



Nguyễn Phúc Khải

CHƯƠNG 12: *POINTER*





Các nội dung:

- Khái niệm
- Thao tác trên POINTER
- POINTER và mảng
- Đối số của hàm là pointer - truyền đối số theo số dạng tham số biến
- Hàm trả về pointer và mảng
- Chuỗi ký tự
- Pointer và việc định vị bộ nhớ động
- Mảng các pointer



Các nội dung:

- Pointer của pointer
- Đối số của hàm MAIN
- Pointer trỏ đến hàm
- Ứng dụng





KHÁI NIỆM

- Một biến có kiểu pointer có thể lưu được dữ liệu trong nó, là địa chỉ của một đối tượng đang khảo sát. Đối tượng đó có thể là một biến, một chuỗi hoặc một hàm.



■ Ví dụ 13.1: Chương trình đổi trị

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

```
void Swap (int doi_1, int doi_2);
```

```
main()
```

```
{ int a = 3, b = 4;
```

```
printf (“Trước khi gọi hàm, a = %d, b = %d.\n”,a,b);
```

```
Swap (a, b); // Gọi hàm đổi trị
```

```
printf (“Sau khi gọi hàm, a = %d, b = %d.\n”,a,b);}
```

```
void Swap (int doi_1, int doi_2)
```

```
{ int temp = doi_1;
```

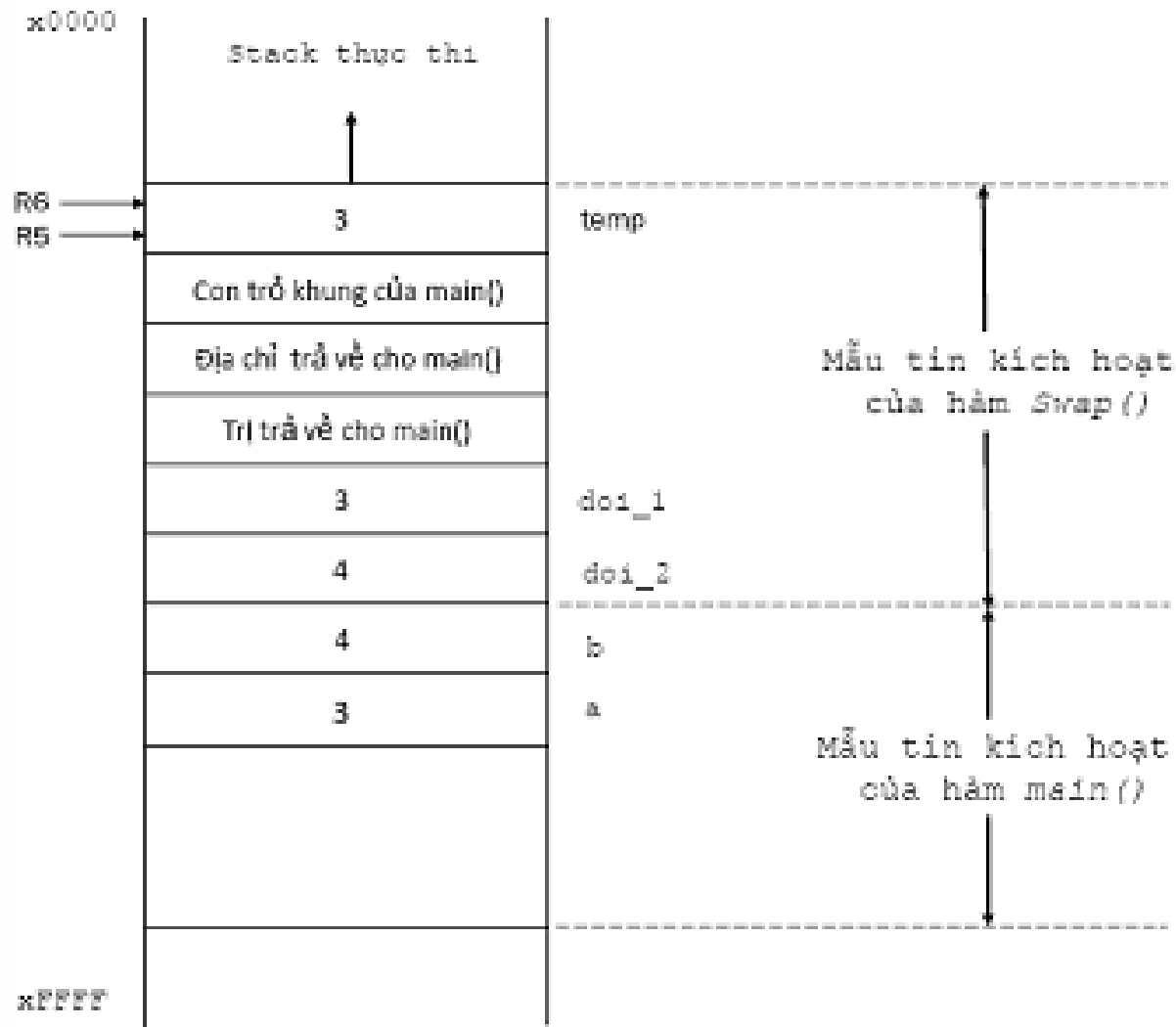
```
doi_1 = doi_2 ;
```

```
doi_2 = temp ; }
```



KHÁI NIỆM

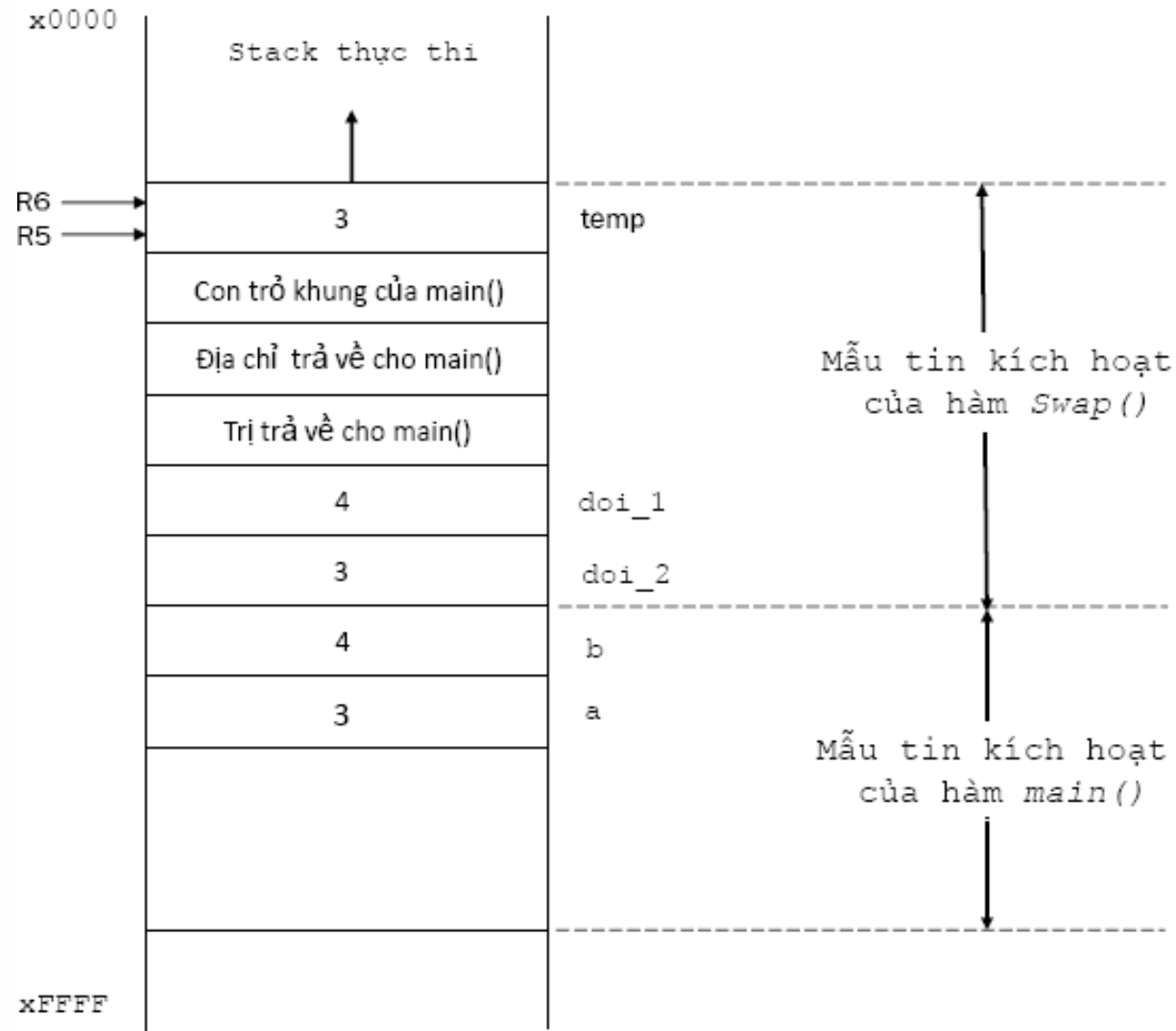
- Hình ảnh stack thực thi khi điều khiển chương trình đang ở dòng `doi_1 = doi_2 ;`





