ABES Engineering College
Estd. 2000
College Code-032

Data Structure Training

Structure & Union

Department of CSE, CS & IT
ABES Engineering College, Ghaziabad

Data Structure Training

(Structure)

INTRODUCTION

- ✓ Arrays are used for the storage of **Homogeneous data**.
- ✓ Hence we have **user defined data types** likes structures, unions, enumerations to store data with different types
- ✓ One of the similarities between arrays and **structures** is that both of them contain a finite number of elements. Thus, array types and structure types are collectively known as aggregate types.
- ✓ **Unions** are similar to structures in all aspects except the manner in which their constituent elements are stored.
- ✓ In structures, separate memory is allocated to each element, while in unions all the elements share the same memory.

Data Structure Training

(Structure)

STRUCTURES

A **structure** is a collection of variables under a single name and provides a convenient way of grouping several pieces of related information together.

It can be used for the storage of heterogeneous data.

Three important tasks of working with structures:

- ❑ Defining a structure type i.e. creating a new type.
- ❑ Declaring variables and constants (i.e. **objects**) of the newly created type.
- ❑ Using and performing operations on the objects of the structure type.

Data Structure Training
(Structure)

DEFINING A STRUCTURE

The general form of **structure type definition** is:

[storage_class_specifier][type_qualifier] ***struct*** *[structure_tag_name]*
{
type member_name1;
type member_name2;
………
} *[variable_name];*

Data Structure Training

(Structure)

Important points about structure definition:

1. The terms enclosed within the square brackets are optional and might not be present in a structure definition statement. But the term in **BOLD** is mandatory.
2. A structure definition consists of the keyword **struct** followed by an optional identifier name, known as **structure tag-name**, and a **structure declaration-list** enclosed within the braces.

```
struct book      //←Structure tag-name is book
{
    char title [25];      //←Structure declaration-list
    char author[20];
    int pages;
    float price;
};
```

```
struct   //←Structure tag-name not present
{
    char title[25];      //←Structure declaration-list
    char author[20];
    int pages;
    float price;
};
```

Data Structure Training

(Structure)

3. The structure definition defines a **new type**, known as **structure type**. After the definition of the structure type, the keyword struct is used to declare its variables.

4. Since the tag-name of a structure is an identifier, all the rules for writing an identifier name are applicable for writing the structure tag-name.

5. The structure declaration-list consists of declarations of one or more variables, possibly of different types. The variable names declared in the structure declaration-list are known as **structure members** or **fields**.

6. Structure members can be variables of the basic types (e.g. char, int, float etc.), pointer types (e.g. char* etc.) or **aggregate type** (i.e. arrays or other structure types)

Data Structure Training

(Structure)

```
#include<stdio.h>
func();                  //←Declaration of the function func

main()
{
struct coord   ←The type struct coord is defined in the scope local to the function
{
    int x,y;
 };

struct coord pt1, pt2;← Declaring variables pt1 and pt2 of the created type struct coord
//←Other statements in the function main
//← …………………………
}

func()
{
struct coord pt3;      //←The tag name coord is not visible here
//←Other statements in the function func
}
```

Output:

Compilation errors

Data Structure Training

(Structure)

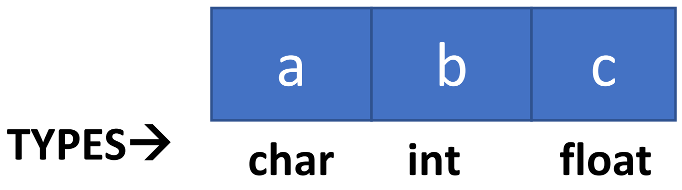
7. A structure declaration-list cannot contain a member of void type or **incomplete type** or function type. Hence, **a structure definition cannot contain an instance of itself**. However, it may contain a pointer to an instance of itself. Such a structure is known as **self referential structure**.

8. A structure definition can have infinite number of members. Practically it depends on the translation limits of the compiler.

9. It is possible to use the shorthand declaration to declare two or more structure members of the same type.

```
struct book
{
    char title [25], author[20]; // ←Shorthand declaration
    int pages;
    float price;
};
```

Data Structure Training (Structure)

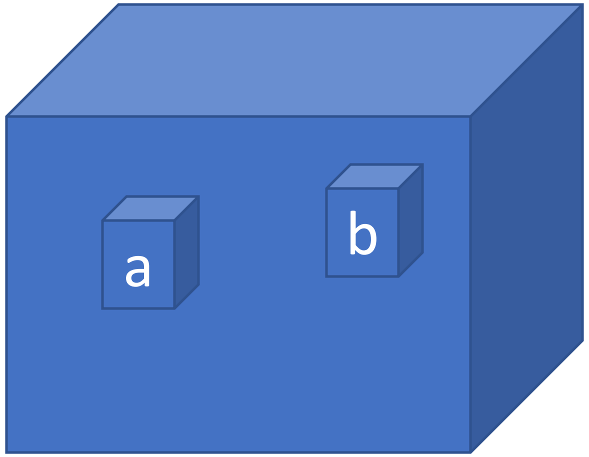



```
struct record
{       //← Structure declaration list consists of variables of  different types
char a;
int b;
float c;
};
```

INTERPRETATION OF SOME RULES

(VALID)

A STRUCTURE CAN HAVE DATA OF DIFFERENT TYPES




```
struct box
{
 struct box a;
 struct box b;
};
```

(INVALID)

A STRUCTURE CANNOT CONTAIN AN INSTANCE OF ITSELF

Data Structure Training (Structure)

A name consists of two names: first name and last name.


```
struct name
{
    char first_name[20];
    char last_name[20];
};  ←Type struct name is complete now onwards
```

A phonebook entry consists of name of a person and his mobile number.


```
struct phonebook_entry
{
    struct name person_name; //←Member of complete type struct
    char mobile_no[10];        // name can be created.
};
```

(VALID)

A STRUCTURE CAN CONTAIN MEMBERS OF OTHER COMPLETE TYPES

A node of a linked list consists of integer data and a pointer to a node.


```
struct node
{
    int data;
    struct node* ptr; //instance of itself
};
```

(VALID)

A STRUCTURE CAN CONTAIN POINTER TO ITSELF

Data Structure Training

(Structure)

10. The name of a structure member can be same as the structure tag-name without any conflict, since they can always be distinguished from the context.

