Chapter 2
Các cấu trúc dữ liệu cơ bản

Các cấu trúc dữ liệu cơ bản

- **Container** : là các cấu trúc dữ liệu cho phép lưu trữ và lấy dữ liệu độc lập với nội dung của dữ liệu.
- Các container phân biệt theo thứ tự lấy mà chúng hỗ trợ.
- Hai loại container quan trọng, thứ tự lấy của chúng phụ thuộc vào thứ tự chèn:
  - **Stack**: hỗ trợ lấy theo thứ tự vào sau ra trước – Last In First Out
  - **Queue**: hỗ trợ lấy theo thứ tự vào trước ra trước – First In First Out

2.5 Ngăn xếp - stack

2.5 Stack

- Định nghĩa
- Ứng dụng
- Cài đặt bằng mảng
- Cài đặt bằng danh sách móc nối
- Định giá biểu thức

Định nghĩa stack

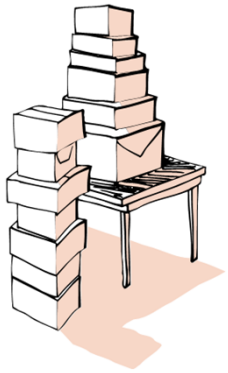
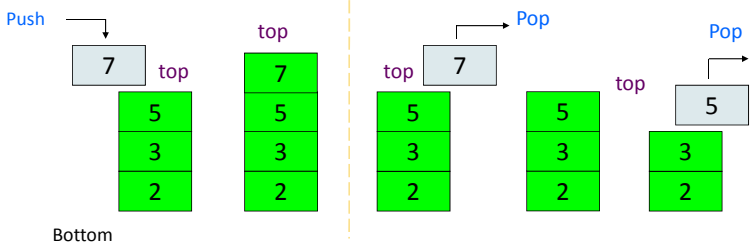


Figure 2.1. Stacks

- Stack: là cấu trúc hiệu quả để lưu trữ và lấy dữ liệu theo thứ tự vào sau, ra trước (LIFO)
- Top: đỉnh stack
- Bottom: đáy stack

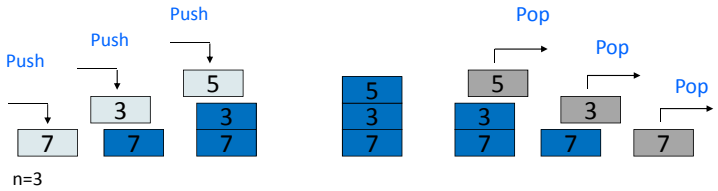
Định nghĩa stack

- Hai thao tác cơ bản:
  - Push: thao tác thêm một phần tử vào stack
  - Pop: thao tác lấy một phần tử ra khỏi stack



Ví dụ

- Ví dụ: đọc vào một danh sách gồm n số nguyên (n<100), và in ra các số đã đọc theo thứ tự ngược



Nhập vào: 7, 3, 5

In ra: 5, 3, 7

Ví dụ

* Thực hiện lần lượt các thao tác sau :
  1. Push(3);
  2. Push(5);
  3. Pop();
  4. Push(6);
  5. Pop();
  6. Pop();
* Kết quả thu được ? Vẽ hình stack minh họa

Cài đặt stack

Cài đặt stack

- Các phương thức cơ bản của stack
  - push() : thêm một phần tử mới vào stack
  - pop() : lấy một phần tử khỏi stack
  - empty() : kiểm tra xem stack có rỗng hay không
  - top() : trả về giá trị phần tử ở đỉnh stack
- Lưu trữ stack
  - Lưu trữ kế tiếp dùng mảng
  - Lưu trữ sử dụng danh sách móc nối

Lưu trữ bằng mảng

```
#include <stdio.h>
#define MAX 10 /* số lượng phần tử tối đa*/
#include <stdlib.h>
```

- Kiểm tra stack rỗng:

```
int empty(int stack[], int count)
```

  - Kiểm tra danh sách được lưu bởi mảng stack.
  - Trả về giá trị 1 nếu stack rỗng(không có phần tử nào)
  - Ngược lại thì trả về 0

Hàm empty()

```
int empty(int stack[], int count)
{
    if(count <0) return 1;
    else return 0;
}
```

Hàm push()