KHÁI NIỆM

- Hình ảnh stack thực thi khi điều khiển đến cuối chương trình





THAO TÁC TRÊN POINTER

- Cú pháp để khai báo biến pointer:
kiểu *tên_biến_pointer
- Với:
 - **kiểu** có thể là kiểu bất kỳ, xác định kiểu dữ liệu có thể được ghi vào đối tượng mà con trỏ đang trỏ đến.
 - **tên_biến_pointer** là tên của biến con trỏ, một danh hiệu hợp lệ.



THAO TÁC TRÊN POINTER

- Biến hoặc đối tượng mà con trỏ đang trỏ đến có thể được truy xuất qua tên của biến con trỏ và dấu "*" đi ngay trước biến con trỏ, cú pháp cụ thể như sau:

*** tên_biến_con_trỏ**





THAO TÁC TRÊN POINTER

- Khai báo biến pointer - pointer hằng:
- Trong ngôn ngữ C, một toán tử lấy địa chỉ của một biến đang làm việc, toán tử này là một dấu **&** (ampersand), tạm gọi là toán tử lấy địa chỉ. Cú pháp như sau:

& biến

- với **biến** là một biến thuộc kiểu bất kỳ, nhưng không được là biến thanh ghi.



THAO TÁC TRÊN POINTER

- Ví dụ:
- Nếu có một biến đã được khai báo là:
`int he_so_a;`
- thì
`&he_so_a`
- sẽ là địa chỉ của biến `he_so_a`.



THAO TÁC TRÊN POINTER

- **Ví dụ:** Xét ví dụ sau:

```
int object;  
int *pint;  
object = 5;  
pint = &object;  
printf("%d",*pint);
```





THAO TÁC TRÊN POINTER

■ Ví dụ:

```
void * pvoid;
```

```
int a, * pint;
```

```
double b, * pdouble;
```

```
pvoid = (void *) &a;
```

```
pint = (int *) pvoid;
```

```
(*pint) ++;
```

```
pvoid = (void *) &b;
```

```
pdouble = (double *) pvoid;
```

```
(*pdouble) -- ;
```





THAO TÁC TRÊN POINTER

- **Các phép toán trên pointer:**
- Có thể cộng, trừ một pointer với một số nguyên (int, long,...). Kết quả là một pointer.
- **Ví dụ :**

```
int *pi1, *pi2, n;  
pi1 = &n;  
pi2 = pi1 + 3;
```





THAO TÁC TRÊN POINTER

- **Ví dụ:** Cho khai báo:

```
int a[20];
```

```
int *p;
```

```
p = &a[0];
```

```
p += 3;
```

```
/* p lưu địa chỉ phần tử a[0 + 3], tức &a[3] */
```



THAO TÁC TRÊN POINTER

- KHÔNG thể thực hiện các phép toán nhân, chia, hoặc lấy dư một pointer với một số, vì pointer lưu địa chỉ, nên nếu thực hiện được điều này cũng không có một ý nghĩa nào cả.
- Phép trừ giữa hai pointer vẫn là một phép toán hợp lệ, kết quả là một trị thuộc kiểu int biểu thị khoảng cách (số phần tử) giữa hai pointer đó.



THAO TÁC TRÊN POINTER

- **Ví dụ:** Xét chương trình ví dụ sau:

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

```
#include <conio.h>
```

```
main()
```

```
{  int *p1, *p2; int a[10];
```

```
    p1 = &a[0];
```

```
    p2 = &a[5];
```

```
    printf ("Dia chi cua bien a[0] la: %p\n", p1);
```

```
    printf ("Dia chi cua bien a[5] la: %p\n", p2);
```

```
    printf ("Khoang cach giua hai phan tu la %d int\n", p2 -  
    p1);
```

```
    getch();
```



THAO TÁC TRÊN POINTER

- **Ví dụ:** Cho các khai báo sau:

```
int * a1;
```

```
char * a2;
```

```
a1 = 0; /* Chương trình dịch sẽ nhắc nhở lệnh này */
```

```
a2 = (char *)0;
```

```
if( a1 != a2) /* Chương trình dịch sẽ nhắc nhở kiểu của  
đối tượng */
```

```
{
```

```
    a1 = (int *) a2; /* Hợp lệ vì đã ép kiểu */
```

```
}
```



THAO TÁC TRÊN POINTER

■ Ví dụ:

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

```
#include <conio.h>
```

```
main()
```

```
{
```

```
    int *pint, a = 0x6141;
```

```
    char *pchar;
```

```
    clrscr();
```

```
    pint = &a;
```

```
    pchar = (char *) &a;
```



THAO TÁC TRÊN POINTER

```
printf ("Tri cua bien pint la: %p\n", pint);  
printf ("Tri cua bien pchar la: %p\n", pchar);  
printf ("Doi tuong pint dang quan ly la %X \n", *pint);  
printf ("Doi tuong pchar dang quan ly la %X \n", *pchar);  
pchar++;  
printf ("Doi tuong pchar dang quan ly la %X \n", *pchar);  
getch();  
}
```



THAO TÁC TRÊN POINTER

- C cho phép khai báo một biến pointer là hằng hoặc đối tượng của một pointer là hằng.
- **Ví dụ:** Các khai báo sau đây là biến pointer hằng:

1. `int a, b;`

`int * const pint = &a;`

`pint = &b;` \rightarrow

C báo lỗi

2. `char * const ps = "ABCD";`



THAO TÁC TRÊN POINTER

- Các khai báo sau đây cho thấy đối tượng của một pointer là hằng:

1. `int i;`

`const int * pint;`

`pint = &i;`

`i = 1;`

`(*pint) ++;` ← C báo lỗi

`i ++;`

2. `const char * s = "ABCD"` hoặc

`char const * s = "ABCD";`



THAO TÁC TRÊN POINTER

- Phân biệt:
 - **const trước *** : pointer chỉ đến đối tượng hằng
 - *** trước const** : pointer hằng





POINTER VÀ MẢNG

- Trong C, *tên mảng là một hằng pointer tới phần tử có kiểu là kiểu của biến thành phần dưới nó một bậc trong mảng,*
- VD: tên của mảng một chiều của các **int** là pointer chỉ tới int, tên của mảng hai chiều của các int là pointer chỉ tới mảng một chiều là hàng các int trong mảng.
- Trong trường hợp mảng một chiều, tên mảng chính là địa chỉ của phần tử đầu tiên của mảng. Do đó, ta hoàn toàn có thể truy xuất mảng bằng một pointer



POINTER VÀ MẢNG

- Ví dụ:

```
int a[3];  
int (*p)[3];  
p[0] = &a ;
```

- Ví dụ: Cho khai báo sau:

```
int a[10];  
int *pa;  
*(a + 0) chính là a[0],  
*(a + 1) chính là a[1],  
....
```

*(a + i) chính là a[i].

Có thể gán:

pa = a;

hoặc **pa = &a[0];**



POINTER VÀ MẢNG

Khi đó, $(pa + 0)$ sẽ chỉ đến phần tử $a[0]$,
 $(pa + 1)$ sẽ chỉ đến phần tử $a[1]$,
...
 $(pa + i)$ sẽ chỉ đến phần tử $a[i]$.