11. Two different structure types may contain members of the same name without any conflict.

12. Since structure definition does not reserve any memory space for the structure members, it is not possible to initialize the structure members during the structure definition.

```
struct book
{
    char title [30]="India 2020: A Vision for the new millennium ";
    char author[20]="A P J Abdul Kalam";
    int pages=400;
    float price=225.50;
};
```

INVALID

Data Structure Training

(Structure)

13. If a structure definition does not contain a structure tag-name, the created structure type is **unnamed**. It is not possible to declare its objects after its definition.

Thus, the objects of unnamed structure type should be declared only at the time of structure definition.

```
struct
{
    char title [25];
    char author[20];
    int pages;
    float price;
} book1; //←Declaration of structure variable book1
---------------------------------------------------------------
```

```
const struct
{
    char title [25];
    char author[20];
    int pages;
    float price;
} book={"Programming C", "Anirudh", 450, 225.50};
//←Creation of qualified constant book
```

Data Structure Training

(Structure)

14. A structure type definition can optionally have a storage class specifier and type qualifiers. However, the type qualifiers and storage class specifier (**except typedef**) can only be used in a structure definition if the structure objects are also declared at the same time.

```
#include<stdio.h>
static struct point
{
    int x;
    int y;
};
main()
{
    struct point pt1;
    //←Other statements
 }
```

Output:
Compilation error "Storage class 'static' not allowed here"

Remarks:
The storage class specifiers except typedef cannot be used in a structure type definition, if the objects are not declared at the time of structure definition.

16. Since a structure definition is a statement, it must always be terminated with a semicolon.

Data Structure Training
(Structure)

DECLARING STRUCTURE OBJECTS

Variables and constants can be declared either at the time of structure definition or after the structure definition.

The general form of declaring structure objects is:
[storage class specifier] [type qualifier] **struct named_structuretype identifier_name** [=intialization_list [,…]]**;**

Important points about the structure object declaration:

1. The terms enclosed within the square brackets are optional and might not be present in a structure object declaration statement.
2. A structure object declaration consists of:
   - The keyword struct for declaring structure variables. const qualifier can also be used for declaring structure constants.
   - The tag-name of the defined structure type.
   - Comma separated list of identifiers. Initialization of a constant is must.
   - A terminating semicolon.

Data Structure Training

(Structure)

The following **structure variable declarations are valid:**

struct book c_book, algorithm_book;
const struct book c_book, algorithm_book={“C Programming”, “Anirudh”, 450,225.5};

3. The objects of the defined structure type cannot be declared without using the keyword struct.

4. Upon the declaration of a structure object, the amount of the memory space allocated to it is equal to the sum of the memory space required by all of its members.

5. The structure members are assigned memory addresses in increasing order, with the first structure member starting at the beginning address of the structure itself.

6. **Initializing members of a structure object:** Like variables and array elements, the members of a **structure object** can also be initialized at the compile time.
   - The members of a structure object can be initialized by providing an **initialization list**.
   - The order of initializers must match the order of structure members in the structure definition.

Data Structure Training (Structure)

- Type of initializer should be same as the type of corresponding structure member in structure definition.
- The number of initializers in an initialization list can be less than the number of members in a structure object. rest of the members will automatically be initialized with 0 .
- Nested structures and arrays can be initialized by using nested braces.

7. A structure object declaration can optionally have a type qualifier.

```
#include<stdio.h>
struct point
{
    int x;
    int y;
};
main()
{
    const struct point pt={2,3};
    pt.x=20;
    pt.y=40;
    //←Other statements……..
 }
```

Output:

Compilation errors “Cannot modify a constant object in function main()”

Data Structure Training (Structure)

8. A structure object declaration can optionally have a storage class specifier.

- If a structure object is declared with a storage class specifier other than typedef, the properties resulting from the storage class specifier except with respect to linkage, also apply to the members of the object.
- The structure objects declared with register storage class specifier are treated as automatic (i.e. auto) variables.

```
#include<stdio.h>
struct point
{
    int x;
    int y;
};
main()
{
    struct point pt1;
    static struct point pt2;
    printf("The coordinates of pt1 are %d,%d\n", pt1.x, pt1.y);
    printf("The coordinates of pt2 are %d,%d\n", pt2.x, pt2.y);
  }
```

Output:

The coordinates of pt1 are 9495,19125

The coordinates of pt2 are 0,0

Data Structure Training

(Structure)

OPERATIONS ON STRUCTURES

The operations that can be performed on an object (i.e. variable or constant) of a structure type are classified into two categories:

- *Aggregate operations*
- *Segregate operations*

AGGREGATE OPERATIONS

An aggregate operation treats an operand as an entity and operates on the entire operand as a whole instead of operating on its constituent members.

The following are the four aggregate operations that can be applied on an object of a structure type:

- ✓ *Accessing members of an object of a structure type*
- ✓ *Assigning a structure object to a structure variable*
- ✓ *Address of a structure object*
- ✓ *Size of a structure (i.e. either structure type or a structure object)*

Data Structure Training
(Structure)

ACCESSING MEMBERS OF AN OBJECT OF A STRUCTURE TYPE

The members of a structure object can be accessed by using:

- ✓ Direct member access operator (i.e. **.**, also known as **dot operator**).
- ✓ Indirect member access operator† (i.e. ->, also known as **arrow operator**)

Important points about the use of dot operator:

1. The dot operator accesses a structure member via structure object name while arrow operator accesses a structure member via pointer to the structure.
   The general form of using dot operator is:
   structure_object_name.structure_member_name

2. The dot operator is a binary operator

3. The first operand of the dot operator should have qualified or unqualified structure type and the second operand should be the name of a member of that type.

