CHƯƠNG 3. Stack – Queue – Link List

NỘI DUNG:

3.1. Danh sách liên kết đơn (Single Linked List)

3.2. Danh sách liên kết đơn vòng (Circular Single Linked List)

3.3. Danh sách liên kết kép (Double Linked List)

3.4. Danh sách liên kết kép vòng (Circualar Double Linked List)

3.5. Ngăn xếp (Stack)

3.6. Hàng đợi (Queue)

3.7. Hàng đợi vòng (Curcular Queue)

3.8. CASE STUDY

3.1. Danh sách liên kết đơn

3.1.1. Định nghĩa. Tập hợp các node thông tin (khối dữ liệu) được tổ chức rời rạc trong bộ nhớ. Trong đó, mỗi node gồm hai thành phần:

- Thành phần dữ liệu (infor): dùng để lưu trữ thông tin của node.
- Thành phần con trỏ (pointer): dùng để trỏ đến node dữ liệu tiếp theo.

Một số vấn đề cần thảo luận:

- Tại sao phải xây dựng danh sách liên kết đơn:
  - Vấn đề bộ nhớ.
  - Vấn đề thêm phần tử.
  - Vấn đề loại bỏ phần tử.
- Khi nào sử dụng danh sách liên kết đơn?
- So sánh danh sách liên kết đơn và mảng?

3.1.2. Biểu diễn danh sách liên kết đơn

Sử dụng kiểu dữ liệu cấu trúc tự trỏ để định nghĩa mỗi node của danh sách liên kết đơn. Giả sử thành phần thông tin của mỗi node được định nghĩa như một cấu trúc Item:

```
typedef struct {
        <Kiểu 1>     <Thành viên 1>;
        <Kiểu 2>     <Thành viên 2>;
        . . . . . . . . . . . . . . . . . . . . . . . . . . . ;
        <Kiểu N>     <Thành viên N>;
} Item;
```

Khi đó, mỗi con trỏ đến một node được định nghĩa như sau:

```
typedef struct node {
        Item    Infor; //Thông tin của mỗi node;
        struct  node *next;
} *List;
```

3.1.3. Các thao tác trên danh sách liên kết đơn

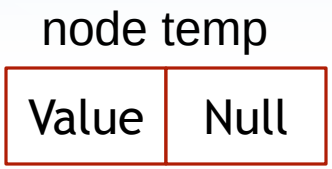
- Khởi tạo danh sách liên kết đơn: đưa trạng thái danh sách liên kết đơn về trạng thái rỗng. Ta gọi thao tác này là Init().
- Cấp phát miền nhớ cho một node: khi thực hiện thêm node vào danh sách thì node cần thêm vào cần trỏ đến một miền nhớ cụ thể thông qua các thao tác cấp phát bộ nhớ.
- Thêm node vào đầu bên trái danh sách liên kết đơn.
- Thêm node vào đầu bên phải theo chiều con trỏ next.
- Thêm node vào node giữa danh sách liên kết đơn.
- Loại node cuối bên trái danh sách liên kết đơn.
- Loại node cuối bên phải theo chiều con trỏ next.
- Loại node ở giữa danh sách liên kết đơn.
- Duyệt thông tin của danh sách liên kết đơn.
- Tìm node trên danh sách liên kết đơn.

Lớp các thao tác trên danh sách liên kết đơn (DSLKĐ):

```
struct node { // biểu diễn node
        int info; //thành phần thông tin của node
        struct node *next; //thành phần con trỏ của node
}*start; // danh sách liên kết đơn: *start.
class single_llist { //Biểu diễn lớp llist
    public:
        node* create_node(int);//Tạo một node cho danh sách liên kết đơn
        void insert_begin(); //Thêm node vào đầu DSLKĐ
        void insert_pos(); //Thêm node tại vị trí ch trước trên DSLKĐ
        void insert_last(); //Thêm node vào cuối DSLKĐ
        void delete_pos(); //Loại node tại vị trí cho trước trên DSLKĐ
        void sort(); //Sắp xếp nội dung các node theo thứ tự tăng dần
        void search(); //Tìm kiếm node trên DSLKĐ
        void update(); //Sửa đổi thông tin của node trên DSLKĐ
        void reverse(); //Đảo ngược danh sách liên kết đơn
        void display(); //Hiển thị nội dung DSLKĐ
        single_llist(){//Constructor của lớp llist.
                start = NULL;
        }
};
```

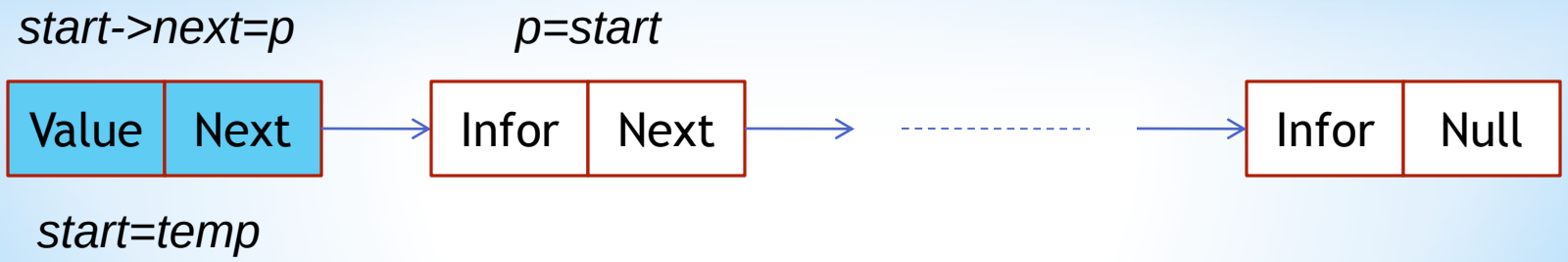
Khởi tạo một node cho DSLKĐ:

```
node *single_llist::create_node(int value){
        struct node *temp, *s; //Khai báo hai con trỏ node *temp, *s
        temp = new(struct node); //Cấp phát miền nhớ cho temp
        if (temp == NULL){ //Nếu không đủ không gian nhớ
                cout<<"Không đủ bộ nhớ để cấp phát"<<endl;
                return 0;
        }
        else {
                temp->info = value;//Thiết lập thông tin cho node temp
                temp->next = NULL; //Thiết lập liên kết cho node temp
                return temp;//Trả lại node temp đã được thiết lập
        }
}
```



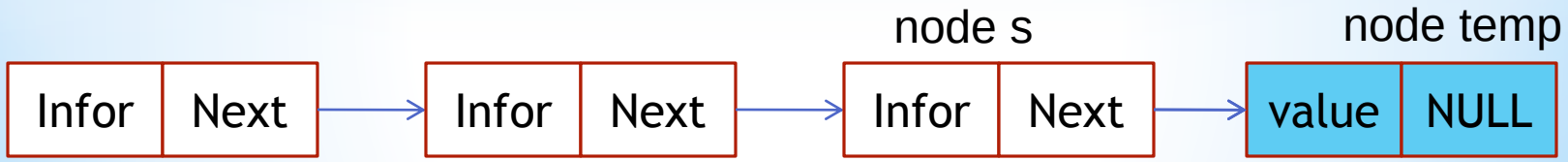
Chèn node vào đầu DSLKĐ:

```
void single_llist::insert_begin(){ //Chèn node vào đầu DSLKĐ
    int value; cout<<“Nhập giá trị node:"; cin>>value; //Giá trị node cần chèn
    struct node *temp, *p; //Sử dụng hai con trỏ temp và p
    temp = create_node(value);//Tạo một node với giá trị value
    if (start == NULL){ //Nếu danh sách rỗng
            start = temp; //Danh sách chính là node temp
            start->next = NULL; //Không có liêt kết với node khác
    }
    else { //Nếu danh sách không rỗng
            p = start; //p trỏ đến node đầu của start
            start = temp; //start được trỏ đến temp
            start->next = p;//start trỏ tiếp đến gốc cũ
    }
    cout<<“Hoàn thành thêm node vào đầu DSLKĐ"<<endl;
}
```



Thêm node vào cuối DSLKĐ:

```
void single_llist::insert_last(){//Thêm node vào cuối DSLKĐ
        int value;
        cout<<“Nhập giá trị cho node: ";cin>>value; //Nhập giá trị node
        struct node *temp, *s; //Sử dung hai con trỏ temp và s
        temp = create_node(value);//Tạo node có giá trị value
        s = start; //s trỏ đến node đầu danh sách
        while (s->next != NULL){ //Di chuyển s đến node cuối cùng
                s = s->next;
        }
        temp->next = NULL; //Temp không chỏ đi đâu nữa
        s->next = temp; //Thiết lập liên kết cho s
        cout<<“Hoàn thành thêm node vào cuối"<<endl;
}
```



Thêm node vị trí pos:

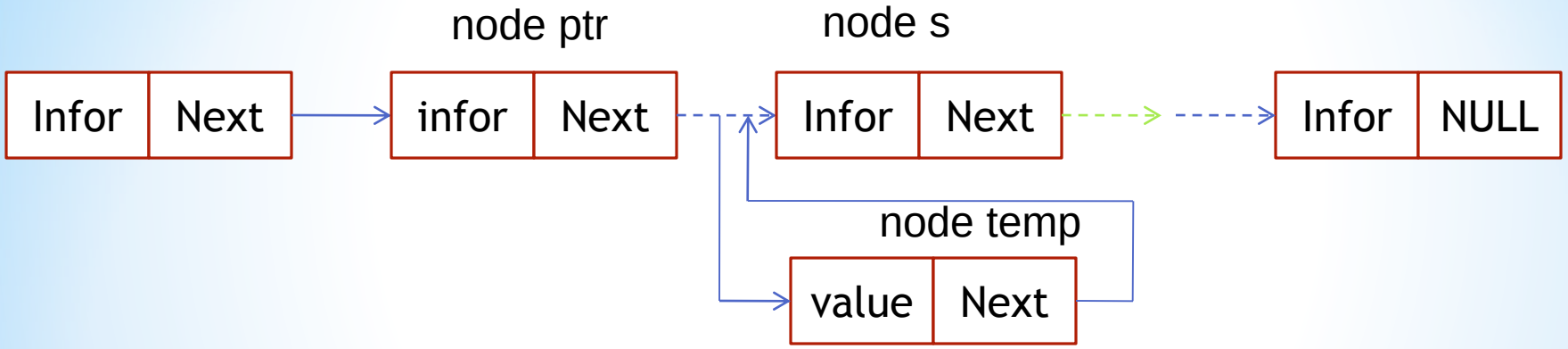
```
void single_llist::insert_pos(){//Thêm node vào vị trí pos
        int value, pos, counter = 0; cout<<"Nhap gia tri node:";cin>>value;
        struct node *temp, *s, *ptr; temp = create_node(value);//Tạo node
        cout<<“Nhập vị trí node cần thêm: ";cin>>pos;
        int i; s = start; //s trỏ đến node đầu tiên
        while (s != NULL){ //Đếm số node của DSLKĐ
                s = s->next; counter++;
        }
        if (pos == 1){ //Nếu pos là vị trí đầu tiên
                if (start == NULL){ //Trường hợp DSLKĐ rỗng
                        start = temp; start->next = NULL;
                }
                else { ptr = start;  start = temp;  start->next = ptr;   }
        }
        else if (pos > 1  && pos <= counter){ //Trường hợp pos hợp lệ
                s = start; //s trỏ đến node đầu tiên
                for (i = 1; i < pos; i++){  ptr = s;  s = s->next; }
                ptr->next = temp;  temp->next = s; //Thiết lập LK cho node
        }
        else { cout<<“Vượt quá giới hạn DSLKĐ"<<endl; }
}
```

Loại node ở vị trí pos:

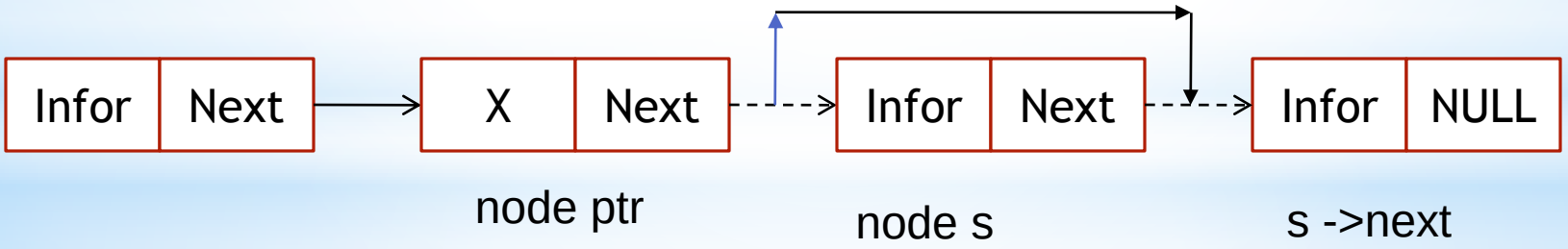
```
void single_llist::delete_pos(){//Loai phan tu o vi tri cho truoc
        int pos, i, counter = 0;
        if (start == NULL){  cout<<“Không thực hiện được"<<endl; return; }
        cout<<“Vị trí cần loại bỏ:";cin>>pos;
        struct node *s, *ptr;  s = start; //s trỏ đến đầu danh sách
        if (pos == 1){//Nếu vị trí loại bỏ là node đầu tiên
                start = s->next; s->next=NULL; free(s);
        }
        else {
                while (s != NULL) {  s = s->next; counter++; } //Đếm số node
                if (pos > 0 && pos <= counter){ //Nếu vị trí hợp lệ
                        s = start;//s trỏ đến node đầu của danh sách
                        for (i = 1;i < pos;i++){ ptr = s;  s = s->next;  }
                        ptr->next = s->next; //Thiết lập liên kết cho node
                }
                else { cout<<"Vi tri ngoai danh sach"<<endl; }
                free(s);
                cout<<"Node da bi loai bo"<<endl;
        }
}
```

Giải thích thêm về hai thao tác chèn node và loại bỏ node:

Thêm node vào vị trí pos:



Loại node ở vị trí pos:



Sửa đổi nội dung node:

```
void single_llist::update(){//Sửa đổi thông tin của node
        int value, pos, i;
        if (start == NULL){ //Nếu danh sách rỗng
                cout<<"Không thực hiện được"<<endl;  return;
        }
        cout<<"Nhập vị trí node cần sửa:";cin>>pos;
        cout<<"Giá trị mới của node:";cin>>value;
        struct node *s, *ptr; //Sử dụng hai con trỏ s và ptr
        s = start; //s trỏ đến node đầu tiên
        if (pos == 1) { start->info = value;} //Sửa luôn node đầu tiên
        else { //Nếu không phải là node đầu tiên
                for (i = 0;i < pos - 1;i++){//Chuyển s đến vị trí pos-1
                        if (s == NULL){//Nếu s là node cuối cùng
                                cout<<"Vị trí "<<pos<<" không hợp lệ"; return;
                        }
                        s = s->next;
                }
                s->info = value;  //Sửa đổi thông tin cho node
        }
        cout<<"Hoàn thành việc sửa đổi"<<endl;
}
```

