CHƯƠNG 11
MẢNG

CHƯƠNG 11
MẢNG

CHƯƠNG 11
MẢNG

11.1 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. Như vậy, để truy xuất một phần tử của mảng, ta cần biết được cước số của nó.

Trong bộ nhớ, các phần tử của mảng được cấp phát ô nhớ có địa chỉ liên tiếp nhau.

CHƯƠNG 11
MẢNG

11.1 KHÁI NIỆM

C cũng cho phép lập trình viên khai báo và làm việc trên mảng một chiều (singledimensional array) và mảng nhiều chiều (multidimensional array). Số phần tử trên một chiều được gọi là kích thước của chiều đó.

CHƯƠNG 11
MẢNG

11.2 KHAI BÁO MẢNG

1- Mảng một chiều

Cú pháp khai báo mảng một chiều như sau:

kiểu tên_mảng [kích_thước];

Với kích_thước là một hằng số nguyên cụ thể, cho biết số phần tử trong chiều đang xét.

Trong C, cước số các phần tử của mảng luôn đi từ 0 trở đi, nên mảng một chiều có **n** phần tử thì cước số các phần tử của mảng là 0,..., n-1.

CHƯƠNG 11
MẢNG

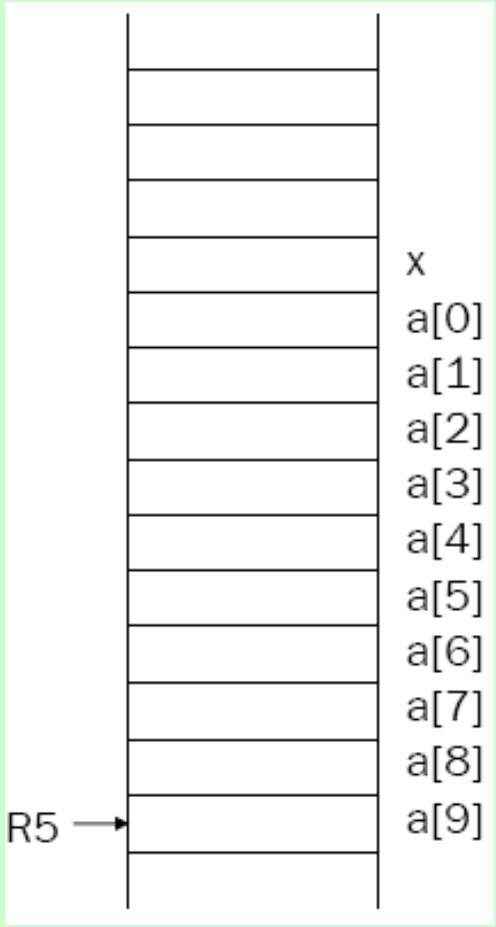
11.2 KHAI BÁO MẢNG

1- Mảng một chiều

Ví dụ: Cho khai báo sau:

```
int  a[10], x;
```

Như vậy mảng a có 10 phần tử **int**, các phần tử đó là a[0], a[1], …, a[9]. Các phần tử này được cấp phát vị trí trong bộ nhớ như hình 12.1 sau.



CHƯƠNG 11
MẢNG

11.2 KHAI BÁO MẢNG

1- Mảng một chiều

Lệnh

```
a[5] = a[3] + 1;
```

có mã LC-3 như sau:

```
ADD     R0, R5, #-9    ; R0 = &a[0]: địa chỉ của a[0]
LDR     R1, R0, #3     ; R1 = a[3]
ADD     R1, R1, #1     ; tăng 1
STR     R1, R0, #5     ; a[5] = R1, tức a[5] = a[3] + 1.
```

CHƯƠNG 11
MẢNG

11.2 KHAI BÁO MẢNG

1- Mảng một chiều

Lệnh

a[5] = 7;

có mã LC-3 như sau:

```
AND     R0, R0, #0
ADD     R0, R0, #7    ; R0 = 7
ADD     R1, R5, #-9   ; R1 = &a[0]: địa chỉ của phần tử a[0]
STR     R0, R1, #5    ; a[5] = R0
```

