValue)) vitri++;
    if (vitri == ListNum+1) return -1; //không tìm thấy
    else  return (List[vitri].Key == KeyValue ? vitri : -1);
}
```

• Độ phức tạp:

- Độ phức tạp trung bình : O(n/2)
- Độ phức tạp xấu nhất : O(n)

◊ *Ghi chú : n* ký hiệu số phần tử của danh sách.

d. Tìm kiếm nhị phân

Phương pháp này chỉ áp dụng với danh sách đặc. Điều kiện để có thể áp dụng giải thuật là danh sách phải được sắp xếp thứ tự theo trường tìm kiếm *Key*.

Giả sử ta phải tìm phần tử có trường *Key = KeyValue*.

Hàm *Search* trả về chỉ số của phần tử tìm thấy đầu tiên, hoặc trả về giá trị -1 nếu không tìm thấy.

Trong PASCAL

```
function Search(KeyValue: KeyType): integer;
var found: boolean;
    i, min, max : integer;
begin
    found := false;
    min := 1;
    max := ListNum;
    while (min <= max) and (not found) do
      begin
        i := (min + max) div 2;
        if List[i].key = KeyValue then
            found := true
        else
          if List[i].key < KeyValue then
            min := i + 1
          else
            max := i – 1;
      end;
    if found then
        search := i
    else
        search := –1; {không tìm thấy}
end;
```

Trong C

```
int Search(KeyType KeyValue)
{
    int found;
    int i, min, max ;
    found = 0;
    min = 1;
    max = ListNum;
    while ((min <= max) && (!found))
      {
        i = (min + max) / 2;
        if (List[i].key == KeyValue)
          found = 1;
        else
          if (List[i].key < KeyValue)
            min = i + 1;
          else
            max = i – 1;
      }
    if ( found)
        return i ;
    else
        return –1;
}
```

◊ Độ phức tạp thuật toán là : $O(\log_2(n))$.

e. Tìm kiếm nội suy

Yêu cầu để có thể áp dụng giải thuật là danh sách phải được sắp xếp thứ tự.

Giải thuật là cải tiến của phương pháp tìm kiếm nhị phân.

Xét biểu thức tính điểm giữa i ở thuật toán nhị phân

$$(min + max)\ div\ 2$$

Biểu thức trên có thể xấp xỉ bằng

$$min + 0.5\ (max - min)$$

Trong thuật toán này ta thay hệ số 0.5 bằng

$$k = \frac{KeyValue - List[\min].key}{List[\max].key - List[\min].key}$$

◊ Trong PASCAL:

```
function Search(KeyValue: KeyType): integer;
var found: boolean;
    i, min, max : integer;
    k : real;
begin
    found := false;
    min := 1;
    max := ListNum;
    while (min <= max) and (not found) do
        begin
            k := (KeyValue - List[min].key) / (List[max].key - List[min].key);
            i := min + round(k * (max - min));
            if (List[i].key = KeyValue) then found := true
            else
                if (List[i].key < KeyValue) then
                    min := i + 1
                else
                    max := i - 1;
        end;
    if found then
        search := i
    else
        search := -1;
end;
```

◊ Trong C:

```
int Search(KeyType KeyValue)
{
    int found ;
    int i, min, max ;
```

```
        float k ;
        found = 0;
        min = 1;
        max = ListNum;
        while ((min <= max) && !found)
            {
                k = (KeyValue - List[min].key) / (List[max].key - List[min].key);
                i = min + k * (max - min);
                if  (List[i].key == KeyValue)
                    found = 1;
                else
                    if  (List[i].key < KeyValue)
                        min = i + 1;
                    else
                        max = i - 1;
            }
        if (found) then
            return  i;
        else
            return -1;
    }
```

◊ Độ phức tạp là : $O(\log_2(n))$

f. Chèn phần tử vào danh sách

Giả sử ta chèn phần tử mới *NewElem* vào vị trí thứ *k*. Khi đó các phần tử từ List[*k*] đến List[*ListNum*] được di chuyển ra sau 1 vị trí và số phần tử danh sách *ListNum* tăng lên 1.

Trong PASCAL	Trong C
procedure Insert(k:integer;NewElem:Element); *var i: integer;* *begin* *for i:= ListNum downto k do* *List[i+1] := List[i];* *List[k] := NewElem;* *ListNum := ListNum + 1;* *end;*	*void Insert(int k, Element NewElem)* { *int i;* *for (i = ListNum; i >= k; i--)* *List[i+1] = List[i];* *List[k] = NewElem;* *ListNum++;* }

◊ Độ phức tạp

- Độ phức tạp trung bình : $O(n/2)$
- Độ phức tạp xấu nhất : $O(n)$

g. Loại phần tử khỏi danh sách

Giả sử ta loại phần tử *List*[*k*] khỏi danh sách. Khi đó các phần tử từ *List*[*k*+1] đến *List*[*ListNum*] được di chuyển về trước 1 vị trí và số phần tử danh sách *ListNum* giảm đi 1.

Trong PASCAL

```
procedure Delete(k: integer);
var  i: integer;
begin
     for i:= k to ListNum-1 do
          List[i] := List[i+1];
     ListNum  := ListNum - 1;
end;
```

Trong C

```
void Delete(int k)
{
     int  i;
     for  (i  =  k;  i  <=  ListNum–1;  i++)
          List[i] = List[i+1];
     ListNum--;
}
```

◊ Độ phức tạp

- Độ phức tạp trung bình : O(*n*/2)
- Độ phức tạp xấu nhất : O(*n*)

h. Trộn danh sách

Giả sử có hai danh sách số nguyên *a*[1..*m*] và *b*[1..*n*] đã sắp xếp tăng dần. Hãy trộn hai danh sách này thành danh sách *c*[1..*m*+*n*] cũng được sắp xếp tăng dần.

Trong PASCAL