Tìm kiếm node trên DSLKĐ:

```
void single_llist::search(){//Tìm kiếm node
        int value, pos = 0; bool flag = false;
        if (start == NULL){
                cout<<“Danh sách rỗng thì tìm cài gì?"<<endl;
                return;
        }
        cout<<“Nội dung node cần tìm:";cin>>value;
        struct node *s; s = start;//s trỏ đến đầu danh sách
        while (s != NULL){ pos++;
                if (s->info == value){//Nếu s->infor là value
                        flag = true;
                        cout<<“Tìm thấy "<<value<<" tại vị trí "<<pos<<endl;
                }
                s = s->next;
        }
        if (!flag) {
                cout<<“Giá trị"<<value<<“không tồn tại"<<endl;
        }
}
```

Hiển thị nội dung DSLKĐ:

```
void single_llist::display(){//Hiển thị nội dung DSLKĐ
        struct node *temp; //Sử dụng một con trỏ temp
        if (start == NULL){ // Nếu danh sách rỗng
                cout<<"Có gì đâu mà hiển thị"<<endl;
                return;
        }
        temp = start; //temp trỏ đến node đầu trong DSLKĐ
        cout<<"Nội dung DSLKĐ: "<<endl;
        while (temp != NULL) { //Lặp cho đến node cuối cùng
                cout<<temp->info<<"->"; //Hiển thị thành phần thông tin
                temp = temp->next; //Trỏ đến node kế tiếp
        }
        cout<<"NULL"<<endl; //Cuối cùng chắc chắn sẽ là NULL
}
```

Sắp xếp nội dung các node của DSLKĐ:

```
void single_llist::sort(){//Sắp xếp nội dung các node
        struct node *ptr, *s; //Sử dụng hai con trỏ ptr và s
        int value; //Giá trị trung gian
        if (start == NULL){//Nếu danh sách rỗng
                cout<<"Có gì đâu mà sắp xếp"<<endl;
                return;
        }
        ptr = start;//ptr trỏ đến node đầu danh sách
        while (ptr != NULL){ //Lặp nếu ptr khác rỗng
                for (s = ptr->next;s !=NULL;s = s->next){ //s là node kế tiếp
                        if (ptr->info > s->info){
                                value = ptr->info;
                                ptr->info = s->info;
                                s->info = value;
                        }
                }
                ptr = ptr->next;
        }
}
```

Đảo ngược các node trong DSLKĐ:

```
void single_llist::reverse(){//Đảo ngược danh sách
        struct node *ptr1, *ptr2, *ptr3; //Sử
        if (start == NULL) {//Nếu danh sách rỗng
                cout<<"Ta không cần đảo"<<endl; return;
        }
        if (start->next == NULL){//Nếu danh sách chỉ có một node
                 cout<<"Đảo ngược là chính nó"<<endl; return;
        }
        ptr1 = start; //ptr1 trỏ đến node đầu tiên
        ptr2 = ptr1->next;//ptr2 trỏ đến node kế tiếp của ptr1
        ptr3 = ptr2->next;//ptr3 trỏ đến nod kế tiếp của ptr2
        ptr1->next = NULL;//Ngắt liên kết ptr1
        ptr2->next = ptr1;//node ptr2 bây giờ đứng trước node ptr1
        while (ptr3 != NULL){//Lặp nếu ptr3 khác rỗng
                ptr1 = ptr2;  //ptr1 lại bắt đầu tại vị trí ptr2
                ptr2 = ptr3;  //ptr2 bắt đầu tại vị trí ptr3
                ptr3 = ptr3->next; //ptr3 trỏ đến node kế tiếp
                ptr2->next = ptr1; //Thiết lập liên kết cho ptr2
        }
        start = ptr2; //node đầu tiên bây giờ là ptr2
}
```

3.1.4. Ứng dụng của danh sách liên kết

- Xây dựng các lược đồ quản lý bộ nhớ:
  - Thuật toán Best Fit:
  - Thuật toán First Fit:
  - Thuật toán Best Availbale.
- Biểu diễn ngăn xếp :
  - Danh sách L + { Add-Top, Del-Top}.
  - Danh sách L + { Add-Bottom, Del-Bottom}.
- Biểu diễn hàng đợi:
  - Danh sách L + { Add-Top, Del-Bottom}.
  - Danh sách L + { Add-Bottom, Del-Top}.
- Biểu diễn cây.
- Biểu diễn đồ thị.
- Biểu diễn tính toán.

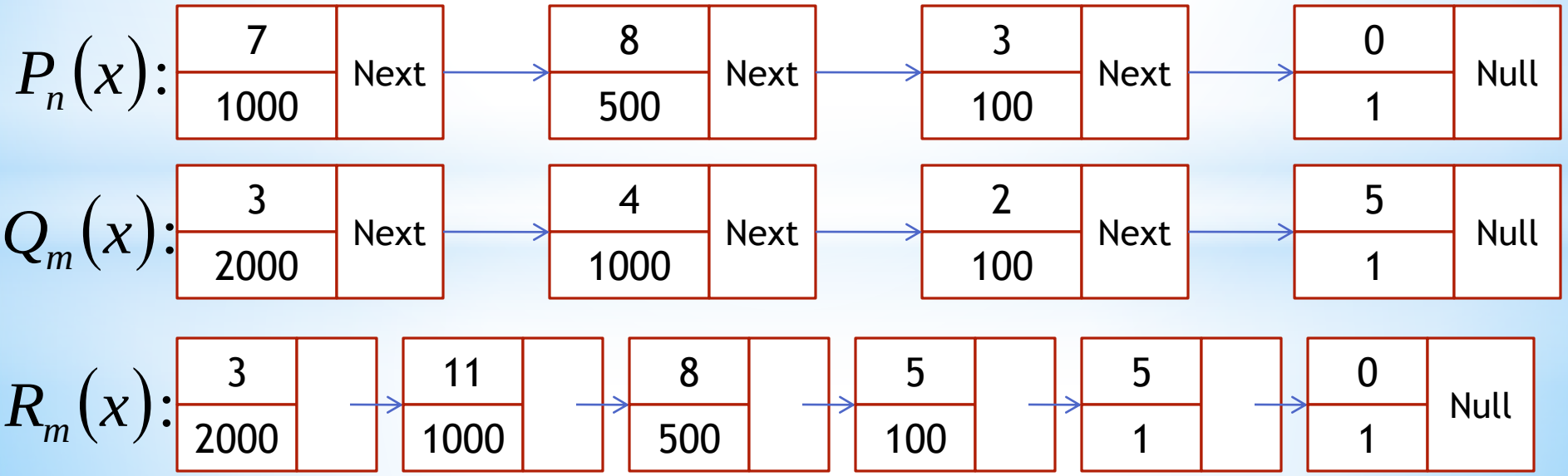
Ví dụ. Thuật toán cộng hai đa thức $R = P_n(x) + Q_m(x)$.

$$P_n(x) = 7x^{1000} + 8x^{500} + 3x^{100} + 1$$

$$Q_m(x) = 3x^{2000} + 4x^{1000} + 2x^{100} + 5x$$

Biểu diễn mỗi số hạng của đa thức:

```
typedef struct node {
        float hsomueso;
        float ;
        struct node *next;
} *dathuc;
```



Thuật toán Cong_Dathuc (Dathuc *P, Dathuc *Q):

Bước 1 (Khởi tạo): R = ∅;

Bước 2 (lặp):

```
while ( P≠∅ && Q≠∅ ) {
        if( P→Somu > Q → Somu ) {
                R = R →P; P = P →next;
        }
        else if (P→Somu < Q → Somu ) {
                R = R →Q; Q = Q →next;
        }
        else {    P → heso = P → heso + Q → heso;
                R = R →P; P = P →next; Q = Q →next;
        }
}
```

Bước 3 (Hoàn chỉnh đa thức):

```
if (P ≠∅ ) R = R = R →P;
 if (Q ≠∅ ) R = R = R →Q;
```

Bước 4 (Trả lại kết quả):

```
Return (R);
```

3.2. Danh sách liên kết đơn vòng (Circualar single linked List)

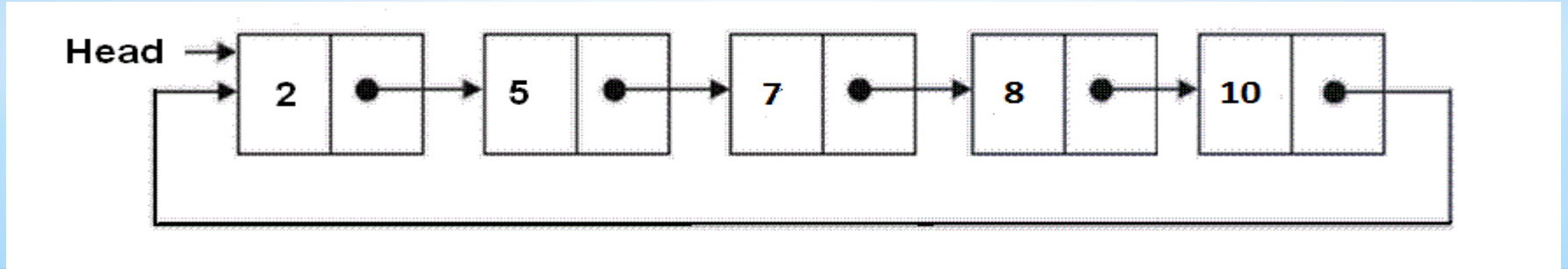
3.2.1. Định nghĩa. Là danh sách liên kết đơn trong đó tất cả các node liên kết với nhau thành một vòng tròn. Không có con trỏ NULL ở node cuối cùng mà được liên kết với node đầu tiên.

Một số tính chất của danh sách liên kết đơn vòng:

- Mọi node đều là node bắt đầu. Tao có thể duyệt tại bất kỳ node nào và chỉ dừng khi ta lặp lại một node đã duyệt.
- Dễ dàng trong việc cài đặt hàng đợi.
- Dễ dàng phát triển các ứng dụng thực hiện vòng quang danh sách.

3.2.2. Biểu diễn: Giống như biểu diễn của danh sách liên kết đơn.

```
struct node {//Biểu diễn node
        int info; //thành phần thông tin của node
        struct node *next; //thành phần con trỏ của node
}*last; //Danh sách liên kết vòng: last
```



3.2.3. Các thao tác trên danh sách liên kết đơn vòng

- Tạo lập node cho danh sách liên kết đơn vòng.
- Thêm node đầu tiên cho sách liên kết đơn vòng.
- Thêm node vào sau một node khác.
- Loại bỏ node trên danh sách liên kết đơn vòng.
- Tìm kiếm node trên danh sách liên kết đơn vòng.
- Hiển thị nội dung danh sách liên kết đơn vòng.
- Sửa đổi nội dung node cho danh sách liên kết đơn vòng.
- Sắp xếp nội dung các node trên danh sách liên kết đơn vòng.

Khai báo danh sách liên kết vòng: giống như danh sách liên kết thông thường.

```
struct node {//Biểu diễn node của danh sách liên kết đơn vòng
        int info; //Thành phần thông tin của node
        struct node *next; //Thành phần con trỏ của node
}*last;
```

3.2.3. Các thao tác trên danh sách liên kết đơn vòng

```
class circular_llist{//Mô tả lớp danh sách liên kết đơn vòng
    public:
        void create_node(int value); //Tạo node cho DSLKĐ vòng
        void add_begin(int value); //Thêm node đầu tiên
        void add_after(int value, int position); //Thêm node sau vị trí pos
        void delete_element(int value); //Loại bỏ node
        void search_element(int value); //Tìm kiếm node
        void display_list(); //Hiển thị node
        void update(); //Sửa đổi nội dung node
        void sort(); //Sắp xếp node
        circular_llist(){//Constructor của lớp circular_llist
            last = NULL;
        }
};
```

Tạo node cho danh sách liên kết đơn vòng:

```
void circular_llist::create_node(int value){//Tạo danh sách liên kết vòng
        struct node *temp; //Khai báo con trỏ  temp
        temp = new(struct node);//Cấp phát bộ nhớ cho con trỏ temp
        temp->info = value;//Thiết lập giá trị cho node temp
        if (last == NULL){//Nếu danh sách rỗng
                last = temp; //last chính là temp
                temp->next = last;//Temp trỏ vòng lại last
        }
        else   {//Nếu danh sách không rỗng
                temp->next = last->next; //thiết lập liên kết cho temp
                last->next = temp; //thiết lập liên kết cho last
                last = temp; //thiết lập liên kết vòng cho last
        }
}
```

Chèn node vào đầu cho danh sách liên kết đơn vòng:

```
void circular_llist::add_begin(int value){//Chèn node vào đầu
        if (last == NULL){//Nếu danh sách rỗng
                cout<<“Chưa tạo node đầu cho danh sách"<<endl;
                return;
        }
        struct node *temp; //Khai bào con trỏ temp
        temp = new(struct node); //Cấp phát bộ nhớ cho node temp
        temp->info = value; //Thiết lập giá trị cho temp
        temp->next = last->next; //Thiết lập liên kết cho temp
        last->next = temp; //Thiết lập liên kết cho last
}
```

Chèn node vào sau vị trí pos:

```
void circular_llist::add_after(int value, int pos){//Chèn node vào sau vị trí pos
        if (last == NULL){//Nếu danh sách rỗng
                cout<<"Không thể thực hiện được."<<endl;
                return;
        }
        struct node *temp, *s;   s = last->next; //s trỏ đến node tiếp theo
        for (int i = 0;i < pos-1; i++){//Di chuyển đến vị trí pos-1
                s = s->next;
                If (s == last->next){ //Nếu s lại quay về đầu
                        cout<<"Số node của danh sách bé hơn";
                        cout<<pos<<" trong danh sách"<<endl;
                        return;
                }
        }
        temp = new(struct node); temp->next = s->next;
        temp->info = value;  s->next = temp;
        if (s == last){ //Nếu s là node cuối cùng
                last=temp;
        }
}
```

Loại bỏ node trong danh sách:

```
void circular_llist::delete_element(int value){//Loại node trong danh sách
        struct node *temp, *s;  s = last->next;
        if (last->next == last && last->info == value){ //Nếu DS chỉ có một node
                temp = last;  last = NULL;  free(temp); return;
        }
        if (s->info == value) {//Nếu s là node đầu tiên
                temp = s; last->next = s->next;free(temp);return;
        }
        while (s->next != last){//Loai node ở giữa
                If (s->next->info == value) {
                        temp = s->next; s->next = temp->next;free(temp);
                        cout<<"Phần tử  "<<value<<“đã loại bỏ”<<endl; return;
                }
                s = s->next;
        }
        If (s->next->info == value){ //Nếu s là node cuối cùng
                temp=s->next;s->next=last->next;free(temp);last=s;return;
        }
        cout<<“Node"<<value<<" không tồn tại trong danh sách"<<endl;
}
```

Tìm node trong danh sách:

```
void circular_llist::search_element(int value){//Tìm node trong DSLK vòng
        struct node *s; int counter = 0;
        s = last->next; //s là node tiếp theo
        while (s != last) { //Lặp trong khi s chưa phải cuối cùng
                counter++;
                if (s->info == value){ //Nếu node s có giá trị value
                        cout<<“Tìm thấy node "<<value;
                        cout<<" ở vị trí "<<counter<<endl;
                        return;
                }
                s = s->next;
        }
        if (s->info == value){ //Nếu node cuối cùng là value
                counter++;
                cout<<“Tìm thấy node "<<value;
                cout<<" ở vị trí "<<counter<<endl;
                return;
        }
        cout<<“Giá trị "<<value<<" không có trong danh sách"<<endl;
}
```