Như vậy, các đối tượng

$*(pa + 0)$ chính là $*(a + 0)$, cũng là $a[0]$
 $*(pa + 1)$ chính là $*(a + 1)$, cũng là $a[1]$
...
 $*(pa + i)$ chính là $*(a + i)$, cũng là $a[i]$



POINTER VÀ MẢNG

- Có thể truy xuất đến các đối tượng của mảng mà pointer đang trỏ đến như sau:

$pa[0]$ chính là phần tử $a[0]$

$pa[1]$ chính là phần tử $a[1]$

...

$pa[i]$ chính là phần tử $a[i]$.





POINTER VÀ MẢNG

- Phân biệt rõ ràng giữa mảng và pointer:
 - Một mảng, sau khi được khai báo và định nghĩa, đã được cấp một vùng nhớ trong bộ nhớ của máy tính và địa chỉ chính là tên mảng.
 - Một biến pointer, sau khi được khai báo, thì vùng nhớ được cấp chỉ là bản thân biến pointer, còn trị bên trong nó là một địa chỉ rác.
 - Ngoài ra, biến pointer có một địa chỉ trong bộ nhớ, còn không thể lấy địa chỉ của tên mảng.



POINTER VÀ MẢNG

- Ví dụ: Sau khi khai báo

`int a[10];`

- thì trong bộ nhớ, a là một hằng pointer, hay một địa chỉ cố định
- Vì tên mảng là một hằng pointer
`a++;`
- là không hợp lệ.

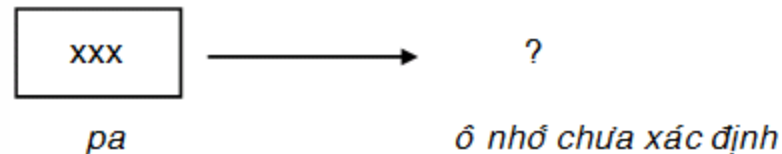


POINTER VÀ MẢNG

- Ví dụ:

```
int *pa;
```

- thì trong bộ nhớ



- do đó lệnh gán
- `*pa = 2;` là không có ý nghĩa, dù C không báo lỗi trong trường hợp này.



POINTER VÀ MẢNG

- Từ mảng hai chiều trở đi, việc dùng biến pointer để truy xuất mảng là khá phức tạp, chúng ta cần phải luôn nhớ là các thao tác trên pointer luôn diễn ra trên cùng một bậc quản lý đối tượng, nghĩa là chúng ta phải luôn biết pointer mà chúng ta sử dụng đang quản lý đối tượng kiểu nào.





POINTER VÀ MẢNG

■ Ví dụ:

```
#define MAX_ROW  20
#define MAX_COL  30
int array[MAX_ROW][MAX_COL];
int row, col;
int (*parr) [MAX_ROW][MAX_COL];
/* biến con trỏ mảng hai chiều */
parr = &array;
```




POINTER VÀ MẢNG

- Với khai báo trên, danh hiệu array là hàng pointer hai lần pointer chỉ tới phần tử đầu tiên của mảng *array*, tức ta có ***array* chính là *array[0][0]*.
- Với mảng một chiều, *array[row]* chính là **(array+row)*, tức *array[row] ≡ *(array+row)*.
- Với mảng hai chiều, ta có *array[0]* ($= *array$ hay $*(array + 0)$) là con trỏ chỉ tới hàng 0 trong mảng (và dĩ nhiên *array[row]* là con trỏ chỉ tới hàng *row* trong mảng, ...). Như vậy, ta có **array[0]* chính là ***array* và cũng chính là *array[0][0]*.



POINTER VÀ MẢNG

- Khi đó, ta có:

$\text{array}[\text{row}][\text{col}] \equiv *(\text{array}[\text{row}] + \text{col})$, tức

$\text{array}[\text{row}][\text{col}] \equiv *(\text{int } (*) \text{ array} + \text{row} * \text{MAX_COL} + \text{col})$

$\text{array}[\text{row}][\text{col}] \equiv *(*(\text{array} + \text{row}) + \text{col})$

$\text{array}[\text{row}][\text{col}] \equiv *(*(\text{int } (*)[\text{MAX_COL}] \text{ parr} + \text{row}) + \text{col})$





ĐỔI SỐ CỦA HÀM LÀ POINTER - TRUYỀN ĐỔI SỐ THEO SỐ DẠNG THAM SỐ BIẾN

- Hàm Swap() ở đoạn chương trình trên có thể viết lại như sau:

```
void Swap (int *doi_1, int *doi_2)
```

*/*pointer là đối số của hàm, để nhận địa chỉ của đối số thật*/*

```
{ int temp;  
    temp = * doi_1;  
    * doi_1 = * doi_2;  
    * doi_2 = temp;  
}
```





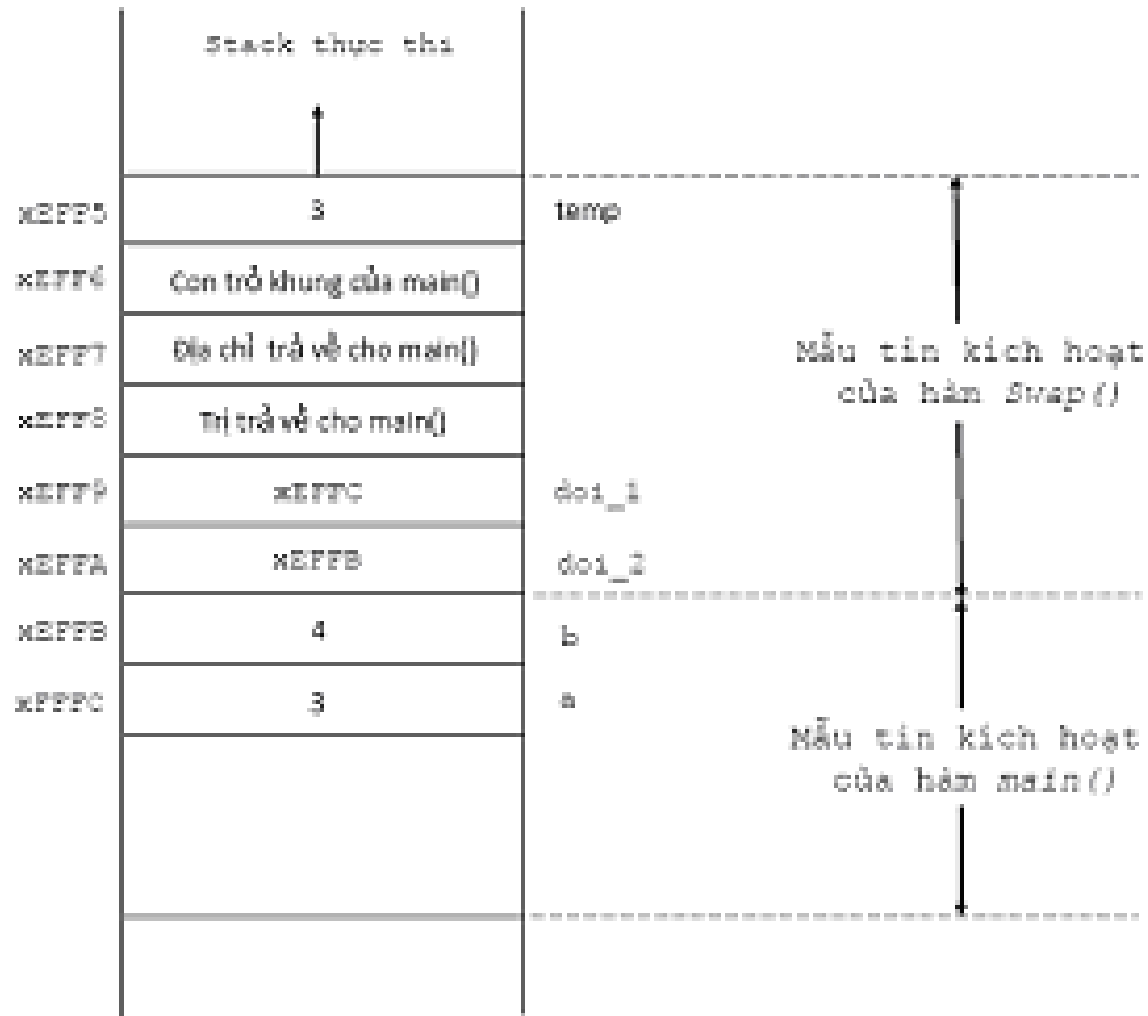
ĐỔI SỐ CỦA HÀM LÀ POINTER - TRUYỀN ĐỔI SỐ THEO SỐ DẠNG THAM SỐ BIẾN

```
#include <stdio.h>
#include <conio.h>
void Swap (int *doi_1, int *doi_2);
main ()
{
    int a = 3, b = 4;
    clrscr();
    printf ("Tri cua hai bien a va b la: %d %d\n", a, b);
    Swap (&a, &b);
    printf ("Sau khi goi ham Swap, tri cua a va b la: %d  

        %d\n", a, b);
    getch(); }
```

