Chương VIII

MẢNG

CBGD: ThS.Trần Anh Dũng

1

KHÁI NIỆM

Mảng là một biến cấu trúc trong đó có nhiều phần tử cùng kiểu, mỗi phần tử là một biến thành phần của mảng. Mỗi biến thành phần này là một biến bình thường và có cước số (subscript) để phân biệt giữa phần tử này và phần tử kia.

C cũng cho phép lập trình viên khai báo và làm việc trên mảng một chiều (singledimensional array) và mảng nhiều chiều (multidimensional array).

CBGD: ThS.Trần Anh Dũng

2

KHAI BÁO MẢNG

1. Mảng một chiều

kiểu tên_mảng [kích_thước];

- **kiểu** là kiểu dữ liệu của các phần tử của mảng,
- **tên_mảng** là một danh hiệu không chuẩn, được dùng để truy xuất mảng,
- **kích_thước** là một hằng số nguyên cụ thể, cho biết số phần tử trên chiều đang xét.

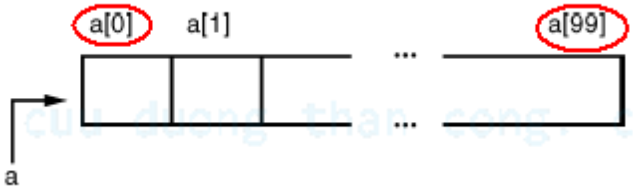
CBGD: ThS.Trần Anh Dũng

3

KHAI BÁO MẢNG

1. Mảng một chiều

Ví dụ int a[100];



CBGD: ThS.Trần Anh Dũng

4

KHAI BÁO MẢNG

1. Mảng một chiều

Ví dụ Viết chương trình nhập một dãy các số nguyên, tìm số lớn nhất trong dãy số đó

```
#include <stdio.h>
#include <conio.h>
main()
    {
        int i, n, max, vtmax;
        int a[100];
        clrscr();
        printf ("Chuong trinh thu mang \n");
        printf ("Moi ban nhap so phan tu cua mang: ");
        scanf ("%d", &n);
        printf ("Moi nhap cac phan tu cua mang:");
        for (i = 0; i < n; i++)
            scanf ("%d", &a[i]);
        max = a[0];
        vtmax = 0;
        for (i = 1; i < n; i++)
            if (max < a[i])
            {
                max = a[i];
                vtmax = i;
            }
        printf ("Phan tu co cuoc so %d co tri lon nhat la %d\n", vtmax, max);
        getch()
    }
```

CBGD: ThS.Trần Anh Dũng

5

KHAI BÁO MẢNG

1. Mảng nhiều chiều

kiểu tên_mảng [kích_thước_chiều1] [kích_thước_chiều2] [...];

- **kiểu** là kiểu dữ liệu của các phần tử của mảng, tương tự như mảng một chiều.
- **tên_mảng** là một danh hiệu không chuẩn, được dùng để truy xuất mảng.
- **kích_thước_chiều1, kích_thước_chiều2,...** là kích thước của các chiều trong mảng. Số kích thước này theo lý thuyết thì không hạn chế.

Array size too large

CBGD: ThS.Trần Anh Dũng

6

KHAI BÁO MẢNG

Ví dụ Khai báo mảng hai chiều a

```
int a[4][3];
```

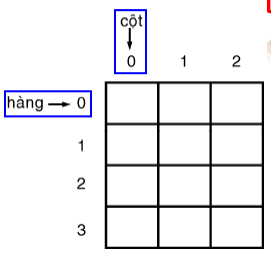
Như vậy mảng a có 4 × 3 phần tử int, các phần tử đó là

a[0][0] a[0][1] a[0][2]

a[1][0] a[1][1] a[1][2]

a[2][0] a[2][1] a[2][2]

a[3][0] a[3][1] a[3][2]



CBGD: ThS.Trần Anh Dũng

7

KHAI BÁO MẢNG

Ví dụ

Viết chương trình tạo và in ra màn hình ma trận có dạng sau:

1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1


```
#include <stdio.h>
#include <conio.h>
#define MAX 20
```