Hiển thị nội dung các node trong danh sách:

```
void circular_llist::display_list(){//Hiển thị nội dung các node trong DS
        struct node *s;
        if (last == NULL){
                cout<<“Không có gì để hiển thị"<<endl;
                return;
        }
        s = last->next; //s là node kế tiếp
        cout<<“Nội dung DSLKV: "<<endl;
        while (s != last){ //Lặp trong khi s chưa phải cuối cùng
                cout<<s->info<<"->"; //Hiển thị nội dung node s
                s = s->next; //s trỏ đến node tiếp theo
        }
        cout<<s->info<<endl; //Hiển thị node cuối cùng
}
```

Sửa đổi nội dung node:

```
void circular_llist::update(){//Sửa đổi nội dung node
        int value, pos, i;
        if (last == NULL){ //Nếu danh sách rỗng
                cout<<"Ta không làm gì được"<<endl;
                return;
        }
        cout<<"Nhập vị trí node cần sửa: ";cin>>pos;
        cout<<"Giá trị mới của node: ";cin>>value;
        struct node *s;
        s = last->next; //s là node tiếp theo
        for (i = 0;i < pos - 1;i++){//Chuyển đến vị trí pos-1
                If (s == last){ //Nếu s quay trở lại đầu
                        cout<<"Số node nhỏ hơn  "<<pos<<endl;
                        return;
                }
                s = s->next;
        }
        s->info = value;
        cout<<"Node đã được sửa đổi"<<endl;
}
```

Sắp xếp nội dung node:

```
void circular_llist::sort(){//Sắp xếp nội dung các node
        struct node *s, *ptr;  int temp;
        if (last == NULL){ //Nếu danh sách rỗng
                cout<<"Có gì đâu mà sắp xếp"<<endl;    return;
        }
        s = last->next; //s là node kế tiếp
        while (s != last){ //Lặp nếu s không phải là last
                ptr = s->next; //ptr là node kế tiếp của s
                while (ptr != last->next) { //Lặp đến node cuối cùng
                        if (ptr != last->next) {
                                if (s->info > ptr->info)  {
                                        temp = s->info;s->info = ptr->info;
                                        ptr->info = temp;
                                }
                        }
                        else {  break; }
                        ptr = ptr->next;
                }
                s = s->next;
        }
}
```

Bài tập 1. Hoàn thành việc xây dựng các thao tác cơ bản trên danh sách liên kết đơn, bao gồm:

- Khởi tạo danh sách liên kết đơn.
- Chèn node vào đầu danh sách liên kết đơn.
- Chèn node vào cuối danh sách liên kết đơn.
- Chèn node vào vị trí xác định trong danh sách liên kết đơn.
- Loại node tại vị trí Pos trong danh sách liên kết đơn.
- Sửa đổi nội dung node trong danh sách liên kết đơn.
- Sắp xếp các node của danh sách liên kết đơn.
- Đảo ngược các node trong danh sách liên kết đơn.
- Tìm kiếm vị trí của node trong danh sách liên kết đơn.
- Hiển thị nội dung trong danh sách liên kết đơn.

Bài tập 2. Hoàn thành bài tập 1 sử dụng C++ STL .

Bài tập 3. Xây dựng tập thao tác trên đa thức dựa vào danh sách liên kết đơn.

Bài tập 4. Xây dựng các phép toán với số lớn bằng danh sách liên kết đơn.

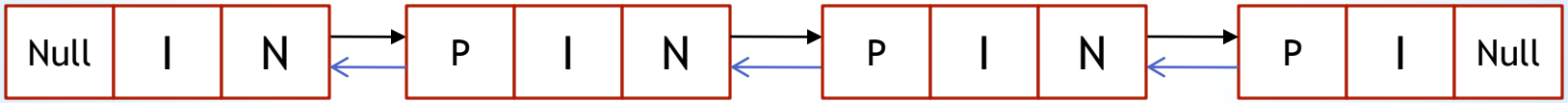
Bài tập 5. Hoàn thành các thao tác trên danh sách liên kết đơn vòng

- Tạo lập node cho danh sách liên kết đơn vòng.
- Thêm node đầu tiên cho sách liên kết đơn vòng.
- Thêm node vào sau một node khác.
- Loại bỏ node trên danh sách liên kết đơn vòng.
- Tìm kiếm node trên danh sách liên kết đơn vòng.
- Hiển thị nội dung danh sách liên kết đơn vòng.
- Sửa đổi nội dung node cho danh sách liên kết đơn vòng.
- Sắp xếp nội dung các node trên danh sách liên kết đơn vòng.

3.3. Danh sách liên kết kép

3.3.1. Định nghĩa. Tập hợp các node (khối dữ liệu) được tổ chức rời rạc trong bộ nhớ. Trong đó, mỗi node gồm ba thành phần:

- Thành phần dữ liệu (infor): dùng để lưu trữ thông tin của node.
- Thành phần con trỏ prev: dùng để trỏ đến node dữ liệu sau nó.
- Thành phần con trỏ next: dùng để trỏ đến node dữ liệu trước nó.



3.3.2. Biểu diễn danh sách liên kết kép

```
typedef struct node {
        Item   Infor; //Thành phần dữ liệu của node
        struct  node *prev; //Thành phần con trỏ sau
        struct  node *next; //Thành phần con trỏ trước
}*L;
```

3.3.3. Các thao tác trên danh sách liên kết kép

- Khởi tạo danh sách liên kết kép.
- Cấp phát miền nhớ cho một node.
- Thêm node vào đầu bên trái danh sách liên kết kép.
- Thêm node vào đầu bên phải danh sách liên kết kép.
- Thêm node vào node giữa danh sách liên kết kép.
- Loại node cuối bên trái danh sách liên kết kép.
- Loại node cuối bên phải danh sách liên kết kép.
- Loại node ở giữa danh sách liên kết kép.
- Duyệt trái danh sách liên kết kép.
- Duyệt phải danh sách liên kết kép.
- Tìm node trên danh sách liên kết kép…

Dưới đây là một số thao tác cơ bản trên danh sách liên kết kép.

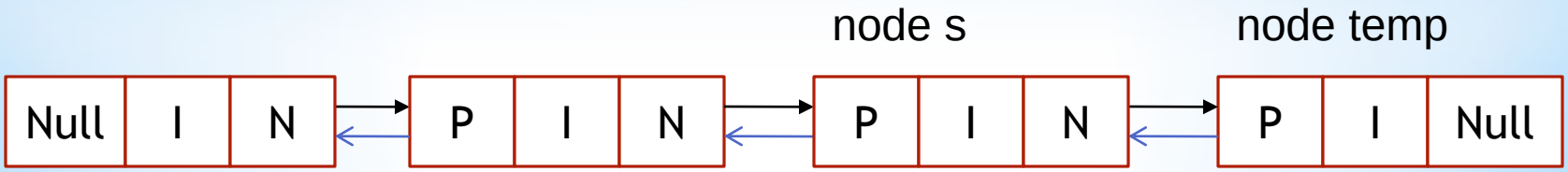
Tập thao tác trên danh sách liên kết kép:

```
struct node {//Biểu diễn node
        int info; //Thành phần thông tin của node
        struct node *next; // Thành phần con trỏ đến node trước nó
        struct node *prev; //Thành phần con trỏ đến node sau nó
}*start; //Biến danh sách liên kết kép

class double_llist {//Mô tả lớp các thao tác
    public:
        void create_list(int value);//Tạo node cho DSLK kép
        void add_begin(int value);//Thêm node vào đầu danh sách
        void add_after(int value, int position);//Thêm node vào sau position
        void delete_element(int value);//Loại node có thông tin value
        void search_element(int value);//Tìm node có thông tin value
        void display_dlist();//Hiển thị danh sách liên kết kép
        void count();//Đếm số node
        void reverse();//Đảo ngược danh sách
        double_llist(){//Constructor của lớp
            start = NULL;
        }
};
```

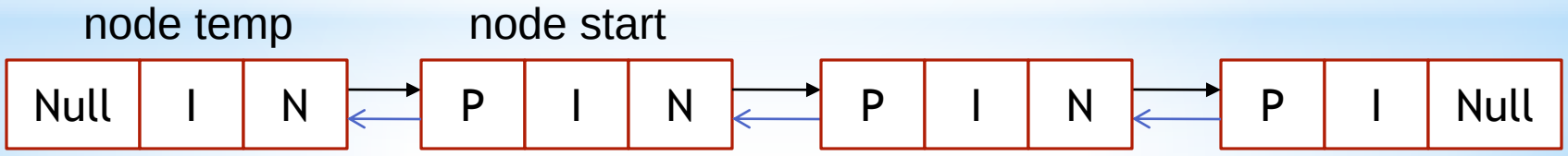
Tạo node cuối cùng cho danh sách liên kết kép:

```
void double_llist::create_list(int value){ //Tạo node cuối cho DSLK kép
        struct node *s, *temp; //Sử dụng hai con trỏ s và temp
        temp = new(struct node); //Cấp phát miền nhớ cho temp
        temp->info = value; //Thiết lập thành phần thông tin cho temp
        temp->next = NULL; //Thiết lập liên kết tiếp theo cho temp
        if (start == NULL){ //Nếu danh sách rỗng
                temp->prev = NULL; //Thiết lập liên kết sau cho temp
                start = temp; //Node đầu tiên trong danh sách là temp
        }
        else { //Nếu danh sách không rỗng
                s = start; // s trỏ đến start
                while (s->next != NULL) //Lặp trong khi chưa đến node cuối
                        s = s->next;
                s->next = temp; //thiết lập liên kết tiếp theo cho node cuối
                temp->prev = s;//Thiết lập liên kết sau cho temp
        }
}
```



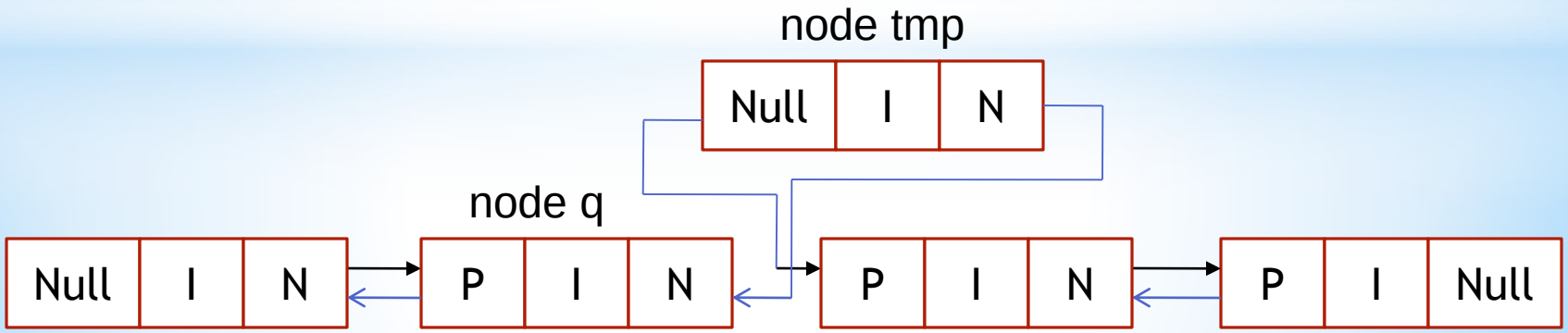
Thêm node vào đầu danh sách liên kết kép:

```
void double_llist::add_begin(int value){//Thêm node vào sau vị trí pos
        if (start == NULL){ //Nếu danh sách rỗng
                cout<<"Không phải làm gì."<<endl;
                return;
        }
        struct node *temp; //Sử dụng con trỏ temp
        temp = new(struct node); //Cấp phát miền nhớ cho temp
        temp->prev = NULL; //temp->prev được thiết lập là Null
        temp->info = value; //Thiết lập thông tin cho temp
        temp->next = start; // temp->next là start
        start->prev = temp; //start->prev là temp.
        start = temp; //Node đầu tiên bây giờ là temp
        cout<<"Node đã thêm vào đầu"<<endl;
}
```



Thêm node vào sau vị trí pos:

```
void double_llist::add_after(int value, int pos){//Thêm node vào sau vị trí pos
        if (start == NULL){ cout<<"Danh sách rỗng."<<endl;  return; }
        struct node *tmp, *q;   int i;  q = start;
        for (i = 0;i < pos - 1;i++) {//Chuyển q đến vị trí pos
                q = q->next;
                if (q == NULL) { cout<<"Số node nhỏ hơn "<<pos<<"<<endl;
                        return;
                }
        }
        tmp = new(struct node); tmp->info = value;//Thiết lập thông tin cho tmp
        if (q->next == NULL) { //Nếu q là node cuối cùng
                q->next = tmp; tmp->next = NULL;  tmp->prev = q;
        }
        else  { //Nếu q không phải node cuối cùng
                tmp->next = q->next;  tmp->next->prev = tmp;
                q->next = tmp; tmp->prev = q;
        }
}
```



Loại node có thông tin value:

```
void double_llist::delete_element(int value){//Loại node có giá trị value
        struct node *tmp, *q; //sử dụng hai con trỏ tmp và q
        if (start->info == value){    //Nếu value là thông tin node đầu tiên
                tmp = start;   start = start->next;   start->prev = NULL;
                cout<<"Node đầu tiên đã bị loại bỏ"<<endl; free(tmp);  return;
        }
        q = start; //q trỏ đến node đầu tiên
        while (q->next->next != NULL) { //Chuyển đến node trước của q->next
                if (q->next->info == value){//Nếu node trước của q->next là value
                        tmp = q->next; q->next = tmp->next;
                        tmp->next->prev = q;
                        cout<<"Node đã loại bỏ"<<endl;free(tmp); return;
                }
                q = q->next;
        }
        if (q->next->info == value){//Nếu value là node cuối cùng
                tmp = q->next;  free(tmp);  q->next = NULL;
                cout<<"Node cuối cùng đã bị loại bỏ"<<endl;
                return;
        }
        cout<<"Node  "<<value<<" không có thực"<<endl;
}
```

Hiển thị và đếm số node của danh sách:

```
void double_llist::display_dlist(){//Hiển thị nội dung danh sách
        struct node *q;
        if (start == NULL){ //Nếu danh sách rỗng
                cout<<"Không có gì để hiển thị"<<endl;
                return;
        }
        q = start; //Đặt q là node đầu tiên trong danh sách
        cout<<"Nội dung danh sách liên kết kép :"<<endl;
        while (q != NULL){ //Lặp cho đến node cuối cùng
                cout<<q->info<<" <-> "; //Hiển thị thông tin node
                q = q->next; //q trỏ đến node tiếp theo
        }
        cout<<"NULL"<<endl;
}
void double_llist::count(){ //Đếm số node của danh sách
        struct node *q = start;
        int cnt = 0;
        while (q != NULL){
                q = q->next;
                cnt++;
        }
        cout<<"Số node: "<<cnt<<endl;
}
```