```
procedure  TwoListMerging;
var  i1,i2,i : integer;
begin
     i1:=1; i2:=1;i:=1;
     while ((i1 <=m) and (i2 <=n)) do
     begin
          if (a[i1]< b[i2]) then
            begin c[i]:=a[i1]; inc(i1)  end
          else
            begin c[i]:=b[i2]; inc(i2)  end;
          inc(i);
     end;
     while (i1<=m) do{kiểm tra danh sách a}
          begin c[i]:=a[i1]; inc(i1);inc(i) end;
     while (i2<=n) do{kiểm tra danh sách b}
          begin c[i]:=b[i2]; inc(i2);inc(i) end;
end;
```

Trong C

```
void TwoListMerging()
{
     int i1,i2,i;
     i=i1=i2=1;
     while ((i1 <=m) && (i2 <=n))
     {
          if (a[i1]< b[i2])
               c[i++]=a[i1++];
          else
               c[i++]=b[i2++];
     }
   while (i1<=m)//kiểm tra danh sách a
          c[i++]=a[i1++];
   while (i2<=n)//kiểm tra danh sách b
          c[i++]=b[i2++];
}
```

◊ Độ phức tạp là : O(m+n)

3. Đặc điểm của danh sách đặc

a. Ưu điểm

Sử dụng 100% dung lượng ô nhớ để lưu trữ thông tin (không tốn bộ nhớ lưu địa chỉ như danh sách liên kết).

Truy xuất trực tiếp phần tử *List*[*i*].

Dễ dàng tìm kiếm phần tử bằng phương pháp nhị phân, nội suy, nếu danh sách có thứ tự.

Dễ dàng cài đặt các phương pháp sắp xếp.

b. Nhược điểm

Lãng phí bộ nhớ.

Không phù hợp với phép chèn và loại bỏ. Số lần di chuyển trung bình cho một phép chèn hoặc loại bỏ là n/2.

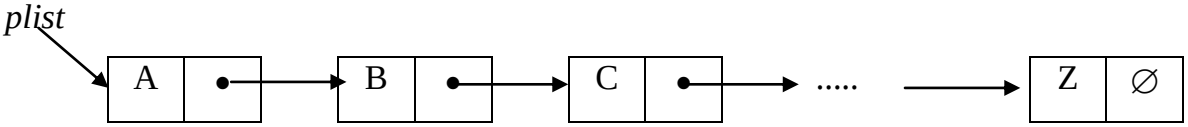
III. DANH SÁCH LIÊN KẾT ĐƠN

1. Định nghĩa và khai báo

a. Định nghĩa: *danh sách liên kết đơn* (*simple linked list*) là danh sách mà các phần tử được nối kết với nhau nhờ các địa chỉ liên kết.

Mỗi phần tử của danh sách gọi là *nút* gồm có hai phần: Phần thông tin và phần kết nối chứa địa chỉ của nút tiếp theo.

Hình sau minh hoạ danh sách liên kết. Mỗi nút có 2 ô, ô trái chứa thông tin, mũi tên ô phải chỉ liên kết đến nút kế tiếp. Ký hiệu ∅ có nghĩa là nút không kết nối đến phần tử khác.



b. Khai báo

Trong PASCAL	Trong C
Type InfoType = <kiểu dữ liệu chứa trong nút>; *NodePointer = ^Node;* *Node = record* *info : InfoType;* *next : NodePointer;* *end;* *Var* *plist, p : NodePointer; {plist là con trỏ danh sách}* *.....* *new(p);{cấp phát nút động cho p}* *...* *dispose(p); {giải phóng nút p}*	*typedef <kiểu dữ liệu chứa trong nút> InfoType;* *struct Node {* *InfoType info;* *struct Node *next;* *};* *typedef struct Node *NodePointer;* *NodePointer plist, p;/* plist là con trỏ danh sách */* *…* *p = (NodePointer)malloc(sizeof(struct Node)); /*cấp phát nút động cho p*/* *...* *free(p); //giải phóng nút p*

2. Các phép toán

a. Khởi tạo: danh sách rỗng, *plist* là con trỏ danh sách.

Trong PASCAL	Trong C
procedure Initialize; *begin* *plist := nil;* *end;*	*void Initialize()* { *plist = NULL;* }

b. Duyệt danh sách

Duyệt các phần tử thoả mãn <*điều kiện*> để thực hiện công việc <*xử lý*>.

Trong PASCAL

```
procedure Traverse;
var  p : NodePointer;
begin
     p := plist;
     while  (p <> nil)  do
          begin
               if  <điều kiện> then
                    <xử lý>
               p := p^.next;
          end;
end;
```

Trong C

```
void Traverse()
{
     NodePointer   p;
     p = plist;
     while  (p != NULL)
          {
               if  <điều kiện>
                    <xử lý>;
               p = p->next;
          }
}
```

◊ *Ví dụ*. Tính số nút của danh sách.

Trong PASCAL

```
function ListNum  : integer;
var count:integer; p:NodePointer;
begin
     p:=plist; count:=0;
     while  (p <> nil)  do
          begin
               count:=count+1;
               p := p^.next;
          end;
     ListNum := count;
end;
```

Trong C

```
int ListNum()
{
     NodePointer   p; int count;
     p = plist; count = 0;
     while  (p != NULL)
          {
               count++;
               p = p->next;
          }
     return count;
}
```

c. Tìm kiếm

Giả sử ta phải tìm phần tử có *info = InfoValue*.

Hàm *Search* trả về địa chỉ của phần tử tìm thấy đầu tiên, hoặc trả về giá trị *nil* (trong PASCAL) hoặc NULL (trong C) nếu không tìm thấy.

• *Trường hợp danh sách không thứ tự*

Trong PASCAL