CHƯƠNG 11
MẢNG

11.2 KHAI BÁO MẢNG

1- Mảng một chiều

Còn lệnh

a[x+1] = a[x] + 2;

với *x* là biến đang chứa trị là chỉ số nào đó cần làm việc, có mã LC-3 như sau:

```
LDR     R0, R5, #-10  ; R0 = x
ADD     R1, R5, #-9   ; R1 = &a[0]
ADD     R1, R0, R1    ; R1 = &a[x]
LDR     R2, R1, #0    ; R2 = a[x]
ADD     R2, R2, #2    ; cộng thêm 2
```

CHƯƠNG 11
MẢNG

11.2 KHAI BÁO MẢNG

1- Mảng một chiều

```
LDR     R0, R5, #-10  ; R0 = x
ADD     R0, R0, #1    ; R0 = x+1
ADD     R1, R5, #-9   ; R1 = &a[0]
ADD     R1, R0, R1    ; R1 = &a[x+1]
STR     R2, R1, #0    ; a[x+1] = R2
```

CHƯƠNG 11
MẢNG

11.2 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();
```

CHƯƠNG 11
MẢNG

11.2 KHAI BÁO MẢNG

```
        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 %d co tri lon nhat la %d\n", vtmax, max);
        getch()
}
```

CHƯƠNG 11
MẢNG

11.2 KHAI BÁO MẢNG

2- Mảng nhiều chiều

Cú pháp khai báo mảng nhiều chiều như sau:

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

Khi dịch C báo lỗi
Array size too large ?

CHƯƠNG 11
MẢNG

11.2 KHAI BÁO MẢNG

2- Mảng nhiều chiều

Ví dụ: Khai báo mảng hai chiều a

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

Như vậy mảng a có 4x3 phần tử int, các phần tử đó là

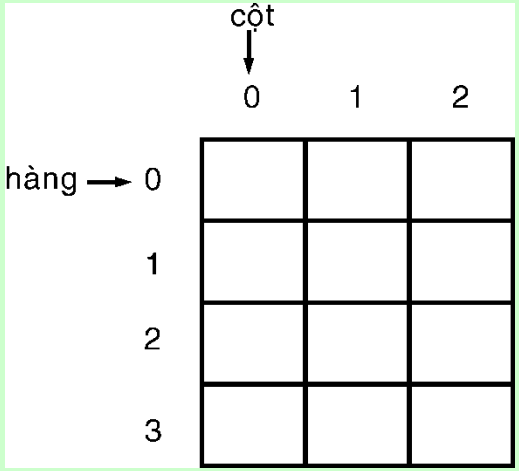
```
a[0][0] a[0][1] a[0][2]
a[1][0] a[1][1] a[1][2]
a[2][0] a[2][1] a[2][2]
a[3][0] a[3][1] a[3][2]
```

CHƯƠNG 11
MẢNG

11.2 KHAI BÁO MẢNG

2- Mảng nhiều chiều

Các phần tử này được sắp trong bộ nhớ theo thứ tự 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],....



CHƯƠNG 11
MẢNG

11.2 KHAI BÁO MẢNG

2- Mảng nhiều chiều

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

CHƯƠNG 11
MẢNG

11.2 KHAI BÁO MẢNG

2- Mảng nhiều chiều

```
#include <stdio.h>
#include <conio.h>
#define MAX 20
main()
{       int i, j;
        int a[MAX][MAX];
        int n;
        clrscr();
        printf ("Chuong trinh thu mang \n");
        printf ("Moi ban nhap cap cua ma tran: ");
```

CHƯƠNG 11
MẢNG

11.2 KHAI BÁO MẢNG

2- Mảng nhiều chiều

```
        scanf ("%d", &n);
        for (i = 0; i < n; i++)
                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");}
                        getch ()                }
}
```

CHƯƠNG 11
MẢNG

11.2 KHAI BÁO MẢNG

2- Mảng nhiều chiều

Ví dụ : Cho các khai báo sau