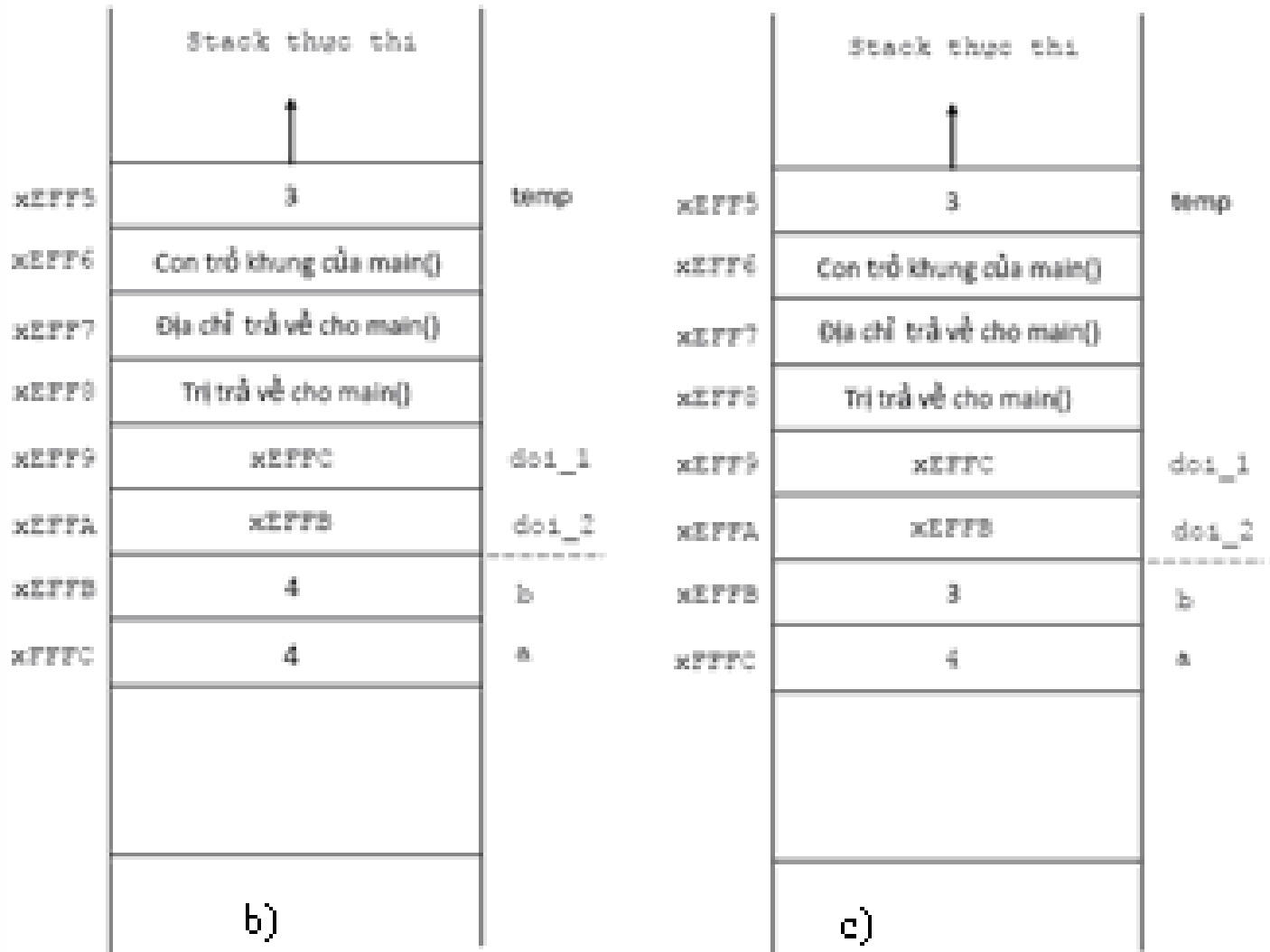


ĐỐI SỐ CỦA HÀM LÀ POINTER - TRUYỀN ĐỐI SỐ THEO SỐ DẠNG THAM SỐ BIẾN





ĐỔI SỐ CỦA HÀM LÀ POINTER - TRUYỀN ĐỔI SỐ THEO SỐ DẠNG THAM SỐ BIẾN





ĐỐI SỐ CỦA HÀM LÀ POINTER - TRUYỀN ĐỐI SỐ THEO SỐ DẠNG THAM SỐ BIẾN

- Trong thư viện chuẩn của C cũng có nhiều hàm nhận đối số vào theo địa chỉ, ví dụ hàm **scanf()**, nhập trị vào cho biến từ bàn phím.



- Viết hàm nhập 3 số từ bàn phím và hàm nhập mảng 1 chiều. Trong hàm main(), gọi 2 hàm trên để nhập dữ liệu và xuất giá trị ra màn hình.



HÀM TRẢ VỀ POINTER VÀ MẢNG

- Một pointer có thể được trả về từ hàm, nếu pointer trỏ đến đối tượng là mảng, thì hàm trả về mảng; nếu pointer chỉ trỏ đến biến bình thường, thì hàm chỉ trả về con trỏ trỏ đến biến thường, cú pháp khai báo hàm như sau:

kiểu *tên_hàm (danh_sách_khai_báo_đối_số);

- với **kiểu** là kiểu của đối tượng mà pointer được hàm trả về trỏ đến.



HÀM TRẢ VỀ POINTER VÀ MẢNG

- **Ví dụ:** Có khai báo

`int *lon_nhat (int a, int b, int c);`

- thì hàm `lon_nhat()` trả về một địa chỉ, địa chỉ đó có thể là địa chỉ của một **int** hoặc địa chỉ của một mảng các **int**, việc sử dụng địa chỉ theo đối tượng nào là do nơi gọi.



HÀM TRẢ VỀ POINTER VÀ MẢNG

- **Ví dụ:** Thiết kế hàm nhập trị cho mảng các int

`int *nhap_tri (int *num)`

`{`

`static int a[10]; int i, n;`

`printf ("Nhap kích thước mảng:");`

`scanf ("%d", &n);`

`*num = n;`

`printf ("Nhap trị cho %d phần tử của mảng:", n);`

`for (i = 0; i < n; i++)`

`scanf ("%d", &a[i]);`

`return a; /* a là địa chỉ đầu mảng cần trả về */`

`}`



HÀM TRẢ VỀ POINTER VÀ MẢNG

- **Ví dụ:** Chương trình sử dụng hàm nhập trị mảng

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

```
#include <conio.h>
```

```
int *nhap_tri(int *num);
```

```
main()
```

```
{  int *pint, so_phan_tu, i;
```

```
    clrscr();
```

```
    pint = nhap_tri (&so_phan_tu);
```

```
    printf ("Cac phan tu cua mang la:");
```

```
    for (i =0; i <so_phan_tu; i++)
```

```
        printf ("%d", pint[i]);
```

```
    getch();      }
```



CHUỖI KÝ TỰ

- **Nhập trị chuỗi:**
- Việc nhập trị cho chuỗi bao gồm hai bước: đầu tiên cần khai báo một nơi trống để chứa chuỗi, sau đó dùng một hàm nhập trị để lấy chuỗi.
- **Hàm *gets()*** đọc các ký tự đến khi nào gặp ký tự quy định hàng mới (tức ký tự '\n', tức khi ta ấn phím ENTER) thì kết thúc việc nhập. Sau đó hàm này lấy tất cả các ký tự đã nhập trước ký tự '\n', gắn thêm vào cuối chuỗi một ký tự NUL ('\0') và trả chuỗi cho chương trình gọi.