Data Structure Training

(Structure)

USE OF DOT OPERATOR

```
#include<stdio.h>
struct coord            //←Definition of type struct coord
{                       //← Creation of new type for 2-D coordinate
int x,y;
};

main()
{
struct coord pt1={4,5};     //←pt1 is a variable of type struct coord
const struct coord pt2={2,3};//←pt2 is a qualified constant of type struct coord
int tx, ty;
printf("Enter values of translation vector:\n");
scanf("%d %d",&tx, &ty);
printf("After translation, coordinates are:\n");
printf("Pt1 (%d,%d)\n", pt1.x+tx, pt1.y+ty);
printf("Pt2 (%d,%d)\n", pt2.x+tx, pt2.y+ty);
}
```

Output:

Enter values of translation vector:

4 2

After translation, coordinates are:

Pt1 (8,7)

Pt2 (6,5)

Data Structure Training

(Structure)

Assigning a structure Object to structure Variable

Like simple variables, a structure variable can be assigned with or initialized with another structure object of the same structure type.

```
#include<stdio.h>
struct book          //←structure definition
{
    char title[25];
    char author[20];
    int price;
};

main()
{struct book b1={“Cutting Stone”, “Abraham”, 200}:
    struct book b2=b1;
    struct book b3;
    b3=b2;     //←Assigning a structure variable to a structure variable
    printf(“%s by %s is of Rs. %d rupees\n”, b1.title, b1.author, b1.price);
    printf(“%s by %s is of Rs. %d rupees\n”, b2.title, b2.author, b2.price);
    printf(“%s by %s is of Rs. %d rupees\n”, b3.title, b3.author, b3.price);
 }
```

Output:

Cutting Stone by Abraham is of Rs. 200
Cutting Stone by Abraham is of Rs, 200
Cutting Stone by Abraham is of Rs. 200

Data Structure Training
(Structure)

Important points about the structure variable assignment:

1. A structure variable can be assigned with or initialized with a structure object of the same type. If the type of assigning or initializing structure object is not same as the type of structure variable on the left side of assignment operator, there will be a compilation error.

2. The assignment operator assigns (i.e. copies) values of all the members of the structure object on its right side to the corresponding members of the structure variable on its left side one by one. It performs **member-by-member copy.**

3. The structure assignment does not copy any padding bits.

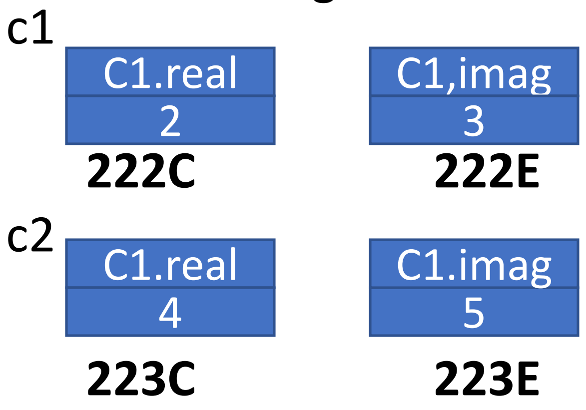
4. Due to member-by-member copy behavior of the assignment operator on the structure variables, structure objects can be passed by value to functions and can also be returned from functions.

Data Structure Training

(Structure)

ADDRESS OF A STRUCTURE OBJECT

The address-of operator (&) when applied on a structure object gives its ***base address***. It can also be used to find the addresses of the constituting members of a structure object.




```
#include<stdio.h>
struct complex
{
int real, imag;
};
main()
{
  struct complex c1={2,3};
  const struct complex c2={4,5};
  printf(“Address of c1 is %p\n”,&c1);
  printf(“Address of its real part is %p\n”,&c1.real);
  printf(“Address of its imaginary part is %p\n”,&c1.imag);
  printf(“Address of c2 is %p\n”,&c2);
  printf(“Address of its real part is %p\n”,&c2.real);
  printf(“Address of its imaginary part is %p\n”,&c2.imag); }
```

Output:

Address of c1 is 233F:222C

Address of its real part is 233F:222C

Address of its imaginary part is 233F:222E

Address of c2 is 233F:223C

Address of its real part is 233F:223C

Address of its imaginary part is 233F:223E

Data Structure Training

(Structure)

There are two different ways of storing the members of a structure variable:

→**Byte aligned:**

In this every structure member starts from a new byte (i.e. they can appear at any byte boundary).

In byte alignment, the data members are stored next to each other.

a	b		c	d			
1001	1011	1100	1001	1111	1101	1010	1000
2000	**2001**	**2002**	**2003**	**2004**	**2005**	**2006**	**2007**

```
struct type
{
    char a;
    int b;
    char c;
    float d;
}var;
```

Data Structure Training

(Structure)

→Machine-word boundary aligned:

In order to increase the performance of the code, the compiler aligns the members of a structure object with the storage boundaries whose **addresses are multiple of their respective sizes**.

```
struct type
{
    char a;
    int b;
    char c;
    int d;
}var;
```

a		b		c		d	
1001	H	1011	1100	1001	H	1111	1101
2400	**2401**	**2402**	**2403**	**2404**	**2405**	**2406**	**2407**

- ***H REPRESENTS HOLES***

The compiler fills these vacant holes with random bytes known as **padding bytes**. Thus, the process by which C compiler inserts unused bytes after the structure members to ensure that each member is appropriately aligned is called **structure padding**.

Data Structure Training

(Structure)

Important points about structure padding:

1. The members of a structure object are always stored in the **order in which they are declared.** They will never be reordered to improve the alignment and save padding.

2. The padding can only appear in-between two structure members (i.e. internal padding) or after the last structure member (i.e. trailing padding). In no case, it can appear before the first member of the structure object.

3. Whether the members of a structure object will be byte aligned or machine-word boundary aligned, depends upon the compiler, its configuration, the working environment and the underlying machine.

