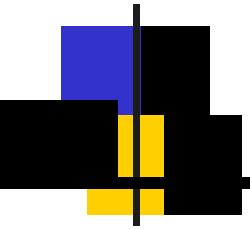
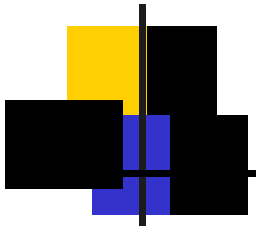


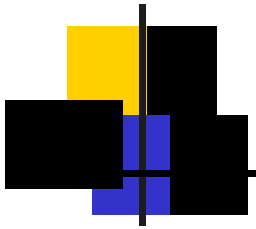
Advanced Data types and Sorting

Session 11



Objectives - 1

- Explain structures and their use
- Define structures
- Declare structure variables
- Explain how structure elements are accessed
- Explain how structures are initialized
- Explain how assignment statements are used with structures
- Explain how structures can be passed as arguments to functions
- Use arrays of structures
- Explain the initialization of structure arrays

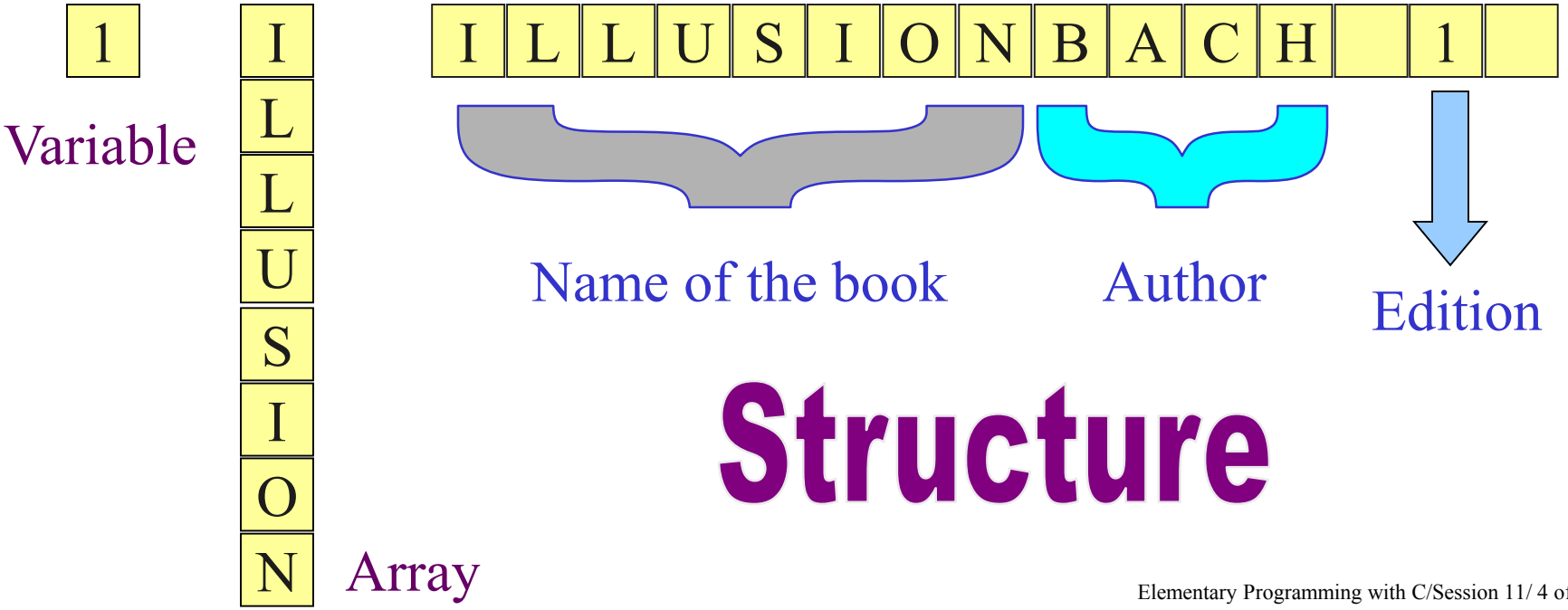


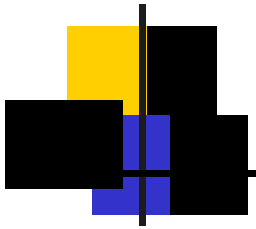
Objectives - 2

- Explain pointers to structures
  Explain how structure pointers can be passed as arguments to functions
- Explain the typedef keyword
- Explain array sorting with the 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
- The structure could hold as many of these items as desired





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

Declaring Structure Variables

- Once the structure has been defined, one or more variables of that type can be declared
- Example: **struct cat books1;**
- The statement sets aside enough memory to hold all items in the structure

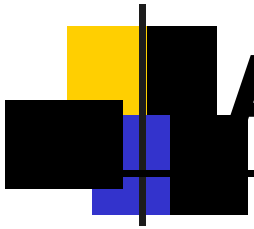
Other ways

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

```
struct cat books1, books2;
```

or

```
struct cat books1;
struct cat books2;
```



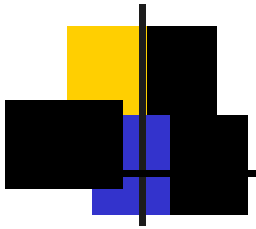
Accessing Structure Elements

- 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", books1.bk_name);
```



Initializing Structures

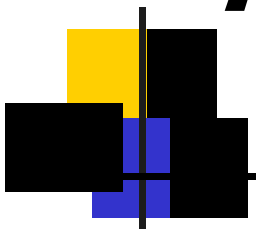