Đảo ngược danh sách liên kết kép:

```
void double_llist::reverse(){//Đảo ngược danh sách liên kết kép
        struct node *p1, *p2; //Sử dụng hai con trỏ p1, p2
        p1 = start; //Đặt p1 trỏ đến node đầu tiên
        p2 = p1->next; //Đặt p2 trỏ đến node kế tiếp của p1
        p1->next = NULL; //Ngắt liên kết next của p1
        p1->prev = p2; //Thiết lập liên kết sau cho p1
        while (p2 != NULL) {
                p2->prev = p2->next; //Thiết lập liên kết sau cho p2
                p2->next = p1; //Thiết lập liên kết trước cho p2
                p1 = p2; //Đặt p1 vào p2
                p2 = p2->prev; //Thiết lập lại liên kết sau cho p2
        }
        start = p1; //Thiết lập node cuối cùng
        cout<<"Danh sách đã đảo ngược"<<endl;
}
```

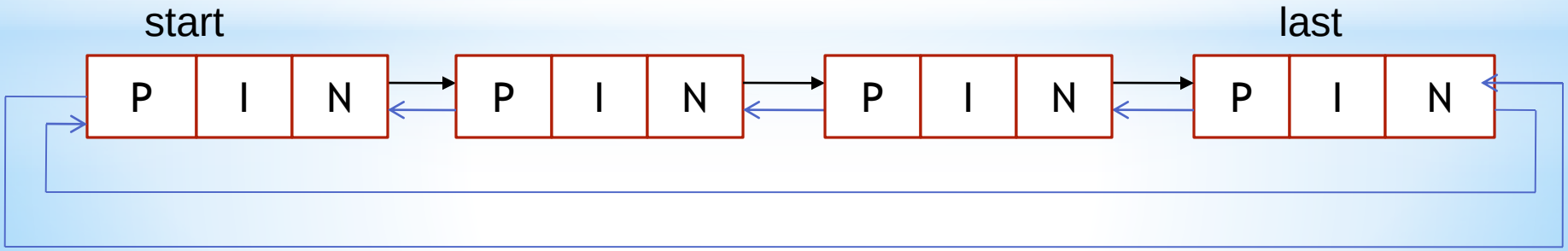
3.4. Danh sách liên kết kép vòng

3.4.1. Định nghĩa: Danh sách liên kết kép vòng là một danh sách liên kết kép sao cho: con trỏ next của node cuối cùng trỏ đến node đầu tiên và con trỏ prev của node đầu tiên trỏ đến node cuối cùng.

3.4.2. Biểu diễn danh sách liên kết kép vòng

Giống như biểu diễn của danh sách kép thông thường nhưng sử dụng con trỏ start luôn là node đầu tiên, con trỏ last luôn là node cuối cùng.

```
struct node {//Biểu diễn node
        int info; //Thành phần thông tin của node
        struct node *next; //Thành phần con trỏ next
        struct node *prev; //Thành phần con trỏ prev
}*start, *last; //start là node đầu tiên, last là node cuối cùng
int  counter =0; //ghi nhận số node của danh sách liên kết vòng
```



3.4.3. Các thao tác trên danh sách liên kết kép vòng

- Tạo node cho danh sách liên kết kép vòng.
- Chèn node vào đầu danh sách liên kết kép vòng.
- Chèn node vào cuối danh sách liên kết kép vòng.
- Chèn node giữa cuối danh sách liên kết kép vòng.
- Thêm node vào node giữa danh sách liên kết kép vòng.
- Loại node tại vị trí bất kỳ trên danh sách liên kết kép vòng.
- Tìm node tại vị trí bất kỳ trên danh sách liên kết kép vòng.
- Sửa đổi thông tin node trên danh sách liên kết kép vòng.
- Hiển thị thông tin trên danh sách liên kết kép vòng.
- Sắp xếp thông tin trên danh sách liên kết kép vòng.

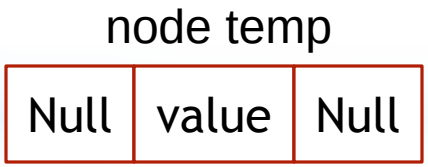
Dưới đây là một cách cài đặt cho các thao tác cơ bản trên danh sách liên kết kép vòng.

Mô tả lớp thao tác trên danh sách liên kết kép vòng:

```
class double_clist {//Mô tả lớp double-clisst
    public:
        node *create_node(int); //Tạo node có giá trị value
        void insert_begin(); //Chèn node vào đầu DSLK kép vòng
        void insert_last(); //Chèn node vào cuối DSLK kép vòng
        void insert_pos(); //Chèn node vào giữa DSLK kép vòng
        void delete_pos(); //Loại node tại vị trí bất kỳ
        void search(); //Tìm node tại vị trí bất kỳ
        void update(); //Sửa đổi thông tin node tại vị trí bất kỳ
        void display(); //Hiển thị nội dung DSLK kép vòng
        void reverse(); // Đảo ngược DSLK kép vòng
        void sort(); // Sắp xếp DSLK kép vòng
        double_clist() {//Constructor DSLK kép vòng
            start = NULL; last = NULL;
        }
};
```

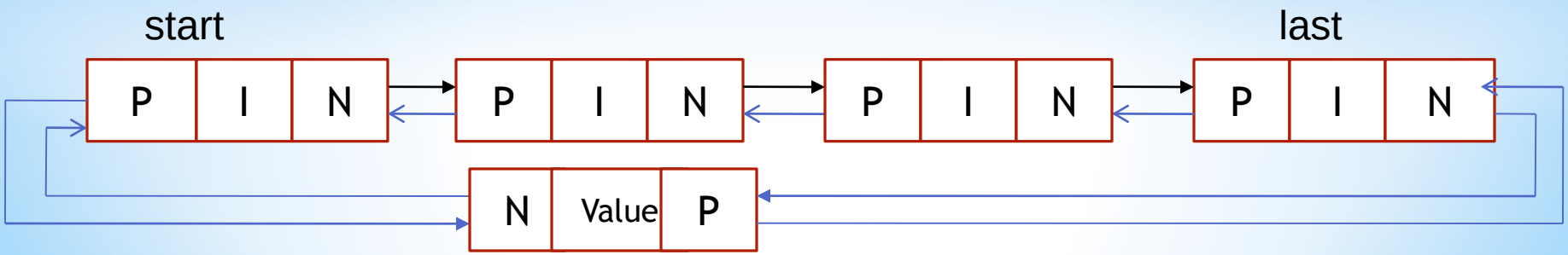
Tạo node có giá trị value trên danh sách liên kết kép vòng:

```
node* double_clist::create_node(int value){//Tạo node có giá trị value
        counter++; //Tăng số node
        struct node *temp; //Sử dụng con trỏ temp
        temp = new(struct node); //Cấp phát miền nhớ cho node
        temp->info = value; //Thiết lập giá trị cho node
        temp->next = NULL; //Thiết lập liên kết next
        temp->prev = NULL; //Thiết lập liên kết prev
        return temp;
}
```



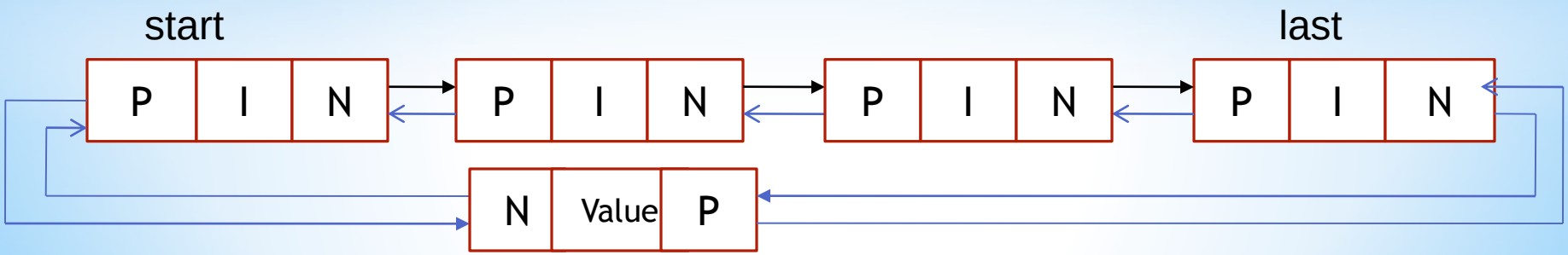
Chèn node vào đầu danh sách liên kết kép vòng:

```
void double_clist::insert_begin(){//Chèn node vào đầu DSLK kép vòng
        int value; cout<<endl<<“Nhập nội dung node: ";cin>>value;
        struct node *temp; // Khai báo con trỏ temp
        temp = create_node(value); // Tạo node có giá trị value
        if (start == last && start == NULL){//Nếu danh sách rỗng
                cout<<“Node được chèn là node duy nhất"<<endl;
                start = last = temp; //node đầu và nde cuối trùng nhau
                start->next = last->next = NULL;
                start->prev = last->prev = NULL;
        }
        else {    //Thiết lập liên kết cho temp, start, last
                temp->next = start; start->prev = temp; start = temp;
                start->prev = last; last->next = start;
                cout<<“Node đã được chèn vào đầu"<<endl;
        }
}
```



Chèn node vào cuối danh sách liên kết kép vòng:

```
void double_clist::insert_last(){//Chèn node vào cuối
        int value;cout<<endl<<"Giá trị node chèn vào cuối: ";cin>>value;
        struct node *temp;//Khai báo node temp
        temp = create_node(value);//Tạo node temp có giá trị value
        if (start == last && start == NULL){ //Nếu danh sách rỗng
                cout<<"Chèn node vào danh sách rỗng"<<endl;
                start = last = temp;start->next = last->next = NULL;
                start->prev = last->prev = NULL;
        }
        else {
                last->next = temp; temp->prev = last; last = temp;
                start->prev = last; last->next = start;
        }
}
```



Chèn node tại vị trí bất kỳ trên danh sách liên kết kép vòng:

```
void double_clist::insert_pos(){    int value, pos, i;
        cout<<endl<<"Giá trị node cần chèn: ";cin>>value;
        cout<<endl<<"Vị trí node cần chèn: "; cin>>pos;
        struct node *temp, *s, *ptr; temp = create_node(value);
        if (start == last && start == NULL){//Nếu danh sách rỗng
                if (pos == 1){// Nếu vị trí cần chèn là 1
                        start = last = temp;start->next = last->next = NULL;
                        start->prev = last->prev = NULL;
                }
                else{ counter--;return; }
        }
        else  {   if (counter < pos){ counter--; return;  }
                s = start;
                for (i = 1;i <= counter;i++){  ptr = s;  s = s->next;
                        if (i == pos – 1) {ptr->next = temp;temp->prev = ptr;
                                temp->next = s; s->prev = temp;
                                cout<<"Hoàn thành"<<endl; break;
                        }
                }
        }
}
```

Loại node tại vị trí bất kỳ trên danh sách liên kết kép vòng:

```
void double_clist::delete_pos(){ int pos, I; node *ptr, *s;
        if (start == last && start == NULL){
                cout<<“Không phải làm gì.”;return;
        }
        cout<<endl<<“Vị trí cần loại bỏ: "; cin>>pos;
        if (counter < pos){ cout<<“Vị trí không hợp lệ"<<endl;return;}
        s = start; //s là node đầu tiên
        if(pos == 1){//Nếu là vị trí đầu tiên
                counter--;  last->next = s->next;
                s->next->prev = last;start = s->next; free(s);
                cout<<“Node đầu tiên đã bị loại"<<endl;
                return;
        }
        for (i = 0;i < pos - 1;i++ ){  //Di chuyển đến vị trí pos-1
                s = s->next;  ptr = s->prev;
        }
        ptr->next = s->next; s->next->prev = ptr;
        if (pos == counter) {  last = ptr; }
        counter--; free(s);
        cout<<“Node đã bị loại bỏ"<<endl;
}
```

Sửa đổi node tại vị trí bất kỳ trên danh sách liên kết kép vòng:

```
void double_clist::update(){//Sửa đổi thông tin cho node
        int value, i, pos;
        if (start == last && start == NULL){
                cout<<"Danh sách rỗng"<<endl;return;
        }
        cout<<endl<<"Vị trí node cần sửa: ";cin>>pos;
        cout<<"Giá trị mới: ";cin>>value;
        struct node *s;
        if (counter < pos){ cout<<"Vị trí không hợp lệ"<<endl;return;}
        s = start;  // s là node đầu tiên
        if (pos == 1) {//Nếu vị trí sửa đổi là vị trí đầu tiên
                s->info= value;cout<<"Node đã được  sửa đổi "<<endl;return;
        }
        for (i=0;i < pos - 1;i++) {//Di chuyển s đến vị trí pos-1
                s = s->next;
        }
        s->info = value;// Cập nhật thông tin sửa đổi
        cout<<"Thông báo sửa thành công"<<endl;
}
```

Tìm kiếm node tại vị trí bất kỳ trên danh sách liên kết kép vòng:

```
void double_clist::search(){//Tìm kiếm node
        int pos = 0, value, i;
        bool flag = false;
        struct node *s;
        if (start == last && start == NULL){
                cout<<"Danh sách rỗng"<<endl;
                return;
        }
        cout<<endl<<"Nội dung node cần tìm: ";cin>>value;
        s = start; //s là node đầu tiên
        for (i = 0;i < counter;i++){  pos++;
                if (s->info == value){
                        cout<<"Tìm thấy "<<value<<" tại vị trí: "<<pos<<endl;
                        flag = true;
                }
                s = s->next;
        }
        if (!flag)
                cout<<"Không tìm thấy"<<endl;
}
```

Sắp xếp node trên danh sách liên kết kép vòng:

```
void double_clist::sort(){//Sap xep
        struct node *temp, *s;   int value, i;
        if (start == last && start == NULL){
                cout<<"Danh sách rỗng"<<endl;
                return;
        }
        s = start; //s là node đầu tiên
        for (i = 0;i < counter;i++){
                temp = s->next;
                while (temp != start){
                        if (s->info > temp->info){
                                value = s->info;
                                s->info = temp->info;
                                temp->info = value;
                        }
                        temp = temp->next;
                }
                s = s->next;
        }
}
```

Hiển thị nội dung danh sách liên kết kép vòng:

```
void double_clist::display(){//Hiển thị nội dung danh sách
        int i;
        struct node *s;
        if (start == last && start == NULL){
                cout<<"Danh sách rỗng"<<endl;
                return;
        }
        s = start;//s là node đầu tiên
        for (i = 0;i < counter-1;i++){
                cout<<s->info<<"<->"; //Hiển thị thông tin của s
                s = s->next; //s trỏ đến node tiếp theo
        }
        cout<<s->info<<endl; //Hiển thị node cuối cùng
}
```

Đảo ngược danh sách liên kết kép vòng:

```
void double_clist::reverse(){//Dao nguoc
        if (start == last && start == NULL){
                cout<<“Danh sách rỗng"<<endl;
                return;
        }
        struct node *p1, *p2;
        p1 = start;
        p2 = p1->next;
        p1->next = NULL;
        p1->prev = p2;
        while (p2 != start){
                p2->prev = p2->next;
                p2->next = p1;
                p1 = p2;
                p2 = p2->prev;
        }
        last = start; start = p1;
        cout<<“Danh sách đã được đảo ngược"<<endl;
}
```

Bài tập 6. Hoàn thành các thao tác trên danh sách liên kết kép

- Khởi tạo danh sách liên kết kép.
- Cấp phát miền nhớ cho một node.
- Thêm node vào đầu bên trái danh sách liên kết kép.
- Thêm node vào đầu bên phải danh sách liên kết kép.
- Thêm node vào node giữa danh sách liên kết kép.
- Loại node cuối bên trái danh sách liên kết kép.
- Loại node cuối bên phải danh sách liên kết kép.
- Loại node ở giữa danh sách liên kết kép.
- Duyệt trái danh sách liên kết kép.
- Duyệt phải danh sách liên kết kép.
- Tìm node trên danh sách liên kết kép…

Dưới đây là một số thao tác cơ bản trên danh sách liên kết kép.