- Thêm một phần tử mới vào stack

int push(int stack[], int &count, int value)

- Thêm một phần tử mới vào **stack** lưu bởi mảng **stack[]**
- Nếu **stack** chưa đầy thì thêm **value** vào **stack** và tăng số lượng phần tử trong **stack** thêm một, trả về giá trị 0 (thêm thành công)
- Ngược lại thì trả về giá trị -1(thêm không thành công)

Hàm push()

```
int push(int stack[], int &count, int value)
{
   if(count < MAX-1 )
   {
      count = count + 1;
      stack[count] = value;
      return 0;
   }
  else
   {
      printf("stack da day!\n");
      return -1;
   }
}
```

Hàm pop()

- Lấy một phần tử khỏi stack

```
int pop(int stack[], int &count, int &value)
```

- Lấy một phần tử ra khỏi **stack** lưu bởi mảng **stack[]**
- Nếu **stack** khác rỗng thì lấy một phần tử ra khỏi stack, giá trị phần tử đó chứa trong **value,** và giảm số lượng phần tử trong **stack** đi một, trả về giá trị 0 (lấy thành công)
- Ngược lại thì trả về giá trị -1(không thành công)

Hàm pop()

```
int pop(int stack[], int &count, int &value)
{
   if(count >= 0 )
   {
      value = stack[count];
      count = count - 1;
     return 0;
   }
   else
   {
      printf("Stack rong, khong the lay duoc!\n");
      return -1;
   }
}
```

Hàm top()

- Trả về giá trị phần tử đang ở đầu stack

```
int top(int stack[], int count, int &value)
```

- Lấy giá trị phần tử ở đầu **stack** lưu bởi mảng **stack[]**
- Nếu **stack** khác rỗng thì lấy giá trị phần tử ở đầu stack gán cho value, trả về giá trị 0 (lấy thành công)
- Ngược lại thì trả về giá trị -1(không thành công)

Hàm top()

```
int top(int stack[], int count, int &value)
{
    if(empty(stack,count)==1) return -1;
    else
    {
        value = stack[count];
        return 0;
    }
}
```

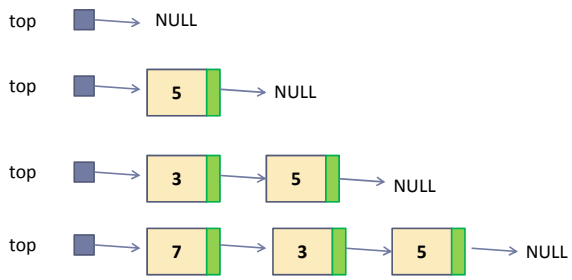
Ví dụ

```
int main()
{
  int stack[MAX];
  int count = -1; //stack rong
  int n,value;

  push(stack,count,10);
  push(stack,count,5);
  push(stack,count,7);

  while(empty(stack,count)==0)
  {
     pop(stack,count,value);
     printf("%d ",value);
  }
  return 0;
}
```

Lưu trữ bằng danh sách móc nối



- Thêm lần lượt 5, 3 ,7 vào stack rỗng

Lưu trữ bằng danh sách móc nối

- Nhận xét:
  - Thao tác push() : thêm phần tử vào đầu danh sách móc nối
  - Thao tác pop() : xóa một phần tử ở đầu danh sách móc nối
- Cấu trúc nút

```
typedef struct node
{
    int data;
    struct node *pNext;
} NODE;
```

Hàm empty()

```
int empty(NODE *pTop)
{
    if(pTop == NULL) return 1;
    else return 0;
}
```

Hàm push()

```
int push(NODE *&pTop, int value)
{
    NODE *ptr = (NODE*)malloc(sizeof(NODE));
    if(ptr!=NULL)
    {
        ptr->data=value;
        ptr->pNext=pTop;
        pTop=ptr;
        return 0;
    }
    else
    {
        printf("het bo nho !\n");
        return -1;
    }
}
```

Lưu trữ bằng danh sách móc nối

- Bài tập: Viết nốt các hàm

```
int pop (NODE *&pTop, int &value)
int top (NODE *pTop, int &value)
```

Ứng dụng của stack

Ứng dụng của stack

- Xử lý gọi hàm trong C/C++
- Stack trong máy tính
  - Sử dụng để tính giá trị biểu thức
  - Xử lý ngắt
- Trong các chương trình biên dịch (compiler)
- Trong trình duyệt web, trình soạn thảo văn bản