- Like variables and arrays, structure variables can be initialized at the point of declaration

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

- Variables **emp1** and **emp2** of the type **employee** can be declared and initialized as:

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

Assignment Statements Used with Structures-1

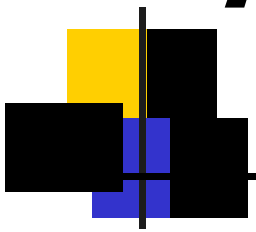


- It is possible to assign the values of one structure variable to another variable of the same type using a simple assignment statement

- For example, if **books 1** and **books2** are structure variables of the same type, the following statement is valid

```
books2 = books1;
```

Assignment Statements Used with Structures - 2

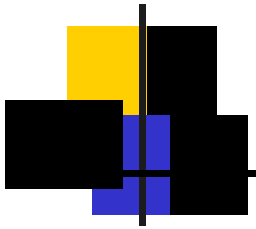


- In cases where direct assignment is not possible, the in-built function **memcpy()** can be used
- Syntax:

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

- Example:

  memcpy (&books2, &books1, sizeof(struct cat));



Structures within Structures

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

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

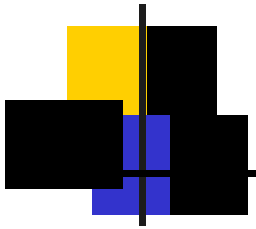
- To access the elements of the structure the format will be similar to the one used with normal structures,

```
issl.borrower
```

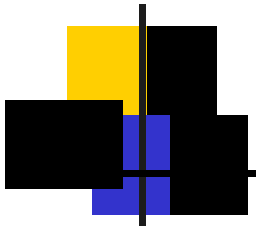
- To access elements of the structure cat, which is a part of another structure issue,

```
issl.books.author
```

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 Structures

- 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 cat books[50];
```

- To the access the variable author of the fourth element of the array **books**:

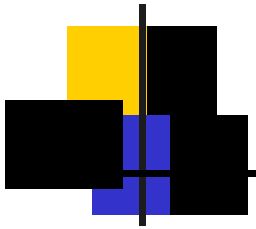
```
books[4].author
```

Initialization of Structure Arrays

- 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}
      };
```

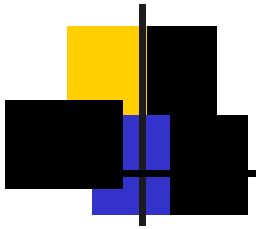


Pointers to Structures

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

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



The typedef keyword

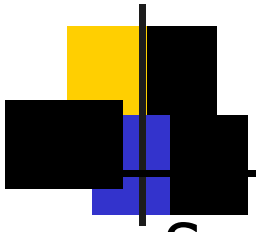
- A new data type name can be defined by using the keyword typedef
- It does not create a new data type, but defines a new name for an existing type
- Syntax:

```
typedef type name;
```

- Example:

```
typedef float deci;
```

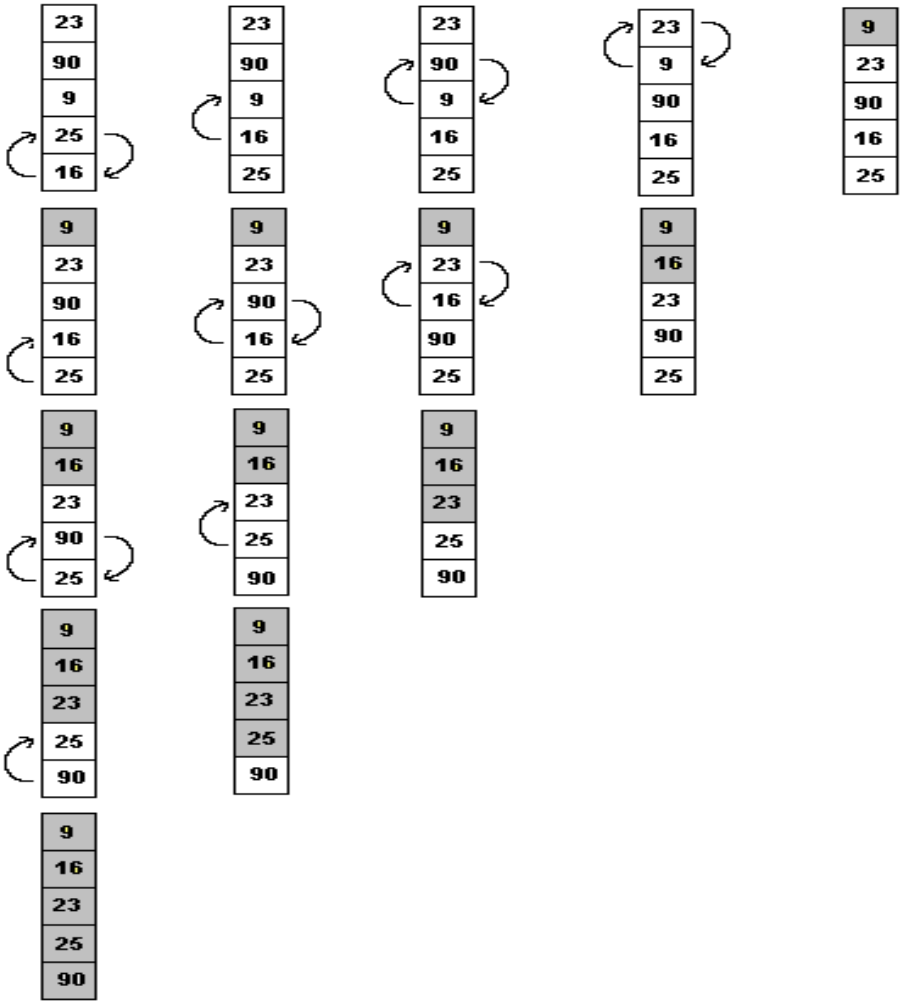
- typedef cannot be used with storage classes



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



Bubble Sort-2

Example

```
#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.....