CHUỖI KÝ TỰ

- *Hàm scanf()* cũng cho phép nhập chuỗi qua định dạng nhập **%s**. Việc nhập chuỗi sẽ kết thúc khi hàm scanf() gặp một trong các ký tự khoảng trắng, ký tự tab hay ký tự xuống hàng đầu tiên mà nó gặp. Đây chính là điểm khác nhau giữa hai hàm nhập chuỗi gets() và scanf().



CHUỖI KÝ TỰ

```
#include <stdio.h>
#include <conio.h>
main()
{ char ten1[41], ten2[41];
  clrscr();
  printf(Moi ban nhap hai ten: );
  scanf ("%s %s", ten1, ten2);
  printf("A! Chao hai ban %s va %s \n", ten1, ten2);
  getch(); }
```



CHUỖI KÝ TỰ

- **Xuất chuỗi**
- Để xuất chuỗi, hai hàm thường hay được dùng là **puts()** và **printf()**.
- **Hàm puts**: ta chỉ cần cung cấp cho hàm đối số là địa chỉ của chuỗi cần in. Hàm này sẽ đọc từng ký tự của chuỗi và in ra màn hình cho đến khi gặp ký tự NUL thì in ra màn hình thêm một ký tự xuống hàng nữa. Prototype của hàm này như sau: **int puts (char * s);**



CHUỖI KÝ TỰ

- *Hàm printf ()* cũng cho phép xuất chuỗi ra màn hình nếu ta dùng định dạng xuất "%s" cho nó. Hàm này sẽ không tự động in thêm ký tự xuống hàng mới như hàm puts().





CHUỖI KÝ TỰ

- **Gán trị cho chuỗi:**
- Việc gán trị cho biến chuỗi thực tế là việc chép từng ký tự từ hằng chuỗi hoặc biến chuỗi đã biết sang một biến chuỗi khác. Trong C, thao tác này được thực hiện nhờ hàm `strcpy()`, hàm này có prototype trong file `string.h` như sau:

`char *strcpy(char *dest, const char *src);`



■ Ví dụ:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
main()
{ char ten1[41], ten2[41]; clrscr();
  printf("Moi ban nhap ten: ");
  gets(ten1);
  strcpy (ten2, ten1);
  printf("A! Chao ban: %s", ten2);
  getch(); }
```



- **Lấy chiều dài chuỗi**
- Trong C, để lấy chiều dài chuỗi ta dùng hàm **strlen()**. Prototype của hàm này trong file **string.h**:

`size_t strlen(const char *s);`

- với `size_t` là một kiểu nguyên, C quy định đây là **unsigned**.



CHUỖI KÝ TỰ

- Xét chương trình nhập một chuỗi, đổi các ký tự thường của chuỗi đó thành ký tự hoa, in chuỗi đó ra lại màn hình.





- **Nối chuỗi:**
- Để nối hai chuỗi lại, C có hàm chuẩn **strcat()**.
Hàm này nhận hai chuỗi làm đối số, và một bản sao của chuỗi thứ hai sẽ được chép nối vào cuối của chuỗi thứ nhất, để tạo ra chuỗi mới. Chuỗi thứ hai vẫn không có gì thay đổi.
Prototype của hàm này trong file string.h:

```
char *strcat(char *dest, const char *src);
```



- So sánh chuỗi
- Hàm này là **strcmp()**, có prototype trong file string.h:
int strcmp(const char *s1, const char*s2);
- Hàm này so sánh hai chuỗi s1 và s2 và trả về một trị là:
 - số dương nếu $s1 > s2$
 - số âm nếu $s1 < s2$
 - số 0 nếu $s1 == s2$



■ Ví dụ: Xét chương trình ví dụ sau đây

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
main()
{ clrscr();
  printf("%d \n", strcmp("QUAN", "quan"));
  printf("%d \n", strcmp("QUAN", "QUAN"));
  printf("%d \n", strcmp("quan", "QUAN"));
  printf("%d \n", strcmp("quang", "quanG"));
  printf("%d \n", strcmp("quang", "quan"));    getch();
}
```




■ Chương trình cho xuất liệu:

-32

0

32

32

103





POINTER VÀ VIỆC ĐỊNH VỊ BỘ NHỚ ĐỘNG

- C cho phép khai báo các biến động, các biến này khi cần thì xin chỗ, không cần thì giải phóng vùng nhớ cho chương trình sử dụng vào mục đích khác. Các biến động này được cấp phát trong vùng nhớ heap, là vùng đáy bộ nhớ (vùng còn lại sau khi đã nạp các chương trình khác xong), và được quản lý bởi các biến pointer



POINTER VÀ VIỆC ĐỊNH VỊ BỘ NHỚ ĐỘNG

- Trong C có hai hàm chuẩn để xin cấp phát bộ nhớ động **malloc()** và **calloc()**, cả hai hàm đều có prototype nằm trong file `alloc.h` hoặc `stdlib.h` như sau:

```
void *malloc(size_t size);
```

```
void *calloc(size_t nitems, size_t size);
```