Data Structure Training(Structure)

Use of *sizeof* Operator on structures

When the **sizeof** operator is applied to an operand of a structure type, the result is the total number of bytes that an object of such type will occupy in the memory. The following are the important points about the use of sizeof operator on structures:

- The general form of sizeof operator is:

  sizeof expression* or *sizeof(expression)

  sizeof(type i.e. structure_type)

```
#include<stdio.h>
struct pad
{
     char a;
     int b;
     char c;
     float d;};
main()
{
struct pad var;
printf("Objects of type struct pad will take %d bytes\n", sizeof(struct pad));
printf("Structure variable var takes %d bytes\n",sizeof var);
}
```

Output:

Objects of type struct pad will take 8 bytes

Structure variable var takes 8 bytes

Data Structure Training

(Structure)

- The result of sizeof operator when applied on a structure is equal to the sum of the size of all of its members. It also **includes the space taken by internal and trailing padding.**

The pragma directive can be used to turn the structure padding on or off.

```
#include<stdio.h>
#pragma option -a
struct pad
{
    char a;
    int b;
    char c;
    float d;
};
main()
{
struct pad var;
printf("Objects of type struct pad will take %d bytes\n", sizeof(struct pad));
printf("Structure variable var takes %d bytes\n",sizeof var);
}
```

Output:

Objects of type struct pad will take 10 bytes

Structure variable var takes 10 bytes

Data Structure Training

(Structure)

Equating structure OBJECTS of the same type

The use of **equality operator** on the operands of a structure type is **not allowed** and **leads to a compilation error**.

Important points about the application of equality operator on the objects of a structure type:

- Unlike arrays, the members of a structure object may not be stored in contiguous memory locations.
- Due to the structure padding, the operation of equality operator on structures is restricted and this is a general rule. Even in byte alignment compilation error is there.
- The application of relational operators like **>=, <=, >, < and != is not allowed** on structures.
- Whether two structure objects are equal or not, can be determined by **comparing all the members of the structure objects separately**.

Data Structure Training (Structure)

```
struct pad
{
    char a;
    int b;
    float c;
};
main()
{
struct pad var1={'A', 2, 2.5}, var2={'A', 2, 2.5};
const struct pad var3={'B',3,5.5}, var4={'C',7,9.5};
printf("Checking equality of structure objects:\n");
if(var1.a==var2.a && var1.b==var2.b && var1.c==var2.c)
    printf("Structure variables are equal\n");
else
    printf("Structure variables are unequal\n");
if(var3.a==var4.a && var3.b==var4.b && var3.c==var4.c)
    printf("Structure constants are equal\n");
else
    printf("Structure constants are unequal\n");
}
```

Output:

Checking equality of structure objects:

Structure variables are equal

Structure constants are unequal

Data Structure Training (Structure)

segregate operations

Segregate operation operates on the individual members of a structure object. These are like normal objects

```
struct book
{
     char title[25];
     char author[20];
     int pages;
     float price;};

main()
{ struct book book1;
 printf("Enter title, author name, pages and price of book1:\n");
 gets(book1.title);
 gets(book1.author);
 scanf("%d %f",&book1.pages,& book1.price);
 flushall();
printf("The cost of books is increased by 10%\n\n");
book1.pages+=100;
book1.price=book1.price*110/100;
printf("In second edition: book1 has %d pages\n",book1.pages);
 printf("The second edition of book1 is of Rs. %f\n",book1.price);}
```

Output:

Enter title, author name, pages and price of book1:
The Book of Wisdom
Stephen W. K. Tan
480, 225
The cost of books is increased by 10%
In second edition: book1 has 580 pages
The second of book1 is of Rs. 247.500000

Data Structure Training

(Structure)

POINTERS TO STRUCTURES

Pointers to structures have the following advantages:

1. It **is easier to manipulate the pointers** to structures than manipulating structures themselves.

2. Passing pointer to a structure as an argument to a function is efficient as compared to passing structure to a function. It also requires less data movement as compared to passing the structure to a function.

3. Some wondrous data structures (e.g. linked lists, trees etc.) use the structures containing pointers to structures. Pointers to structures play an important role in their successful implementation.

Data Structure Training
(Structure)

Declaring pointer to a structure

The general form of declaring a pointer to a structure is:

[storage_class_specifier] [type qualifier] ***struct named_structure_type* identifier_name****[=l-value, ….];*

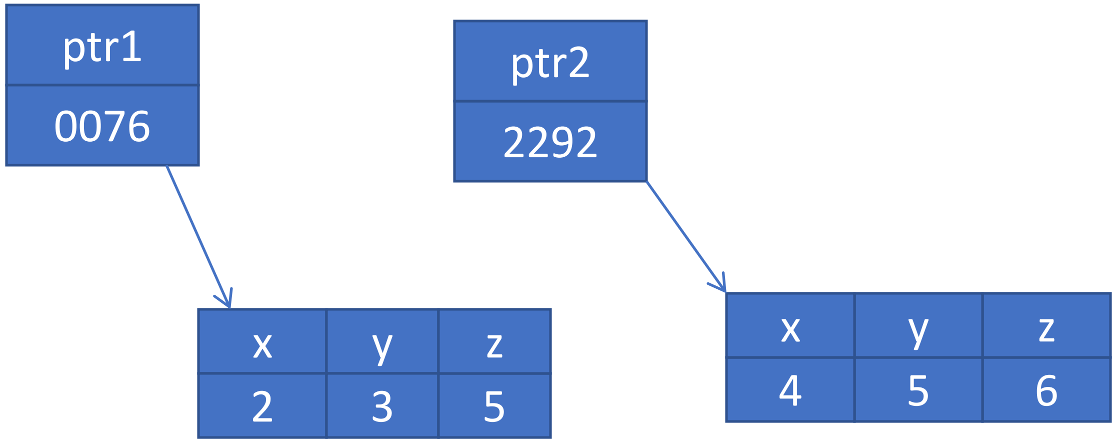
Important points about declaring a pointer to a structure:

1. The terms enclosed within the square brackets are optional and might not be present in a declaration statement.

2. A pointer to a structure type can be declared in a separate declaration statement only if the structure type is named.

3. The declared structure pointer can optionally be initialized with an l-value. The initializing l-value should be of appropriate type.

