Advanced Data types and Sorting

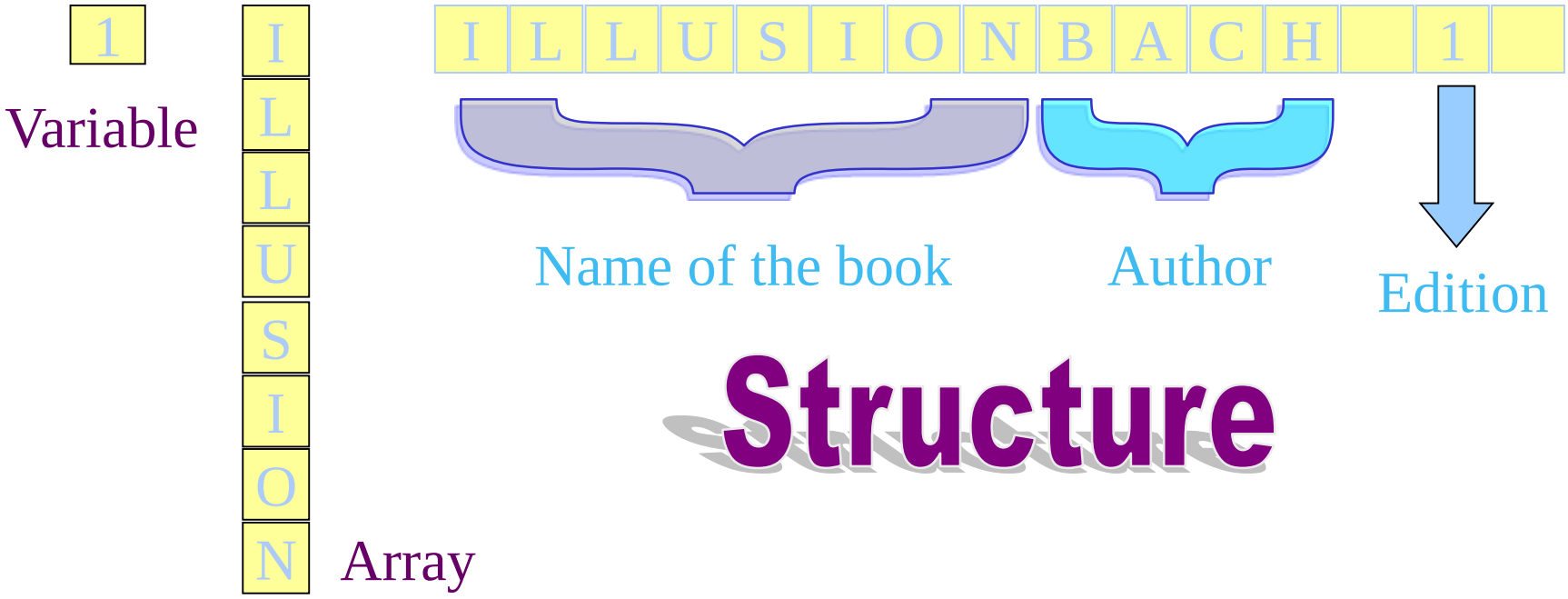
SESSION 11

Objectives

- Explain structures and their use
- Define structures, structure variables
- Explain how structure elements are accessed
- Explain how structures are initialized
- Explain how structures can be passed as arguments to functions
- Use arrays of structures
- Explain pointers to structures
- Explain the typedef keyword
- Explain Selection sort and Bubble sort methods

Structures

- A structure consists of a number of data items, which need not be of the same data type, grouped together



Define a structure

- A structure definition forms a template for creating structure variables
- The variables in the structure are called structure elements or structure members

Example:

```
struct Book  {
   char  title[25];
   char  author[20];
   int   edition;
   float price;
};
```

Declare structure variable

- Once the structure has been defined, one or more variables of that type can be declared
- Example: struct Book b1;
- The statement sets aside enough memory to hold all items in the structure

Other ways

```
struct Book   {
   char  title[25];
   char  author[20];
   int   edition;
   float price;
}b1, b2;
```

```
struct Book b1, b2;
Or
struct Book b1;
struct Book b2;
```

Initialize structure

Like variables and arrays, structure variables can be initialized at the point of declaration.