```
function Search(InfoValue:InfoType):
NodePointer;
var found: boolean; p : NodePointer;
begin
    found := false;
    p := plist;
    while (p <> nil) and (not found) do
        if p^.info = InfoValue then
            found := true
        else
            p := p^.next;
    Search := p;
end;
```

Trong C

```
NodePointer Search(InfoType InfoValue)
{
    int found; NodePointer p;
    found = 0;
    p = plist;
    while ((p != NULL) && (! found))
        {
            if (p->info == InfoValue)
                found = 1;
            else
                p = p->next;
        }
    return p;
}
```

- *Trường hợp danh sách có thứ tự* (*tăng dần*)

Trong PASCAL

```
function Search(InfoValue: InfoType);
NodePointer;
var p : NodePointer;
begin
    p := plist;
    while (p <> nil) and (p^.info <
    InfoValue) do
        p := p^.next;
    if (p <> nil) and (p^.info >
    InfoValue) then
        p := nil;
    search := p;
end;
```

Trong C

```
NodePointer Search(InfoType InfoValue)
{
    NodePointer p;
    p = plist;
    while ((p != NULL) && (p->info <
    InfoValue))
        p = p->next;
    if ((p != NULL) && (p->info >
    InfoValue))
        p = NULL;
    return p;
}
```

d. Chèn phần tử vào danh sách

Giả sử ta chèn phần tử có nội dung *NewInfo* vào danh sách.

- *Trường hợp chèn vào đầu danh sách*

Trong PASCAL

```
procedure Insert(NewInfo: InfoType);
var  p : NodePointer;
begin
     new(p);
     p^.info := NewInfo;
     p^.next := plist;
     plist := p;
end;
```

Trong C

```
void Insert(InfoType NewInfo)
{
     NodePointer p;
     p = (NodePointer)malloc(sizeof(struct
     Node));
     p->info = NewInfo; p->next = plist;
     plist = p;
}
```

- *Trường hợp chèn vào cuối danh sách*

Trong PASCAL

```
procedure  Insert(NewInfo: InfoType);
var  p, q : NodePointer;
begin
     new(p); {tạo nút p}
     p^.info := NewInfo;
     p^.next := nil;
     {tìm nút cuối danh sách q}
     q := plist;
     while  (q^.next <> nil)  do
          q := q^.next;
     q^.next := p; {liên kết nút q đến p;
     p trở thành nút cuối của danh
     sách}
end;
```

Trong C

```
void  Insert(InfoType NewInfo)
{
     NodePointer p, q;
     p = (NodePointer)malloc(sizeof(struct
     Node));
     p->info = NewInfo;
     p->next = NULL;
     // tìm nút cuối danh sách q
     q = plist;
     while  (q->next != NULL)
          q = q->next;
     q->next = p;// liên kết nút q đến p; p trở
thành nút cuối của danh sách
}
```

- *Trường hợp chèn vào sau phần tử q*

Trong PASCAL

```
procedure Insert(NewInfo: InfoType;
q: NodePointer);
var  p : NodePointer;
begin
     new(p);
     p^.info := NewInfo;
     p^.next := q^.next;
     q^.next := p;
end;
```

Trong C

```
void  Insert(InfoType  NewInfo, NodePointer q)
{
     NodePointer  p;
     p = (NodePointer)malloc(sizeof(struct
     Node));
     p->info = NewInfo;
     p->next = q->next;
     q->next = p;
}
```

e. Loại phần tử khỏi danh sách

- *Trường hợp loại phần tử đầu danh sách*

Trong PASCAL

```
procedure Delete;
var  p : NodePointer;
begin
     p := plist;
     plist := p^.next;
     dispose(p);
end;
```

Trong C

```
void Delete()
{
     NodePointer p;
     p = plist;
     plist = p->next;
     free(p);
}
```

- *Trường hợp loại phần tử sau phần tử q*

Trong PASCAL

```
procedure Delete(q : NodePointer);
var  p : NodePointer;
begin
     p := q^.next;
     if  p <> nil then
          begin
               q^.next := p^.next;
               dispose(p);
          end;
end;
```

Trong C

```
void Delete(NodePointer q)
{
     NodePointer p;
     p = q->next;
     if  (p != NULL)
          {
               q->next = p->next;
               free(p);
          }
}
```

f. Ghép nhiều danh sách thành danh sách mới

- Ghép danh sách có chỉ điểm đầu *plist2* vào sau phần tử *q* của danh sách khác có chỉ điểm đầu là *plist1*.

Trong PASCAL

```
procedure Insert(q: NodePointer);
var p : NodePointer;
begin
     if plist2 <> nil then
          begin
               p := plist2;
               {tìm  phần tử  cuối}
               while p^.next <> nil do
                    p := p^.next;
               p^.next := q^.next;
               q^.next := plist2;
          end;
end;
```

Trong C