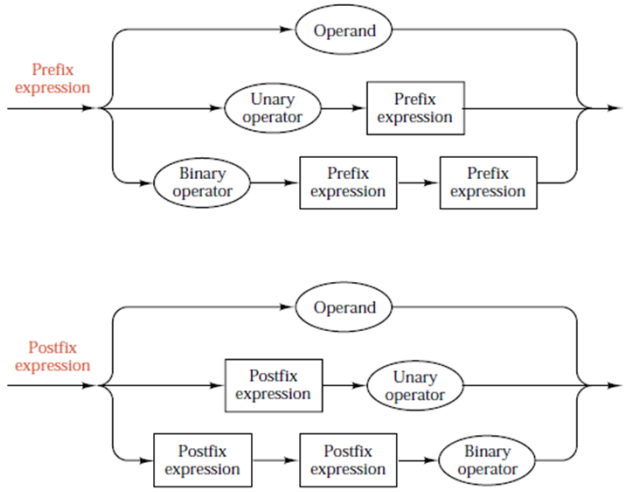
Ứng dụng của stack

- Định giá biểu thức:

  A = b + c * d / e – f

- **Biểu thức trung tố** : toán tử hai ngôi đứng giữa hai toán hạng, toán tử một ngôi đứng trước toán hạng.
- **Biểu thức hậu tố** : Toán tử đứng sau toán hạng
- Ví dụ:

Trung tố	$a * (b - c)/d$
Hậu tố	$abc - * d/$
Tiền tố	$/ * a - bcd$



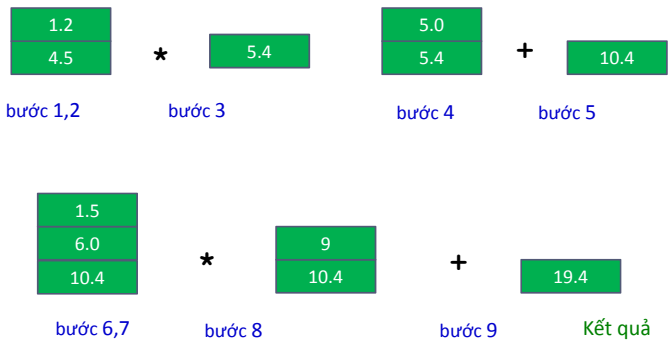
Định giá biểu thức

- **Tính giá trị biểu thức hậu tố** :
  Trung tố : `(4.5 * 1.2) + 5.0 + (6.0 * 1.5)`
  Hậu tố : `4.5 1.2 * 5.0 + 6.0 1.5 * +`
- **Cách tính**: Duyệt biểu thức hậu tố (chỉ gồm toán hạng 2 ngôi)
  - **Gặp toán hạng**: đẩy vào stack
  - **Gặp toán tử 1 ngôi**: lấy ra 1 toán hạng trong stack, áp dụng toán tử lên toán hạng và đẩy kết quả trở lại stack
  - **Gặp toán tử 2 ngôi**: lấy 2 toán hạng ở đỉnh stack theo thứ tự, áp dụng toán tử lên 2 toán hạng đó, kết quả lại đẩy vào stack.

  Kết thúc, đưa ra kết quả là giá trị ở đỉnh stack.

Định giá biểu thức hậu tố

`4.5 1.2 * 5.0 + 6.0 1.5 * +`



Định giá biểu thức hậu tố

$2\ 4\ 9\ \sqrt{}\ -\ +\ 4\ 2\ \hat{}\ *$

Bước	Mô tả	Trạng thái stack	Ghi chú
0	Stack rỗng	$\emptyset$	
1	Gặp 2 (toán hạng)	2	
2	4	$2, 4$	
3	9	$2, 4, 9$	
4	$\sqrt{}$ (toán tử 1 ngôi)	$2, 4, 3$	Thực hiện $\sqrt{9}$
5	$-$ (toán tử 2 ngôi)	$2, 1$	Thực hiện $4 - 3$
6	$+$ (toán tử 2 ngôi)	3	Thực hiện $2 + 1$
7	4	$3, 4$	
8	2	$3, 4, 2$	
9	^ (toán tử 2 ngôi)	$3, 16$	Thực hiện 4^2
10	$*$ (toán tử 2 ngôi)	48	Thực hiện $3 * 16$