Example

```
struct Employee {
   int  no;
   char name[25];
                    };

struct Employee e1 = {346,"Abraham"};
struct Employee e2 = {347,"John"};
```

Access structure elements

- Structure elements are referenced through the use of the **dot operator** (.), also known as the **membership operator**
- Syntax:

```
structure_name.element_name
```

- Example:

```
scanf("%s", e1.name);
```

Assignment statements used with structure -1/2

- It is possible to assign the value of one structure variable to another variable of the same type using a simple assignment statement

- For example, if **b1** and **b2** are structure variables of the same type, the following statement is valid

```
b2 = b1;
```

Assignment statements used with structure - 2/2

- In cases where direct assignment is not possible, the in-built function **memcpy()** can be used

- Syntax:

```
memcpy (char* destn , char &source , int nbytes);
```

- Example:

```
memcpy (&b2 , &b1 , sizeof(struct Book));
```

Structure within structure

It is possible to have one structure within another structure. A structure cannot be nested within itself

```
struct Book {
  char  title[25];
  char  author[20];
  int   edition;
  float price;
};
```

```
struct Issue {
  char  borrower[20];
  char  dateIssue[8];
  struct Book b;
} iNote;

iNote.b.edition=1;
strcpy(iNote.borrower,"Lee");
```

Passing Structures as Arguments

- A structure variable can be passed as an argument to a function
- This facility is used to pass groups of logically related data items together instead of passing them one by one
- The type of the argument should match the type of the parameter

Array of structure

- A common use of structures is in arrays of structures
- A structure is first defined, and then an array variable of that type is declared
- Example:

```
struct Book bookList[50];
```

- To the access the variable author of the fourth element of the array bookList:

```
bookList[3].author
```

Initialization of structure array

- Structure arrays are initialized by enclosing the list of values of its elements within a pair of braces
- Example:

```
struct unit {
   char ch;
   int i;
};

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

Pointers to structures

- Structure pointers are declared by placing an asterisk(*) in front of the structure variable's name
- The **->** operator is used to access the elements of a structure using a pointer
- Example:

```
struct Book *p;
p = &b1;
printf("%s", p->author);
```

- Structure pointers passed as arguments to functions enable the function to modify the structure elements directly

typedef keyword

- A new data type name can be defined by using the keyword typedef
- It does not create a new data type, but defines a new name for an existing type
- Syntax:

```
typedef type name ;
```

- Example:

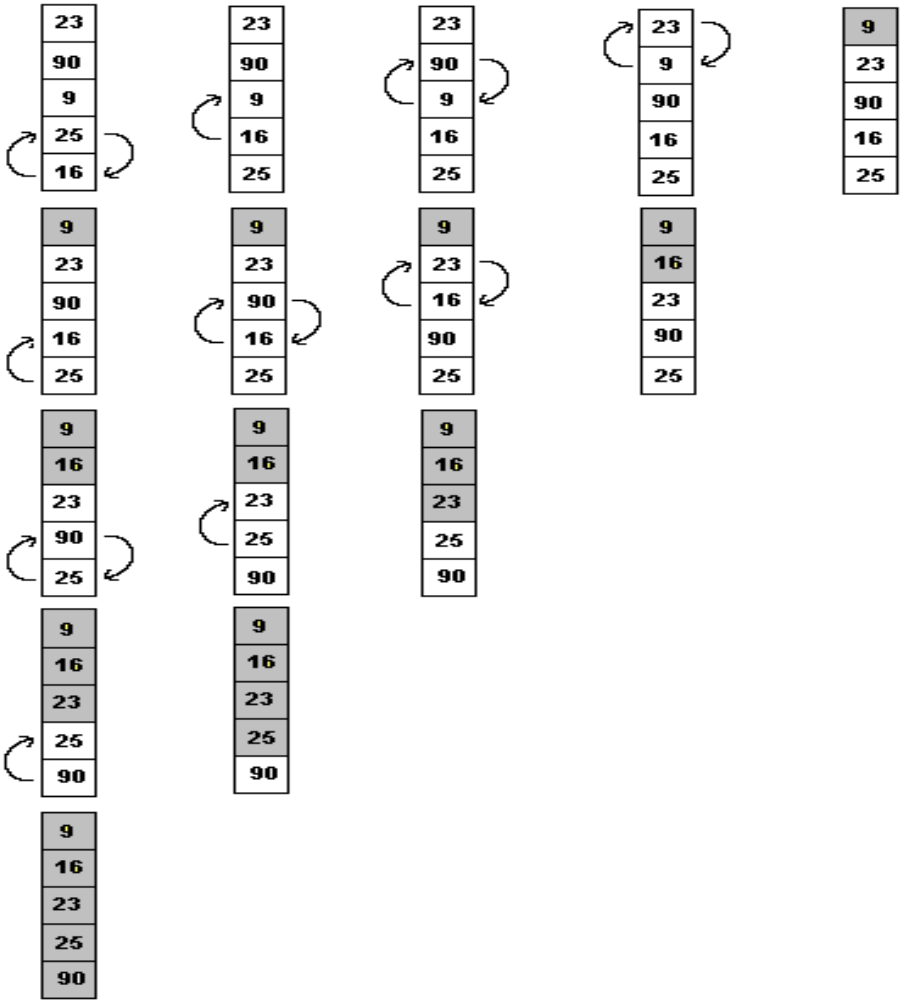
```
typedef struct Book bStruct ;
```

- typedef cannot be used with storage classes

Sorting arrays

- Sorting involves arranging the array data in a specified order such as ascending or descending
- Data in an array is easier to search when the array is sorted
- There are two methods to sort arrays – **Selection Sort** and **Bubble Sort**
- In the selection sort method, the value present in each element is compared with the subsequent elements in the array to obtain the least/greatest value
- In bubble sort method, the comparisons begin from the bottom-most element and the smaller element bubbles up to the top

Bubble sort 1/2



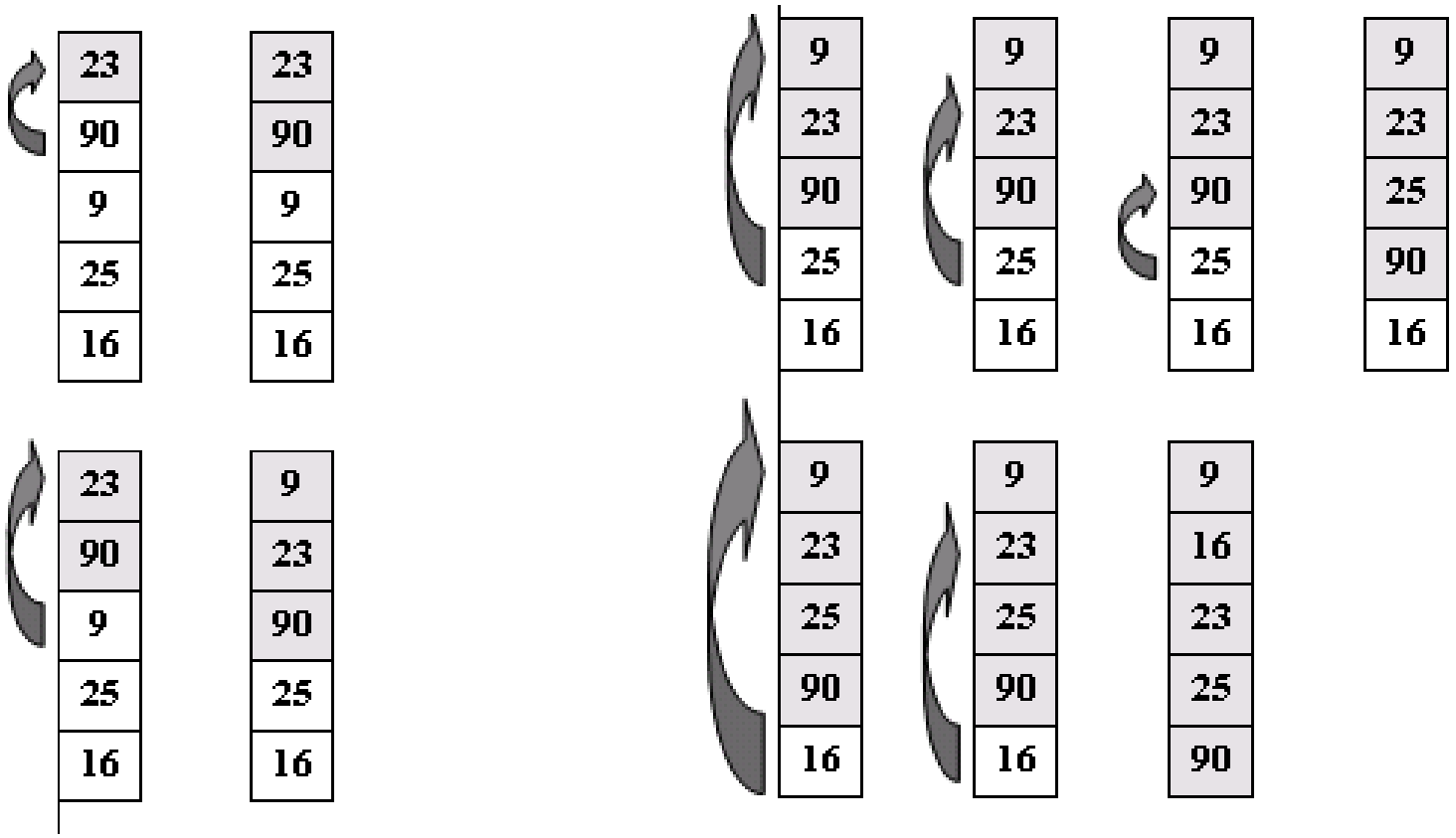
Bubble sort 2/2

```
void main()
{
  int i, j, temp, arr_num[5] = { 23, 90, 9, 25, 16};

  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;
        }
    }

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

Insert sort 1/3



Insert sort 2/3

```
void main(){
    int i, j, arr[5] = { 23, 90, 9, 25, 16 };
    char flag;
    /*Loop to compare each element of 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]);
}
```

Insert sort 3/3

```
void 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;
}
```