11.1 INTRODUCTION

- A pointer is a derived data type in c.
- Pointers contains memory addresses as their values.
- A **pointer** is a variable whose value is the address of another variable, i.e., direct address of the memory location.
- Like any variable or constant, you must declare a pointer before using it to store any variable address.
- Pointers can be used to access and manipulate data stored in the memory.

Advantages

(i) Pointers make the programs simple and reduce their length.

(ii) Pointers are helpful in allocation and de-allocation of memory during the execution of the program.

   Thus, pointers are the instruments dynamic memory management.

(iii) Pointers enhance the execution speed of a program.

(iv) Pointers are helpful in traversing through arrays and character strings. The strings are also arrays of characters terminated by the null character ('\0').

(v) Pointers also act as references to different types of objects such as variables, arrays, functions, structures, etc. In C, we use pointer as a reference.

(vi) Storage of strings through pointers saves memory space.

(vii) Pointers may be used to pass on arrays, strings, functions, and variables as arguments of a function.

(viii) Passing on arrays by pointers saves lot of memory because we are passing on only the address of array instead of all the elements of an array, which would mean passing on copies of all the elements and thus taking lot of memory space.

(ix) Pointers are used to construct different data structures such as linked lists, queues, stacks, etc.

11.2 ACCESSING THE ADDRESS VARIABLE

- The operator & immediately preceding a variable returns the address of the variable associated with it.

Example

p=&quantity;

- Would assign 5000 (the location of quantity) to the variable p.
- The & operator is a address operator.
- The & operator can be used only with a simple variable or an array element.

&125 → pointing at constants

int x[10];   illegal

&x → pointing at array names

&(x+y) → pointing at expressions

- If x is an array then expression such as &x[0] is valid.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int x=125;
float p=20.345;
char a='a';
clrscr();
printf("%d is stored at addr  %u\n",x,&x);
printf("%f is stored at addr  %u\n",p,&p);
printf("%c is stored at addr  %u\n",a,&a);
getch();
}
```

DECLARING POINTER VARIABLES

Syntax

*data_type *pt_name;*

1. The * tells that the variable pt_name is a name of the pointer variable.
2. Pt_name needs a memory location.
3. Pt_name points to a variable of type data_type.

Example

*int *p;*

- Declares the variable p as a pointer variable that points to an integer data type.
- The declarations cause the compiler to alocate memory locations for the pointer variable p.

INITIALIZATION OF POINTER VARIABLES

- The process of assigning the address of a variable to a pointer variable is known as initialization.
- All uninitialized pointers will have some unknown values that will be interpreted as memory addresses.
- They may not be valid addresses or they may point to some values that are wrong.
- Once a pointer variable has been declared we can use the assignment operator to initialize the variable.

Example

1. *int q;*   2. *int q;*   3. *int x,*p=&x*

   *int *p;*   *int *p=&q*

   p=&q;

Illegal statement → *int *p=&x, x;*

- We can also define a pointer variable with an initial value to NULL or 0.

*int *p=null:*

*int *p=0;*

11.3 POINTER FLEXIBILITY

- Pointers are flexible.
- We can make the same pointer to point to different data variables in different statements.

Example

*int x, y, z, *p*

…………….

*p=&x;

…………….

*p=&y;

…………….

*p=&z;

……………

- We can also use different pointers to point to the same data variable.

Example

int x;

*int *p1=&x;*

*int *p2=&x;*

*int *p3=&x;*

………….

- With the exception of NULL and 0, no other constant value can be assigned to a pointer variable.

11.4 ACCESSING A VARIABLE THROUGH ITS POINTERS

- **We can access the value of another variable using the pointer variable.**

Steps:
- Declare a normal variable, assign the value.
- Declare a pointer variable with the same type as the normal variable.
- Initialize the pointer variable with the address of normal variable.
- Access the value of the variable by using asterisk (*****) - it is known as **dereference operator (indirection operators).**

```c
#include <stdio.h>
int main(void)
{
//normal variable
int num = 100;
//pointer variable
int *ptr;
//pointer initialization
ptr = &num;
//pritning the value
printf("value of num = %d\n", *ptr);
return 0;
}
```

EXAMPLE