```
#define MAX 4
int a[MAX][MAX];
int n = 3;/* cấp thực sự cần làm việc của ma trận  */
int i, j;     /* biến là chỉ số mảng  */
/* Nhập trị cho mảng*/
for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)   scanf (“%d”, &a[i][j]);
```

CHƯƠNG 11
MẢNG

11.2 KHAI BÁO MẢNG

2- Mảng nhiều chiều

Giả sử trị nhập vào là:

0	1	2
3	4	5
6	7	8
9	10	11

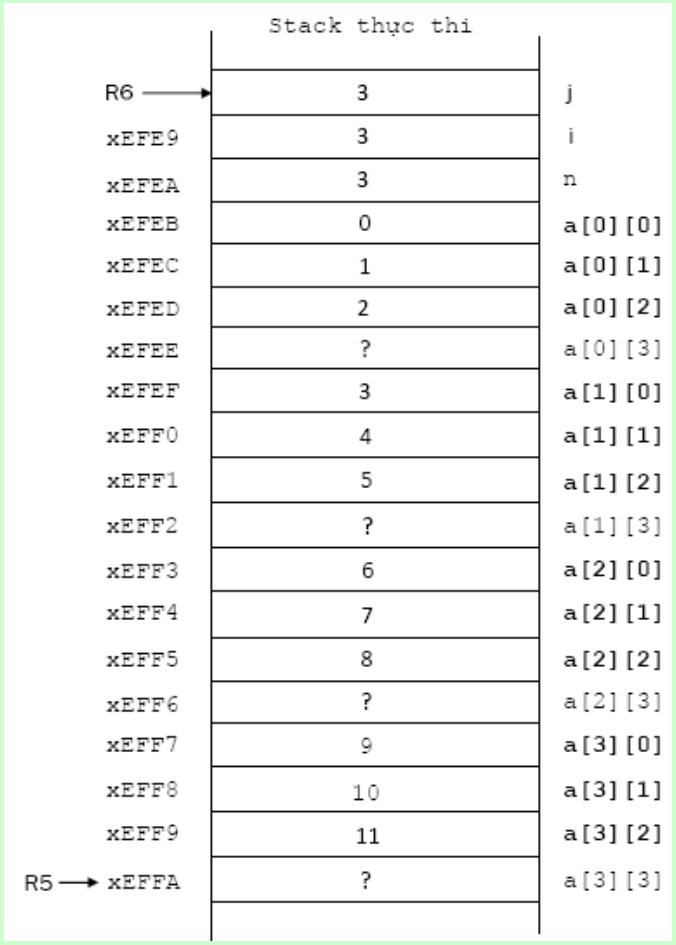
Mảng a[3][3], là một phần của ma trận a[MAX][MAX]

CHƯƠNG 11
MẢNG

11.2 KHAI BÁO MẢNG

2- Mảng nhiều chiều

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



CHƯƠNG 11
MẢNG

11.2 KHAI BÁO MẢNG

2- Mảng nhiều chiều

Ví dụ : Có khai báo

```
int a[10];
```

mà ta lại thực hiện lệnh

```
for (i = 0; i<= 10; i++) a[i] = i;
```

thì trong thực tế không có phần tử a[10], nhưng việc gán cũng được thực hiện, và ô nhớ kế tiếp phần tử a[9] được gán trị.

CHƯƠNG 11
MẢNG

11.2 KHAI BÁO MẢNG

2- Mảng nhiều chiều

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];

Điểm khác biệt?

CHƯƠNG 11
MẢNG

11.2 KHAI BÁO MẢNG

2- Mảng nhiều chiều

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

Ví dụ :

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

CHƯƠNG 11
MẢNG

11.2 KHAI BÁO MẢNG

2- Mảng nhiều chiều

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

Ví dụ :

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

Cả hai gets() và puts() đều có prototype nằm trong file *stdio.h*.

CHƯƠNG 11
MẢNG

11.2 KHAI BÁO MẢNG

2- Mảng nhiều chiều

Ví dụ :

Chương trình truy xuất chuỗi dùng hàm chuẩn của C.

CHƯƠNG 11
MẢNG

11.2 KHAI BÁO MẢNG

2- Mảng nhiều chiều

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

CHƯƠNG 11
MẢNG

11.3 KHỞI ĐỘNG TRỊ CỦA MẢNG

Khi khai báo mảng là biến toàn cục hoặc tĩnh thì mảng có thể được khởi động trị bằng các giá trị hằng.

Ví dụ :

```
int a[5] = {1, 3, 5, 7, 9};
int b[10] = {1, 2, 3, 4, 5};
```

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.

CHƯƠNG 11
MẢNG

11.3 KHỞI ĐỘNG TRỊ CỦA MẢNG

Ví dụ:

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

char s[30] = “I go to school \n”;
char ch[] = “Hello, World!”;
```

