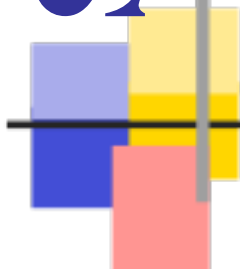


Các kiểu dữ liệu nâng cao - Sắp xếp



Bài 11



Mục tiêu - 1

- Tìm hiểu kiểu dữ liệu cấu trúc và công dụng
- Định nghĩa cấu trúc
- Khai báo các biến kiểu cấu trúc
- Cách truy cập vào các phần tử của cấu trúc
- Khởi tạo biến cấu trúc
- Sử dụng biến cấu trúc trong câu lệnh gán
- Cách truyền tham số cấu trúc
- Sử dụng mảng các cấu trúc
- Tìm hiểu cách khởi tạo mảng các cấu trúc

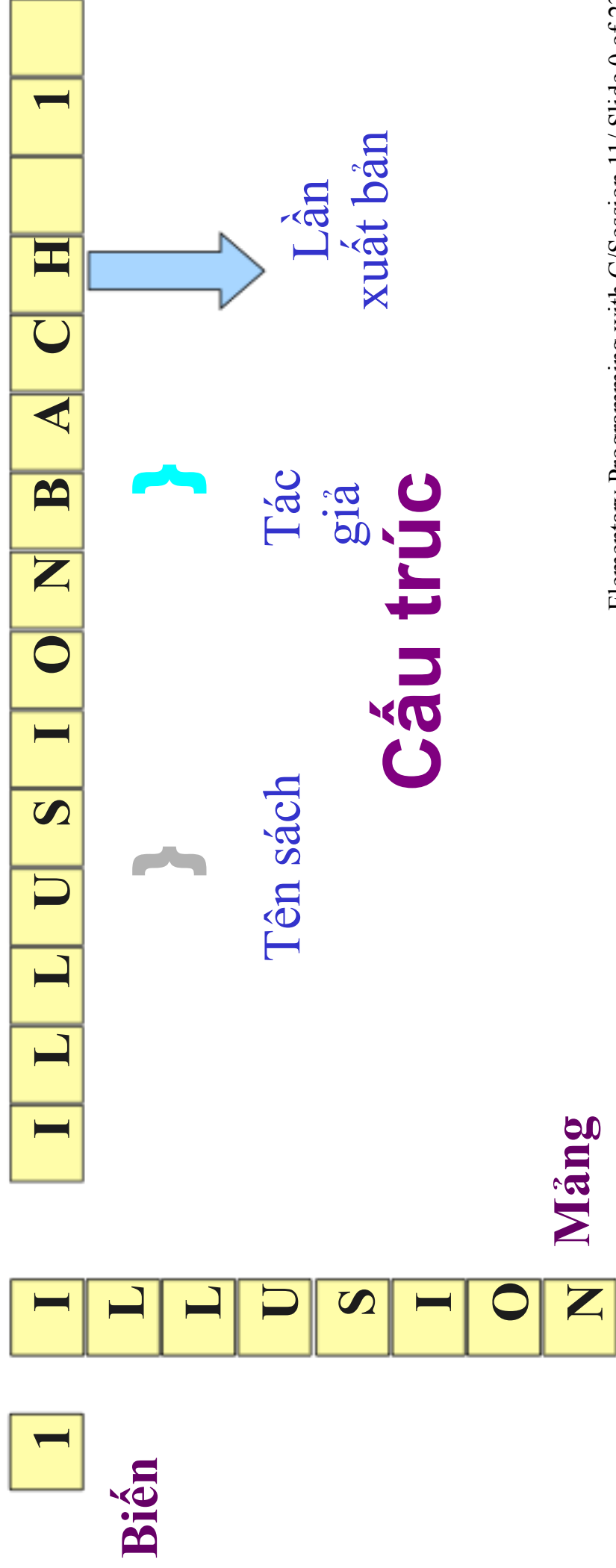


Mục tiêu - 2

- Con trỏ cấu trúc
- Cách truyền tham số kiểu con trỏ cấu trúc
- Tìm hiểu từ khóa typedef
- Sắp xếp mảng bằng phương pháp Bubble sort và Insertion sort.