```c
#include <stdio.h>
void main()
{
int x,y;
int *ptr;
x=10;
ptr=&x;
y=*ptr;
printf("Value of x is %d\n",x);
printf("%d is stored at address %u\n",x,&x);
printf("%d is stored at address %u\n",*&x, &x);
printf("%d is stored at address %u\n",*ptr,ptr);
printf("%d is stored at address %u\n",ptr,&ptr);
printf("%d is stored at address %u\n",y,&y);
*ptr=100;
printf("\nNew value of x =%d\n",x);
}
```

Output

Value of x is 10

10 is stored at address 2996846848
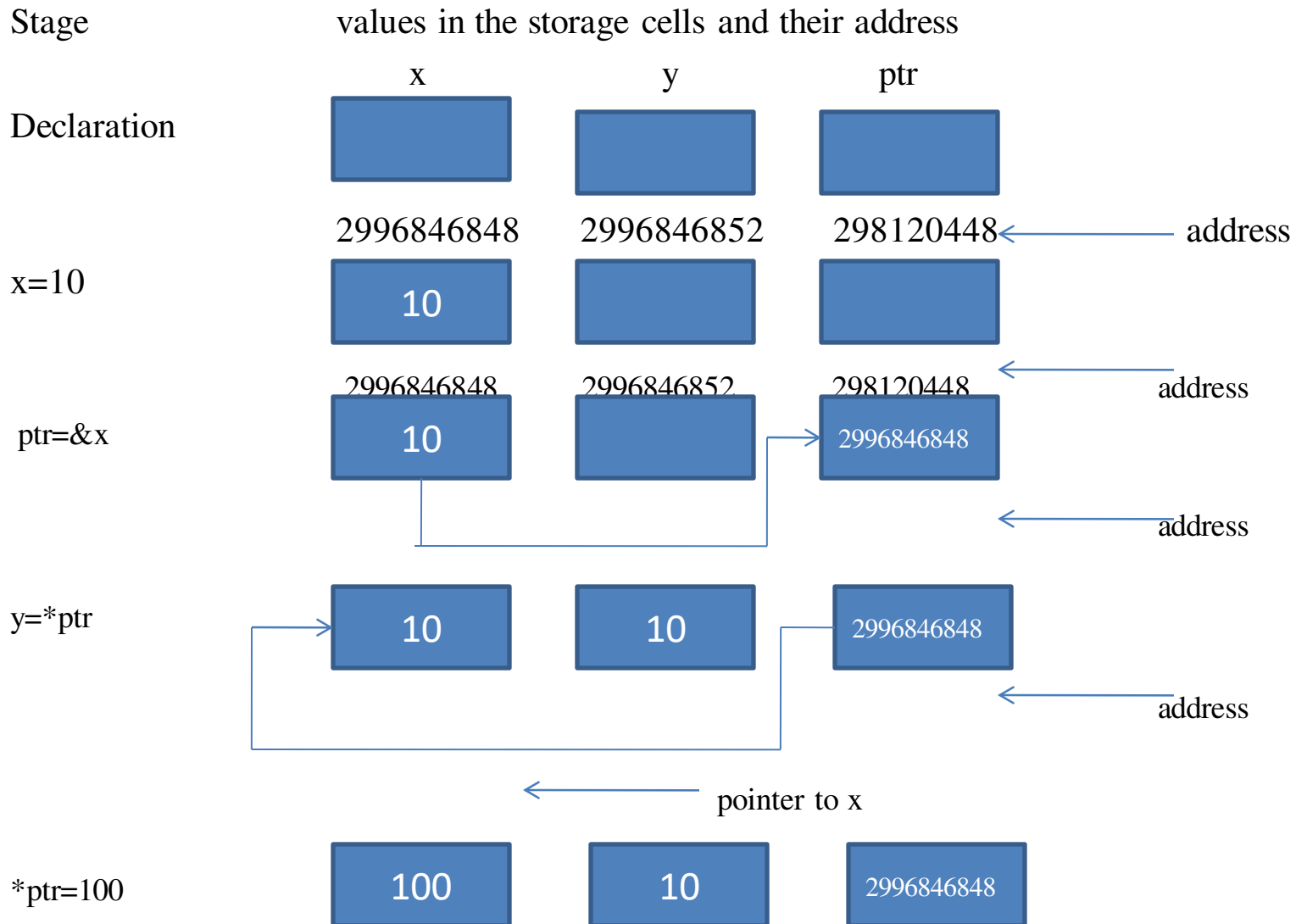10 is stored at address 2996846848
10 is stored at address 2996846848
298120448 is stored at address 2996846856

10 is stored at address 2996846852

New value of x =100

ILLUSTRATION OF POINTER EXPRESSION

Stage   values in the storage cells and their address

	x	y	ptr

Declaration

2996846848   2996846852   298120448 ← ——— address

x=10

10

2996846848   2996846852   298120448 ← ——— address

ptr=&x

10   2996846848

← ——— address

y=*ptr

10   10   2996846848

← ——— address

← ——— pointer to x

*ptr=100

100   10   2996846848

11.5 CHAIN OF POINTER

- Pointer to point to another pointer, thus creating a chain of pointer.

p2	p1	variable
address2	→ address1	→ value
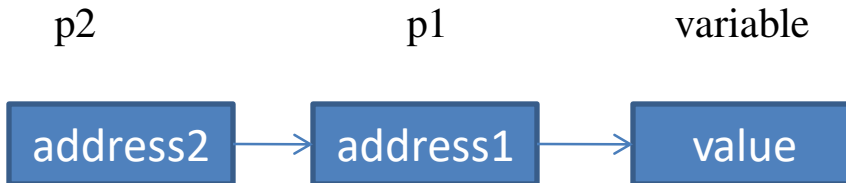
- The pointer variable p2 contains the address of the pointer variable p1, which points to the location that contains the desired value.
- This is known as multiple indirections.
- A variable that is pointer to a pointer must be declared using additional indirection operator symbol in front of the name.

    int **p2;

- The declaration tells the compiler that p2 is a pointer to a pointer of int type.

- The pointer p2 is not a pointer to an integer, but rather a pointer to an integer pointer.
- We can access the target value indirectly pointed to by pointer to a pointer by applying the indirection operator twice.