Định giá biểu thức

- Xây dựng chương trình tính giá trị của một một biểu thức hậu tố được lưu trong một xâu ký tự.
- **Toán hạng**:
  các số nguyên không âm
- **Toán tử** : (xét các toán hạng 2 ngôi đơn giản)
  +, -, *, /, %, ^ (lũy thừa)

Định giá biểu thức

- **int** compute(**int** left, **int** right, **char** op);
  /* Thực hiện tính: "left op right" */

- **int** isOperator(**char** op);
  /* Kiểm tra op có phải là toán tử không?
  op phải là một trong số '+','-','*','/','%','^'
  nếu đúng thì trả về 1, sai thì trả về 0 */

Định giá biểu thức

```
int isOperator(char op)
{
if ((op == '+') || (op == '-') ||
   (op == '*' )|| (op == '%' )||
   (op == '/' )|| (op == '^' ))
     return 1;
   else
     return 0;
}
```

```
int compute(int left, int right, char op)
{
  int value;
  switch(op){
    case '+':value = left + right;
       break;
    case '-':value = left - right;
       break;
    case '*':value = left * right;
       break;
    case '%':value = left % right;
       break;
    case '/':value = left / right;
       break;
    case '^':value = pow(left, right);
       break;
  }
  return value;
}
```

```
int main()
{
    char eq[]="21*56*3/+7-";
int left, right, expValue;
char ch;
NODE *top=NULL;

for(int i=0;i<strlen(eq);i++)
{
        if(isOperator(eq[i])==0)
        {
             //printf("%d\n",eq[i]-'0');
             push(top,eq[i]-'0');
        }
```

```
    else
        {
            pop(top,right);
            pop(top,left);
            expValue=compute(left,right,eq[i]);
printf("%d %c %d = %d\n",left,eq[i],right,expValue);
            push(top,expValue);
        }
    }
    pop(top,expValue);
    printf("%d",expValue);
    getch();
    return 0;
}
```

Chuyển biểu thức dạng trung tố sang hậu tố

Chuyển biểu thức từ dạng trung tố sang hậu tố:

Ví dụ.	Trung tố	Hậu tố
	a*b*c*d*e*f	ab*c*d*e*f*
	1 + (-5) / (6 * (7+8))	1 5 - 6 7 8 + * / +
	(x/y – a*b) * ((b+x) – y)^{y}	x y / a b * – b x + y y ^ – *
	(x*y*z – x^2 / (y*2 – z^3) + 1/z) * (x – y)	xy*z*x2^y2*z3^ – / – 1z/+xy – *

Chuyển biểu thức sạng trung tố sang hậu tố

Toán tử	Mức ưu tiên
^(toán tử 2 ngôi)	6
!, ~ (toán tử 1 ngôi giai thừa và đảo dấu)	6
abs, sin, cos, tan, exp, ln, lg, round, trunc, sqr, sqrt, arctan	6
*, /, % (toán tử 2 ngôi)	5
+, - (toán tử 2 ngôi)	4
==, !=, <, >, ≤, ≥ (toán tử quan hệ)	3
Not (toán tử logic)	2
&&, || (toán tử logic)	1
= (toán tử gán)	0

Chuyển biểu thức sạng trung tố sang hậu tố

- **Duyệt lần lượt biểu thức trung tố từ trái qua phải:**

1. **Gặp toán hạng**: viết sang biểu thức kết quả (là biểu thức hậu tố cần tìm)
2. **Gặp toán tử** (có độ ưu tiên nhỏ hơn 6):
   - nếu **stack rỗng**, hoặc đỉnh stack là toán tử có độ ưu tiên nhỏ hơn, hoặc là **'('** : đẩy toán tử đang xét vào stack
   - ***Ngược lại***: lấy các toán tử ở đỉnh stack có độ ưu tiên lớn hơn hoặc bằng toán tử đang xét lần lượt đưa vào biểu thức kết quả và đẩy toán tử đang xét vào stack
3. **Gặp toán tử** có độ ưu tiên 6, hoặc **'('**: đẩy vào stack
4. Gặp **')'**: lấy tất cả các toán tử trong stack cho đến khi gặp '(' đầu tiên, đưa sang biểu thức kết quả theo đúng thứ tự (không đưa '(,)' vào biểu thức kết quả), và đẩy 1 ký hiệu '(' ra khỏi stack.
5. Nếu duyệt hết biểu thức trung tố, lấy nốt những toán tử trong stack đưa sang biểu thức kết quả (theo đúng thứ tự).