CBGD: ThS.Trần Anh Dũng

8

KHAI BÁO MẢNG

```
main()
    {
        int i, j;                   /* 2 biến điều khiển truy xuất ma trận */
        int a[MAX][MAX];  /* mảng hai chiều lưu ma trận          */
        int n;                      /* cấp của ma trận                     */
        clrscr();
        printf ("Chuong trinh thu mang \n");
        printf ("Moi ban nhap cap cua ma tran: ");
        scanf ("%d", &n);
        for (i = 0; i < n; i++)
```

CBGD: ThS.Trần Anh Dũng

9

KHAI BÁO MẢNG

```
        for (j = 0; j < n; j++)
            if (i == j)
                a[i][i] = 1;
            else
                a[i][j] = 0;
    printf ("Ma tran duoc tao la: \n");
    for (i = 0; i < n; i++)
        {
            for (j = 0; j < n; j++)
                printf ("%d", a[i][j]);
            printf ("\n");          /* xuống hàng về đầu dòng */
        }
    getch ()
}
```

vòng for in ra từng hàng của ma trận

CBGD: ThS.Trần Anh Dũng

10

KHAI BÁO MẢNG

C không có sự phân biệt giữa một biến chuỗi và một mảng các ký tự. Cả hai trường hợp đều được khai báo

```
char tên [chiều_dài];
```

kết thúc bằng ký tự NUL ('\0')

Khi lập trình, lập trình viên cần thêm kí tự NUL vào cuối chuỗi (nếu không sẽ là mảng kí tự)

C có hai hàm chuẩn để truy xuất chuỗi:

- Hàm gets() cho phép nhập một chuỗi có tên để trong đối số hàm này.

```
char s[20];
gets (s);
```

- *Hàm puts()* cho phép xuất một chuỗi có tên để trong đối số hàm này ra màn hình.

```
char s[20];
puts (s);
```

Cả hai hàm đều có prototype nằm trong file stdio.h

KHAI BÁO MẢNG

Ví dụ Chương trình truy xuất chuỗi dùng hàm chuẩn của C

```
#include <stdio.h>
#include <conio.h>
main()
    {
        char s[100];
        clrscr();
        printf ("Moi nhap mot chuoi: ");
        /* nhap chuoi dung ham chuan gets() cua C */
        gets (s);
        printf ("Chuoi da nhap la: ");
        /* xuat chuoi dung ham chuan puts() cua C */
        puts (s);
        getch();
    }
```

KHAI BÁO MẢNG

việc nhập chuỗi do lập trình viên tự nhập bằng vòng lặp

```
#include <stdio.h>
#include <conio.h>
main()
    {
        char s[100];
        int chieu_dai = 0;
        char c;
        s[0] = '\0';
        clrscr();
        printf ("Moi nhap mot chuoi: ");
        /* nhap chuoi bang vong lap */
        do
            {
                c = getch();
                if (c != '\r')
                    {
                        s [chieu_dai++] = c;
                        putch(c);
                    }
            }
        while (c != '\r');
        s [chieu_dai] = '\0';
        printf ("\n");
        printf ("chuoi da nhap la: ");
        /* xuat chuoi dung ham chuan gets() cua C */
        puts (s);
        getch();
    }
```


CBGD: ThS.Trần Anh Dũng

13

KHỞI ĐỘNG TRỊ CỦA MẢNG

Ví dụ int a[5] = {1, 3, 5, 7, 9};

→ a[0] = 1a[1] = 3a[2] = 5a[3] = 7a[4] = 9

Nếu số trị ít hơn số phần tử mảng thì các phần tử còn lại không được khởi động trị, có nghĩa các phần tử này có trị là 0.

Khi khởi động, kích thước mảng có thể không cần xác định, khi đó C sẽ tự động xem mảng có kích thước là số trị để khởi động cho các phần tử của mảng.

Ví dụ :

double a[] = {1.23, –5.67, 9.87, 1.34};

CBGD: ThS.Trần Anh Dũng

14

KHỞI ĐỘNG TRỊ CỦA MẢNG

Đối với chuỗi ký tự, ta có thể khởi động trị bằng một hằng chuỗi có chiều dài không vượt quá kích thước khai báo (kể cả ký tự kết thúc chuỗi NUL '\0'). Khi khởi động, chiều dài chuỗi lúc khai báo không xác định thì chiều dài của chuỗi chính là chiều dài chuỗi đã khởi động cho biến chuỗi.