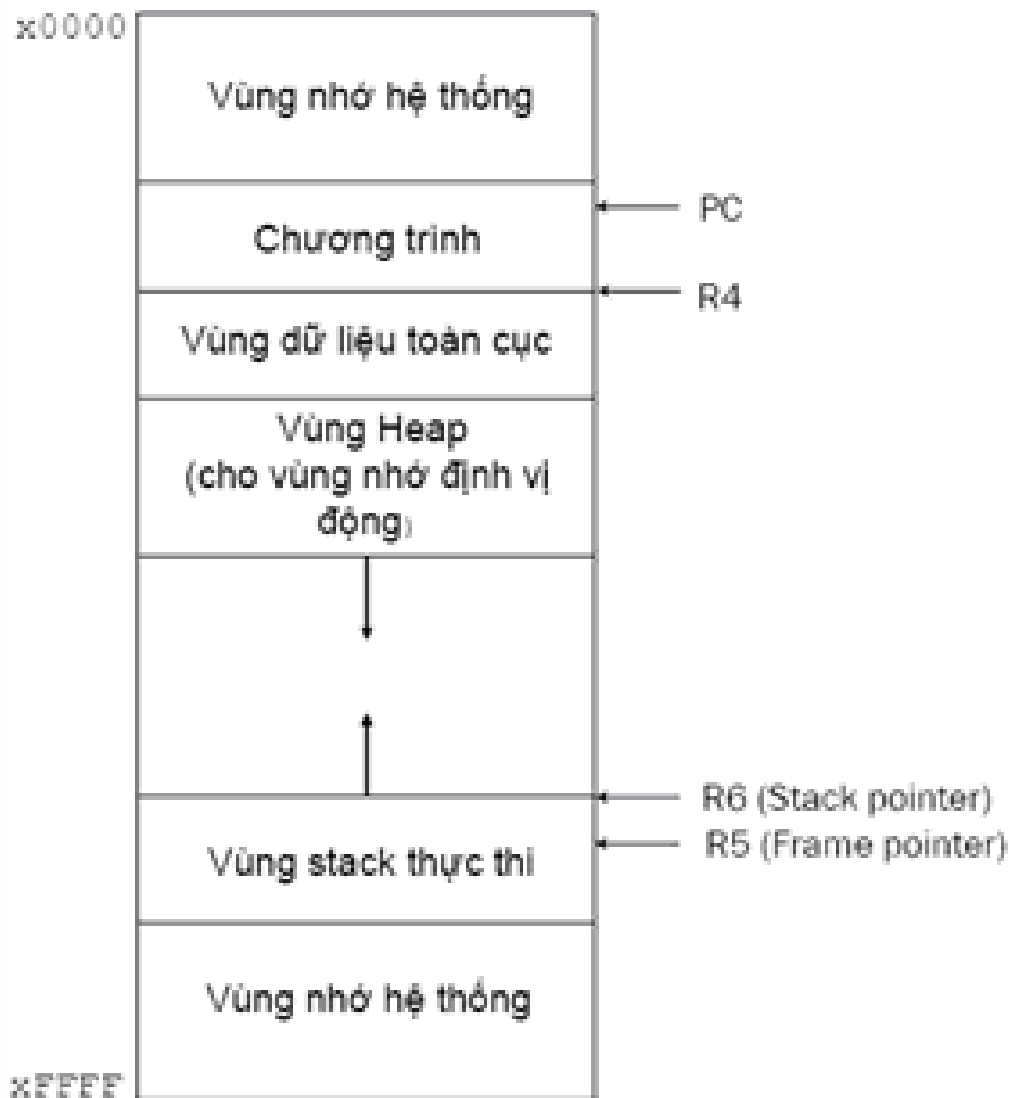
POINTER VÀ VIỆC ĐỊNH VỊ BỘ NHỚ ĐỘNG

- Nếu biến động được xin, sau khi dùng xong, vùng nhớ của nó không được giải phóng thì nó vẫn chiếm chỗ trong bộ nhớ, mặc dù chương trình đã kết thúc. C đưa ra hàm `free()` để giải phóng khối bộ nhớ được xin bằng hàm `malloc()` hoặc `calloc()`.
- Prototype của hàm `free()` trong file `stdlib.h` hoặc `alloc.h` như sau:

`void free (void * block);`



POINTER VÀ VIỆC ĐỊNH VỊ BỘ NHỚ ĐỘNG





POINTER VÀ VIỆC ĐỊNH VỊ BỘ NHỚ ĐỘNG

- **Ví dụ:** Cần xin một khối bộ nhớ có 10 phần tử int, ta viết như sau:

```
int *pint;
```

```
pint = (int *) malloc (10 * sizeof (int));
```

hoặc

```
pint = (int *) calloc (10, sizeof (int));
```



POINTER VÀ VIỆC ĐỊNH VỊ BỘ NHỚ ĐỘNG

```
#include <stdio.h>
#include <conio.h>
#include <process.h>      ← EXIT
#include <alloc.h>
main()
{  int *pint, s = 0, i;
    pint = (int *) calloc (10, sizeof (int));
        if (pint == NULL)
            { printf ("Khong du bo nho \n");
              exit (1);
            }
```



POINTER VÀ VIỆC ĐỊNH VỊ BỘ NHỚ ĐỘNG

```
clrscr();  
printf ("Moi nhap 10 tri vao mang: ");  
for (i = 0; i <10; i++)  
    scanf ("%d", &pint[i]);  
for (i = 0; i <10; i++)  
    s += pint[i];  
printf ("Tong cac phan tu cua mang la: %d \n", s);  
getch();  
free (pint);  
}
```




MẢNG CÁC POINTER

- Cú pháp khai báo mảng các pointer:
 kiểu * tên_mảng [Kích_thước];

- Ví dụ: Khi khai báo

```
int * pint[4];
```

```
int a = 2, b = 3, c = 4, d = 5;
```

```
pint[0] = &a;
```

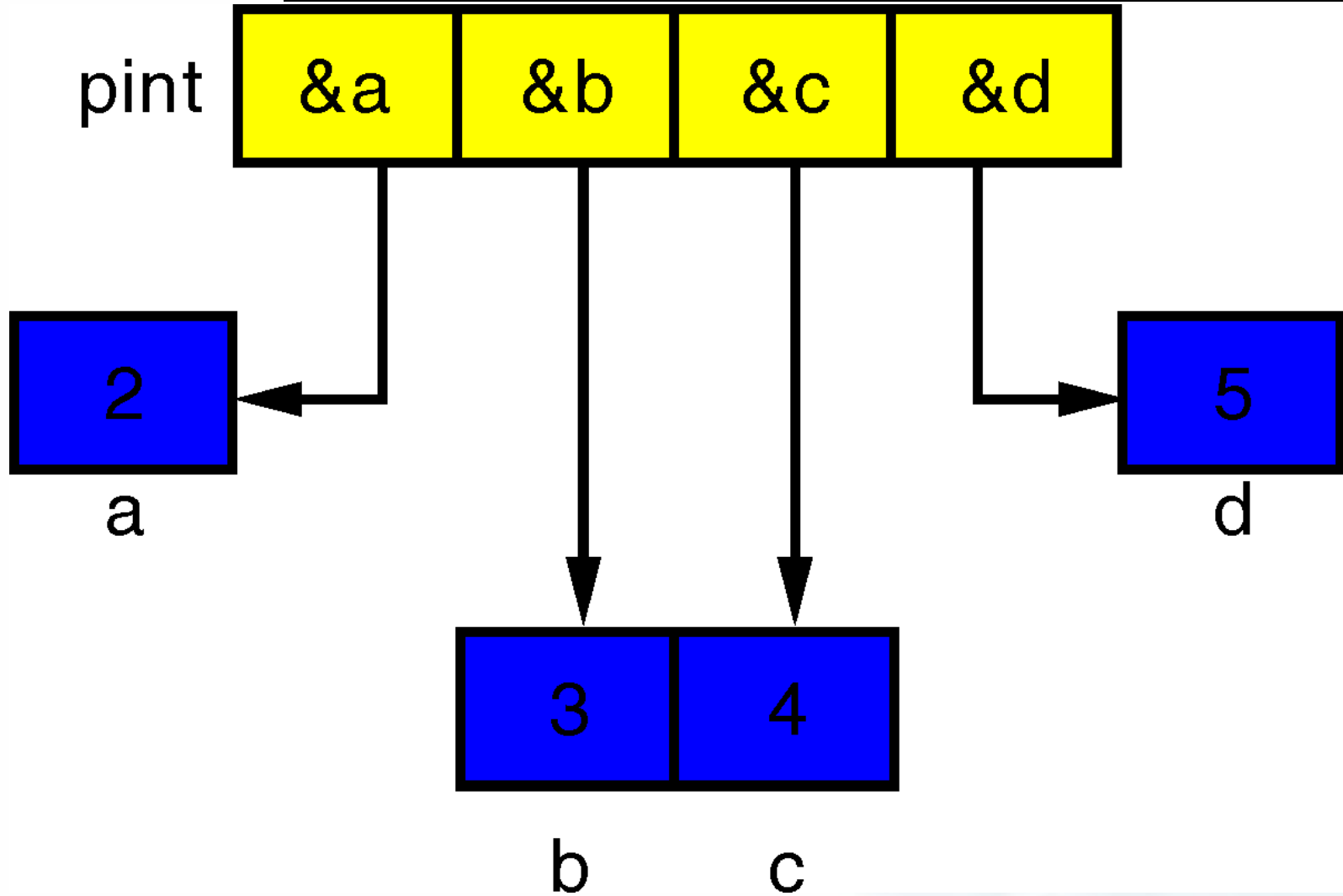
```
pint[1] = &b;
```

```
pint[2] = &c;
```

```
pint[3] = &d;
```



MẢNG CÁC POINTER





MẢNG CÁC POINTER

- Ví dụ: Khi khai báo

```
int * pint[4];
```

```
int a[3], b[3], c[3], d[3];
```

```
pint[0] = a;
```

```
pint[1] = b;
```