Cấu Trúc

- Một cấu trúc bao gồm các mẫu dữ liệu, không nhất thiết cùng kiểu, được nhóm lại với nhau.
- Một cấu trúc có thể bao gồm nhiều mẫu dữ liệu như vậy.





Định Nghĩa Cấu Trúc

- Việc định nghĩa cấu trúc sẽ tạo ra kiểu dữ liệu mới cho phép người dùng sử dụng chúng để khai báo các biến kiểu cấu trúc .
- Các biến trong cấu trúc được gọi là các **phần tử của cấu trúc** hay **thành phần của cấu trúc**
- Ví dụ:

```
struct cat {  
    char bk_name [25];  
    char author [20];  
    int edn;  
    float price;  
};
```



Khái Báo Biến Cấu Trúc

- Khi một cấu trúc đã được định nghĩa, chúng ta có thể khai báo một hoặc nhiều biến kiểu này.
- Ví dụ: **struct cat books1;**
- Câu lệnh này sẽ dành đủ vùng nhớ để lưu trữ tất cả các mục trong một cấu trúc.

```
struct cat {  
    char bk_name[25];  
    char author[20];  
    int edn;  
    float price;  
} books1, books2;
```

Cách

khác struct cat books1,
books2;

hoặc

struct cat books1;
struct cat books2;

Truy Cập Phần Tử của Cấu Trúc

- Các phần tử của cấu trúc được truy cập thông qua việc sử dụng toán tử chấm (.), toán tử này còn được gọi là toán tử thành viên - membership.
- Cú pháp:
`structure_name.element_name`
- Ví dụ:
`scanf("%s", books1.bk_name);`

Khởi Tạo Cấu Trúc

- Giống như các biến khác và mảng, các biến kiểu cấu trúc có thể được khởi tạo tại thời điểm khai báo

```
struct employee
{
    int no;
    char name [20];
};
```

- Các biến **emp1** và **emp2** có kiểu **employee** có thể được khai báo và khởi tạo như sau:

```
struct employee emp1 = {346, "Abraham"};
struct employee emp2 = {347, "John"};
```




Câu Lệnh Gán Sử Dụng Các Cấu Trúc - 1

- Có thể sử dụng câu lệnh gán đơn giản để gán giá trị của một biến cấu trúc cho một biến khác có cùng kiểu
- Chẳng hạn, nếu **books1** và **books2** là các biến cấu trúc có cùng kiểu, thì câu lệnh sau là hợp lệ
books2 = books1;



Câu Lệnh Gán Sử Dụng

Các Cấu Trúc - 2

- Trong trường hợp không thể dùng câu lệnh gán trực tiếp, thì có thể sử dụng hàm tạo sẵn `memcpy()`
- Cú pháp:
`memcpy (char * destn, char &source, int nbytes);`
- Ví dụ:
`memcpy (&books2, &books1, sizeof(struct cat));`

Cấu Trúc Lồng Trong Cấu Trúc

- Một cấu trúc có thể lồng trong một cấu trúc khác. Tuy nhiên, một cấu trúc không thể lồng trong chính nó.

```
struct issue {  
    char borrower [20];  
    char dt_of_issue[8];  
    struct cat books;  
}issl;
```

- Việc truy cập vào các phần tử của cấu trúc này tương tự như với cấu trúc bình thường khác,
`issl.borrower`
- Để truy cập vào phần tử của cấu trúc cat là một phần của cấu trúc issl, `issl.books.author`



Truyền tham số kiểu cấu trúc

- Tham số của hàm có thể là một cấu trúc.
- Là một phương tiện hữu dụng khi muốn truyền một nhóm các thành phần dữ liệu có quan hệ logic với nhau thông qua một biến thay vì phải truyền từng thành phần một
- Kiểu của tham số thực phải trùng với kiểu của tham số hình thức.