Ví dụ : char s[30] = I go to shool \n;

char ch[] = Hello, World!;

Chú ý cần có sự phân biệt giữa chuỗi và mảng các ký tự

Ví dụ :

Chuỗi char s[] = Hello;

được lưu trong bộ nhớ

H	e	l	l	o	\0

mảng char ch[] = {'H', 'e', 'l', 'l', 'o'};

được lưu trong bộ nhớ

H	e	l	l	o

CBGD: ThS.Trần Anh Dũng

15

MẢNG LÀ ĐỐI SỐ CỦA HÀM – MẢNG LÀ BIẾN TOÀN CỤC

Khi khai báo đối số của hàm là mảng, thì kích thước của chiều đầu tiên của mảng không cần xác định cụ thể.

Ví dụ

Thiết kế hàm sắp xếp mảng một chiều theo thứ tự từ lớn tới nhỏ bằng phương pháp select sort.

không nêu cụ thể kích thước mảng trong khai báo đối số hàm

đối số n đóng vai trò kích thước mảng khi gọi hàm

```
void select_sort (int a[ ], int n)
    {
        int i, j; /* 2 biến lặp trong giải thuật select sort */
        int max, vtmax; /* 2 biến giữ số lớn nhất, và vị trí đó */
        int tam; /* biến tạm để hoán đổi trị */
        for (i = 0; i < n-1; i++)
```

CBGD: ThS.Trần Anh Dũng

16

MẢNG LÀ ĐỐI SỐ CỦA HÀM – MẢNG LÀ BIẾN TOÀN CỤC

```
if (max < a[j])                    {
    {                              tam = a[i];
        max = a[j];                a[i] = max;
        vtmax = j;                 a[vtmax] = tam;
    }                              }
if (vtmax ! = i)                   }
                                }
```

Vòng lặp for thứ nhất (biến chạy i):
-**Lần 1:** Tìm số lớn nhất, ghi đầu tiên bên trái
-**Lần 2:** Tìm số lớn nhì, ghi thứ nhì bên trái
- .....

17

MẢNG LÀ ĐỐI SỐ CỦA HÀM – MẢNG LÀ BIẾN TOÀN CỤC

Ví dụ

Thiết kế chương trình nhập ma trận hai chiều, tính tổng từng hàng của ma trận.

```
#include <stdio.h>
#include <conio.h>
#define MAX 20
void nhap_ma_tran (int a[][MAX], int n);
int tong_hang (int a[], int n);
```


18

MẢNG LÀ ĐỐI SỐ CỦA HÀM – MẢNG LÀ BIẾN TOÀN CỤC

```
main()
    {
        int n, i;
        int a[MAX][MAX];
        clrscr();
        printf ("chuong trinh thu ma tran \n");
        printf ("Moi ban nhap cap cua ma tran: ");
        scanf ("%d", &n);
        printf (Moi nhap cac phan tu cua ma tran:\n);
        nhap_ma_tran (a, n);
        printf ("Tong theo tung hang cua ma tran:\n");
        for (i = 0; i < n; i++)
            printf ("Hang thu %d co tong la %d \n", i+1, tong_hang(a[i], n));
        getch();
    }
```


19

MẢNG LÀ ĐỐI SỐ CỦA HÀM – MẢNG LÀ BIẾN TOÀN CỤC

```
void nhap_ma_tran (int a[][MAX], int n)
    {
        int i, j;
        for (i = 0; i < n; i++)
            for (j = 0; j < n; j++)
                scanf ("%d", &a[i][j]);
    }
```

```
int tong_hang (int a[], int n)
    {
        int i, s = 0;
        for (i = 0; i < n; i++)
            s += a[i];
        return s;
    }
```


20

MẢNG LÀ ĐỐI SỐ CỦA HÀM – MẢNG LÀ BIẾN TOÀN CỤC

Chương trình trên sẽ cho màn hình xuất liệu ví dụ là:

```
Chuong trinh thu ma tran
Moi ban nhap cap cua ma tran: 4
Moi nhap cac phan tu cua ma tran:
3   4   5   8
2   3   5   1
2   7   9   0
2   3   3   8
Tong theo tung hang cua ma tran:
Hang thu 1 co tong la 20
Hang thu 2 co tong la 11
Hang thu 3 co tong la 18
Hang thu 4 co tong la 16
```