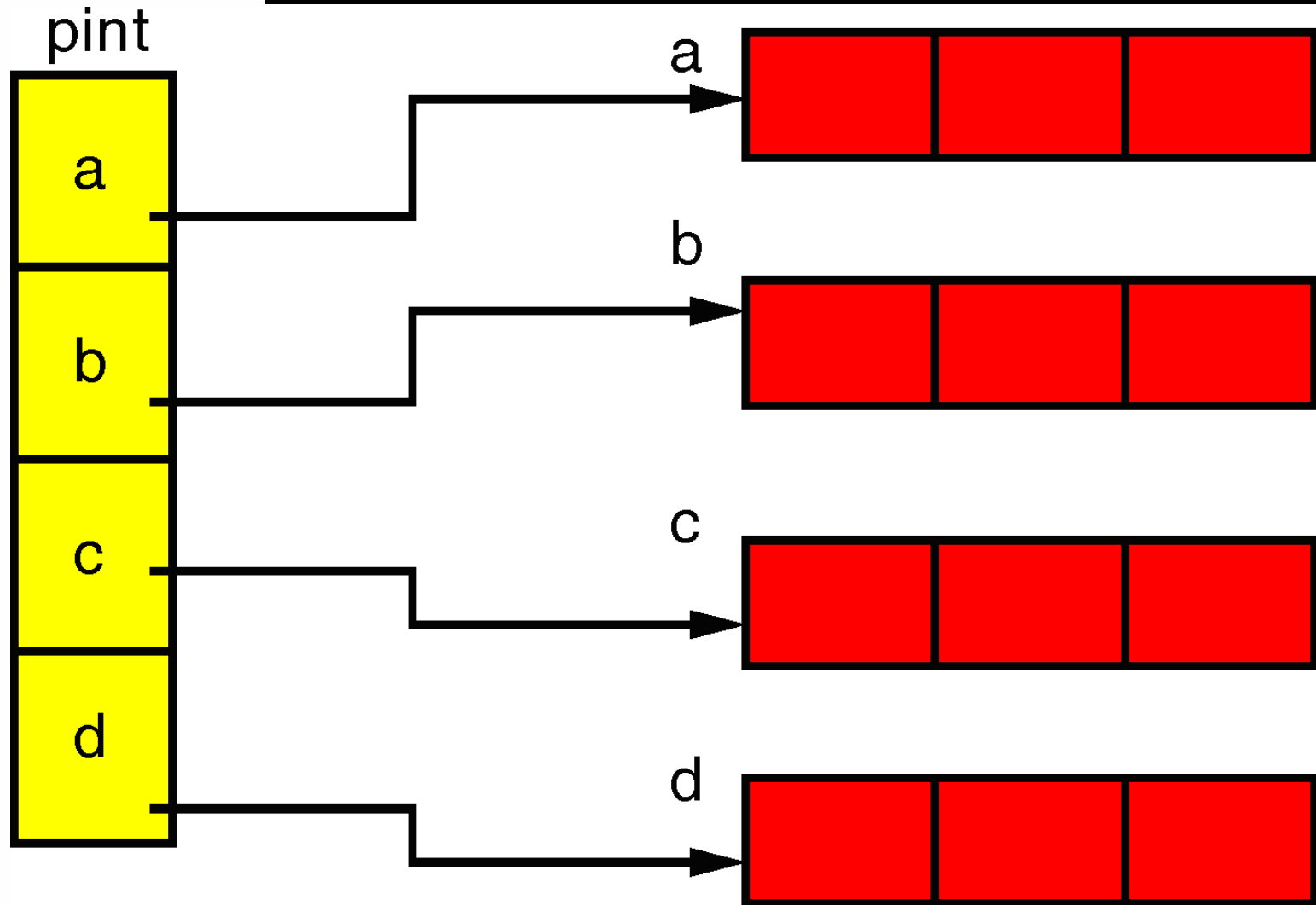
```
pint[2] = c;
```

```
pint[3] = d;
```





MẢNG CÁC POINTER





MẢNG CÁC POINTER

- Một mảng các pointer cũng có thể được khởi động trị nếu mảng là mảng toàn cục hay mảng tĩnh.
- **Ví dụ:**

```
static char *thu[7] = {"Thu 2", "Thu 3", "Thu 4", "Thu 5",  
"Thu 6", "Thu 7", "Chua nhat"};
```





POINTER CỦA POINTER

- Cú pháp khai báo pointer này như sau:

kiểu ** tên_pointer

- Ví dụ:

```
int **pint;
```

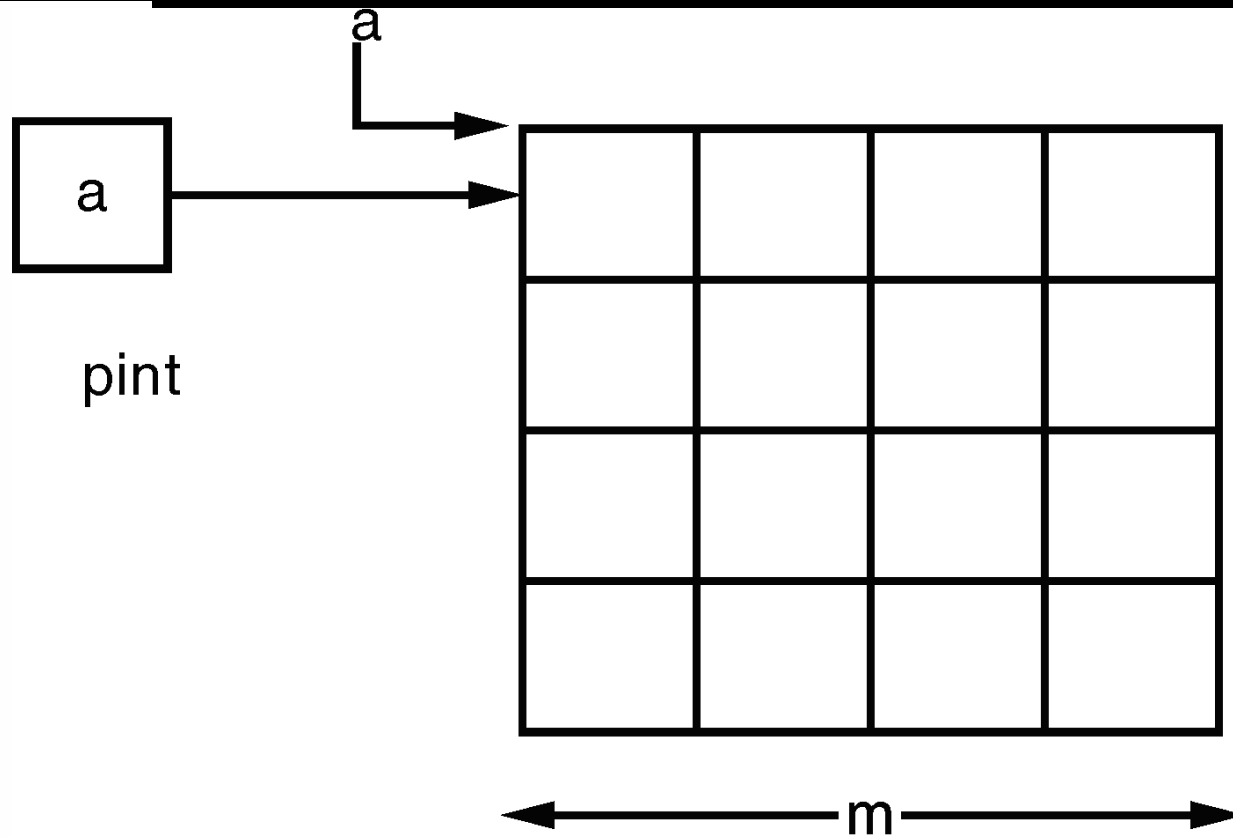
```
int *p;
```

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

- thì `pint = &p;`
hoặc `pint = (int **) &a;`



POINTER CỦA POINTER



- Thay vì truy xuất $a[i][j]$, ta có thể truy xuất $*(pint + m*i + j)$, với m là số phần tử trên một hàng của mảng hai chiều.