Chuyển biểu thức sạng trung tố sang hậu tố

- Ví dụ: Chuyển biểu thức sau sang dạng hậu tố

$$-3+\frac{4^{5^a}}{2}-7$$

- Biểu thức trên được viết lại là

$$-3+4\text{^}(5\text{^}a)/2-7$$

hoặc $-3+4\text{^}5\text{^}a/2-7$

- Chú ý: dấu $-$ trong -3 là toán tử 1 ngôi có độ ưu tiên 6

$-3+4\text{^}5\text{^}a/2-7$

Bước	Mô tả	Trạng thái stack	Kết quả
0	Stack rỗng	∅	∅
1	Gặp $-$ (toán tử 1 ngôi)	$-$	∅
2	Gặp 3 (toán hạng), đưa sang biểu thức kết quả)	$-$	3
3	Gặp $+$ (toán tử 2 ngôi), lấy $-$ khỏi stack và đẩy $+$ vào	$+$	$3\ -$
4	Gặp 4 (toán hạng)	$+$	$3-4$
5	Gặp ^ (toán tử 2 ngôi bậc 6)	$+,\text{^}$	$3-4$
6	Gặp 5	$+,\text{^}$	$3-4\ 5$
7	Gặp ^	$+,\text{^},\text{^}$	$3-4\ 5$
8	Gặp a (toán hạng)	$+,\text{^},\text{^}$	$3-4\ 5\ a$
9	Gặp / (toán tử 2 ngôi), lấy các toán tử có độ ưu tiên lớn hơn hoặc bằng ra khỏi stack, sau đó đẩy / vào	$+,/$	$3-4\ 5\ a\ \text{^}\ \text{^}$

$-3+4\text{^}5\text{^}a/2-7$

Bước	Mô tả	Trạng thái stack	Kết quả
10	Gặp 2	$+,/$	$3-4\ 5\ a\ \text{^}\ \text{^}\ 2$
11	Gặp $-$ (toán tử 2 ngôi), lấy / và $+$ khỏi stack, rồi đẩy $-$ vào	$-$	$3-4\ 5\ a\ \text{^}\ \text{^}\ 2\ /\ +$
12	Gặp 7 (toán hạng)	$-$	$3-4\ 5\ a\ \text{^}\ \text{^}2\ /\ +7$
	Đã duyệt song biểu thức, lấy nốt cac toán tử trong stack đưa sang biểu thức kết quả	∅	$3-4\ 5\ a\ \text{^}\ \text{^}2\ /\ +7-$

Biểu thức kết quả: $3-4\ 5\ a\ \text{^}\ \text{^}2\ /\ +7-$

Chuyển biểu thức sạng trung tố sang hậu tố

- Viết chương trình minh họa chuyển biểu thức dạng trung tố sang dạng hậu tố
- Biểu thức chỉ gồm:
  - Toán tử: $+,-,*,/,\%,\text{^}$
  - Toán hạng: 1 ký tự
    vd. a, b, c, 3, 4, 6,…
  - Dấu ngoặc: (, và)
- Ví dụ:

$$3+\frac{5}{3}+(1-a/b\text{^}2)\ \text{ hay }\ 3+5/3+(1-a/b\text{^}2)$$

Chuyển trung tố sang hậu tố

```
typedef struct node
{
    char data;
    struct node *pNext;
} NODE;
```

Chuyển trung tố sang hậu tố

```
int isOperator(char op)
{
if ((op == '+') || (op == '-') ||
   (op == '*' )|| (op == '%' )||
   (op == '/' )|| (op == '^' ))
     return 1;
   else
     return 0;
}
```

Chuyển trung tố sang hậu tố

```
int priority(char op)
{
    if((op == '+') || (op == '-')) return 4;
    if((op == '*' )|| (op == '%' )||
         (op == '/' )) return 5;
    if(op == '^')return 6;

    return 0;
}
```

```
int main()
{
    char eq[]="(3+5*(7-6*5)+4)^5^6";
    //char eq[]="1+3*4/5";
    char ch;
    char *retVal=(char*)calloc(sizeof(char),strlen(eq)+1);

    NODE *top=NULL;
    int j=0;
```