MẢNG LÀ ĐỐI SỐ CỦA HÀM – MẢNG LÀ BIẾN TOÀN CỤC

Lưu ý:
Khi sử dụng mảng là biến toàn cục, nếu muốn sử dụng trong một module chương trình thì trong module này phải khai báo extern cho mảng biến toàn cục đó ở đầu module

Ví dụ:

Trong module MAIN.C

```
#include <...>
int a[20];
main()
    {
      ...
    }
```

Trong module FUNC.C

```
#include <...>
extern int a[];
void func1 ()
    {
      ...
    }
```

mảng a[] được sử dụng trong module FUNC.C, nhưng được khai báo trong module MAIN.C

CÁC ỨNG DỤNG

1. Sắp xếp mảng

1- Bubble sort

Giải thuật sort này dựa vào nguyên tắc: phần tử nhỏ hơn sẽ "nhẹ hơn" và vì vậy sẽ "nổi" lên trên. Như vậy, đây là phương pháp so sánh trực tiếp hai phần tử trong mảng với nhau, nếu phần tử nào nhỏ sẽ được đổi chỗ sang chỗ có chỉ số (cước số) thấp hơn (nếu việc sắp xếp theo thứ tự từ nhỏ tới lớn).

CÁC ỨNG DỤNG

Ví dụ

```
#include <stdio.h>
#include <conio.h>
#define MAX 20
void nhap_ma_tran (int a[], int n);
void in_ma_tran (int a[], int n);
void bubble_sort (int a[], int n);
main()
{
    int n, i;
    int a[MAX];
```

CÁC ỨNG DỤNG

```
    clrscr();
    printf ("Chuong trinh thu ma tran \n");
    printf ("Moi ban nhap kich thuoc day: ");
    scanf ("%d" &n);
    printf ("Moi nhap cac phan tu cua ma tran:\n");
    nhap_ma_tran (a, n);
    bubble_sort (a, n);
    printf ("Ma tran duoc sap xep la:\n");
    in_ma_tran (a, n);
    getch();
}
```

```
void nhap _ma_tran (int a[], int n)
    {
        int i;
        for (i = 0; i < n; i++)
            scanf ("%d", &a[i]);
    }
void in_ma_tran (int a[], int n)
    {
        int i;
        for (i =0; i <n; i++)
            printf ("%d", a[i]);
    }
```

CBGD: ThS.Trần Anh Dũng

25

CÁC ỨNG DỤNG

```
void bubble_sort (int a[], int n)
    {
        int i, j, temp; /* 2 biến lặp trong giải thuật bubble sort */
        for (i =0; i <n-1; i++)
            {
                printf ("Lan lap thu %d\n", i);
                for (j = n-1; j > i; j - - )
                    if (a[j] < a[j-1])
                        {
                            temp = a[j];
                            a[j] = a[j+1];
                            a[j+1] = temp;
                            in_ma_tran (a, n);
```

Dấu -

CBGD: ThS.Trần Anh Dũng

26

CÁC ỨNG DỤNG

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

Chương trình sẽ cho xuất liệu ví dụ:

```
Chuong trinh thu ma tran
Moi ban nhap kich thuoc day: 5
Moi nhap cac phan tu cua ma tran:
9 -5 7 0 1        <-  mảng nhập
Lan lap thu 0
9 -5 7 0 1}
-5 9 7 0 1}       ->  vòng lặp thứ 0 của i
Lan lap thu 1
-5 9 0 1 7}
-5 0 9 1 7}       ->  vòng lặp thứ 1 của i
Lan lap thu 2
-5 0 1 9 7}       ->  vòng lặp thứ 2 của i
Lan lap thu 3
-5 0 1 7 9}       ->  vòng lặp thứ 3 của i
Ma tran duoc sap xep la:
-5 0 1 7 9
```