Bài tập 7. Hoàn thành các thao tác trên danh sách liên kết kép vòng

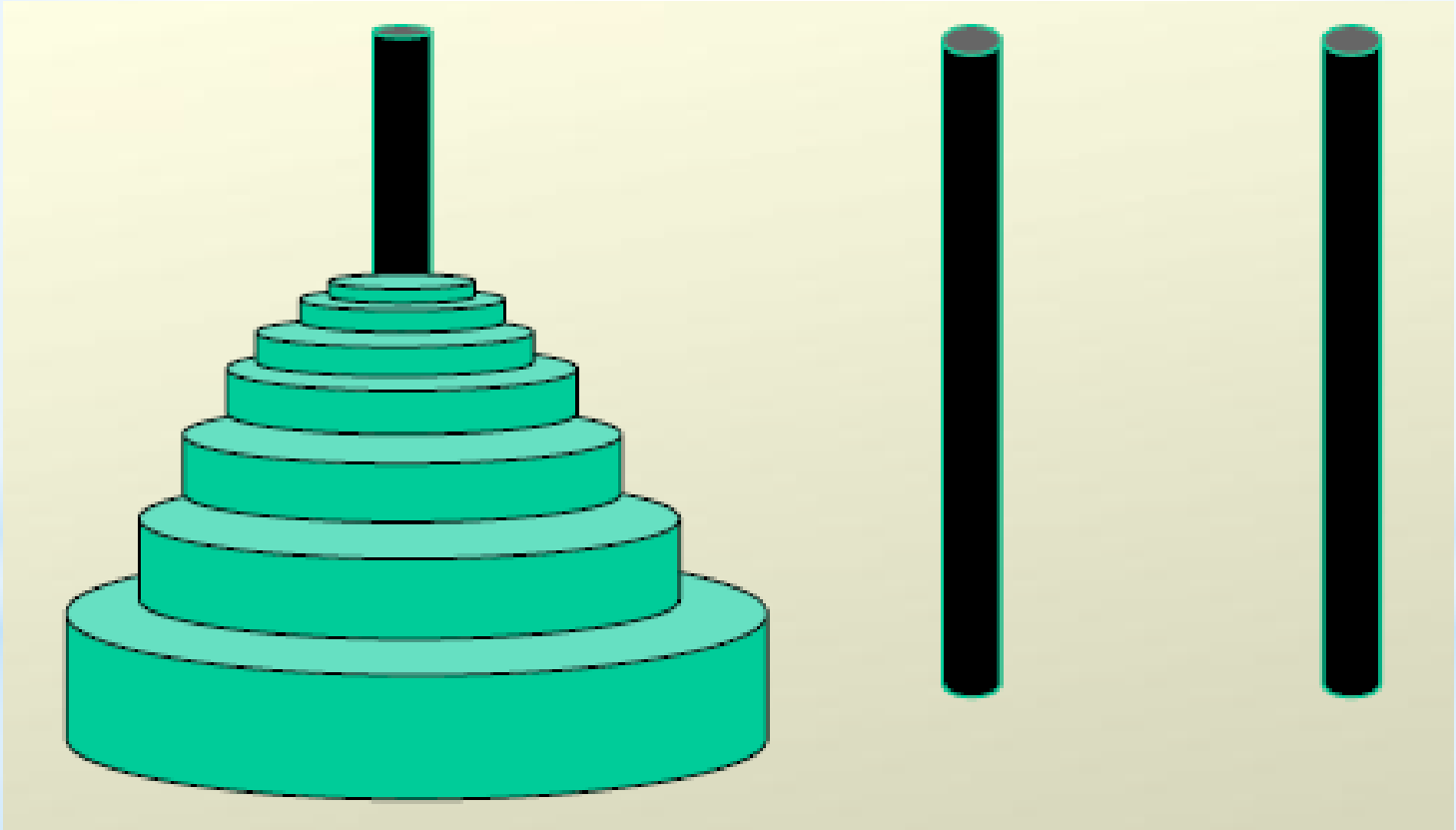
- Tạo node cho danh sách liên kết kép vòng.
- Chèn node vào đầu danh sách liên kết kép vòng.
- Chèn node vào cuối danh sách liên kết kép vòng.
- Chèn node giữa cuối danh sách liên kết kép vòng.
- Thêm node vào node giữa danh sách liên kết kép vòng.
- Loại node tại vị trí bất kỳ trên danh sách liên kết kép vòng.
- Tìm node tại vị trí bất kỳ trên danh sách liên kết kép vòng.
- Sửa đổi thông tin node trên danh sách liên kết kép vòng.
- Hiển thị thông tin trên danh sách liên kết kép vòng.
- Sắp xếp thông tin trên danh sách liên kết kép vòng.

Dưới đây là một cách cài đặt cho các thao tác cơ bản trên danh sách liên kết kép vòng.

3.5. Ngăn xếp (Stack)

3.5.1. Định nghĩa. Tập hợp các node thông tin được tổ chức liên tục hoặc rời rạc nhau trong bộ nhớ và thực hiện theo cơ chế FILO (First – In – Last – Out).

Ví dụ. Bài toán tháp Hà Nội.



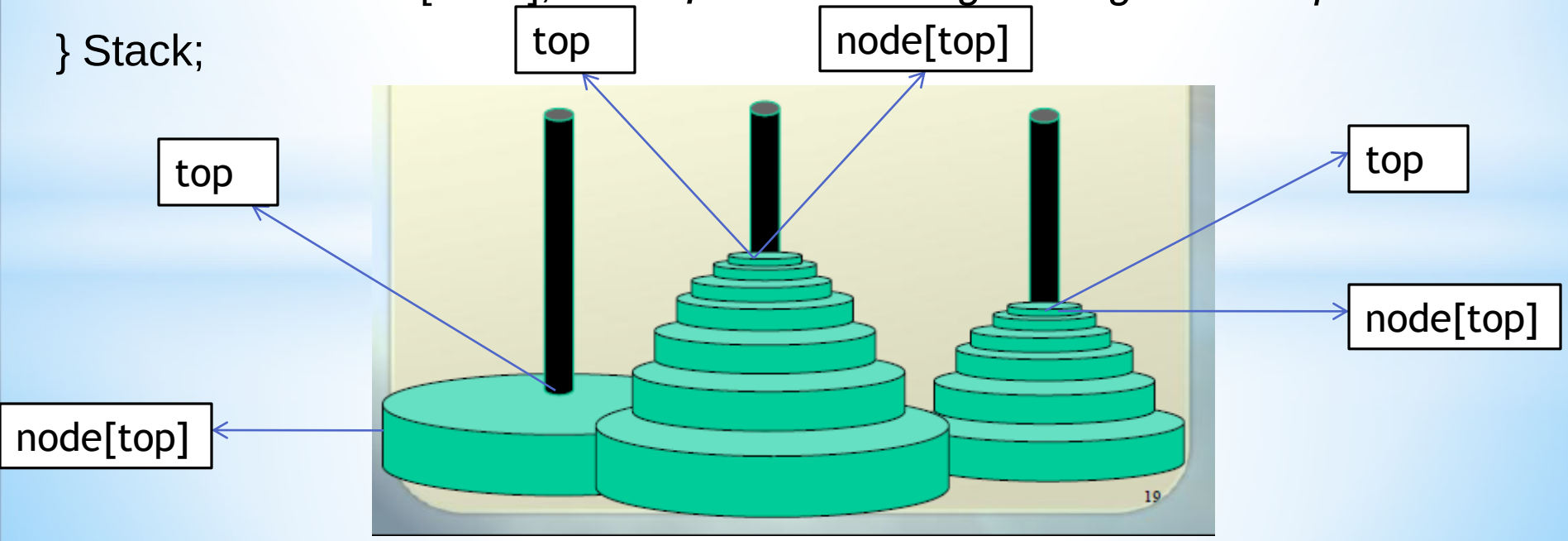
3.5.2. Biểu diễn dựa vào mảng

Có hai phương pháp biểu diễn ngăn xếp:

- **Biểu diễn liên tục**: các phần tử dữ liệu của ngăn xếp được lưu trữ liên tục nhau trong bộ nhớ (Mảng).
- **Biểu diễn rời rạc**: các phần tử dữ liệu của ngăn xếp được lưu trữ rời rạc nhau trong bộ nhớ (Danh sách liên kết).

Ví dụ. Biểu diễn ngăn xếp dựa vào mảng.

```
typedef struct {
        int     top; //Đỉnh đầu của stack nơi diễn ra mọi thao tác
        int     node[MAX]; //Dữ liệu lưu trữ trong stack gồm MAX phần tử
} Stack;
```

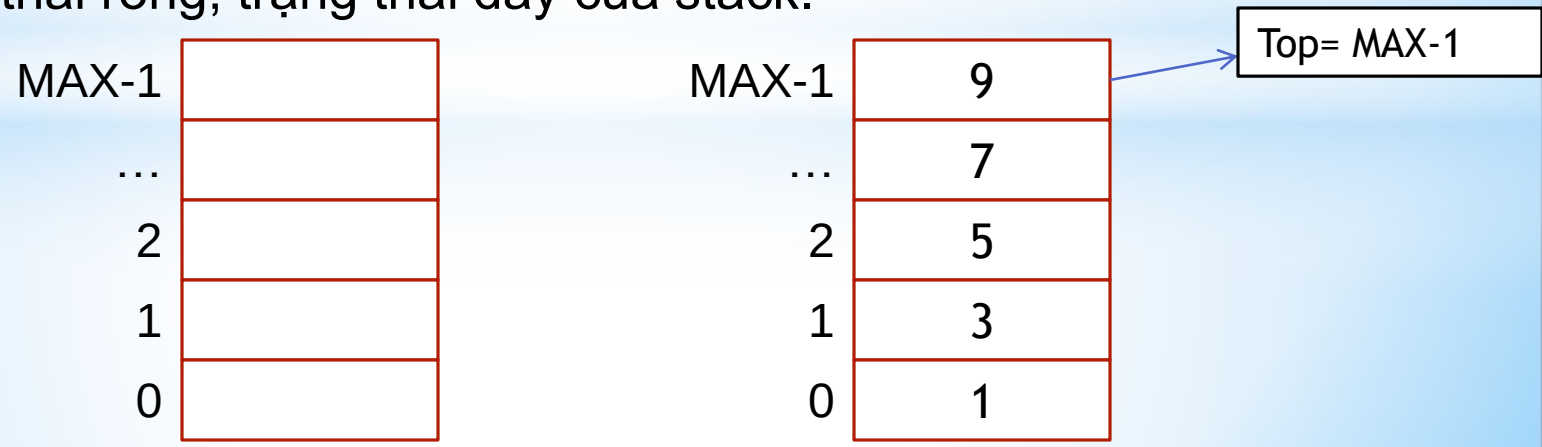


3.5.3. Các thao tác trên stack dựa vào mảng

Các thao tác xây dựng trên stack theo cơ chế Last-In-First-Out bao gồm:

- **Kiểm tra tích rỗng của stack** (*Empty*(*stack s*))**.** Khi ta muốn lấy dữ liệu ra khỏi ngăn xếp thì nó chỉ được thực hiện khi và chỉ khi ngăn xếp không rỗng.
- **Kiểm tra tính đầy của ngăn xếp** (*Full*(*stack s*))**.** Khi ta muốn đưa dữ liệu vào ngăn xếp thì nó chỉ được thực hiện khi và chỉ khi ngăn xếp chưa tràn.
- **Đưa dữ liệu vào ngăn xếp** (*Push*(*stack s, item x*))**.** Thao tác này chỉ được thực hiện khi và chỉ khi ngăn xếp chưa tràn.
- **Đưa dữ liệu vào ngăn xếp** (*Pop*(*stack s*)**.** Thao tác này chỉ được thực hiện khi và chỉ khi ngăn xếp không rỗng.

Ví dụ. Trạng thái rỗng, trạng thái đầy của stack.



Kiểm tra tính rỗng của stack:

Tất cả các thao tác trên stack chỉ được thực hiện tại vị trí con trỏ top. Vì vậy ta qui ước tại vị trí top = -1 stack ở trạng thái rỗng. Thao tác được thực hiện như sau:

```
typedef struct {
        int    top; //con trỏ top
        int    node[MAX]; //Các node của stack
} stack;
int     Empty( stack *s ) { //s là con trỏ đến stack
        if ( stack ->top == -1 )  // Nếu top =-1
            return (1); //Hàm trả lại giá trị đúng
        return(0); //Hàm trả lại giá trị sai
}
```

Kiểm tra tính đầy của stack:

Khi ta muốn lấy dữ liệu ra khỏi ngăn xếp thì ngăn xếp phải chưa tràn. Vì biểu diễn dựa vào mảng, do đó tại vị trí top = MAX -1 thì stack ở trạng thái đầy. Thao tác được thực hiện như sau:

```
typedef struct {
        int    top; //con trỏ top
        int    node[MAX]; //Các node của stack
} stack;
int     Full( stack *s ) { //s là con trỏ đến stack
        if ( stack ->top == MAX-1 )  // Nếu top = MAX -1
            return (1); //Hàm trả lại giá trị đúng
        return(0); //Hàm trả lại giá trị sai
}
```

Đưa dữ liệu vào stack:

Khi muốn đưa dữ liệu vào ngăn xếp thì ta phải kiểm tra ngăn xếp có đầy (tràn) hay không? Nếu ngăn xếp chưa đầy, thao tác sẽ được thực hiện. Nếu ngăn xếp đã đầy, thao tác không được thực hiện. Thao tác được thực hiện như sau:

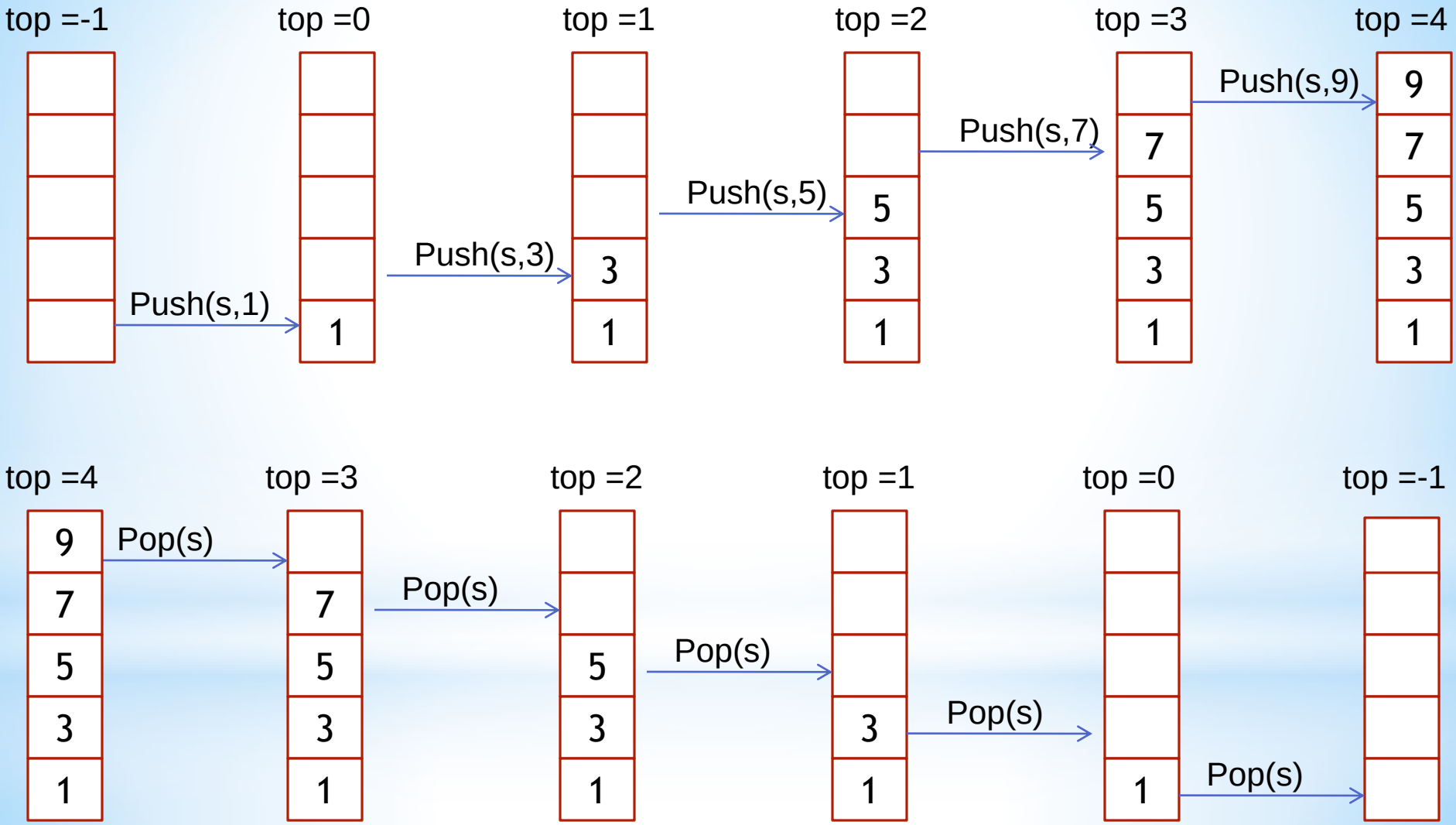
```
typedef struct {
        int    top; //con trỏ top
        int    node[MAX]; //Các node của stack
} stack;
void     Push( stack *s, int   x ) { //x là node cần thêm vào stack
        if ( !Full (s))  // Nếu stack chưa tràn
            s ->top = (s ->top) +1; //Tăng con trỏ top lên 1 đơn vị
            tode[s ->top] = x; //Lưu trữ x tại vị trí top
        }
        else <thông báo tràn stack>;
}
```

Lấy dữ liệu ra khỏi stack:

Khi muốn lấy dữ liệu ra khỏi ngăn xếp thì ta phải kiểm tra ngăn xếp có rỗng hay không? Nếu ngăn xếp không rỗng, thao tác sẽ được thực hiện. Nếu ngăn xếp rỗng, thao tác không được thực hiện. Thao tác được thực hiện như sau:

```
typedef struct {
        int    top; //con trỏ top
        int    node[MAX]; //Các node của stack
} stack;
int     Pop( stack *s ) { //s là con trỏ đến stack
        if ( !Empty (s)) { // Nếu stack không rỗng
            x = Node[s ->top]; //x là nội dung node bị lấy ra
            s ->top = (s ->top) -1; //giảm con trỏ top 1 đơn vị
            return (x); //Trả lại x là node bị loại bỏ
        }
        else { <thông báo stack rỗng>; return (∞); }
}
```

Ví dụ. Cho stack lưu trữ tối đa 5 phần tử. Bắt đầu thực hiện tại trạng thái rỗng.



3.5.4. Các thao tác trên stack dựa danh sách liên kết đơn

Xây dựng Stack bằng danh sách liên kết đơn được thực hiện như sau:

Stack = { DSLK đơn + [<Add-Top:(Push)>; <Del-Top: (Push)>] }

Stack = { DSLK đơn + [<Add-Bottom:(Push)>; <Del-Bottom: (Push)>] }

Lớp các thao tác trên stack được xây dựng như sau:

```
struct node{//Biểu diễn stack
        int info; //Thành phần thông tin của node
        struct node *link; //Thành phần con trỏ của node
}*Stack;
class stack_list{ //Mô tả lớp stack_list
        public:
                node *push(node *, int); //Thêm node
                node *pop(node *); //Loại bỏ node
                void traverse(node *);//Duyệt stack
                stack_list(){   Stack = NULL; }  //Constructor của lớp
};
```

```
node *stack_list::push(node *Stack, int item){ //Thêm node vào stack
        node *tmp; tmp = new (struct node);
        tmp->info = item; tmp->link = Stack; Stack = tmp;
        return Stack;
}
node *stack_list::pop(node *Stack){  node *tmp;
        if (Stack == NULL)  cout<<"Stack rỗng"<<endl;
        else  {  tmp = Stack;
                cout<<“Node đã bị loại bỏ: "<<tmp->info<<endl;
                Stack = Stack->link; free(tmp);
        }
        return Stack;
}
void stack_list::traverse(node *Stack){
        node *ptr; ptr = Stack;
        if (Stack == NULL)  cout<<"Stack rỗng"<<endl;
        else  {  cout<<“Các phần tử của stack :"<<endl;
                while (ptr != NULL)   {
                        cout<<ptr->info<<endl; ptr = ptr->link;
                }
        }
}
```

3.5.5. Ứng dụng của ngăn xếp

- Xây dựng các giải thuật đệ qui.
- Khủ bỏ các giải thuật đệ qui.
- Biểu diễn tính toán.
- Duyệt cây, duyệt đồ thị…

Ví dụ 1. Biểu diễn các biểu thức thức số học dạng hậu tố

- a + b $\Leftrightarrow$ a b +
- a - b $\Leftrightarrow$ a b -
- a * b $\Leftrightarrow$ a b *
- a / b $\Leftrightarrow$ a b /
- (P) $\Leftrightarrow$ P

Ví dụ.

$$(a+b*c)-(a/b+c)=$$
$$=(a+bc*)-(ab/+c)$$
$$=(abc*+)-(ab/c+)$$
$$=abc*+-ab/c+$$
$$=abc*+ab/c+-$$

Ví dụ 1. Tính toán biểu thức số học hậu tố

- a + b $\Leftrightarrow$ a b +
- a - b $\Leftrightarrow$ a b -
- a * b $\Leftrightarrow$ a b *
- a / b $\Leftrightarrow$ a b /
- (P) $\Leftrightarrow$ P

Ví dụ.

$$(a+b*c)-(a/b+c)=$$

$$=(a+bc*)-(ab/+c)$$

$$=(abc*+)-(ab/c+)$$

$$=abc*+-ab/c+$$

$$=abc*+ab/c+-$$

Thuật toán chuyển đổi biểu thức trung tố P thành biểu thức hậu tố?

Bước 1 (Khởi tạo)**:** *stack = ∅; Out = ∅;*

Bước 2 (Lặp) :

For each x∈P do

2.1. Nếu x =‘ (‘ : Push(stack, x);

2.2. Nếu x là toán hạng: x⇒Out;

*2.3. Nếu x ∈ { +, -, *, / }*

y = get(stack);

a) Nếu priority(x)≥priority(y): Push(stack, x);

b) Nếu priority(x)<priority(y):

y = Po(stack); y⇒Out; Push(stack, x);

c) Nếu stack = ∅: Push(stack, x);

2.4. Nếu x =‘)’:

y = Pop(stack);

While (y!=‘(’) do

y⇒Out; y = Pop(stack);

EndWhile;

EndFor;

Bước 3(Hoàn chỉnh biểu thức hậu tố)*:*

While (stack ≠ ∅) do

y = Pop(stack); y⇒Out;

EndWhile;

Bước 4(Trả lại kết quả)*:*

Return(Out).

Kiểm nghiệm thuật toán: P = $(a+b*c)-(a/b+c)$

x∈P	Bước	Stack	Out
x =‘(‘	2.1	(	∅
x =a	2.2	(	a
x =+	2.3.a	(+	a
x =b	2.2	(+	a b
x =*	2.3.a	(+ *	a b
x = c	2.2	(+ *	a b c
x =‘)‘	2.3	∅	a b c * +
x =-	2.2.c	-	a b c * +
x =‘(‘	2.1	- (	a b c * +
x =a	2.2	- (	a b c * + a
x =/	2.2.a	- (/	a b c * + a
x =b	2.2	- (/	a b c * + a b
x =+	2.3.b	- (+	a b c * + a b/
x =c	2.2	- (+	a b c * + a b/c
x =‘)‘	2.4	∅	a b c * + a b/c + -
P = a b c * + a b/c + -			

Thuật toán tính toán giá trị biểu thức hậu tố?

Bước 1 (Khởi tạo)**:**

stack = ∅;

Bước 2 (Lặp) :

For each x∈P do

2.1. Nếu x là toán hạng:

Push(stack, x);

*2.2. Nếu x ∈ { +, -, *, / }*

a) TH2 = Pop(stack, x);

b) TH1 = Pop(stack, x);

c) KQ = TH1 ⊗ TH2;

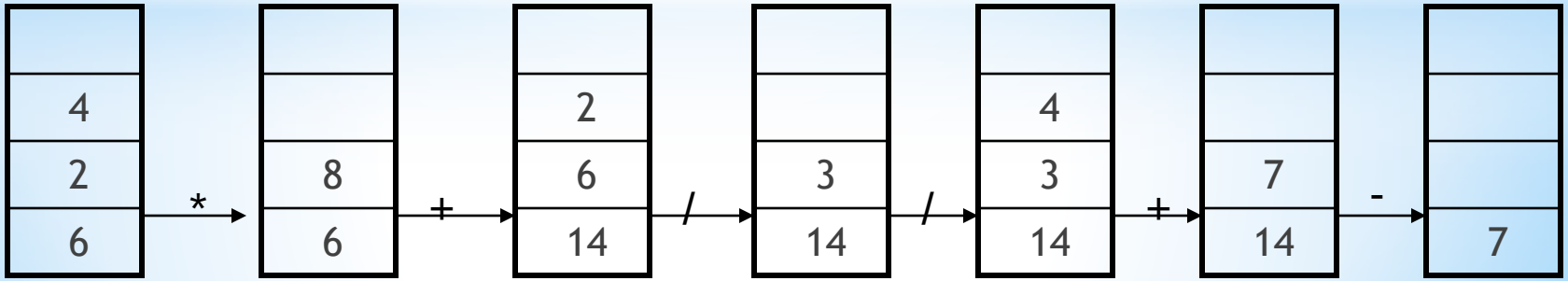
d) Push (stack, KQ);

EndFor;

Bước 4(Trả lại kết quả)*:*

Return(Pop(stack)).

Ví dụ: P = 6 2 4 * + 6 2 / 4 + -



3.6. Hàng đợi (queue)

3.6.1. Định nghĩa hàng đợi (queue).

Tập hợp các node thông tin được tổ chức liên tục hoặc rời rạc nhau trong bộ nhớ và thực hiện theo cơ chế FIFO (First–In–First–Out).

3.6.2. Biểu diễn hàng đợi dựa vào mảng

Có hai phương pháp biểu diễn hàng đợi:

- Biểu diễn liên tục: sử dụng mảng.
- Biểu diễn rời rạc: sử dụng danh sách liên kết.

Trong trường hợp sử dụng mảng, mỗi node của hàng đợi được biểu diễn như một cấu trúc gồm 3 thành viên như sau:

```
typedef struct {
    int     inp; // con trỏ inp biểu diễn lối vào của hàng đợi
    Int     out; // con trỏ out biểu diễn lối ra của hàng đợi
    Int     node[MAX]; //Các node thông tin của hàng đợi
} queue;
```

3.6.3. Các thao tác trên hàng đợi

Các thao tác trên hàng đợi bao gồm:

- Kiểm tra tính rỗng của hàng đợi (Empty(queue q)). Thao tác này được sử dụng khi ta cần đưa dữ liệu vào hàng đợi.
- Kiểm tra tính đầy của hàng đợi (Empty(queue q)). Thao tác này được sử dụng khi ta muốn lấy dữ dữ liệu ra khỏi hàng đợi.
- Thao tác đưa dữ liệu vào hàng đợi (Push(queue q, int x)).
- Thao tác lấy dữ liệu ra khỏi hàng đợi (Pop(queue q)).

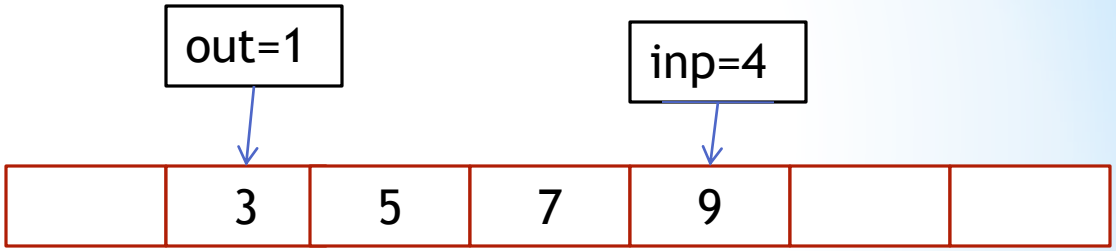
Các kiểu hàng đợi:

- **Hàng đợi tuyến tính**:
  - Các thao tác được xây dựng sao cho con trỏ inp luôn có giá trị lớn hơn con trỏ out (inp > out). Đối với hàng đợi tuyến tính này, các ô nhớ con trỏ inp và con trỏ out đã đi qua sẽ không được sử dụng lại.
  - Các thao tác được xây dựng sao cho con trỏ inp luôn có giá trị lớn hơn con trỏ out (inp > out) và các ô nhớ con trỏ inp đi qua sẽ được phép sử dụng lại bằng cách luôn ngầm định con trỏ out luôn thực hiện tại vị trí out = 0.