Data Structure Training (Structure)




```
#include<stdio.h>
struct coord
{
     int x,y, z;
}pt1={2,3,5}, *ptr1;
main()
{struct coord pt2={4,5,6};
struct coord *ptr2=&pt2;
ptr1=&pt1;
printf("Addresses of pt1 and pt2 are %p %p\n",&pt1,&pt2);
printf("Addresses of ptr1 and ptr2 are %p %p\n",&ptr1,&ptr2);
printf("ptr1 and ptr2 points to %p %p\n",ptr1,ptr2);
printf("Size of type (struct coord) is %d\n", sizeof(struct coord));
printf("Size of type (struct coord*) is %d\n",sizeof(struct coord*));
printf("pt1 and pt2 takes %d bytes\n", sizeof(pt1));
printf("ptr1 and ptr2 takes %d bytes\n", sizeof(ptr1));}
```

Output:

Addresses of pt1 and pt2 are 2397:0076 2397:2292
Addresses of ptr1 and ptr2 are 2397:0D38 2397:228E
ptr1 and ptr2 points to 2397:0076 2397:2292
Size of type (struct coord) is 6
Size of type (struct coord*) is 4
pt1 and pt2 takes 6 bytes
ptr1 and ptr2 takes 4 bytes

Data Structure Training (Structure)

Accessing structure members via pointer to structure

The members of a structure object can be accessed via pointer to a structure object by using one of the following two ways:

- *By using dereference or indirection operator and direct member access operator*
- *By using indirect member access operator (i.e. ->, known as* ***arrow operator****)*

Important points about accessing the structure members via pointer to the structure object:

- The general form to access a structure member via pointer to the structure object using dereference and dot operator is:
- **(*pointer_to_structure_type).structure_member_name**

- It is mandatory to parenthesize the dereference operator and structure pointer

- The members of a structure object can also be accessed via pointer to the structure object by using only one operator, known as **indirect member access operator** or **arrow operator**. The general form of such access is:
  pointer_to_structure_object->structure_member_name

Data Structure Training (Structure)

- ✓ The arrow operator consists of a hyphen (-) followed by a right arrow (>) with no space in-between.
- ✓ The expression **pointer_to_structure_object->structure_member_name** is equivalent to the expression **(*pointer_to_structure_object).structure_member_name.**

```
#include<stdio.h>
struct coord
{
int x, y;
};
main()
{
struct coord pt={2,3};
struct coord *ptr=&pt;
printf("Coordinates of Pt1 are (%d,%d)\n",(*ptr).x, (*ptr).y);
printf("Coordinates of Pt1 are (%d,%d)\n",ptr->x, ptr->y);
}
```

Output:

Coordinates of Pt1 are (2,3)
Coordinates of Pt1 are (2,3)

Data Structure Training (Structure)

ARRAY OF STRUCTURES

```
#define MAXBOOKS 10
struct book
{
     char title[30];
     char author[30];
};
main()
{
 struct book library[MAXBOOKS];    //array of structures
int count=0,i;
char ch;
while(1)
{    printf("Enter the title of the book:\t");
     gets(library[count].title);
     printf("Enter the author's name:\t");
     gets(library[count].author);
    flushall();
     count++;
}
```

Data Structure Training (Structure)

```
if(count==MAXBOOKS)
     {
          printf("Capacity full\n");
          break;    }
   else
 { printf("Do you want to enter(Y/N):\t");
          ch=getche();
          printf("\n");
          if(ch=='y'||ch=='Y')
               continue;
          else
               break;
     }
}
printf("\nFollowing are the books:\n\n");
for(i=0;i<count;i++)
{
printf("%s by %s: \n", library[i].title,  library[i].author);
}
}
```

Output:

Enter the title of the book: The law and the lawyer
Enter the author's name: M K Gandhi
Do you want to enter more(Y/N):   Y
Enter the title of the book: Rise and fall of super powers
Enter the author's name: Paul Kennedy
Do you want to enter more(Y/N):   N

Following are the books in the library:

The law and the lawyer by M K Gandhi
Rise and fall of great powers by Paul Kennedy

Data Structure Training (Structure)

STRUCTURES WITHIN STRUCTURES (NESTED STRUCTURES)

A structure can be nested within another structure. Nested structures are used to create complex data types. Declaration is as follows:

```
#include<conio.h>
struct name
{
    char first_name[20];
    char last_name[20];
};
struct phonebook_entry
{
    struct name person_name;   //nested structure
    char mobile_no[15];
};
```

Data Structure Training (Structure)

Important points about nested structures:

- ✓ The nested structures contain the members of other structure type. The structure types used in the structure definition should be complete.
- ✓ It is even possible to define a structure type within the declaration list of another structure type definition.

```
struct phone_entry
{
    struct name
    {
        char fnam[20];
        char lnam[20];
    } pnam;
    char mno[10];};
main()
{struct phone_entry per1={{"Anil","Kumar"}}};
printf("Enter the mobile number of %s %s\n",per1.pnam.fnam, per1.pnam.lnam);
gets(per1.mno);
printf("\nPhone book entries are:\n");
printf("%s %s:\t%s\n", per1.pnam.fnam, per1.pnam.lnam, per1.mno);}
```

Output:

Enter the mobile number of Anil Kumar
9814456767
Anil Kumar 9814456767

Data Structure Training

(Structure)

FUNCTIONS AND STRUCTURES

Three ways of passing a structure object to a function:

1. *Passing each member of a structure object as a separate argument.*
2. *Passing the entire structure object by value.*
3. *Passing the structure object by address/reference*

PASSING EACH MEMBER OF A STRUCTURE OBJECT AS A SEPARATE ARGUMENT

- A structure object can be passed to a function by passing each member of the structure object.
- The members of the structure object can be passed by value or by **passed by value or by address/reference**.