```
#include <stdio.h>
void main()
{
int x, *p1,**p2;
x=100;
p1=&x;
p2=&p1;
printf("pointer to pointer value %d",**p2);
}
```

Output

pointer to pointer value 100

11.6 POINTER EXPRESSIONS

- Pointer variables can be used in expressions

Example

- If p1 and p2 are properly declared and initialized pointers then the following statements are valid.

y= *p1 * *p2; → *y=(*p1) * (*p2)*

sum= sum + *p1;

z=5* - *p1/ *p2 →*(5* (-(*p1)))/(*p2);*

- There is blank space between / and *p2

*p2= *p2 + 10;

p1+4;

p2-2;

p1-p2;

p1++;

-p2;

sum += *p2;

- In addition to arithmetic operations , the pointer can also be compared using the relational operators.

p1>p2

p1==p2

p1 != p2

- We may not use pointers in division or multiplications.

p1/p2

p1 * p2

p1/3

Example

```c
#include <stdio.h>
int main()
{
  int a, b,*p1, *p2,x,y,z;
  a=10;
  b= 5;
  p1=&a;
  p2=&b;
  x= *p1 * *p2;
  y= *p1 + *p2;
  printf("Address of a = %u\n",a);
  printf("Address of b = %u\n",b);
  printf("a= %d\tb=%d\n",a,b);
  printf("x= %d\ty=%d\n",x,y);
  *p2= *p2 +5;
  *p1= *p1-5;
  z= *p1 * *p2 -7;
  printf("a= %d\tb=%d\n",a,b);
  printf("*p1 = %d\n",*p1);
  printf("*p2 = %d\n",*p2);
  printf("z= %d\n",z);
  return 0;
}
```

Output

Address of a = 71870892
Address of b = 71870896
a= 10   b=5
x= 50   y=15
a= 5   b=10
*p1 = 5
*p2 = 10
z= 43

11.7 POINTER INCREMENT & SCALE FACTOR

p1++;

- The pointer p1 to point to the next value of its type.
- If p1 is an integer pointer with an initial value, say 4020, then the operation p1++, the value of p1 will be 4022.
- Ie, the value increased by the length of the data type that it points to.

char	1 byte
int	2 bytes
float	4 bytes
long int	4 bytes
double	8 bytes

11.8 POINTERS AND ARRAYS

- The address of &x[0] and x is the same. It's because the variable name x points to the first element of the array.
- &x[0] is equivalent to x. And, x[0] is equivalent to *x.
- Similarly, &x[1] is equivalent to x+1 and x[1] is equivalent to *(x+1).
- &x[2] is equivalent to x+2 and x[2] is equivalent to *(x+2).
- Basically, &x[i] is equivalent to x+i and x[i] is equivalent to *(x+i).

Example 1: Pointers and Arrays

```c
#include <stdio.h>
 int main()
{
 int i, x[20], sum = 0,n;
printf("Enter the value of n: ");
scanf("%d",&n);
printf("Enter number one by one\n");
for(i = 0; i < n; ++i)
{
/* Equivalent to  scanf("%d", &x[i]);  */
scanf("%d", x+i);
 // Equivalent to sum += x[i]
 sum += *(x+i);
 }
printf("Sum = %d", sum);
return 0;
}
```

Output
Enter the value of n: 5
Enter number one by one
5
10
15
20
25
Sum = 75

Example :2