Mảng Cấu Trúc

- Một áp dụng thường gặp là mảng cấu trúc
- Một kiểu cấu trúc phải được định nghĩa trước, sau đó một biến mảng có kiểu đó mới được khai báo
- Ví dụ: `struct cat books[50];`
- Để truy cập vào thành phần author của phần tử thứ tư của mảng books:
`books[4].author`

- Mảng cấu trúc được khởi tạo bằng cách liệt kê danh sách các giá trị phần tử của nó trong một cặp dấu móc

- Ví dụ:

```
struct unit {  
    char ch;  
    int i;  
};  
struct unit series [3] =  
    {{ 'a', 100 }, { 'b', 200 }, { 'c', 300 } };
```

Con Trỏ Đến Cấu Trúc

- Con trỏ cấu trúc được khai báo bằng cách đặt dấu * trước tên của biến cấu trúc.
- Toán tử -> được dùng để truy cập vào các phần tử của một cấu trúc sử dụng một con trỏ

- Ví dụ: `struct cat *ptr_bk;`

```
ptr_bk = &books;  
printf("%s", ptr_bk->author);
```

- Con trỏ cấu trúc được truyền vào hàm, cho phép hàm thay đổi trực tiếp các phần tử của cấu trúc.

Từ Khóa `typedef`

- Một kiểu dữ liệu có thể được định nghĩa bằng cách sử dụng từ khóa `typedef`
- Nó không tạo ra một kiểu dữ liệu mới, mà định nghĩa một tên mới cho một kiểu đã có.
- Cú pháp: `typedef type name;`
- Ví dụ: `typedef float deci;`
- `typedef` không thể sử dụng với *storage classes*

Sắp xếp mảng

- Sắp xếp liên quan đến việc thay đổi vị trí các phần tử theo thứ tự xác định như tăng dần hay giảm dần
- Dữ liệu trong mảng sẽ dễ dàng tìm thấy hơn nếu mảng được sắp xếp
- Hai phương pháp sắp xếp mảng được trình bày: Bubble Sort và Insertion Sort
- Trong phương pháp Bubble sort, việc so sánh bắt đầu từ phần tử dưới cùng và phần tử có giá trị nhỏ hơn sẽ chuyển dần lên trên (nổi bọt)
- Trong phương pháp Insertion sort, mỗi phần tử trong mảng được xem xét, và đặt vào vị trí đúng của nó giữa các phần tử đã được sắp xếp





Bubble Sort - tt

```
#include <stdio.h>

void main() {
    int i, j, temp, arr_num[5]={23, 90, 9, 25, 16};
    clrscr();
    for(i=3; i>=0; i--) /* Tracks every pass */
        for(j=4; j>=4-i; j--) {
            /* Compares elements */
            if(arr_num[j]<arr_num[j-1])
            {
                temp=arr_num[j];
                arr_num[j]=arr_num[j-1];
                arr_num[j-1]=temp;
            }
        }
    }
}
```

Contd.....