Sai với giải thuật

CÁC ỨNG DỤNG

2- Quick sort

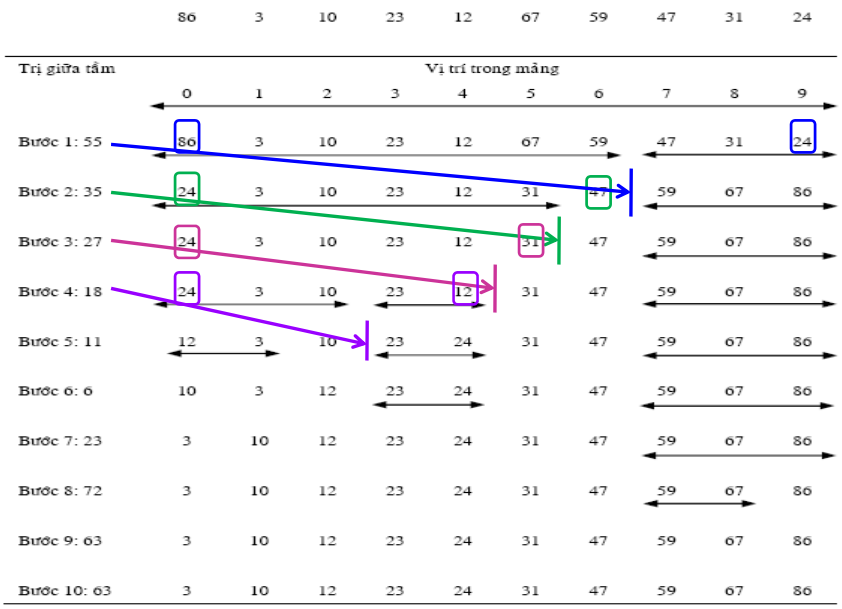
Đây là giải thuật sort được đánh giá là nhanh nhất trong các giải thuật sắp xếp.

Giải thuật này được C khuyến khích sử dụng.

Giải thuật **quick sort** bắt đầu bằng việc tìm một giá trị giữa tầm cho mảng.

Giá trị giữa tầm có thể được tính bằng trung bình cộng của hai phần tử đầu tiên và sau cùng trong phần mảng đang được sắp xếp. Mỗi khi chọn xong giá trị giữa tầm này, giải thuật sẽ chuyển tất cả các phần tử có giá trị nhỏ hơn giá trị giữa tầm sang phần có chỉ số thấp của mảng, và chuyển tất cả các phần tử có giá trị lớn hơn giá trị giữa tầm sang phần có chỉ số cao của mảng.

CÁC ỨNG DỤNG



	86	3	10	23	12	67	59	47	31	24
Trị giữa tầm	Vị trí trong mảng									
	0	1	2	3	4	5	6	7	8	9
Bước 1: 55	86	3	10	23	12	67	59	47	31	24
Bước 2: 35	24	3	10	23	12	31	47	59	67	86
Bước 3: 27	24	3	10	23	12	31	47	59	67	86
Bước 4: 18	24	3	10	23	12	31	47	59	67	86
Bước 5: 11	12	3	10	23	24	31	47	59	67	86
Bước 6: 6	10	3	12	23	24	31	47	59	67	86
Bước 7: 23	3	10	12	23	24	31	47	59	67	86
Bước 8: 72	3	10	12	23	24	31	47	59	67	86
Bước 9: 63	3	10	12	23	24	31	47	59	67	86
Bước 10: 63	3	10	12	23	24	31	47	59	67	86

CBGD: ThS.Trần Anh Dũng

29

CÁC ỨNG DỤNG

Do đó, chính vì điểm mạnh của giải thuật này mà C cung cấp trong thư viện hàm chuẩn qsort(), có prototype trong file stdlib.h như sau:

```
void qsort(void *base, size_t nelem, size_t width, int (*fcmp)(const void *, const void *));
```

với:
- **base** là tên mảng cần sắp xếp
- **nelem** là số phần tử của mảng cần sắp xếp
- **width** là kích thước của một phần tử của mảng
- ***fcmp** là hàm xác định quy luật sắp xếp mảng.