CHƯƠNG 11
MẢNG

11.3 KHỞI ĐỘNG TRỊ CỦA MẢNG

Ví dụ:

Cho khai báo mảng và khởi động trị như sau:

```
int a[][3] = {
                { 11, 12, 13},
                { 21, 22, 23},
                { 31, 32, 33}
            };
```

Với khai báo này, mảng a sẽ có 9 phần tử trong 3 hàng

CHƯƠNG 11
MẢNG

11.3 KHỞI ĐỘNG TRỊ CỦA MẢNG

Ví dụ :

Chuỗi char s[] = “Hello”;

H	e	l	l	o	\0

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

H	e	l	l	o

CHƯƠNG 11
MẢNG

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

Khi khai báo đối số của hàm là mảng, kích thước của chiều đầu tiên của mảng không cần xác định cụ thể. Tuy nhiên từ chiều thứ hai trở đi, kích thước mảng phải xác định.

Tên mảng chính là địa chỉ của mảng, nên việc truyền tên mảng cho hàm chính là truyền địa chỉ thực của mảng nên mọi thay đổi trên mảng trong hàm cũng chính là thay đổi trên mảng thật (truyền theo kiểu tham số biến).

CHƯƠNG 11
MẢNG

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

Ví dụ 12.14 (SGT)

```
void select_sort (int a[ ], int n)
{
....
}
```

Ví dụ 12.15, 12.16 (SGT)

CHƯƠNG 11
MẢNG

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

Ví dụ 12.17: Xét chương trình tính trung bình của các số như sau:

```
#include <stdio.h>
#define MAX 10
int Average (int values[]);
main()
{    int index;
     int mean;
     int a[MAX];
```

CHƯƠNG 11
MẢNG

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

```
printf (“Mời nhập %d số nguyên: ”, MAX);
      // Nhập trị cho mảng
     for (index = 0; index < MAX; index++)
          scanf (“%d”, &a[index]);
     mean = Average (a);
     printf (“Trung bình của các số này là %d.\n”, mean);
}
```