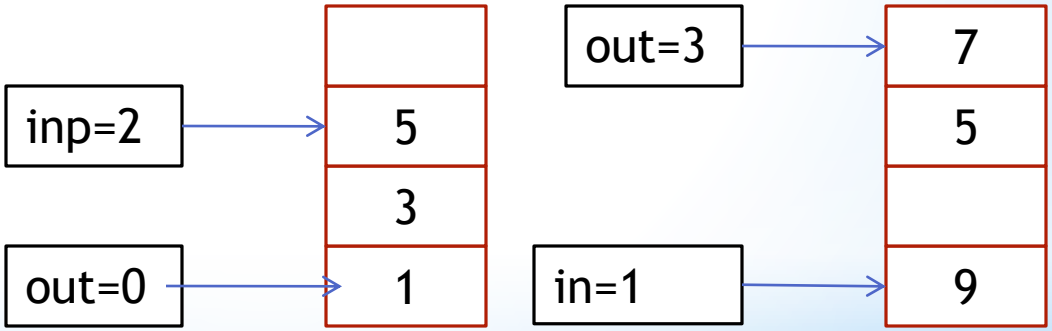
•**Hàng đợi vòng**: Các thao tác được xây dựng sao cho các ô nhớ con trỏ inp và con trỏ out đã đi qua sẽ được sử dụng lại. Đối với hàng đợi vòng, nhiều trạng thái inp<out và nhiều trạng thái inp>out.

• **Hàng đợi ưu tiên**: mỗi phần tử của hàng đợi được gắn với một độ ưu tiên sao cho phần tử nào có độ ưu tiên cao sẽ được lưu trữ gần con trỏ out nhất.

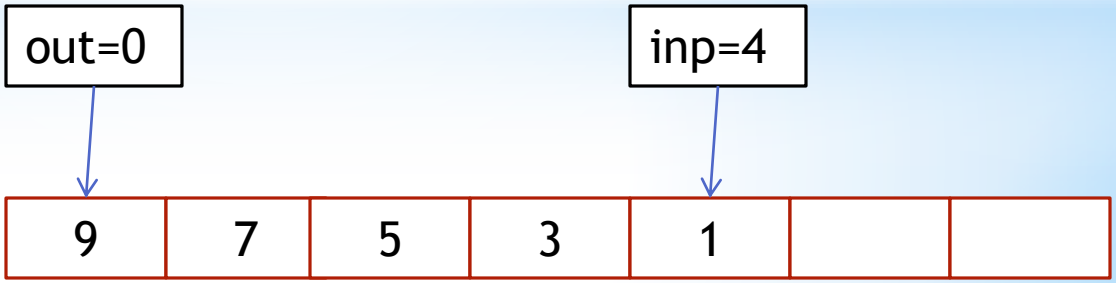
Hàng đợi tuyến tính:



Hàng đợi vòng:



Hàng đợi ưu tiên:



Biểu diễn hàng đợi:

```
typedef struct {
    int     inp; // con trỏ inp biểu diễn lối vào của hàng đợi
    Int     out; // con trỏ out biểu diễn lối ra của hàng đợi
    Int     node[MAX]; //Các node thông tin của hàng đợi
} queue;
```

Kiểm tra tính rỗng của hàng đợi: hàng đợi tuyến tính rỗng khi nó ở trạng thái khởi đầu (inp = out = -1) hoặc khi con trỏ in = out. Như vậy, trong cả hai trường hợp inp = out thì trạng thái hàng đợi rỗng.

```
int   Empty( queue  *q) { //q là con trỏ kiểu queue
    If ( q ->inp == q ->out) //Nếu q ->inp = q ->out
        return (1); //Trả lại giá trị đúng
    return(0); //Trả lại giá trị sai
}
```

Kiểm tra tính đầy của hàng đợi: hàng đợi tuyến tính đầy khi con trỏ inp ở vị trí MAX-1 (inp = MAX -1).

```
int   Full( queue  *q) { //q là con trỏ kiểu queue
      If ( q ->inp == MAX -1 ) //Nếu q ->inp = MAX-1
            return (1); //Trả lại giá trị đúng
      return(0); //Trả lại giá trị sai
}
```

Đưa dữ liệu vào hàng đợi:

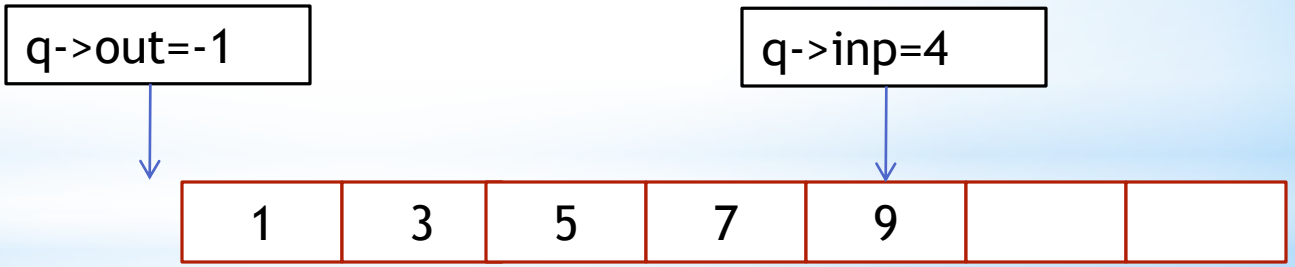
```
void Push( queue  *q, int   x) { //x là dữ liệu cần lưu trữ trong queue
      If ( !Full(q)) //Nếu đúng q chưa đầy: q->in <MAX-1
            q ->inp = (q ->in) +1; //Nhấc con trỏ in lên 1 đơn vị
            q->node[q->inp] = x; //Lưu trữ x tại vị trí q->inp
      }
      else <Thông báo tràn queue>;
}
```

Lấy dữ liệu ra khỏi hàng đợi:

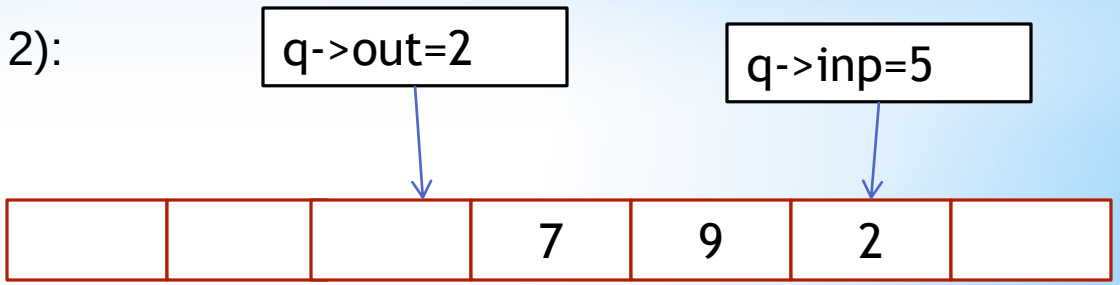
```
int Pop( queue  *q) { //q là con trỏ kiểu queue
    If ( !Empty (q)) //Nếu đúng q không rỗng
        q ->out = (q ->out) +1; //Nhấc con trỏ out lên 1 đơn vị
        int x = q->node[q->inp]; // x là giá trị node được lấy ra
        return(x); //Trả lại giá trị node loại bỏ
    }
    else { <Thông báo stack rỗng>; return (∞); }
}
```

Hoạt động của Push và Pop:

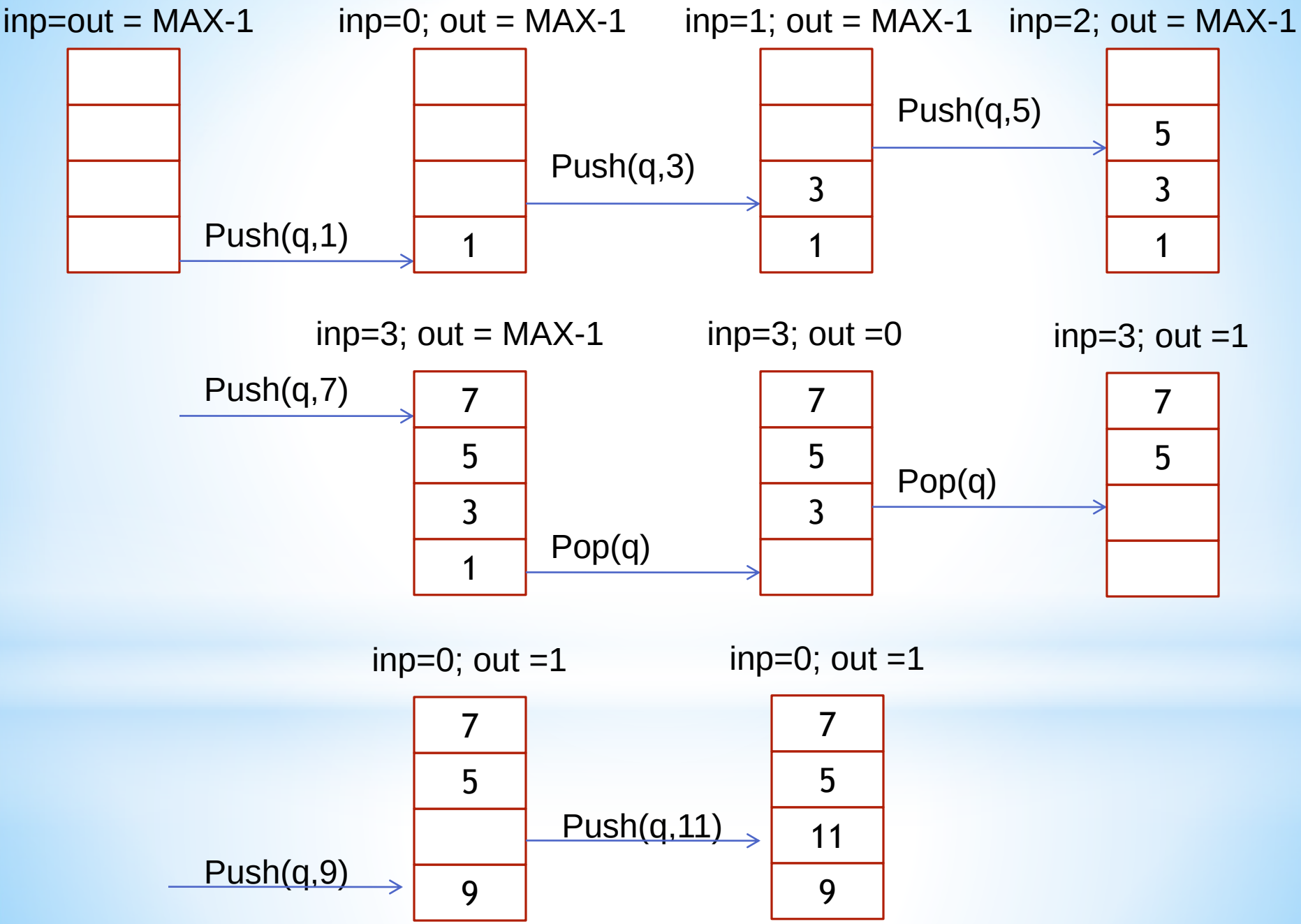
Push(q,1); Push(q,3); Push(q,5); Push(q,7); Push(q,9);



Pop (q); Pop (q); Pop(q); Push(q, 2):



Hàng đợi vòng: Hàng đợi rỗng: inp=out = MAX-1



3.6.4. Cài đặt hàng đợi bằng danh sách liên kết đơn

Cài đặt hàng đợi bằng danh sách liên kết đơn được thực hiện bằng cách:

Queue = { DSLKĐ + [Add-Top + Del-Bottom] }

Queue = { DSLKĐ + [Add-Bottom + Del-Top] }

Biểu diễn hàng đợi bằng danh sách liên kết đơn:

```
struct node {
        int data;
        node *next;
}*front = NULL,*rear = NULL,*p = NULL,*np = NULL;
```

Thao tác đưa phần tử vào hàng đợi:

```
void push(int x) {np = new node; //Cấp phát bộ nhớ cho np
        np->data = x;  np->next = NULL; //Thiết  lập liên kết cho np
        if(front == NULL) { //Nếu lối vào là rỗng
                front = rear = np; //
                rear->next = NULL;
        }
        else   {
                rear->next = np; rear = np;
                rear->next = NULL;
        }
}
```

Thao tác lấy phần tử ra khỏi hàng đợi:

```
int remove() { //Lấy phần tử ra khỏi hàng đợi
        int x;
        if(front == NULL)  { //Nếu hàng đợi rỗng
                cout<<"Hàng đợi rỗng"<<endl;
        }
        else   { //Nếu hàng đợi không rỗng
                p = front; //q là node sau cùng
                x = p->data; //Nội dung node q
                front = front->next; //Chuyển front đến node kế tiếp
                delete(p);//Giải phóng p
                return(x); //trả lại kết quả
        }
}
```

3.6.5.Cài đặt hàng đợi vòng bằng mảng

Biểu diễn lớp hàng đợi vòng:

```
class Circular_Queue { //Mô tả lớp Circular_Queue
    private:
        int *cqueue_arr;
        int front, rear; //Lối vào và lối ra hàng đợi
    public:
        Circular_Queue(){ //Constructor
            cqueue_arr = new int [MAX];
            rear = front = -1;
        }
        void    Push(int item);
        void    Pop(void);
        void    Display(void);
}
```

Thao tác Push:

```
void Circular_Queue :: Push(int item) { //Thêm phần tử vào hàng đợi vòng
        if ((front == 0 && rear == MAX-1) || (front == rear+1))  {
                cout<<"Tràn hàng đợi"<<endl;  return;
        }
        if (front == -1) {  front = 0;  rear = 0; }
        else    {
                if (rear == MAX – 1)    rear = 0;
                else   rear = rear + 1;
        }
        cqueue_arr[rear] = item ;
}
```

Thao tác Pop:

```
void Circular_Queue :: Pop() { //Loại phần tử khỏi hàng đợi vòng
        if (front == -1) {cout<<"Queue rỗng"<<endl;  return ;}
        cout<<"Phần tử bị loại bỏ là : "<<cqueue_arr[front]<<endl;
        if (front == rear) { front = -1; rear = -1; }
        else   {
                if (front == MAX – 1)    front = 0;
                else    front = front + 1;
        }
}
```

Thao tác Hiển thị hàng đợi vòng:

```
void Circular_Queue :: Display() { //Hiển thị hàng đợi vòng
        int front_pos = front, rear_pos = rear;
        if (front == -1) { cout<<"Hàng đợi rỗng"<<endl; return; }
        cout<<"Các phần tử của hàng đợi :";
        if (front_pos <= rear_pos)  {
                while (front_pos <= rear_pos) {
                        cout<<cqueue_arr[front_pos]<<"  "; front_pos++;
                }
        }
        else {
                while (front_pos <= MAX – 1) {
                        cout<<cqueue_arr[front_pos]<<"  "; front_pos++;
                }
                front_pos = 0;
                while (front_pos <= rear_pos) {
                        cout<<cqueue_arr[front_pos]<<"  "; front_pos++;
                }
        }
        cout<<endl;
}
```

3.6.5. Ứng dụng của hàng đợi

- Biểu diễn tính toán
- Duyệt cây
- Duyệt đồ thị.
- Xây dựng các thuật toán lập lịch.

Ví dụ 1. Bài toán người sản xuất và nhà tiêu dùng.

Ví dụ 2. Thuật toán Best Fit.

Ví dụ 3. Thuật toán Best Availble.