CBGD: ThS.Trần Anh Dũng

30

CÁC ỨNG DỤNG

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define MAX 20
void nhap_ma_tran (int a[], int n);
void in_ma_tran (int a[], int n);
int sort_function (const void *a, const void *b);
main()
        {
            int n;
            int a[MAX];
            clrscr();
            printf ("Chuong trinh sap xep ma tran bang QUICK SORT \n");
            printf ("Moi ban nhap kich thuoc cua may: ");
            scanf ("%d", &n);
            printf ("Moi nhap cac phan tu cua ma tran:\n");
            nhap_ma_tran (a, n);
            qsort (a, n, sizeof(a[0]), sort_function);
            printf ("Ma tran duoc sap xep la:\n");
            in_ma_tran (a, n);
            getch();
```


CBGD: ThS.Trần Anh Dũng

31

CÁC ỨNG DỤNG

```
void nhap_ma_tran (int a[], int n)
        {
            int i;
            for (i = 0; i < n; i++)
                scanf ("%d", &a[i]);
        }
void in_ma_tran (int a[], int n)
        {
            int i;
            for (i = 0; i < n; i++)
                printf ("%d", a[i]);
        }
```

```
int sort_function (const void *a, const void *b)
        {
            if (*(int*)a > *(int*)b)
                return 1;
            else if (*(int *)a == *(int *)b)
                return 0;
            else
                return -1;
        }
```


CBGD: ThS.Trần Anh Dũng

32

CÁC ỨNG DỤNG

Chương trình này cho xuất liệu ví dụ:

```
Chuong trinh sap xep ma tran bang QUICK SORT
Moi ban nhap kich thuoc day: 10
Moi nhap cac phan tu cua ma tran:
86 3 10 23 12 67 59 47 31 24
Ma tran duoc sap xep la:
3 10 12 23 24 31 47 59 67 86
```

Chương trình trên có sử dụng một số khái niệm về biến con trỏ, về địa chỉ của mảng,..., mà trong chương sau độc giả sẽ biết rõ hơn.

33

STACK

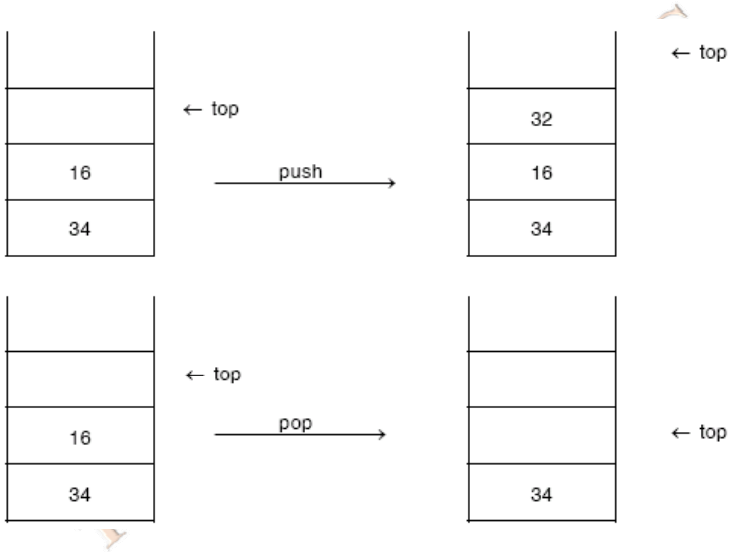
Stack (tạm dịch là ngăn xếp) là một kiểu cấu trúc dữ liệu do lập trình viên tự lập ra, khi cần, lập trình viên có thể thêm một phần tử vào stack, hoặc xóa một phần tử ra khỏi stack. Đặc điểm của cấu trúc dữ liệu này là dữ liệu được ghi vào hoặc lấy ra khỏi stack theo trật tự **vào trước ra sau** (last-in first-out), có nghĩa là phần tử được ghi vào stack trước sẽ được lấy ra sau và ngược lại.

Các thao tác cần có để làm việc trên stack:

- Khởi động stack, tương ứng với hàm init_stack() cần thiết kế.
- Các hàm để xem stack rỗng, đầy, hay xem trị trên đỉnh stack.
- Đẩy một phần tử vào stack, tương ứng hàm push() cần thiết kế.
- Lấy một phần tử từ đỉnh stack ra, tương ứng với hàm pop() cần thiết kế.

34

STACK



CBGD: ThS.Trần Anh Dũng

35

STACK

Để truy xuất stack, ta dùng một biến để quản lý đỉnh stack, biến đó là top. Khi biến này có trị:

- MAXTACK -1 (là một trị đã được khai báo sẵn, cho biết kích thước tối đa của stack) ➔ stack đang đầy, không thể đẩy thêm một phần nào vào stack nữa được.
- 0 thì stack đang rỗng ➔ không thể lấy trị từ stack ra được

CBGD: ThS.Trần Anh Dũng

36

STACK