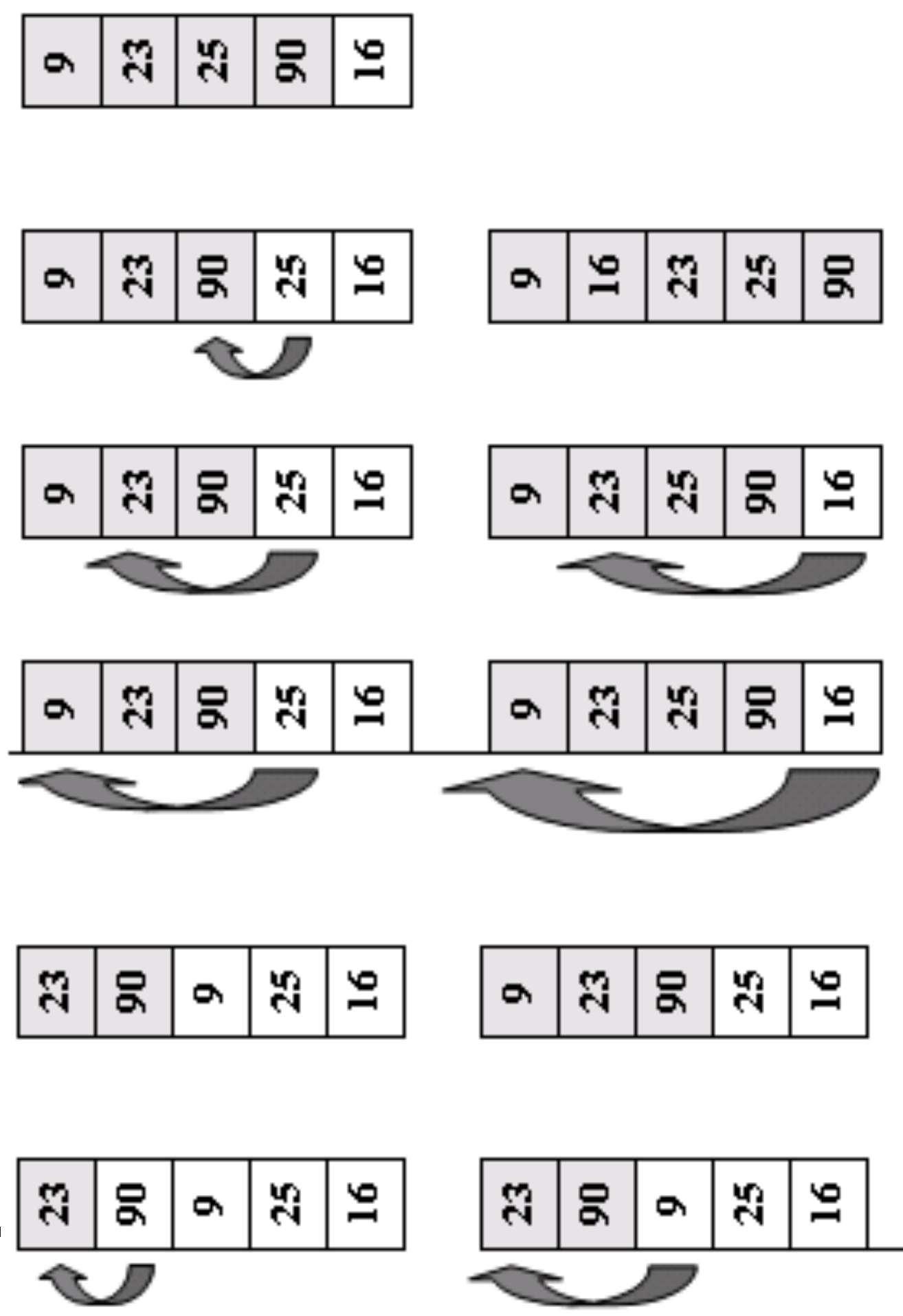


Bubble Sort - tt

```
printf("\nThe sorted array");  
for(i=0;i<5;i++)  
    printf("\n%d", arr_num[i]);  
  
    getch();  
}
```



Insertion Sort





Insertion Sort - tt

```
#include<stdio.h>
void main() {
    int i, j, arr[5] = { 23, 90, 9, 25, 16 };
    char flag;
    clrscr();
    /*Loop to compare each element of the unsorted part of the array*/
    for(i=1; i<5; i++)
    /*Loop for each element in the sorted part of the array*/
        for(j=0, flag='n'; j<i&&flag=='n'; j++) {
            if(arr[j]>arr[i]) {
                /*Invoke the function to insert the number*/
                insertnum(arr, i, j);
                flag='y';
            }
        }
    printf("\n\nThe sorted array\n");
    for(i=0; i<5; i++)
        printf("%d\t", arr[i]);
    getch();
}
```



Insertion Sort-3

```
insertnum(int arrnum[], int x, int y) {  
    int temp;  
    /*Store the number to be inserted*/  
    temp=arrnum[x];  
    /*Loop to push the sorted part of the array  
    down from the position where the number has to  
    inserted*/  
    for(;x>y; x--) arrnum[x]=arrnum[x-1];  
    /*Insert the number*/  
    arrnum[x]=temp;  
}
```