CHƯƠNG 11
MẢNG

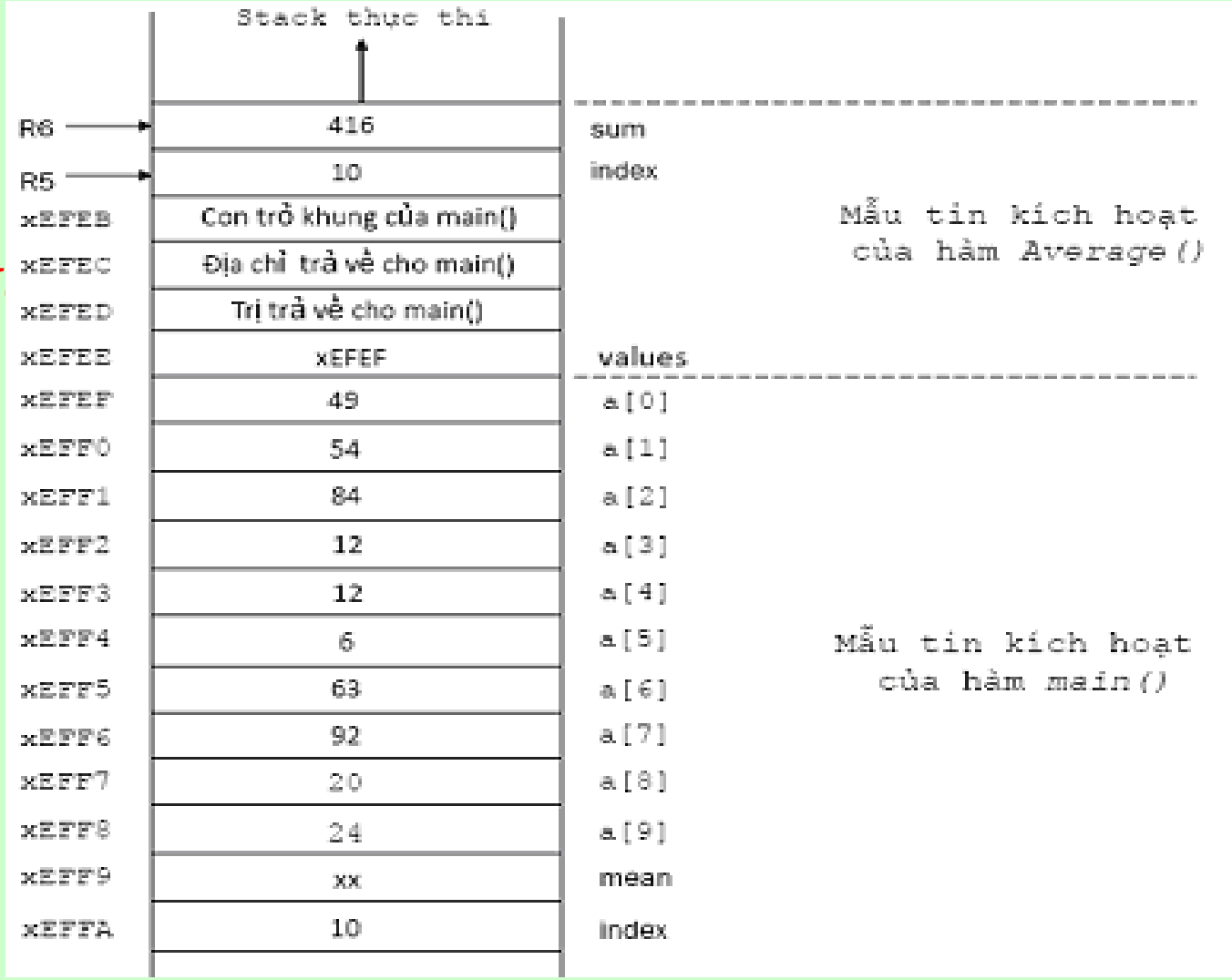
11.4 MẢNG LÀ ĐỐI SỐ CỦA HÀM MẢNG LÀ BIẾN TOÀN CỤC

```
int Average (int values[])
{
    int index;
    int sum = 0;

    for (index = 0; index < MAX; index++)
        sum += values[index];
    return (sum/MAX);
}
```

CHƯƠNG 11
MẢNG

11.4 MẢNG LÀ ĐỐI SỐ CỦA HÀM MẢNG LÀ BIẾN TOÀN CỤ



CHƯƠNG 11
MẢNG

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

```
#include <stdio.h>
#define MAX 10
int Average (int values[], int number);
main()
{    int index;
     int mean;
     int n;
     int a[MAX];
```

CHƯƠNG 11
MẢNG

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

```
// Nhập số số nguyên cần làm việc
    do
    {    printf (“Bạn muốn làm việc với bao nhiêu số nguyên? (0
< n <= 10): ”);
        scanf (“%d”, &n);
        if (n <= 0 || n > 10)
            printf (“Sai trị. Nhập lại.\n”);
    } while (n <= 0 || n > 10);
    printf (“Mời nhập %d số nguyên: ”, n);
```

CHƯƠNG 11
MẢNG

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

```
// Nhập trị cho mảng
    for (index = 0; index < n; index++)
        scanf ("%d", &a[index]);
    mean = Average (a, n);
    printf ("Trung bình của các số này là %d.\n", mean);
}
```

CHƯƠNG 11
MẢNG

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

```
int Average (int values[], int number)
{
    int index;
    int sum = 0;
    for (index = 0; index < number; index++)
        sum + = values[index];
    return (sum/number);
}
```

CHƯƠNG 11
MẢNG

11.5 CÁC ỨNG DỤNG

11.5.1 Sắp xếp mảng

Trong phần trên, để sắp xếp một mảng, giải thuật được nêu ra là giải thuật **select sort**, trong mục này ta sẽ xét thêm hai giải thuật nữa là giải thuật **bubble sort** và **quick sort**.

CHƯƠNG 11
MẢNG

11.5 CÁC ỨNG DỤNG

11.5.1 Sắp xếp mảng

1- Bubble sort (ví dụ 5.17)

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).