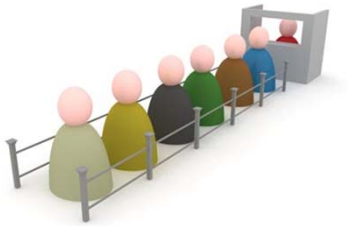
```
for(int i=0;i<strlen(eq); i++)
    {
        //printf("%c  ",eq[i]);
        if((eq[i]!='(')&&(eq[i]!=')'))
            if(isOperator(eq[i])==0)//toan hang
            {
                retVal[j]=eq[i];
                j++;
            }
```

```
else //toan tu
{
    if(top==NULL || top->data=='(' || priority(eq[i])==6 ||
       priority(eq[i])>priority(top->data)) push(top,eq[i]);
    else
        {
        ch = top->data;
        while(top !=NULL && priority(eq[i]) <= priority(ch)) {
            retVal[j]=pop(top);
            j++;
            if(top!=NULL) ch = top->data;
        }
        push(top,eq[i]);
        }
}
```

```
 else //dau ( hoac )
     {
         if(eq[i]=='(') push(top,eq[i]);
         else
         {
             ch=pop(top) ;
             while(ch!='(')
             {
                 retVal[j]=ch;
                 j++;
                 ch=pop(top) ;
             }
         }
     }
 }
```

```
while(top!=NULL) //lay not cac toan tu con lai
    {
         ch=pop(top) ;
         retVal[j]=ch;
         j++;
    }
    retVal[j]='\0';

    printf("\n\n%s",retVal);
    getch();
    return 0;
}
```



2.6 Queue – Hàng đợi

2.6 Queue

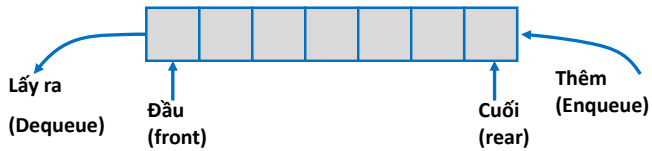
- Định nghĩa
- Ứng dụng
- Cài đặt queue bằng mảng
- Cài đặt queue bằng danh sách móc nối

Định nghĩa

- **Queue**: là cấu trúc hiệu quả để lưu trữ và lấy dữ liệu theo thứ tự vào trước ra trước (FIFO)

Định nghĩa

- Phần tử được lấy ra ở phần đầu queue (front), và được thêm vào ở cuối queue (rear)



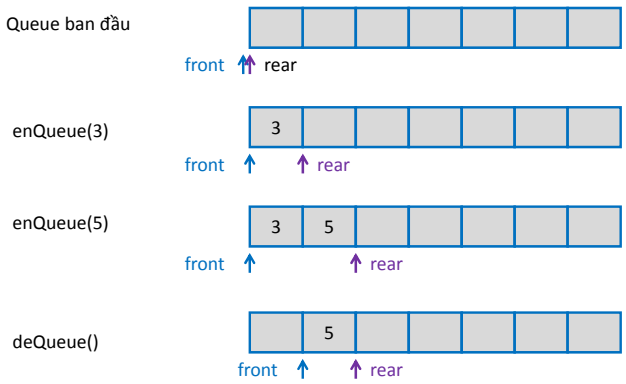
Ứng dụng

- Trong hệ điều hành:
  - Hàng đợi xử lý sự kiện
  - Quản lý tiến trình
  - Tổ chức bộ đệm bàn phím
  - Xử lý các lệnh
  - ….
- Khử đệ quy
- Lưu vết quá trình tìm kiếm, quay lui, vét cạn ….