Data Structure Training (Structure)

```
#include<stdio.h>
struct complex
{
      int re;
      int im;
};
add_complex(int, int, int, int);                 →by value
mult_complex(int*,int*,int*,int*);               →by reference
main()
{
      struct complex no1, no2;
      printf("Enter the real and imaginary parts of 1st number:\t");
      scanf("%d %d",&no1.re, &no1.im);
      printf("Enter the real and imaginary parts of 2nd number:\t");
      scanf("%d %d",&no2.re, &no2.im);
      add_complex(no1.re, no1.im, no2.re, no2.im);
      mult_complex(&no1.re, &no1.im, &no2. re, &no2.im);
}
add_complex(int a, int b, int c, int d)
{-------------------------------
--------------------}
```

Data Structure Training

(Structure)

PASSING A STRUCTURE OBJECT BY VALUE

- Method of passing every member of the structure object to a function is highly inefficient, unmanageable and infeasible if the number of members in a structure object to be passed is large.
- The member-by-member copy behavior of the assignment operator when applied on structures makes it possible to pass a structure object to, and return a structure object from a function by value.
- all the members of a structure object are passed together instead of being passed individually.
- As the structure objects are passed by value, the changes made in the called function are not reflected back to the calling function.
- If the number of members in a structure object is quite large, this method involves large data movement. In such a case, the method of passing structure object via pointer will be more efficient

Data Structure Training (Structure)

```
struct point
{
    int x;
    int y;
};
reflectpoint(struct point);
main()
{
    struct point pt;
    printf(“Enter the x and y coordinates of the point:\t”);
    scanf(“%d %d”,&pt.x, &pt.y);
    reflectpoint(pt);
    printf(“The x and y coordinates of the reflected point: (%d,%d)”,pt.x, pt.y);
}
reflectpoint(struct point pt)   //structure passed
{
    int temp;
    temp= pt.x;
    pt.x=pt.y;
    pt.y=temp;
}
```

Output:

Enter the x and y coordinates of the point: 4 7

The x and y coordinates of the reflected point: (4,7)

Data Structure Training

(Structure)

PASSING A STRUCTURE OBJECT BY ADDRESS/ REFERENCE

- If the number of members in a structure object is quite large, it is beneficial to pass the structure object by address/reference.

- In this method of passing a structure object, instead of passing the entire structure object, only the address of a structure object is passed. Hence, this method of structure passing requires less data movement.

- Since a structure object is passed by address, the changes made in the called function are reflected back to the calling function.

Data Structure Training (Structure)

```
#include<stdio.h>
struct point
{
    int x;
    int y;
};
reflectpoint(struct point*);
main()
{
    struct point pt;
    printf("Enter the x and y coordinates of the point:\t");
    scanf("%d %d",&pt.x, &pt.y);
    reflectpoint(&pt);
    printf("The x and y coordinates of the reflected point: (%d,%d)",pt.x, pt.y);
}
reflectpoint(struct point* pt)   //passed by reference
{
    int temp;
    temp= pt->x;
    pt->x=pt->y;
    pt->y=temp;
}
```

Output:

Enter the x and y coordinates of the point: 4 7

The x and y coordinates of the reflected point: (4,7)

Data Structure Training (Structure)

TYPEDEF AND STRUCTURES

Storage class specifier typedef can be used for creating syntactically convenient names (i.e. aliases). Thus, it can be used to create an alias for the defined structure type so that the keyword struct is not required repeatedly to declare the structure objects.

```
#include<stdio.h>
struct name
{
    char first_name[20];
    char last_name[20];
} ;
typedef struct name name; //←typedef name is same as structure tag-name
main()
{
name person;
printf("Enter the first name and the last name of the person:\n");
scanf("%s %s",person.first_name, person.last_name);
printf("The name of the person is %s %s",person.first_name, person.last_name);
}
```

Output:

Enter the first name and the last name of the person:
Arvind Mishra
The name of the person is Arvind Mishra

Data Structure Training (Union)

UNIONS

A union is a collection of one or more variables, possibly of different types.

The only difference between structures and unions is in the terms of storage of their members. In structures, separate memory is allocated to each member, while in unions, all the members of an object share the same memory.
Operations on union objects: All the operations on union objects are applied in the same way as they are applied on the structure objects.

Important points about unions:

1. **Defining a union type:** A union type is defined in the same way as a structure type, with only difference that the keyword union is used instead of the keyword struct .

2. **Declaring union objects:** Objects of a union type can be declared either at the time of union type definition or after the union type definition in a separate declaration statement.

Data Structure Training

(Union)

The general form of declaring a union object is:

[storage class specifier] [type qualifier] ***union named_union_type identifier_name*** *[=intialization_list [,…]]****;***

Important points about a union object declaration:

1. The terms enclosed with the square brackets are optional and might not be present in a union variable declaration statement.

2. A union object declaration consists of:
   - The keyword union for declaring union variables. It can also be used in conjunction with const qualifier for declaring a union constant.
   - The tag-name of the defined union type.
   - Comma separated list of identifiers. The variables can optionally be initialized by providing initialization lists. However, the initialization of constants is must.
   - A terminating semicolon.

Data Structure Training (Union)

3. Size of a union object or union type: Upon the declaration of a union object, the amount of memory allocated to it is the amount necessary to contain its largest member.

4. Address-of a union object: The members of a union object are stored in the memory in such a way that they overlap each other. All the members of a union object start from the same memory location.

5. Initialization of a union object: Since the members of a union object share the same memory, the union object can hold the value of only one of its member at a time.

```
#include<stdio.h>
union variables
{
     char a;
     int b;
     float c;} ;
main()
{union variables var={'A', 2, 2.5};
  printf("The values of the members are %c %d %f", var.a, var.b, var.c);   }
```

Output:

Compilation error "Declaration syntax error in function main()"

Data Structure Training (Structure)

What will be the output of the following code?

```c
#include<stdio.h>
int main(void) {
struct foo {
 int a;
 int b;
} f1[2];
struct foo f2[3];
f1[0].a = 2;
f2[0].a = 3;
printf("%d", f1[0].a);
printf("%d", f2[0].b);
return 0;
```

What would be the output of the above code ? Choose the correct option.

A. 2 2

B. 2 3

C. 3 2

D. 2 0

Data Structure Training (Structure)

What will be the output of the following code?

```
#include<stdio.h>
int main(void) {
struct foo {
 int a;
 int b;
} f1[2];
struct foo f2[3];
f1[0].a = 2;
f2[0].a = 3;
printf("%d", f1[0].a);
printf("%d", f2[0].b);
return 0;
```

What would be the output of the above code ? Choose the correct option.

A. 2 2

B. 2 3

C. 3 2

D. 2 0

Data Structure Training (Structure)

What will be the output of the following code?

```c
#include<stdio.h>
int main(void) {
struct foo {
 int a;
 int b;
} f1[2];
struct foo f2[3];
f1[0].a = 2;
f2[0].a = 3;
printf("%d", f1[0].a++);
printf("%d", ++f2[0].a);
return 0;
}
```

What would be the output of the above code ? Choose the correct option.