CHƯƠNG 11
MẢNG

11.5 CÁC ỨNG DỤNG

11.5.1 Sắp xếp mảng

1- Bubble sort (ví dụ 5.17)

```
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
Lan lap thu 0
9 -5 0 7 1
-5 9 0 7 1
Lan lap thu 1
-5 9 0 1 7
-5 0 9 1 7
```

CHƯƠNG 11
MẢNG

11.5 CÁC ỨNG DỤNG

11.5.1 Sắp xếp mảng

1- Bubble sort (ví dụ 5.17)

Lan lap thu 2
-5 0 1 9 7
Lan lap thu 3
-5 0 1 7 9
Ma tran duoc sap xep la:
-5 0 1 7 9

CHƯƠNG 11
MẢNG

11.5 CÁC ỨNG DỤNG

11.5.1 Sắp xếp mảng

2- Quick sort

Đây là giải thuật sort được đánh giá là nhanh nhất trong các giải thuật sắp xếp. Nguyên tắc của giải thuật này như sau:

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

CHƯƠNG 11
MẢNG

11.5 CÁC ỨNG DỤNG

11.5.1 Sắp xếp mảng

2- Quick sort

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. 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.

CHƯƠNG 11
MẢNG

	86	3	10	23	12	67	59	47	31	24
Trị giữa tâm	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

CHƯƠNG 11
MẢNG

11.5 CÁC ỨNG DỤNG

11.5.1 Sắp xếp mảng

3- Select sort

Giải thuật sẽ tìm giá trị lớn nhất đưa về vị trí có cước số thấp nhất, sau đó tìm giá trị lớn thứ nhì đưa về vị trí có cước số thấp nhì, quá trình diễn ra tương tự cho đến hết

Ví dụ 8.14 (SGT)

CHƯƠNG 11
MẢNG

11.5 CÁC ỨNG DỤNG

11.5.2 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).

CHƯƠNG 11
MẢNG

11.5 CÁC ỨNG DỤNG

11.5.2 Stack

Các thao tác cần có để làm việc trên stack:

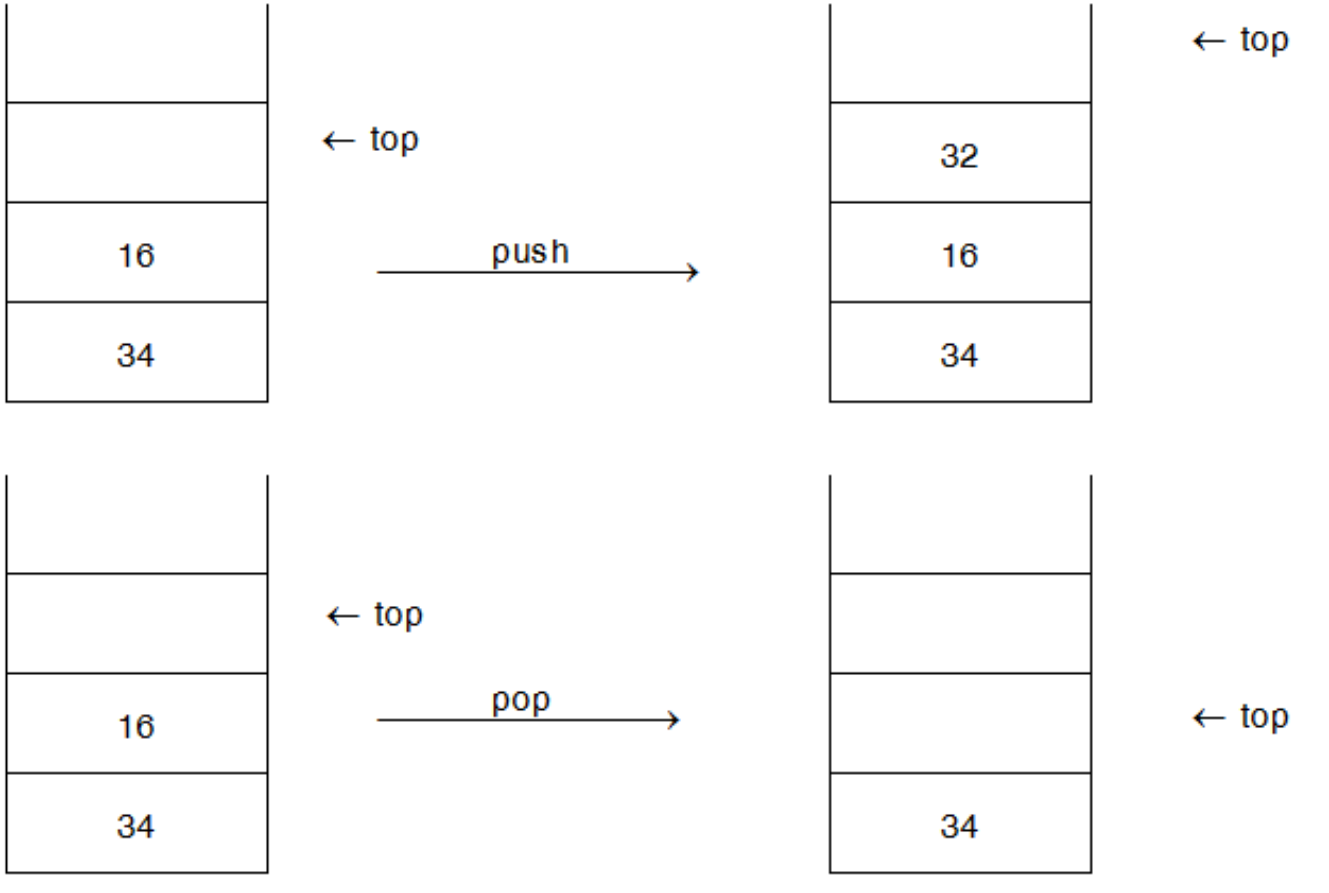
- Khởi động stack, tương ứng với hàm init_stack() cần thiết kế.
- Các hàm để xem stack rỗng, đầy, hay xem trị trên đỉnh stack.
- Đẩy một phần tử vào stack, tương ứng hàm push() cần thiết kế.
- Lấy một phần tử từ đỉnh stack ra, tương ứng với hàm pop() cần thiết kế.