Cài đặt queue

- Các phương thức cơ bản:
  - enQueue() : thêm một phần tử mới vào queue
  - deQueue() : lấy một phần tử khỏi queue
  - empty() : kiểm tra xem queue có rỗng hay không
  - front() : trả về giá trị phần tử ở đầu queue, mà không hủy nó
- Lưu trữ queue
  - Lưu trữ kế tiếp dùng mảng
  - Lưu trữ sử dụng danh sách móc nối

Lưu trữ bằng mảng



Hàm Empty()

- Kiểm tra queue rỗng:

```
int Empty(int queue[], int front, int rear)
```

- Kiểm tra hàng đợi được lưu bởi mảng queue .
- Trả về giá trị 1 nếu queue rỗng(không có phần tử nào)
- Ngược lại thì trả về 0

Hàm Empty()

```
int Empty(int queue[], int front, int
  rear)
{
    if(rear==front) return 1;
    else return 0;
}
```

Hàm enQueue()

- Thêm một phần tử mới vào queue

int enQueue(int queue[], int &rear, int value)

- Thêm một phần tử mới vào **queue** lưu bởi mảng **queue[]**
- Nếu **queue** chưa đầy thì thêm **value** vào queue, trả về giá trị 0 (thêm thành công)
- Ngược lại thì trả về giá trị -1(thêm không thành công)

```
int enQueue(int queue[], int &rear, int value)
{
   if(rear < MAX-1)
   {
      rear= rear +1;
      queue[rear] = value;
      return 0;
   }
   else
   {
      printf("queue da day !\n");
      return -1;
   }
}
```

Hàm deQueue()

- Lấy một phần tử khỏi queue

int deQueue(int queue[], int &front, int rear, int &value)

- Lấy một phần tử ra khỏi **queue** lưu bởi mảng **queue[]**
- Nếu queue khác rỗng thì lấy một phần tử ra khỏi queue, giá trị phần tử đó chứa trong **value**, trả về giá trị 0 (lấy thành công)
- Ngược lại thì trả về giá trị -1(không thành công)

Hàm deQueue()

```
int deQueue(int queue[], int &front, int rear, int
   &value)
{
  if(front == rear)
  {
    printf("Queue rong !\n");
    return -1;
  }

  front = front + 1;
  value = queue[front];
}
```

Hàm Front()

- Trả về giá trị phần tử đang ở đầu queue

int front(int queue[], int front, int rear, int &value)

- Lấy giá trị phần tử ở đầu **queue**
- Nếu **queue** khác rỗng thì lấy giá trị phần tử ở đầu **queue** gán cho value, trả về giá trị 0 (lấy thành công)
- Ngược lại thì trả về giá trị -1(không thành công)

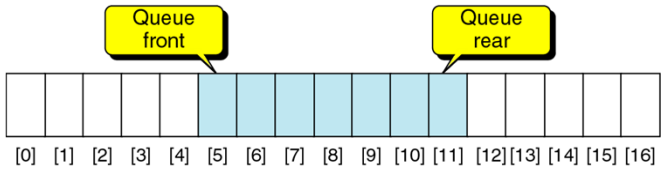
Hàm Front()

```
int Front(int queue[], int front, int rear, int &value)
{
  if(front == rear)
  {
   printf("Queue rong !\n");
   return -1;
  }
  value = queue[front+1];
}
```

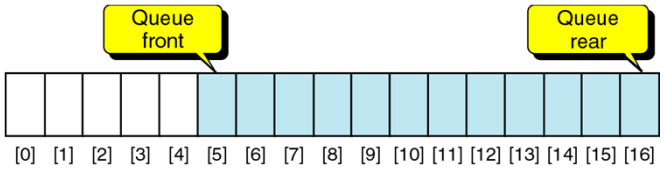
```
#define MAX 10
int main()
{
   int queue[MAX];
   int front,rear;
   int n,value;
   front=rear=(-1);
   enQueue(queue,rear,10);
   if(empty(queue,front,rear)==1)
        printf("Queue dang rong!\n");
   else {
        Front(queue,front,rear,value);
        printf("Dau queue : %d\n",value);
        }

   getch();
}
```