A. 3 3

B. 3 4

C. 2 3

D. 2 4

Data Structure Training (Structure)

What will be the output of the following code?

```c
#include<stdio.h>
int main(void) {
struct foo {
 int a;
 int b;
} f1[2];
struct foo f2[3];
f1[0].a = 2;
f2[0].a = 3;
printf("%d", f1[0].a++);
printf("%d", ++f2[0].a);
return 0;
}
```

What would be the output of the above code ? Choose the correct option.

A. 3 3

B. 3 4

C. 2 3

D. 2 4

Data Structure Training (Structure)

What will be the output of the following code?

```c
#include<stdio.h>
int main(void) {
    struct books {
        int pages;
        char str[4];
    } b;
    printf("%lu", sizeof(b));
    return 0;
}
```

What would be the output of the above code ? Choose the correct option.

A. 4

B. 7

C. 8

D. Error

Data Structure Training (Structure)

What will be the output of the following code?

```
 #include<stdio.h>
int main(void) {
    struct books {
        int pages;
        char str[4];
    } b;
    printf("%lu", sizeof(b));
    return 0;
}
```

What would be the output of the above code ? Choose the correct option.

A. 4

B. 7

C. 8

D. Error

Data Structure Training (Structure)

What will be the output of the following code?

```
#include<stdio.h>
int main(void) {
    struct book {
        int pages;
        char str[3];
   } b;
    printf("%lu", sizeof(b));
    return 0;
}
```

What would be the output of the above code ? Choose the correct option.

A. 4

B. 7

C. 8

D. Error

Data Structure Training (Structure)

What will be the output of the following code?

```
#include<stdio.h>
int main(void) {
    struct book {
        int pages;
        char str[3];
    } b;
    printf("%lu", sizeof(b));
    return 0;
}
```

What would be the output of the above code ? Choose the correct option.

A. 4

B. 7

C. 8

D. Error

Data Structure Training (Structure)

What will be the output of the following code?

```c
#include<stdio.h>
int main(void) {
   struct book {
       int pages;
       char str[3];
       char str1[3];
       float price;
   } b;
   printf("%lu", sizeof(b));
   return 0;
}
```

What would be the output of the above code ? Choose the correct option.

A. 12

B. 16

C. 15

D. 10

Data Structure Training (Structure)

What will be the output of the following code?

```c
#include<stdio.h>
int main(void) {
   struct book {
       int pages;
       char str[3];
       char str1[3];
       float price;
   } b;
   printf("%lu", sizeof(b));
   return 0;
}
```

What would be the output of the above code ? Choose the correct option.

A. 12

B. 16

C. 15

D. 10

Data Structure Training (Structure)

```c
#include<stdio.h>
struct student
{
  int x;
  static int y;
};

int main()
{
  printf("%d", sizeof(struct student));
  return 0;
}
```

Consider the following C program & Choose the correct option(Assume Integer Size 8 Byte) .

A. 8

B. 16

C. Syntax Error

D. Compiler error

Data Structure Training (Structure)

```c
#include<stdio.h>
struct student
{
  int x;
  static int y;
};

int main()
{
  printf("%d", sizeof(struct student));
  return 0;
}
```

Consider the following C program & Choose the correct option(Assume Integer Size 8 Byte) .

A. 8

B. 16

C. Syntax Error

D. Compiler error

Data Structure Training (Structure)

```
struct {
  short arr[10];
  union {
    float a;
    long b;
  }x;
} y;
```

Consider the following C program & Choose the correct option(Assume short 2B,float 4B,long 8B what is Memory requirement for variable y) .

A. 32

B. 28

C. 22

D. Compiler error

Data Structure Training (Structure)

```c
struct {
  short arr[10];
  union {
    float a;
    long b;
  }x;
} y;
```

Consider the following C program & Choose the correct option(Assume short 2B,float 4B,long 8B what is Memory requirement for variable y) .

A. 32

B. 28

C. 22

D. Compiler error

Data Structure Training (Structure)

```c
#include<stdio.h>
struct student
{
  int x;
  struct student next;
};

int main()
{
  struct student temp;
  temp.x = 10;
  temp.next = temp;
  printf("%d", temp.next.x);
  return 0;
}
```

Consider the following C program & Choose the correct option.

A. 10

B. 16

C. Syntax Error

D. Compiler error

Data Structure Training (Structure)

```c
#include<stdio.h>
struct student
{
  int x;
  struct student next;
};

int main()
{
  struct student temp;
  temp.x = 10;
  temp.next = temp;
  printf("%d", temp.next.x);
  return 0;
}
```

Consider the following C program & Choose the correct option.

A. 10

B. 16

C. Syntax Error

D. Compiler error

Data Structure Training (Structure)

```c
#include<stdio.h>
union test
{
  int x;
  char arr[4];
  int y;
};

int main()
{
  union test t;
  t.x = 0;
  t.arr[1] = ‘x';
  printf("%s", t.arr);
  return 0;
}
```

Consider the following C program & Choose the correct option.

A. Nothing will be printed

B. garbage

C. x

D. Compiler error

Data Structure Training (Structure)

```
#include<stdio.h>
union test
{
  int x;
  char arr[4];
  int y;
};

int main()
{
  union test t;
  t.x = 0;
  t.arr[1] = ‘x';
  printf("%s", t.arr+1);
  return 0;
}
```

Consider the following C program & Choose the correct option.

A. Nothing will be printed

B. garbage

C. x

D. Compiler error

Data Structure Training (Structure)

```
#include<stdio.h>
struct Point
{
int x, y, z;
};
int main()
{
struct Point p1 = {.y = 0, .z = 1, .x = 2};
printf("%d %d %d", p1.x, p1.y, p1.z);
return 0;
}
```

Consider the following C program & Choose the correct option.