POINTER CỦA POINTER

```
#include <stdio.h>
#define MAX_ROW 3
#define MAX_COL 3
main()
{   int row, col;
    int *pint1;
    int a2d [MAX_ROW][MAX_COL] = { {0, 1, 2},
                                     {10, 11, 12},
                                     {20, 21, 22} };

    int **pint2;
    int (*pa2d)[MAX_ROW][MAX_COL];
    /* Thu dia chi cua pointer va mang 2 chieu */
```




POINTER CỦA POINTER

```
pint1 = a2d[1];
pa2d = &a2d;
pint2 = (int **)&a2d;
printf ("pint1 = a2d[1] = %p\n", pint1);
printf ("*( *( ( int (*)[MAX_COL] ) pint2 + 1)+ 2)=
%d\n", *( *( ( int  (*)[MAX_COL] ) pint2 + 1)+ 2));
printf ("*( *(a2d + 1) + 2) = %d\n", *( *(a2d + 1) + 2));
printf ("*pint1[2] = %d\n", pint1[2]);
printf ("(*pa2d)[1][2] = %d\n", (*pa2d)[1][2]);
printf ("Tri cua cac phan tu trong mang truy xuat qua
pointer 2 lan:\n");
```



POINTER CỦA POINTER

```
for (row = 0; row < MAX_ROW; row ++)  
{   for (col = 0; col < MAX_COL; col ++)  
    printf ("%d  ", *( *( ( int (*)[MAX_COL] ) pint2 +  
row)+ col));  
    printf ("\n");   }  
getchar();}
```

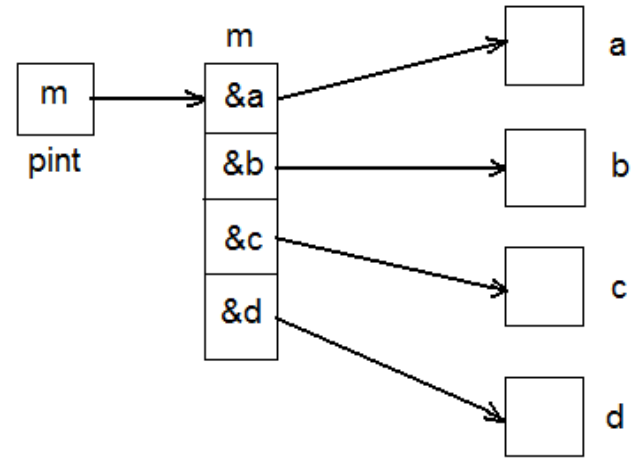




POINTER CỦA POINTER

■ Ví dụ:

```
int *m[4];  
int a = 1, b = 2, c = 3, d = 4; int **pint;  
pint = m;  
m[0] = &a; m[1] = &b; m[2] = &c;  
m[3] = &d;
```



- Thay vì truy xuất trực tiếp a, b, ..., ta có thể dùng pointer ***(pint[i])**



POINTER CỦA POINTER

■ Ví dụ:

```
int ** pi;
```

```
int * pint[4];
```

```
int a[3], b[3], c[3], d[3];
```

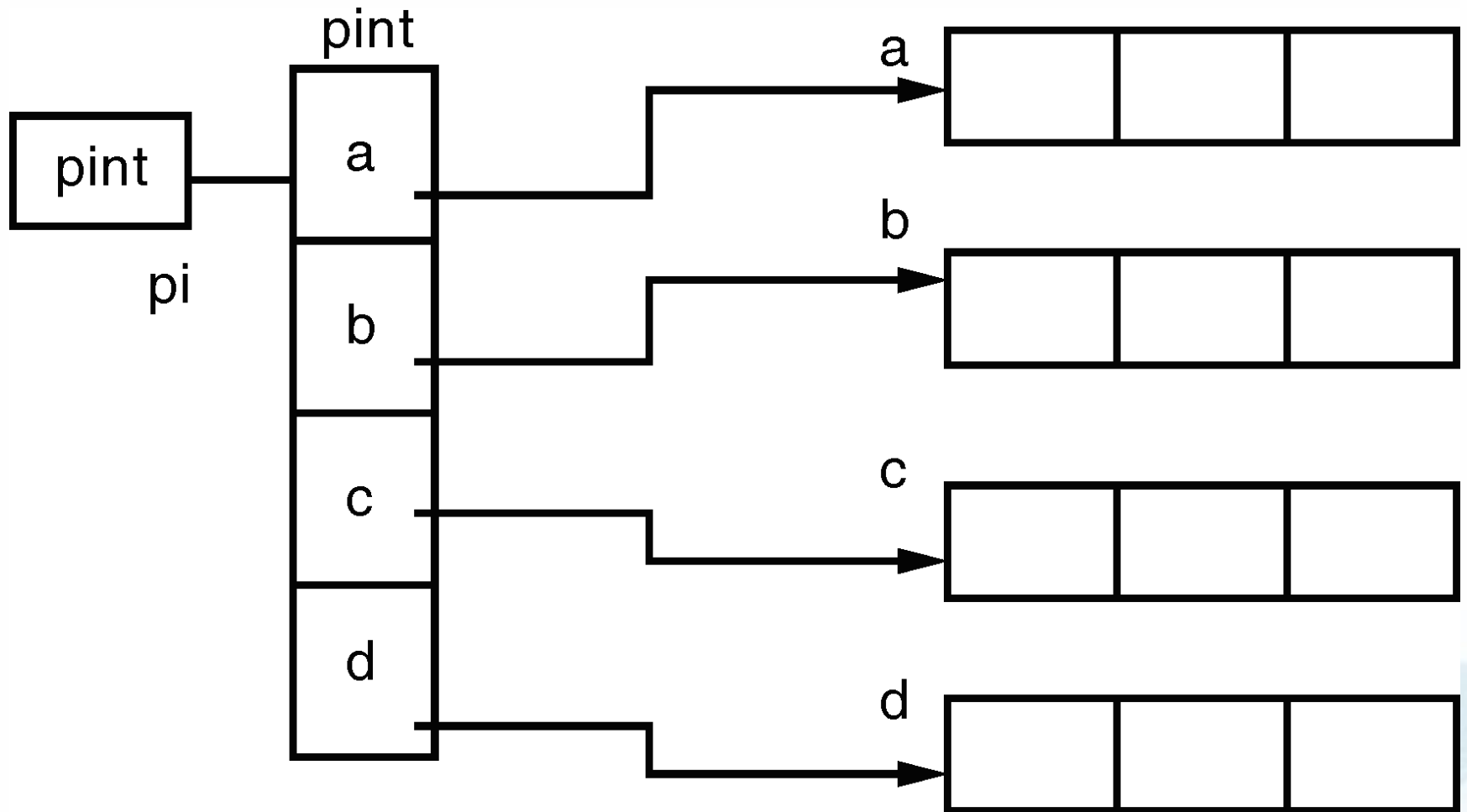
```
pi = pint;
```

```
pint[0] = a; pint[1] = b; pint[2] = c; pint[3] = d;
```





POINTER CỦA POINTER





ĐỔI SỐ CỦA HÀM MAIN

- C hoàn toàn cho phép việc nhận đối số vào hàm `main()`, có hai đối số C đã quy định theo thứ tự:
 - **int argc**: đối số cho biết số tham số đã nhập, kể cả tên chương trình.
 - **char *argv[]**: mảng các pointer trỏ đến các chuỗi là tham số đi theo sau tên chương trình khi chạy chương trình từ DOS.



ĐỔI SỐ CỦA HÀM MAIN

```
#include <stdio.h>
#include <conio.h>
main (int argc, char *argv[])
{  int i;
   clrscr();
   printf ("Cac doi so cua chuong trinh la: \n");
   printf ("Ten chuong trinh la: %s \n", argv[0]);
   if ( argc > 1 )
       for (i = 1; i < argc; i++)
           printf ("Doi so thu %d: %s \n", i, argv[i]);
   getch();
}
```



ĐỔI SỐ CỦA HÀM MAIN

- Nếu nhập từ bàn phím như sau:

C:\>thu_main tin thu 123

- thì chương trình cho xuất liệu là:

Cac doi so cua chuong trinh la:

Ten chuong trinh la: C:\thu_main.exe

Doi so thu 1: tin Doi so thu 2: thu

Doi so thu 3: 123





POINTER TRỞ ĐẾN HÀM

- Cú pháp khai báo một pointer chỉ tới hàm:
 kiểu (* tên_pointer) (kiểu_các_đối_số);
- Nếu khai báo **int (* p_function) (int, int);**
- và đã có hàm

```
int cong (int a, int b)
```

```
{    ...
```

```
}
```

- Ta có thể:

```
p_function = cong;
```

```
tong = (*p_function) (m, n);
```



Bài tập

- Viết chương trình nhập vào một dãy số nguyên (chưa biết có bao nhiêu số nguyên). Loại bỏ các số nguyên bị lặp lại. In ra dãy số mới này.





Kết thúc chương 12