CHƯƠNG 11
MẢNG

11.5 CÁC ỨNG DUNG

11.5.2 Stack

Ví dụ:



CHƯƠNG 11
MẢNG

11.5 CÁC ỨNG DỤNG

11.5.2 Stack

Chương trình ứng dụng stack (SGT)

CHƯƠNG 11
MẢNG

11.5 CÁC ỨNG DỤNG

11.5.3 Queue

Queue là một cấu trúc dữ liệu, trong đó 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. Dữ liệu vào ra queue theo trật tự vào đầu tiên ra **đầu tiên** (first-in first-out).

CHƯƠNG 11
MẢNG

11.5 CÁC ỨNG DỤNG

11.5.3 Queue

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

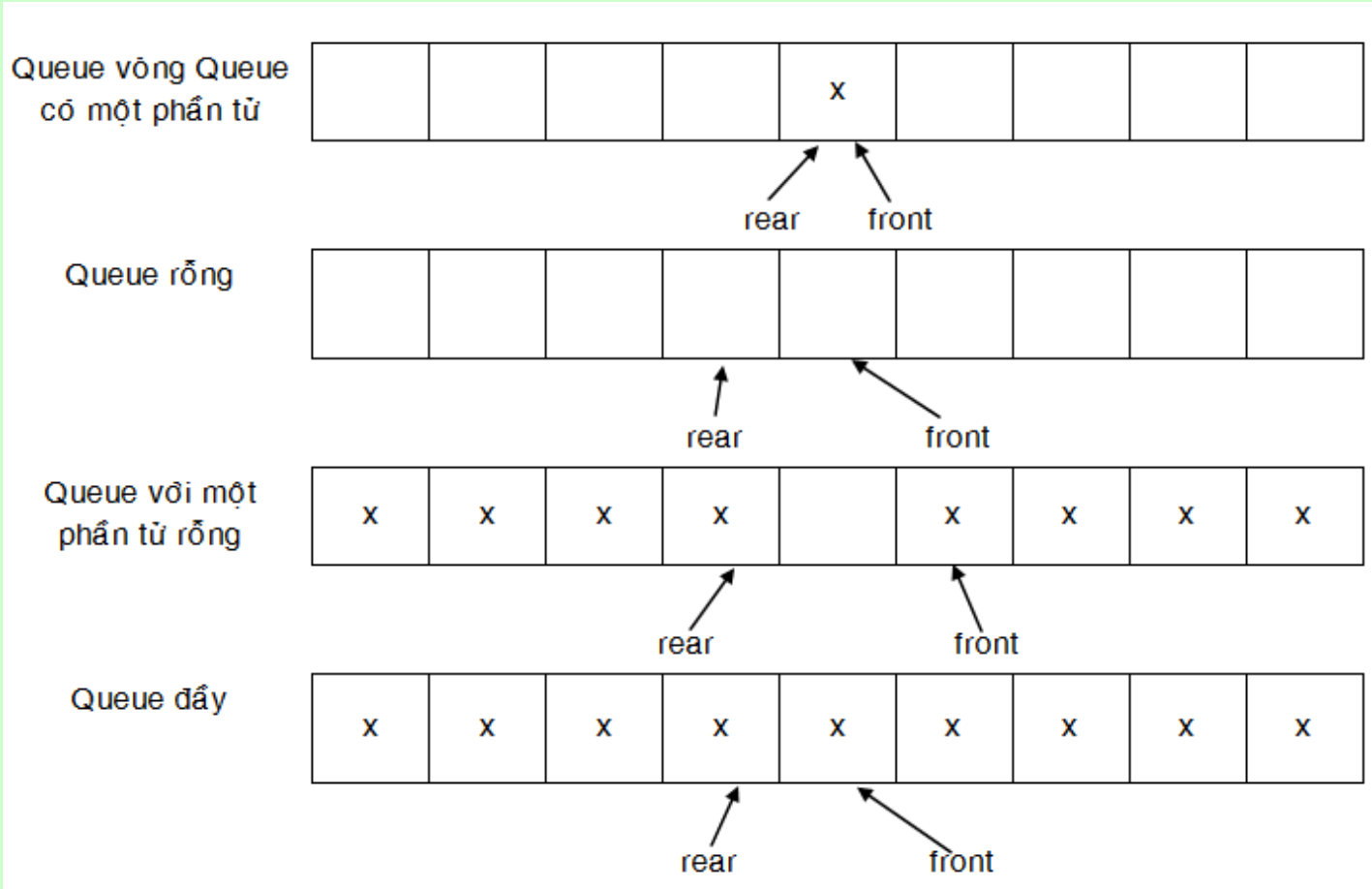
- Khởi động queue, tương ứng với hàm init_queue() cần thiết kế.
- Các hàm để xem queue rỗng, đầy.
- Thêm một phần tử vào queue, tương ứng hàm addqueue() cần thiết kế.