A. 2 0 1

B. 0 1 2

C. 1 0 2

D. Compiler error

Data Structure Training (Structure)

```c
#include<stdio.h>
struct Point
{
int x, y, z;
};
int main()
{
struct Point p1 = {.y = 0, .z = 1, .x = 2};
printf("%d %d %d", p1.x, p1.y, p1.z);
return 0;
}
```

Consider the following C program & Choose the correct option.

A. 2 0 1

B. 0 1 2

C. 1 0 2

D. Compiler error

Data Structure Training (Structure)

```c
#include <stdio.h>
struct date {
  unsigned int d;
  unsigned int m;
  unsigned int y;
};

int main()
{
  printf("Size of date is %lu bytes\n", sizeof(struct date));
  struct date dt = { 04, 01, 2021 };
  printf("Date is %d/%d/%d", dt.d, dt.m, dt.y);
}
```

What is the size of date in byte Assume that unsigned int takes 32 bit.

A. 12
B. 24
C. 4
D. Compiler error

Data Structure Training (Structure)

```c
#include <stdio.h>
struct date {
  unsigned int d;
  unsigned int m;
  unsigned int y;
};

int main()
{
  printf("Size of date is %lu bytes\n", sizeof(struct date));
  struct date dt = { 04, 01, 2021 };
  printf("Date is %d/%d/%d", dt.d, dt.m, dt.y);
}
```

What is the size of date in byte Assume that unsigned int takes 32 bit.

A. 12
B. 24
C. 4
D. Compiler error

Data Structure Training (Structure)

```c
#include <stdio.h>
struct date {
  unsigned int d : 5;
  unsigned int m : 4;
  unsigned int y;
};

int main()
{
  printf("Size of date is %lu bytes\n", sizeof(struct date));
  struct date dt = { 04, 01, 2021 };
  printf("Date is %d/%d/%d", dt.d, dt.m, dt.y);
  return 0;
}
```

What is the size of date in byte Assume that unsigned int takes 32 bit.

A. 8

B. 24

C. 12

D. Compiler error

Data Structure Training (Structure)

```c
#include <stdio.h>
struct date {
  unsigned int d : 5;
  unsigned int m : 4;
  unsigned int y;
};

int main()
{
  printf("Size of date is %lu bytes\n", sizeof(struct date));
  struct date dt = { 04, 01, 2021 };
  printf("Date is %d/%d/%d", dt.d, dt.m, dt.y);
  return 0;
}
```

What is the size of date in byte Assume that unsigned int takes 32 bit.

A. 8

B. 24

C. 12

D. Compiler error

Data Structure Training (Structure)

```c
#include <stdio.h>
struct date {
  int d : 5;
  int m : 4;
  int y;
};

int main()
{
  printf("Size of date is %lu bytes\n", sizeof(struct date));
  struct date dt = { 31, 12, 2021 };
  printf("Date is %d/%d/%d", dt.d, dt.m, dt.y);
  return 0;
}
```

What is the output when date is printed Assume that unsigned int takes 32 bit.

A. 04/01/2021

B. -1/-4/2021

C. 4/1/2021

D. Compiler error

Data Structure Training (Structure)

```
#include <stdio.h>
struct date {
  int d : 5;
  int m : 4;
  int y;
};

int main()
{
  printf("Size of date is %lu bytes\n",
    sizeof(struct date));
  struct date dt = { 31, 12, 2021 };
  printf("Date is %d/%d/%d", dt.d, dt.m, dt.y);
  return 0;
}
```

What is the output when date is printed . Assume that unsigned int takes 32 bit.

A. 04/01/2021

B. -1/-4/2021

C. 4/1/2021

D. Compiler error

Data Structure Training (Structure)

```c
#include <stdio.h>
struct date {
  int d : 5;
  int m : 4;
  int y;
};

int main()
{
  printf("Size of date is %lu bytes\n",
    sizeof(struct date));
  struct date dt = { 31, 12, 2021 };
  printf("Date is %d/%d/%d", dt.d, dt.m, dt.y);
  return 0;
}
```

What is the size of date Assume that unsigned int takes 32 bit.

A. 8

B. 12

C. 24

D. Compiler error

Data Structure Training (Structure)

```c
#include <stdio.h>
struct date {
  int d : 5;
  int m : 4;
  int y;
};

int main()
{
  printf("Size of date is %lu bytes\n", sizeof(struct date));
  struct date dt = { 31, 12, 2021 };
  printf("Date is %d/%d/%d", dt.d, dt.m, dt.y);
  return 0;
}
```

What is the size of date Assume that unsigned int takes 32 bit.

A. 8

B. 12

C. 24

D. Compiler error

Data Structure Training (Structure)

```
#include <stdio.h>
struct test1 {
  unsigned int x : 5;
  unsigned int y : 8;
};

int main()
{
  printf("Size of test1 is %lu bytes\n", sizeof(struct test1));
   return 0;
}
```

What is the size of test1 in bytes Assume that unsigned int takes 32 bit.

A. 4

B. 12

C. 24

D. Compiler error

Data Structure Training (Structure)

```c
#include <stdio.h>
struct test1 {
  unsigned int x : 5;
  unsigned int y : 8;
};
int main()
{
  printf("Size of test1 is %lu bytes\n", sizeof(struct test1));
   return 0;
}
```

What is the size of test1 in bytes Assume that unsigned int takes 32 bit.

A. 4

B. 12

C. 24

D. Compiler error

Data Structure Training (Structure)

```c
#include <stdio.h>
struct test2 {
  unsigned int x : 5;
  unsigned int : 0;
  unsigned int y : 8;
};

int main()
{
  printf("Size of test2 is %lu bytes\n", sizeof(struct test2));
  return 0;
}
```

What is the size of test2 in bytes Assume that unsigned int takes 32 bit.

A. 4

B. 8

C. 12

D. Compiler error

Data Structure Training (Structure)

```c
#include <stdio.h>
struct test2 {
  unsigned int x : 5;
  unsigned int : 0;
  unsigned int y : 8;
};

int main()
{
  printf("Size of test2 is %lu bytes\n", sizeof(struct test2));
  return 0;
}
```

What is the size of test2 in bytes Assume that unsigned int takes 32 bit.

A. 4
B. 8
C. 12
D. Compiler error

Thank You