```
#include <stdio.h>
#include <conio.h>
$define MAXSTACK 100
int top;
void push (int x, int stack[]);
void pop (int *x, int stack[]);
int empty (void);
int full (void);
void init_stack (void);
main()
{
    int s[MAXSTACK], i, so_tri, tri;
    clrscr();
    printf (Nhap bao nhieu tri vao stack: );
    scanf (%d, &so_tri);
    init_stack();
    printf("Stack da duoc khoi dong, de nghi nhap %d tri\n", so_tri);
    for (i =0; i < so_tri; i++)
        {
            scanf("%d", &tri);
            push(tri, s);
        }
```

CBGD: ThS.Trần Anh Dũng

37

STACK

```
        printf ("tri trong stack: ");
        for (i = 0; < so_tri; i++)
            {
                pop(&tri, s);
                printf("%d", tri);
            }
        getch();
    }
void push (int x, int stack[])
    {
        if (top == MAXSTACK 1)
            printf ("Stack day \n");
        else
            {
                top++;
                stack[top] = x;
            }
    }
void pop (int *x, int stack[])
    {
        if (top ==0)
            printf (Stack rong \n);
        else
            {
                *x = stack[top];
                top;
            }
    }
```

CBGD: ThS.Trần Anh Dũng

38

STACK

```
int empty (void)
        {
            return top == 0;
        }
int full (void)
        {
            return top == MAXSTACK;
        }
void init_stack (void)
        {
            top = 0;
        }
```

```
Nhap bao nhieu tri vao stack: 9
Stack da duoc khoi dong, de nghi nhap 9 tri
4 0 -4 7 97 3 1 34 123
Tri trong stack: 123 34 1 3 97 7 -4 0 4
```

CBGD: ThS.Trần Anh Dũng

39

QUEUE

Queue:

- ✓ Là một cấu trúc dữ liệu.
- ✓ Việc thêm dữ liệu vào được thực hiện ở một đầu, còn việc lấy một phần tử ra khỏi queue được thực hiện ở đầu kia theo trật tự vào đầu tiên ra đầu tiên (first-in first-out).
- ✓ Phần tử đầu tiên ra khỏi queue gọi là front, phần tử sau cùng ra khỏi queue gọi là rear.
- ✓ Queue có nhiều loại, tuy nhiên loại queue được sử dụng trong lập trình nhiều vẫn là queue vòng. Biến đếm count để biết được số phần tử đang có trong queue.

CBGD: ThS.Trần Anh Dũng

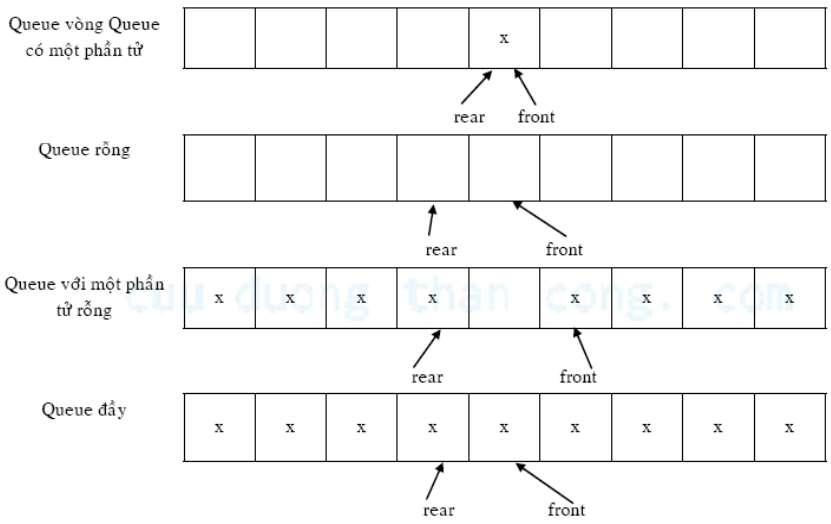
40

QUEUE

Tương tự như đối với stack, các thao tác cần có để làm việc trên queue:

- Khởi động queue, tương ứng với hàm init_queue() cần thiết kế.
- Các hàm để xem queue rỗng, đầy.
- Thêm một phần tử vào queue, tương ứng hàm addqueue() cần thiết kế.
- Lấy một phần tử ra khỏi queue, tương ứng với hàm deletequeue() cần thiết kế.

QUEUE



QUEUE

VD:

```
#include <stdio.h>
#include <conio.h>
#define MAXQUEUE 100
int count, front, rear;
void addqueue (int x, int queue[]);
void deletequeue (int *x, int queue[]);
int empty (void);
```


QUEUE

```
int full (void);
void init_queue (void);
main()
    {
        int q[MAXQUEUE], i, so_tri, tri;
        clrscr();
        printf ("Nhap bao nhieu tri vao queue: ");
        scanf ("%d", &so_tri);
        init_queue();
        printf("Queue da duoc khoi dong, de nghi nhap %d tri\n" }
        for (i = 0; i < so_tri; i++)
            {
                scanf("%d", &tri);
                addqueue(tri, q);
            }
        printf ("Tri trong queue duoc lay ra: ");
        for (i = 0; i < so_tri; i++)
            {
                delequeue (&tri, q);
                printf("%d", tri);
            }
        getch();
```

QUEUE

```
void addqueue (int x, int queue[])
{
    if (count == MAXQUEUE)
        printf (QUEUE day \n);
    else
    {
        count++;
        rear ++;
        if (rear == MAXQUEUE)
            rear = 0;
        queue[rear] = x;
    }
}
```

```
void deletequeue (int *x, int queue[])
{
    if (count == 0)
        printf ("Queue rong \n");
    else
    {
        count;
        *x = queue[front];
        if (front == MAXQUEUE -1)
            front = 0;
        else
            front++;
    }
}
```

CBGD: ThS.Trần Anh Dũng

45

QUEUE

```
int empty (void)
{
    return count == 0;
}
int full (void)
{
    return count == MAXQUEUE;
}
```

```
void init_queue (void)
{
    count = 0;
    front = 0;
    rear = -1;
}
```

CBGD: ThS.Trần Anh Dũng

46

BÀI TẬP

1. Nhập một ma trận n x n bất kỳ, sắp xếp lại ma trận sao cho các trị lớn nhất trên từng hàng, nằm trên đường chéo của ma trận.

2. Viết chương trình tạo và in ra màn hình ma trận có dạng sau:

```
a  b  c  d
b  c  d  a
c  d  a  b
d  a  b  c
← n →
```

n : tham số nhập

3. Nhập một chuỗi bất kỳ từ bàn phím, xóa tất cả các ký tự khoảng trắng thừa của chuỗi và in ra màn hình chuỗi mới.

4. Nhập một dãy số từ bàn phím. Viết hai hàm để in ra màn hình biểu đồ ngang và biểu đồ dọc của các dấu * tương ứng với các số nhập trong dãy số.

Ví dụ: Nhập 2 4 3;

Xuất:

```
 * *          * * *
* * * *       * * *
* * *          * *
                 *
```


QUEUE

5. Viết chương trình tạo và in ra màn hình tam giác PASCAL cấp n, với n nhập từ bàn phím.

6. Viết chương trình tạo ma trận nghịch đảo n x n.

7. Viết chương trình giải hệ phương trình tuyến tính bằng phương pháp Gauss.

8. Nhập một ma trận vuông bất kỳ, tính tổng các hàng, các cột, các đường chéo.

9. Nhập một ma trận bất kỳ. In ra màn hình các trị và vị trí của các số nguyên tố có trong mảng đó.

10. Viết các hàm đổi từ số sang chuỗi, và từ chuỗi sang số.