Ví dụ 4. Bữa tối của 5 nhà triết học (The Dinner of Five Philosophers).

Ví dụ 5. Thuật toán Round-Robin.

3.7. Hàng đợi ưu tiên

3.7.1. **Định nghĩa**. Hàng đợi ưu tiên (Priority Queue) là một mở rộng của cấu trúc dữ liệu hàng đợi có những đặc tính sau:

- Mỗi phần tử của hàng đợi được gắn với độ ưu tiên của nó.
- Node có độ ưu tiên cao hơn sẽ đứng trước node có độ ưu tiên thấp hơn trong hàng đợi.
- Nếu hai phần tử có độ ưu tiên giống nhau thì thứ tự ưu tiên tuân theo luật của hàng đợi thông thường (vào trước ra trước).

3.7.2. Biểu diễn

```
struct node{ //Cấu trúc một node của hàng đợi ưu tiên
        int priority; //Mức độ ưu tiên của node
        int info; //Thông tin của node
        struct node *link; //Con trỏ liên kết của node
};
```

3.7.3. Thao tác trên hàng đợi ưu tiên

- Thao tác Push: đưa vào hàng đợi với mức độ ưu tiên của nó.
- Thao tác Pop: loại bỏ phần tử có mức độ ưu tiên cao nhất.

Lớp biểu các thao tác trên hàng đợi ưu tiên được cài đặt bằng C++ như sau:

Biểu diễn node của hàng đợi ưu tiên:

```
struct node{
        int priority; //Mức độ ưu tiên của node
        int info; //Thông tin của node
        struct node *link; //Liên kết của node
};
```

Lớp thao tác trên hàng đợi ưu tiên:

```
class Priority_Queue{
  private:      node *front; //thành phần private
  public:
        Priority_Queue(){ front = NULL;  } //Constructor lớp Priority_Queue
        void Push(int item, int priority); //Đưa phần tử queue
        void Pop(void) ; //Loại bỏ phần tử có độ ưu tiên cao nhất
        void display(); //Hiển thị hàng đợi ưu tiên
};
```

Đưa phần tử có mức độ ưu tiên vào hàng đợi:

```
void Priority_Queue::Push(int item, int priority) {
        node *tmp, *q; //Sử dụng hai con trỏ tmp và q
        tmp = new node; //Cấp phát miền nhớ cho tmp
        tmp->info = item; //Thiết lập nội dung cho tmp
        tmp->priority = priority; //Thiết lập độ ưu tiên cho tmp
        if (front == NULL || priority < front->priority){ //Nếu hàng đợi rỗng
                tmp->link = front; //Node đầu tiên là tmp
                front = tmp; //Hiện tại hàng đợi có một node
        }
        else {    q = front; //q trỏ đến front
                while (q->link != NULL && q->link->priority <= priority)
                        q=q->link; //Tìm vị trí thích hợp cho độ ưu tiên
                tmp->link = q->link; //Thiết lập liên kết cho tmp
                q->link = tmp;//thiết lập liên kết cho q
        }
}
```

Loại bỏ phần tử ra khỏi hàng đợi:

```
void Priority_Queue::Pop(){ node *tmp; //Sử dụng con trỏ tmp
        if(front == NULL) cout<<"Hàng đợi rỗng\n";
        else {  tmp = front; //tmp trỏ đến front
                cout<<"Node loại bỏ là: "<<tmp->info<<endl;
                front = front->link; free(tmp);//Giải phóng tmp
        }
}
```

Hiển thị hàng đợi ưu tiên:

```
void Priority_Queue::display(){    node *ptr;  ptr = front;
        if (front == NULL)   cout<<"Hàng đợi rỗng"<<endl;
        else{     cout<<"Nội dung hàng đợi:"<,endl;
                cout<<"Độ ưu tiên các node:"<<endl;
                while(ptr != NULL){ cout<<ptr->priority<<" "<<ptr->info<<endl;
                        ptr = ptr->link;
                }
        }
}
```

3.8. Hàng đợi hai điểm cuối (double ended queue):

3.8.1.Định nghĩa

Hàng đợi dqueue(*hàng đợi hai điểm cuối*) là loại hàng đợi đặc biệt cho phép thêm và loại bỏ phần tử tại điểm cuối của hàng đợi.

3.8.2. Biểu diễn dqueue

```
struct node {
        int info; //Thông tin của node
        node *next; //Con trỏ đến node tiếp theo
        node *prev;//Con trỏ đến node sau nó

}*head, *tail;
```

head: là con trỏ đến đỉnh đầu; tail là con trỏ đến đỉnh cuối.

3.8.3. Ứng dụng của dqueue

- dqueue hỗ trợ các thao tác cho cả stack và queue.
- dqueue hỗ trợ các phép quay theo chiều kim đồng hồ hoặc bán nửa kiem đồng hồ.
- Các thao tác thêm và loại bỏ được thực hiện hiệu quả với độ phức tạp hằng số.

3.8.4. Ngôn ngữ lập trình

- STL của C++ hỗ trợ các thao tác trên dqueue.
- Java cung cấp interface cho dqueue.

3.8.5. Các thao tác trên dqueue

Các thao tác trên dqueu bao gồm:

- Insert-Front(): thêm một node vào đầu dqueue.
- Insert-Last(): thêm một node cuối đầu dqueue.
- Del-Front(): loại một node ở đầu dqueue.
- Del-Last(): Loại một node ở cuối dqueue
- Display-Front(): hiển thị dqueue bắt đầu từ đầu trước
- Display-Last(): hiển thị dqueue bắt đầu từ đầu sau
- GetFront(): nhận node ở đầu dqueue
- GetLasst(): nhận node cuối dqueue.
- isEmpty(): kiểm tra tính rỗng của dqueue.
- isFull(): kiểm tra tính tràn của dqueue.

Dưới đây là một cách cài đặt cho lớp dqueue:

Biểu diễn dqueue:

```
struct node{
        int info; //Thông tin của node
        node *next; //Thành phần con trỏ trước
        node *prev; //Thành phần con trỏ sau
}*head, *tail;
```

Mô tả lớp dqueue:

```
class dqueue {
    public:
        int top1, top2; //top1 là con trỏ vào; top2 là con trỏ ra.
        void insert_front(); //Thêm node vào đầu trước của dqueue
        void insert_last(); //Thêm node vào đầu sau của dqueue
        void del_front(); //Loại node ở đầu trước của dqueue
        void del_last(); //Loại node vào đầu trước của dqueue
        void display_front(); //Hiển thin nội dung dqueue bắt đầu từ đầu trước
        void display_last(); //Hiển thin nội dung dqueue bắt đầu từ đầu tsau
        dqueue(){ //Constructor của dqueue
                top1 = 0;
                top2 = 0;
                head = NULL;
                tail = NULL;
        }
};
```

Thao tác Insert-front:

```
void dqueue::insert_front(){ struct node *temp;int value;
        if (top1 + top2 >= 50){
                cout<<“Tràn hàng đợi"<<endl;return;
        }
        if (top1 + top2 == 0){ //Nếu dqueue rỗng
                cout<<“Nhập giá trị node: “;cin>>value;
                 head = new (struct node); head->info = value;
                head->next = NULL; //Thiết lập liên kết trước cho head
                head->prev = NULL; //Thiết lập liên kết tsau cho head
                tail = head; // Node trước và node sau là một
                top1++; //top1 tăng lên 1 đơn vị
        }
        else {    cout<<“Nhập giá trị node: "; cin>>value;
                temp = new (struct node); temp->info = value;
                temp->next = head; //Thiết lập liên kết trước cho temp
                temp->prev = NULL; // Thiết lập liên kết sau cho temp
                head->prev = temp; Thiết lập liên kết sau cho head
                head = temp; //Node bắt đầu bây giờ chính là temp
                top1++;
        }
}
```

Thao tác Insert-Last:

```
void dqueue::insert_last(){ struct node *temp;    int value;
        if (top1 + top2 >= 50){
                cout<<“Tràn hàng đợi"<<endl; return;
        }
        if (top1 + top2 == 0){ //Nếu dqueue rỗng
                cout<<“Nhập giá trị node: “; cin>>value;
                head = new (struct node);head->info = value;
                head->next = NULL;//Thiết lập liên kết trước cho head
                head->prev = NULL; //Thiết lập liên kết sau cho head
                tail = head; //Node đầu và node cuối là một
                top1++;
        }
        else {  cout<<“Nhập giá trị node: “;cin>>value;
                temp = new (struct node);temp->info = value;
                temp->next = NULL; //Thiết lập liên kết trước cho tmp
                temp->prev = tail; //Thiết lập liên kết sau cho tmp
                tail->next = temp; //Node sau cũ được trỏ đến tmp
                tail = temp; //Node sau mới bây giờ là tmp
                top2++;
        }
}
```

Thao tác Del-Front:

```
void dqueue::del_front(){
        struct node *tmp =head;
        if (top1 + top2 <= 0){ //Nếu dqueue rỗng
                cout<<"Deque rỗng"<<endl;
                return;
        }
        head = head->next; //head được di chuyển lên một node
        head->prev = NULL; //Ngắt liên kết sau của head
        top1--; free (tmp);
}
```

Thao tác Del-Last:

```
void dqueue::del_last(){struct node *tmp =tail;
        if (top1 + top2 <= 0){
                cout<<"Deque Underflow"<<endl;
                return;
        }
        tail = tail->prev;
        tail->next = NULL;
        top2--; free(tmp);
}
```

Thao tác Display-Front:

```
void dqueue::display_front(){ struct node *temp;
        if (top1 + top2 <= 0){
                cout<<“Dqueue rỗng"<<endl;return;
        }
        temp = head; //temp trỏ đến head
        cout<<“Bắt đầu duyệt từ node đầu:"<<endl;
        while (temp != NULL){
                cout<<temp->info<<" “; temp = temp->next;
        }
 }
```

Thao tác Display-Last:

```
void dqueue::display_last(){ struct node *temp;
        if (top1 + top2 <= 0){
                cout<<“Hàng đợi rỗng"<<endl;return;
        }
        cout<<“Duyệt từ node cuối:"<<endl;
        temp = tail; //temp trỏ đến tail
        while (temp != NULL){
                cout<<temp->info<<" “; temp = temp->prev;
        }
}
```

Bài tập 8. Xây dựng các thao tác trên ngăn xếp dựa vào mảng.
Bài tập 9. Xây dựng các thao tác trên ngăn xếp dựa vào danh sách liên kết đơn.
Bài tập 10. Xây dựng các thao tác trên ngăn xếp dựa vào C++ STL.
Bài tập 11. Kiểm tra xâu ký tự có phải là biểu thức chính qui hay không?
Bài tập 12. Chuyển đổi biểu thức số học từ trung tố thành hậu tố?
Bài tập 13. Tính toán giá trị biểu thức hậu tố?
Bài tập 14. Xây dựng tập thao tác trên hàng đợi dựa vào mảng?
Bài tập 15. Xây dựng tập thao tác trên hàng đợi dựa vào danh sách liên kết?
Bài tập 16. Xây dựng tập thao tác trên hàng đợi ưu tiên dựa vào mảng?
Bài tập 17. Xây dựng tập thao tác trên hàng đợi ưu tiên dựa vào danh sách liên kết?
Bài tập 18. Xây dựng tập thao tác trên hàng đợi vòng dựa vào mảng?
Bài tập 19. Xây dựng tập thao tác trên hàng đợi vòng dựa vào danh sách liên kết?
Bài tập 20. Xây dựng tập thao tác trên dqueu?
Bài tập 21. Xây dựng tập thao tác trên hàng đợi dựa và C++STL?

Bài tập 3. Hoàn thành các thao tác trên danh sách liên kết đơn vòng

- Tạo lập node cho danh sách liên kết đơn vòng.
- Thêm node đầu tiên cho sách liên kết đơn vòng.
- Thêm node vào sau một node khác.
- Loại bỏ node trên danh sách liên kết đơn vòng.
- Tìm kiếm node trên danh sách liên kết đơn vòng.
- Hiển thị nội dung danh sách liên kết đơn vòng.
- Sửa đổi nội dung node cho danh sách liên kết đơn vòng.
- Sắp xếp nội dung các node trên danh sách liên kết đơn vòng.

Bài tập 4. Hoàn thành bài tập 3 bằng C++ STL .

RELAXATION

Cho cặp số S và T là các số nguyên tố có 4 chữ số (Ví dụ S = 1033, T = 8197 là các số nguyên tố có 4 chữ số). Hãy viết chương trình tìm cách dịch chuyển S thành T thỏa mãn đồng thời những điều kiện dưới đây:

- Mỗi phép dịch chuyển chỉ được phép thay đổi một chữ số của số ở bước trước đó (ví dụ nếu S=1033 thì phép dịch chuyển S thành 1733 là hợp lệ);
- Số nhận được cũng là một số nguyên tố có 4 chữ số (ví dụ nếu S=1033 thì phép dịch chuyển S thành 1833 là không hợp lệ, và S dịch chuyển thành 1733 là hợp lệ);
- Số các bước dịch chuyển là ít nhất.

Ví dụ với S = 1033, T = 8179 trong file nguyento.in dưới đây sẽ cho ta file ketqua.out như sau:

nguyento.in	*ketqua.out*
1033 8179	6
	8179 8779 3779 3739 3733 1733 1033

Kế tục thành công với khối lập phương thần bí, Rubik sáng tạo ra dạng phẳng của trò chơi này gọi là trò chơi các ô vuông thần bí. Đó là một bảng gồm 8 ô vuông bằng nhau như Hình 1. Trạng thái của bảng các màu được cho bởi dãy kí hiệu màu các ô được viết lần lượt theo chiều kim đồng hồ bắt đầu từ ô góc trên bên trái và kết thúc ở ô góc dưới bên trái. Ví dụ: trạng thái trong Hình 1 được cho bởi dãy các màu tương ứng với dãy số (1, 2, 3, 4, 5 , 6, 7, 8). Trạng thái này được gọi là trạng thái khởi đầu.

Biết rằng chỉ cần sử dụng 3 phép biến đổi cơ bản có tên là ‘A’, ‘B’, ‘C’ dưới đây bao giờ cũng chuyển được từ trạng thái khởi đầu về trạng thái bất kỳ:

‘A’ : đổi chỗ dòng trên xuống dòng dưới. Ví dụ sau phép biến đổi A, hình 1 sẽ trở thành hình 2:

‘B’ : thực hiện một phép hoán vị vòng quanh từ trái sang phải trên từng dòng. Ví dụ sau phép biển đổi B Hình 1 sẽ trở thành Hình 3.

‘C’ : quay theo chiều kim đồng hồ bốn ô ở giữa. Ví dụ sau phép biến đổi C Hình 1 trở thành Hình 4.

Hình 1

1	2	3	4
8	7	6	5

Hình 2

8	7	6	5
1	2	3	4

Hình 3

4	1	2	3
5	8	7	6

Hình 4

1	7	2	4
8	6	3	5

Ví dụ trạng thái: 2 6 8 4 5 7 3 1 sẽ thực hiện ít nhất là 7 bước như sau:

$$B \rightarrow C \rightarrow A \rightarrow B \rightarrow C \rightarrow C \rightarrow B$$