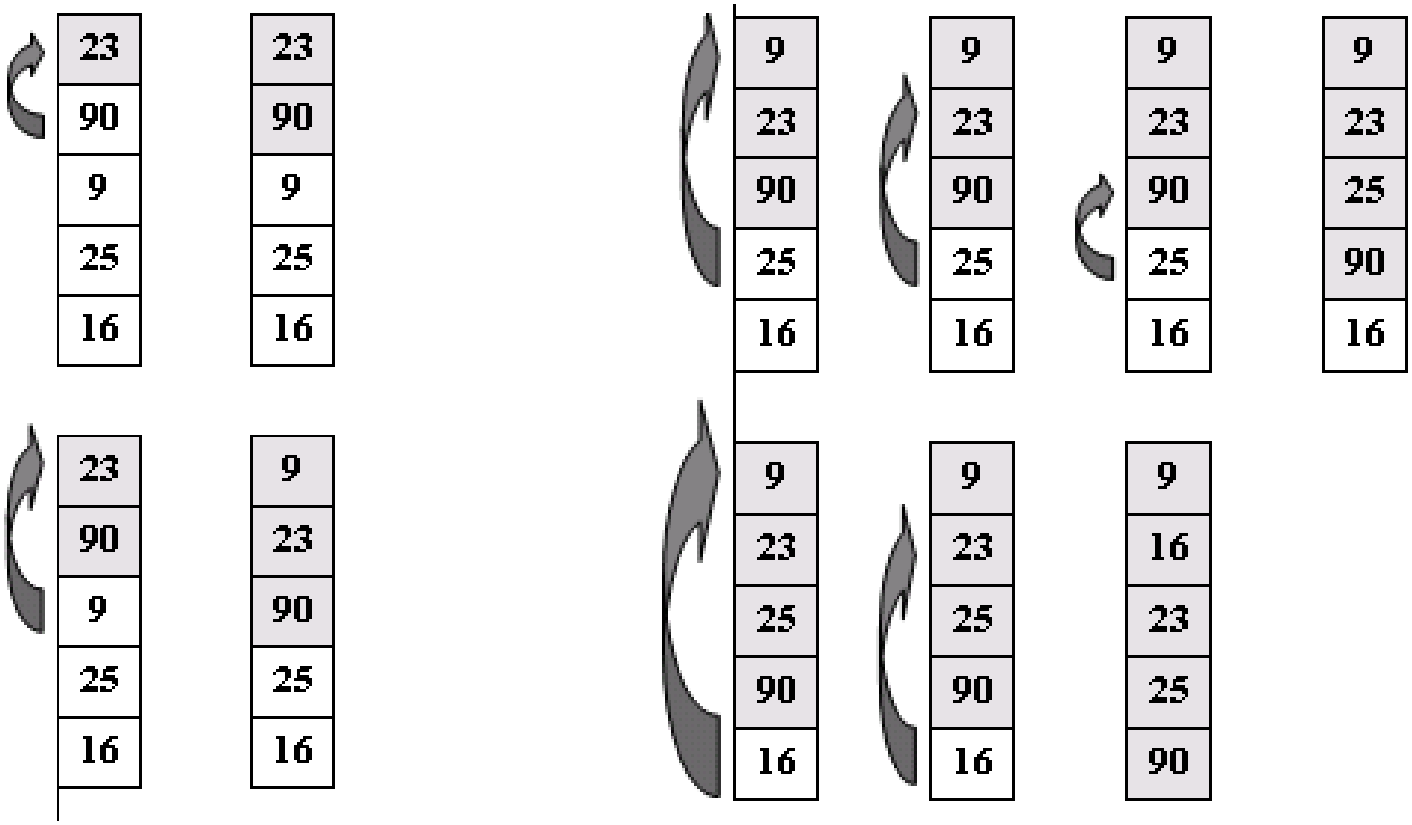
Bubble Sort-3

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

    getch();
}
```

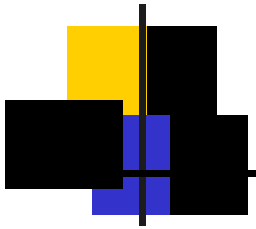
Example

Insertion Sort-1



Insertion Sort-2

```
#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;
}
```