- Lấy một phần tử ra khỏi queue, tương ứng với hàm deletequeue() cần thiết kế.

CHƯƠNG 11
MẢNG

11.5 CÁC ỨNG DỤNG

11.5.3 Queue

Ví dụ:



CHƯƠNG 11
MẢNG

11.5 CÁC ỨNG DỤNG

11.5.3 Queue

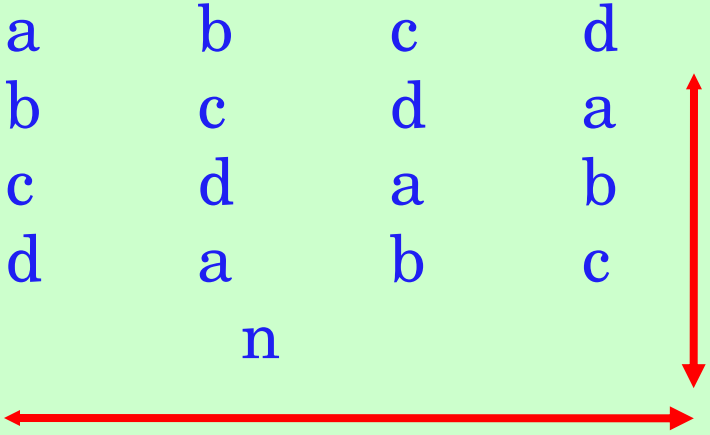
Ví du (SGT)

CHƯƠNG 11
MẢNG

BÀI TẬP CUỐI CHƯƠNG

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:



CHƯƠNG 11
MẢNG

BÀI TẬP CUỐI CHƯƠNG

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;

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

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

CHƯƠNG 11
MẢNG

BÀI TẬP CUỐI CHƯƠNG

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.

CHƯƠNG 11
MẢNG

BÀI TẬP CUỐI CHƯƠNG

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.