```
void Insert(NodePointer q)
{
     NodePointer p;
     if (plist2 != NULL)
          {
               p = plist2;
               /*tìm  phần tử  cuối*/
               while (p->next != NULL)
                    p = p->next;
               p->next = q->next;
               q->next = plist2;
          }
}
```

• Ghép danh sách có chỉ điểm đầu *plist2* vào sau phần tử cuối của danh sách khác có chỉ điểm đầu là *plist1*.

Trong PASCAL	Trong C
procedure Insert; *var p : NodePointer;* *begin* *if plist2 <> nil then* *begin* *p := plist1;* *{*tìm phần tử cuối của danh sách *1}* *while p^.next <> nil do* *p := p^.next;* *p^.next := plist2;* *end;* *end;*	*void Insert*( ) { *NodePointer p;* *if* (*plist2* != *NULL*) { *p = plist1;* /*tìm phần tử cuối danh sách *1*/ *while* (*p->next* != *NULL*) *p = p->next;* *p->next = plist2;* } }

g. Trộn nhiều danh sách thành danh sách mới

Giả sử ta cần trộn hai danh sách có cùng thứ tự tăng dần của trường *Info* với hai chỉ điểm đầu *plist1*, *plist2* thành danh sách thứ tự với chỉ điểm đầu là *plist3*.

Trong PASCAL	Trong C
procedure Merge_List; *var p1,p2,p3 : NodePointer;* *begin* *new*(*plist3*); *p3 := plist3;* *p1 := plist1;* *p2 := plist2;* *while* (*p1 <> nil*) *and* (*p2 <> nil*) *do* *if p1^.info > p2^.info then* *begin* *p3^.next := p2;* *p3 := p2;* *p2 := p2^.next;* *end* *else* *begin* *p3^.next := p1;* *p3 := p1;* *p1 := p1^.next;* *end; {*kết thúc *while}*	*void Merge_List*() { *NodePointer p1,p2,p3 ;* *plist3 =* (*NodePointer*)*malloc*(*sizeof*(*struct Node*)); *p3 = plist3;* *p1 = plist1;* *p2 = plist2;* *while* ((*p1* != *NULL*)&& (*p2* != *NULL*)) *if* (*p1->info > p2->info*) { *p3->next = p2;* *p3 = p2;* *p2 = p2->next;* } *else* { *p3->next = p1;* *p3 = p1;* *p1 = p1->next;*

```
    if  p1 = nil then
        p3^.next := p2
    else
        p3^.next := p1;
    p3 := plist3;
    plist3 := plist3^.next; {gán phần tử
    đầu}
    dispose(p3);
end;
```

```
        }
    if  (p1 == NULL)
        p3->next = p2;
    else
        p3->next = p1;
    p3 = plist3;
    plist3 = plist3->next; //gán phần tử đầu
    free(p3);
}
```

3. Đặc điểm của danh sách liên kết

a. Ưu điểm

- Thích hợp phép chèn, loại bỏ, trộn, ghép danh sách.
- Rất phù hợp với các loại danh sách có nhiều biến động.
- Sử dụng bộ nhớ lưu các nút trong danh sách linh hoạt và tiết kiệm.

b. Nhược điểm

- Tốn vùng nhớ cho chỉ điểm liên kết.
- Không thích hợp cho tìm kiếm.

IV. DANH SÁCH ĐA LIÊN KẾT

1. Định nghĩa

Danh sách đa liên kết là danh sách mà các phần tử được nối kết với nhau nhờ nhiều vùng liên kết.

◊ *Ví dụ:* Danh sách số điện thoại.

Ta cần in danh sách theo khách hàng hoặc theo số điện thoại. Khi đó tổ chức lưu trữ bằng danh sách đa liên kết sẽ thuận lợi hơn.

2. Tổ chức danh sách đa liên kết

Mỗi phần tử của danh sách đa liên kết bao gồm một vùng thông tin *info* và các vùng liên kết *next1*, *next2*, ..., *nextn* và ứng với các vùng liên kết ta có các chỉ điểm đầu *plist1*, *plist2*, ..., *plistn*.

Phần tử của danh sách đa liên kết có dạng:

info	*next1*	*next2*	.....	*nextn*

- ***Khai báo***

Trong PASCAL

```
Type InfoType =      record
                        info1 : ...;
                        info2 : ...;
                            ...
                end;
    NodePointer  = ^Node;
    Node = record
                info : InfoType;
                next1 : NodePointer;
                next2 : NodePointer;
                            . . .
            end;
Var
    plist1, plist2, ... : NodePointer;
```

Trong C

```
typedef  struct data
          { ... info1;
            ... info2;
            ...
          }  InfoType;
struct  Node {
                    InfoType info;
                    struct Node *next1;
                    struct Node *next2;
                    ...
                };
typedef struct Node *NodePointer;
NodePointer plist1, plist2, ... ;// plist1, plist2, ... là con trỏ danh sách
```

3. Các phép toán

a. Khởi tạo: danh sách rỗng.

Trong PASCAL Trong C

Procedure Initialize; *begin* *plist1 := nil;* *plist2 := nil;* *...* *end;*	*void Initialize*() { *plist1 = NULL;* *plist2 = NULL;* *...* }

b. Duyệt danh sách

Duyệt danh sách theo từng liên kết, giải thuật giống như giải thuật duyệt danh sách liên kết đơn.

c. Chèn phần tử vào danh sách

Xét danh sách đa liên kết có *hai* vùng liên kết *next1* và *next2* đã sắp xếp theo thứ tự tăng dần theo *info1* và *info2* trong bản ghi kiểu InfoType.

Giả sử ta chèn phần tử có nội dung *NewInfo* với giá trị *NewInfo1* và *NewInfo2* vào danh sách.

Hàm *Insert_Element* trả về địa chỉ của phần tử mới thêm vào.

◊ Trong PASCAL:

```
Function Insert_Element(NewInfo: InfoType): NodePointer;
var  p , {*phần tử mới thêm vào*}
     q , {*phần tử phụ trợ*}
     before  {*phần tử đứng trước p *}
          : NodePointer;
begin
     new(p);
     p^.info := NewInfo; {* ghi dữ liệu vào phần tử p *}
     {tìm phần tử trước p theo next1 }
     q := plist1;
     while (q <> nil) and (q^.info.info1 < NewInfo1) do
          begin
               before := q;
               q := q^.next1
          end;
     if q = plist1 then
          plist1 := p
     else
          before^.next1 := p;
     p^.next1 := q;
     {tìm phần tử trước p theo next2 }
     q := plist2;
     while (q <> nil) and (q^.info.info2 < NewInfo2) do
```

```
            begin
                before := q;
                q := q^.next2
            end
        if q = plist2 then
            plist2 := p
        else
            before^.next2 := p;
        p^.next2 := q;
        Insert_Element := p
    end;
```

◊ Trong C:

```
    NodePointer  Insert_Element(InfoType NewInfo)
    {
        NodePointer   p , /*phần tử mới thêm vào*/
        q , /*phần tử phụ trợ*/
        before; /*phần tử đứng trước p */
        p = (NodePointer)malloc(sizeof(struct Node));
        p->info = NewInfo;/* ghi dữ liệu vào phần tử p */
        //tìm phần tử trước p theo next1
        q = plist1;
        while ((q != NULL) && (q->info.info1 < NewInfo1))
        {
            before = q;
            q = q->next1;
        }
        if (q == plist1)
            plist1 = p;
        else
            before->next1 = p;
        p->next1 = q;
        //tìm phần tử trước p theo next2
        q = plist2;
        while ((q != NULL) && (q->info.info2 < NewInfo2))
        {
            before = q;
            q = q->next2;
        }
        if (q == plist2)
            plist2 = p;
        else
            before->next2 = p;
        p->next2 = q;
        return  p;
```

}

d. Loại phần tử khỏi danh sách

Xét danh sách đa liên kết có *hai* vùng liên kết *next1* và *next2* đã sắp xếp theo thứ tự tăng dần theo *info1* và *info2* trong bản ghi kiểu InfoType.

Giả sử ta cần loại bỏ phần tử có nội dung *DelInfo*. Thủ tục *Delete_Element* sẽ cài đặt thuật toán xoá phần tử khỏi danh sách.

◊ Trong PASCAL:

```
Procedure Delete_Element(DelInfo : InfoType);
var  q , {* phần tử phụ trợ dò tìm *}
     q1 , {* phần tử tìm thấy theo next1 *}
     before  {* phần tử  đứng trước q *}
          : NodePointer;
begin
     q := plist1;
     { dò tìm theo vùng next1 }
     while (q <> nil) and (q^.info <> DelInfo) do
          if q^.info.info1 > DelInfo.info1 then
               q := nil
          else
               begin
                    before := q;
                    q := q^.next1
               end;
     if q <> nil then { tìm thấy phần tử cần loại bỏ }
          begin
               q1 := q;
               { ngắt next1 }
               if q = plist1 then
                    plist1 := q^.next1
               else
                    before^.next1 := q^.next1;
               { dò tìm theo vùng next2 }
               q := plist2;
               while q <> q1 do
                    begin
                         before := q;
                         q := q^.next2
                    end;
               if q = plist2 then
                    plist2 := q^.next2
               else
                    before^.next2 := q^.next2;
```

```
            end;
        dispose(q)
    end;
```

◊ Trong C:

```
    void Delete_Element(InfoType DelInfo)
    {
        NodePointer  q , /* phần tử phụ trợ dò tìm */
        q1 , /* phần tử tìm thấy theo next1 */
        before; /* phần tử  đứng trước q */
        q = plist1;
        // dò tìm theo vùng next1
        while ((q != NULL) && (q->info != DelInfo))
            if  (q->info.info1 > DelInfo.info1)
                q = NULL;
            else
            {
                before = q;
                q = q->next1;
            }
        if (q != NULL) // tìm thấy phần tử cần loại bỏ
        {
            q1 = q;
            // ngắt next1
            if  (q == plist1)
                plist1 = q->next1;
            else
                before->next1 = q->next1;
            //dò tìm theo vùng next2
            q = plist2;
            while  (q != q1)
            {
                before = q;
                q = q->next2;
            }
            if  (q == plist2)
                plist2 = q->next2;
            else
                before->next2 = q->next2;
        }
        free(q);
    }
```

V. DANH SÁCH LIÊN KẾT KÉP

1. Định nghĩa

Danh sách liên kết kép (*doubly linked list*) là danh sách mà mỗi phần tử có hai vùng liên kết. Một vùng liên kết chỉ đến phần tử đứng ngay trước nó, gọi là *liên kết ngược* (*previous*) và một vùng liên kết chỉ đến phần tử đứng ngay sau nó, gọi là *liên kết thuận* (*next*).

◊ *Ví dụ*. Danh sách có thể thêm vào hoặc lấy ra ở hai đầu.

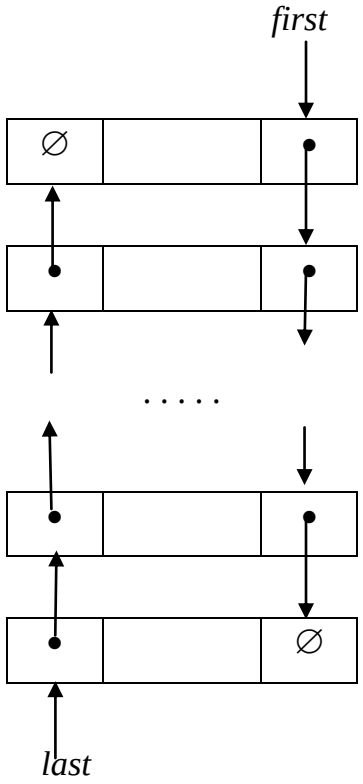
2. Tổ chức danh sách liên kết kép

Mỗi phần tử của danh sách liên kết kép bao gồm một vùng thông tin *info* và các vùng liên kết *previous* và *next* và ứng với các vùng liên kết ta có các chỉ điểm đầu *first* và chỉ điểm cuối *last*.

Phần tử của danh sách liên kết kép có dạng:

previous	*info*	*next*

Tổ chức danh sách liên kết kép như sau:



- ***Khai báo***

Trong PASCAL	Trong C
Type InfoType = record *Info1 : ...;* *Info2 : ...;* *...* *end;* *NodePointer = ^Node;* *Node = record* *info : InfoType;* *previous : NodePointer;* *next : NodePointer;* *end;* *Var first, last : NodePointer;{ con trỏ đầu và con trỏ cuối danh sách}*	*typedef struct data* *{ ... info1;* *... info2;* *...* *} InfoType;* *struct Node {* *InfoType info;* *struct Node *previous;* *struct Node *next;* *};* *typedef struct Node *NodePointer;* *NodePointer first, last, ... ;// con trỏ đầu và con trỏ cuối danh sách*

3. Các phép toán

a. Khởi tạo: danh sách rỗng.

Trong PASCAL	Trong C
procedure Initialize; *begin* *first := nil; last := nil;* *end;*	*void Initialize*() { *first = NULL; last = NULL;* }

b. Duyệt danh sách

Duyệt danh sách theo liên kết thuận hoặc theo liên kết ngược, giải thuật giống như giải thuật duyệt danh sách liên kết đơn.

c. Chèn phần tử vào danh sách:

Giả sử ta chèn phần tử có nội dung *NewInfo*.

- *Trường hợp thêm phần tử vào đầu danh sách*

Hàm *Insert_First* trả về địa chỉ của phần tử mới thêm vào.

Trong PASCAL	Trong C
function Insert_First(NewInfo: InfoType): NodePointer; *var p {*phần tử mới *}: NodePointer;* *begin* *new(p);* *p^.info := NewInfo;{* ghi dữ liệu vào phần tử p *}* *p^.previous := nil;* *p^.next := first;* *if first <> nil then* *first^.previous := p;* *first := p;* *if last = nil then* *last := p;* *Insert_First := p* *end;*	*NodePointer Insert_First(InfoType NewInfo)* { *NodePointer p ;//phần tử mới* *p = (NodePointer)malloc(sizeof(struct Node));* *p->info = NewInfo;// ghi dữ liệu vào phần tử p* *p->previous = NULL;* *p->next = first;* *if (first != NULL)* *first->previous = p;* *first = p;* *if (last = NULL)* *last = p;* *return p;* }

- *Trường hợp thêm phần tử vào sau phần tử q*

Hàm *Insert_Element* trả về địa chỉ của phần tử mới thêm vào.

Trong PASCAL	Trong C
function Insert_Element(NewInfo: InfoType; q: NodePointer): NodePointer; *Var p, {*phần tử mới *} r : NodePointer;* *begin* *new(p);* *p^.info := NewInfo;{* ghi dữ liệu vào phần tử p *}* *p^.next := q^.next;* *p^.previous := q;* *q^.next := p;* *r := p^.next;* *if r <> nil then {thêm vào giữa danh sách}* *r^.previous := p* *else {thêm vào cuối danh sách}* *last := p;* *Insert_Element := p* *end;*	*NodePointer Insert_Element(InfoType NewInfo, NodePointer q)* { *NodePointer p , /*phần tử mới */ r;* *p = (NodePointer)malloc(sizeof(struct Node));* *p->info = NewInfo;// ghi dữ liệu vào phần tử p* *p->next = q->next;* *p->previous = q;* *q->next = p;* *r = p->next;* *if (r != NULL) /*thêm vào giữa danh sách */* *r->previous = p;* *else /*thêm vào cuối danh sách */* *last = p;* *return p;* }

• *Trường hợp thêm phần tử vào danh sách đã có thứ tự*
Hàm *Insert_Element* trả về địa chỉ của phần tử mới thêm vào.

◊ Trong PASCAL:

```
function Insert_Element(NewInfo: InfoType): NodePointer;
var     p , {*phần tử mới thêm vào*}
    q, before : NodePointer;
begin
    new(p);
    p^.info := NewInfo; {* ghi dữ liệu vào phần tử p *}
    {tìm phần tử before ngay trước phần tử p}
    q := first;
    while (q <> nil) and (q^.info < NewInfo) do
        begin
            before := q;
            q := q^.next;
        end;
    if q = first then {thêm vào đầu danh sách}
        begin
            p^.next := first;
            p^.previous := nil;
            if first <> nil then
                first^.previous := p;
            first := p;
            if last = nil then
                last := p;
        end
    else    {thêm vào sau before}
        begin
            p^.next := before^.next;
            p^.previous := before;
            before^.next := p;
            if q <> nil then {thêm vào giữa danh sách}
                q^.previous := p
            else        {thêm vào cuối danh sách}
                last := p;
        end
    Insert_Element := p
end;
```

◊ Trong C:

```
NodePointer  Insert_Element(InfoType NewInfo)
{
```

```
    NodePointer   p , /*phần tử mới thêm vào*/
              q, before;
    p = (NodePointer)malloc(sizeof(struct Node));
    p->info = NewInfo;// ghi dữ liệu vào phần tử p
    {tìm phần tử before ngay trước phần tử p}
    q = first;
    while ((q != NULL) && (q->info < NewInfo))
    {
        before = q;
        q = q->next;
    }
    if (q == first) //thêm vào đầu danh sách
    {
        p->next = first;
        p->previous = NULL;
        if (first != NULL)
            first->previous = p;
        first = p;
        if (last == NULL)
            last = p;
    }
    else    /*thêm vào sau before */
    {
        p->next = before->next;
        p->previous = before;
        before->next = p;
        if (q != NULL) //thêm vào giữa danh sách
            q->previous = p;
        else          //thêm vào cuối danh sách
            last = p;
    }
    return  p;
}
```

d. Loại phần tử khỏi danh sách

Giả sử ta cần loại bỏ phần tử có địa chỉ *p*. Thủ tục *Delete_Element* sẽ xoá phần tử khỏi danh sách.

Trong PASCAL	Trong C
procedure Delete_Element(p : NodePointer); *var* *before, {* phần tử đứng trước p *}* *after {* phần tử đứng sau p *}* *: NodePointer;* *begin* *before := p^.previous;* *after := p^.next;* *if before <> nil then* *before^.next := p^.next;* *if after <> nil then* *after^.previous := p^.previous;* *if p = first then* *first := after;* *if p = last then* *last := before;* *dispose(p)* *end;*	*void Delete_Element(NodePointer p)* { *NodePointer before, /* phần tử đứng trước p */* *after ; // phần tử đứng sau p* *before = p->previous;* *after = p->next;* *if (before != NULL)* *before->next = p->next;* *if (after != NULL)* *after->previous = p->previous;* *if (p == first)* *first = after;* *if (p == last)* *last = before;* *free(p);* }

e. Tìm kiếm phần tử trong danh sách.

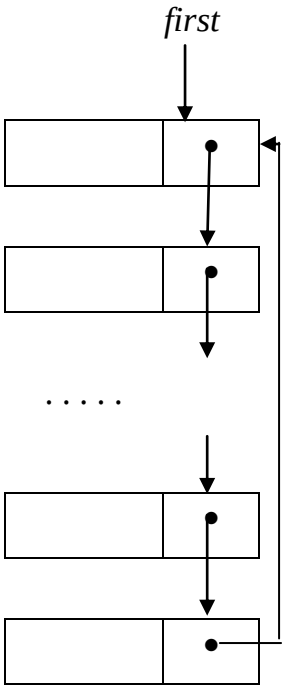
Có thể tìm theo vùng *next* bắt đầu từ chỉ điểm *first* (thứ tự tăng dần) hoặc theo vùng *previous* bắt đầu từ chỉ điểm *last* (thứ tự giảm dần). Giải thuật giống như đối với danh sách liên kết đơn.

VI. DANH SÁCH LIÊN KẾT VÒNG

1. Định nghĩa

Danh sách liên kết vòng là danh sách liên kết mà vùng liên kết của phần tử cuối trỏ đến phần tử đầu.

Có thể biểu diễn các phần tử danh sách liên kết vòng như sau:



• ***Khai báo***

Trong PASCAL	Trong C
Type InfoType = <kiểu dữ liệu chứa trong phần tử>; *NodePointer = ^Node;* *Node = record* *info : InfoType;* *next : NodePointer;* *end;* *Var first, p: NodePointer ;{ first là con trỏ danh sách}*	*struct Node {* *InfoType info;* *struct Node *next;* *};* *typedef struct Node *NodePointer;* *NodePointer first, p;// first là con trỏ danh sách* *p = (NodePointer)malloc(sizeof(struct Node));//cấp phát nút động cho p*

2. Các phép toán trên danh sách

Các phép toán *duyệt, tìm kiếm, chèn, xoá* tương tự như trên danh sách liên kết đơn. Tuy nhiên cần chú ý là vùng liên kết của phần tử cuối trỏ đến phần tử đầu.

Danh sách liên kết vòng thích hợp với các phép tách và ghép danh sách.

- *Thủ tục chèn phần tử vào cuối danh sách*

Trong PASCAL	Trong C
procedure Insert(NewInfo: InfoType); *var p, q : NodePointer;* *begin* *{tạo nút p}* *new(p);* *p^.info := NewInfo;* *p^.next := nil;* *{tìm nút cuối danh sách q}* *q := first;* *while (q^.next <> first) do* *q := q^.next;* *q^.next := p; {liên kết nút q đến p}* *p^.next := first;{ p trở thành nút cuối của danh sách}* *end;*	*void Insert(InfoType NewInfo)* { *NodePointer p, q;* *p = (NodePointer)malloc(sizeof(struct Node));* *p->info = NewInfo;* *p->next = NULL;* *// tìm nút cuối danh sách q* *q = first;* *while (q->next != first)* *q = q->next;* *q->next = p;// liên kết nút q đến p;* *p->next = first;// p trở thành nút cuối của danh sách* }

- *Thủ tục loại phần tử sau phần tử q*

Trong PASCAL	Trong C
procedure Delete(q : NodePointer); *var p,r : NodePointer;* *begin* *p := q^.next;* *if p <> nil then* *begin* *q^.next := p^.next;* *end;* *if p = first then* *begin* *{tìm nút cuối danh sách r}* *r := first;* *while (r^.next <> first) do* *r := r^.next;* *first := p^.next;* *r^.next := first;* *end;* *dispose(p);* *end;*	*void Delete(NodePointer q)* { *NodePointer p, r;* *p = q->next;* *if (p != NULL)* { *q->next = p->next;* } *if (p ==first)* { *r = first;* *while (r->next != first)* *r = r->next;* *first = p->next;* *r->next = first;* } *free(p);* }

◊ *Ví dụ: Bài toán Josephus*

Một nhóm *n* binh sĩ bị kẻ thù bao vây và họ phải chọn một người đi cầu cứu viện binh. Việc chọn người được thực hiện như sau.

Cho trước một số nguyên *s gọi là bước đếm*. Các binh sĩ xếp thành vòng tròn đánh số 1, 2, ... , *n* bắt đầu từ người chỉ huy.

Đếm từ người thứ nhất, khi đạt đến *s* thì binh sĩ tương ứng bị loại ra khỏi vòng.

Lại bắt đầu đếm từ người tiếp theo, khi đạt đến *s* thì binh sĩ tương ứng bị loại ra khỏi vòng.

Quá trình này tiếp tục cho đến khi chỉ còn lại một binh sĩ, và người này chính là người được điều đi cầu cứu viện binh.

Sau đây là chương trình cài đặt giải thuật của bài toán.

Trong PASCAL