Lưu trữ bằng mảng

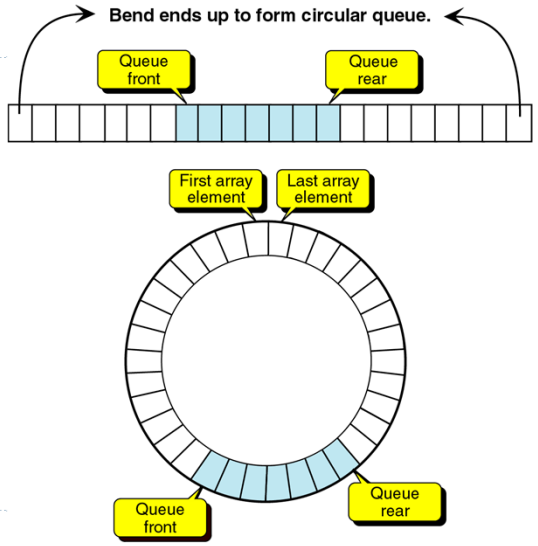


- Queue tăng hết mảng

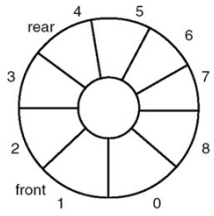


Cần sử dụng mảng rất lớn ?

Sử dụng queue dạng vòng



Queue dạng vòng



- Ban đầu:

```
front=rear=(0);
```

- Thay đổi

```
int Front(int queue[], int front, int rear, int &value)
{
    if(front == rear)
    {
      printf("Queue rong !\n");
      return -1;
    }
    value = queue[(front+1)%MAX];
}
```

Queue dạng vòng

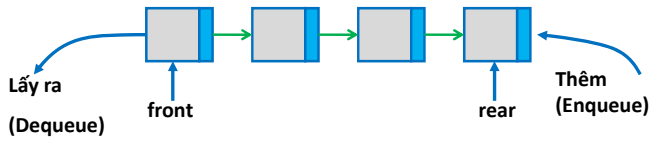
```
int enQueue(int queue[], int front, int &rear, int value)
{
  if(((rear +1)%MAX) == front)
  {
    printf("queue da day, khong the them
  %d!\n",value);
    return -1;
  }
  rear= (rear +1)%MAX;
  queue[rear] = value;
  return 0;
}
```

Queue dạng vòng

```
int deQueue(int queue[], int &front, int rear, int &value)
{
  if(front == rear)
  {
    printf("Queue rong !\n");
    return -1;
  }

  front = (front + 1)%MAX;
  value = queue[front];
}
```

Lưu trữ bằng danh sách móc nối

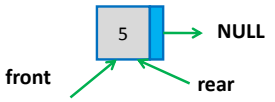


- **enQueue()** : thêm phần tử vào đầu danh sách móc nối
- **deQueue():** xóa phần tử ở cuối danh sách móc nối

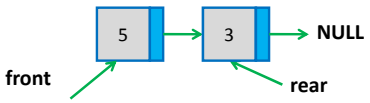
Khởi đầu

front = rear → NULL

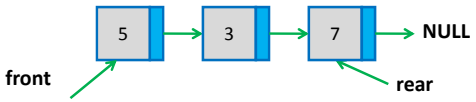
Thêm phần tử thứ nhất



Thêm phần tử thứ hai



Thêm phần tử thứ ba



Lưu trữ bằng danh sách móc nối

- Queue rỗng khi front (hoặc rear) bằng NULL

```
int empty(NODE * front, NODE *rear)
{
   if(front==NULL) return 1;
   else return 0;
}
```

```
int enQueue(NODE *&front, NODE *&rear, int value)
{
    NODE * ptr=(NODE*)malloc(sizeof(NODE));
    if(ptr==NULL)
    {
         printf("Khong con du bo nho !\n");
         return -1;
    }
    ptr->data=value;
    ptr->pNext=NULL;
    if(front==NULL)
    {
        front=ptr;
        rear=front;
    }
    else
    {
        rear->pNext=ptr;
        rear=ptr;
    }
    return 0;
}
```

Lưu trữ bằng danh sách móc nối

- Bài tập: Hoàn thiện nốt hai hàm còn lại

1. `int deQueue(NODE *&front, NODE *&rear, int &value)`
2. `int front(NODE * front, NODE *rear, int &value)`

Ứng dụng

Mô phỏng hàng đợi tại một phòng khám

Tại phòng khám, mỗi người đến được nhận một số theo thứ tự tăng dần, thứ tự phục vụ theo thứ tự các số đó. Giả sử mỗi người khách đến có thời gian phục vụ là t (t là một giá trị nguyên trong khoảng 1-9). Tại mỗi thời điểm xác suất có một khác mới là a (0<a<1).

Ứng dụng

- Cách tạo số ngẫu nhiên

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


srand(time(NULL));
x= rand()%10;  //x nhan gia tri [0,9]
```