```
Type
    NodePointer  = ^Node;
    Node  = record
                info : integer;
                next : NodePointer;
            end;
Var  q, p, first : NodePointer;
    n, {số binh sĩ}
    s, {bước đếm}
    i     : integer;
Procedure Initialize; {*** khởi tạo ***}
begin
    first := nil;
    write('cho số binh sĩ : '); readln(n);
    write('cho số bước đếm: '); readln(s);
end;
Procedure  CreateRing; {Tạo danh sách}
var  p, last : NodePointer;
begin
    {tạo nút đầu tiên}
    new(first);
    first^.info := 1;
    first^.next := nil;
    {nối tiếp  n-1  phần tử }
    last := first;
    for  i := 2 to  n  do
        begin
            new(p);
            p^.info := i;
```

Trong C

```
struct  node  {
                int  v;/*dinh ke*/
                struct node  *link;
            };
typedef  struct node *nodeptr;
int   n, s;
nodeptr first, last;
void addnode(int i);
void init(void); /*** khởi tạo ***/
main()
{
    nodeptr p1, p;
    int k, i;
    clrscr();
    printf("nhap n, s = ");
    scanf("%d %d", &n, &s);
    // Tạo danh sách
    init();
    for (k = 2; k <= n; k++)
        addnode(k);
    //Xóa n-1 nút
    p1 = last;
    for (k = 1; k <= n-1; k++)
      {
        for (i = 1; i < s; i++)
            p1 = p1->link;
        p = p1->link;
        p1->link = p->link;
        free(p);
```

```
            p^.next := nil;
            last^.next := p;
            last := p;
        end;
    {kết nối phần tử cuối đến phần tử đầu}
    last^.next := first;
end;
BEGIN
    Initialize;
    CreateRing;
    {loại dần n - 1 binh sĩ}
    new(p);
    p^.next := first;
    while  p <> p^.next  do
        begin
        {di chuyển đến phần tử thứ s-1}
            for  i := 1  to  s-1  do
                p := p^.next;
        { loại phần tử thứ s khỏi vòng}
            q:=p^.next;
            p^.next := q^.next;
            disposei(q)
        end;
    write('Người được chọn : ', p^.info);
END.
```

```
        }
    printf("\n người còn lại:  %d", p1->v);
}
void addnode(int i)
{
    nodeptr p;
     p = (nodeptr)malloc(sizeof(struct
node));
     p->v = i;
    last->link = p;
    p->link = first;
    last = p;
}
void init(void)
{
    nodeptr  p;
    p = (nodeptr)malloc(sizeof(struct
node));
    p->v = 1;
    first = p;
    last = p;
    last->link = first;
}
```

BÀI TẬP

1. Viết chương trình nhập các số nguyên *n, a, b* ($0 < n \le 100$, $a < b$, $n << b-a < 1000$) và thực hiện các công việc sau:

a) Tạo danh sách đặc *L1* gồm *n* số nguyên ngẫu nhiên trong khoảng [*a*;*b*].

b) Tạo danh sách đặc *L2* gồm *n* số nguyên ngẫu nhiên *khác nhau* trong khoảng [*a*; *b*].

c) Nối *L2* vào *L1* để tạo danh sách *L*.

d) Kiểm tra tính đơn điệu của danh sách *L*.

e) Kiểm tra tính đối xứng của danh sách *L*.

f) Nhập một số nguyên *x*, sau đó tìm phần tử trong danh sách *L* có giá trị bằng *x*. Nếu tìm thấy, chèn phần tử (*x*–1) vào trước phần tử *x* và phần tử (*x*+1) vào sau phần tử *x*. Nếu không tìm thấy, chèn phần tử (*x*–1) vào đầu danh sách và phần tử (*x*+1) vào cuối danh sách.

g) Nhập một số *x*, sau đó xác định xem có bao nhiêu phần tử trong danh sách *L* có giá trị bằng *x*.

h) Loại các phần tử có giá trị bằng *x* ra khỏi danh sách *L*.

i) Tính trung bình cộng các phần tử trong danh sách *L*.

j) Tạo danh sách các số chẵn từ danh sách *L*.

k) Tạo danh sách các số nguyên tố từ danh sách *L*.

l) Tạo danh sách các số chính phương từ danh sách *L*.

m) Loại các phần tử giống nhau ra khỏi danh sách *L* (chỉ giữ lại 1 phần tử duy nhất).

2. Viết chương trình nhập các số nguyên *n, a, b* ($0 < n$ 100, $a < b$, $n << b-a < 1000$) và thực hiện các công việc sau:

a) Tạo danh sách liên kết *L1* gồm *n* số nguyên ngẫu nhiên trong khoảng [*a;b*].

b) Tạo danh sách liên kết *L2* gồm *n* số nguyên ngẫu nhiên khác nhau trong khoảng [*a; b*].

c) Nối *L2* vào *L1* để tạo danh sách *L*.

d) Kiểm tra tính đơn điệu của danh sách *L*.

e) Kiểm tra tính đối xứng của danh sách *L*.

f) Nhập một số *x*, sau đó tìm phần tử trong danh sách *L* có giá trị bằng *x*. Nếu tìm thấy, chèn phần tử (*x* – 1) vào trước phần tử *x* và phần tử (*x*+1) vào sau phần tử *x*. Nếu không tìm thấy, chèn phần tử (*x* – 1) vào đầu danh sách và phần tử (*x*+1) vào cuối danh sách.

g) Nhập một số *x*, sau đó xác định xem có bao nhiêu phần tử trong danh sách *L* có giá trị bằng *x*.

h) Loại các phần tử có giá trị bằng *x* ra khỏi danh sách *L*.

i) Tính trung bình cộng các phần tử trong danh sách *L*.

j) Tạo danh sách liên kết các số chẵn từ danh sách *L*.

k) Tạo danh sách liên kết các số nguyên tố từ danh sách *L*.

l) Tạo danh sách liên kết các số chính phương từ danh sách *L*.

m) Loại các phần tử giống nhau ra khỏi danh sách *L* (chỉ giữ lại 1 phần tử duy nhất).

3. Viết chương trình giải bài toán Josephus với số người và bước đếm nhập từ bàn phím.