```c
#include <stdio.h>
 int main()
 {
int *p,sum,i;
 int n,x[10];
printf ("Enter the value of n\n");
 scanf("%d",&n);
printf("Enter the array elements one by one\n");
for (i=0;i<n;i++)
   scanf("%d",&x[i]);
 p=x;
printf("Elements\t Value\t Address\n");
for (i=0;i<n;i++)
{
 printf("x[%d] %d  %u\n",i,*p,p);
 sum +=*p;
 p++;
}
printf("\n Sum = %d",sum);
printf("\n address of first element (&x[0]) =
      %u",&x[0]);
printf("\n p = %u",p);
return 0;
}
```

output

Enter the value of n

5

Enter the array elements one by one

1

2

3

4

5

 Elements   Value   Address

x[0]

    1   2316341728

x[1]   2   2316341732
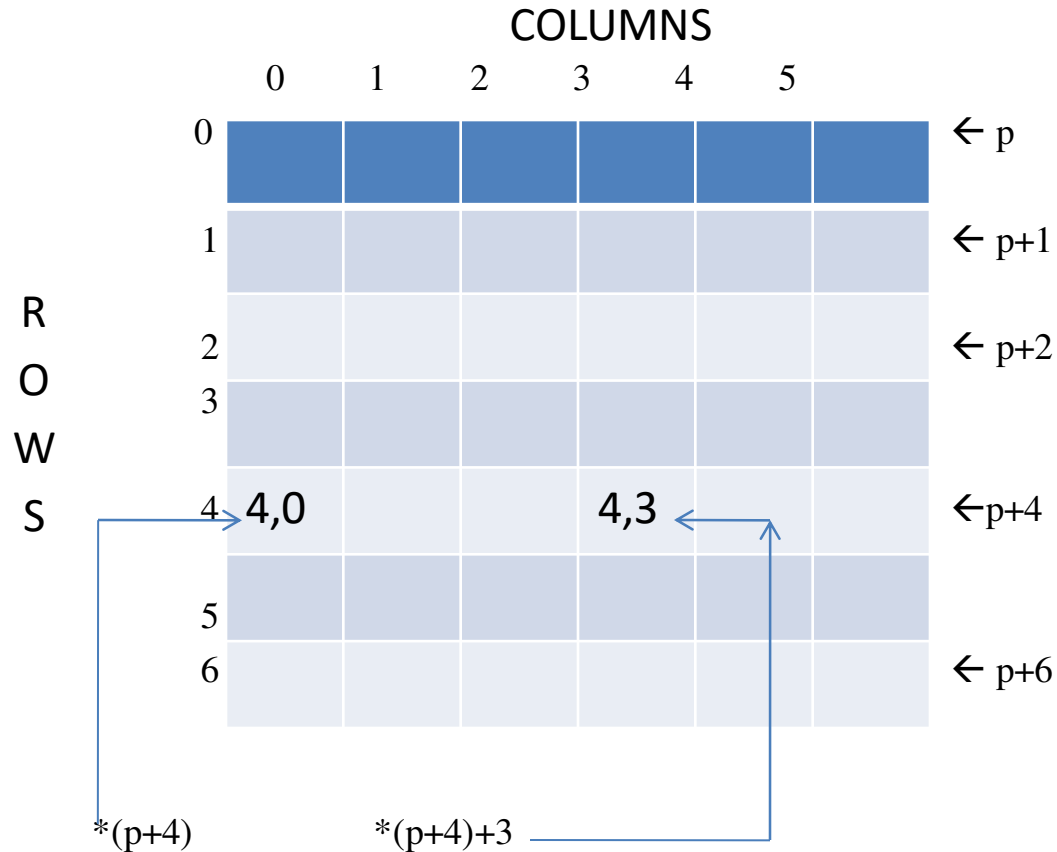
x[2]   3   2316341736

x[3]   4   2316341740

x[4]   5   2316341744

 Sum = 16
 address of first element (&x[0]) = 2316341728

p = 2316341748

- Pointers can be used to manipulate two-dimensional arrays also.
- An two-dimensional array can be represented by the pointer expression as follows
- *(*(a+i)+j) or *(*(p+1)+j)

COLUMNS

	0	1	2	3	4	5	
0							← p
1							← p+1
2							← p+2
3							
4	4,0			4,3			←p+4
5							
6							← p+6

ROWS

*(p+4)   *(p+4)+3

p → pointer to first row

p+i → pointer to ith row

*(p+i) → pointer to first element in the ith row

*(p+i) +j → pointer to jth element in the ith row

((p+i)+j) → valu stored in the ith row and jth columns.

Example

```c
#include<stdio.h>
int main()
{
  int    arr[3][4]    =    {    {11,22,33,44},
    {55,66,77,88},{11,66,77,44}};
  int i, j;
for(i = 0; i < 3; i++)
 {
printf("Address of %d th array %u \n",i , *(arr + i));
 for(j = 0; j < 4; j++)
   {
 printf("arr[%d][%d]=%d\n", i, j, *( *(arr + i) + j) );
   }
```

```c
printf("\n\n");
 }
 // signal to operating system program ran fine
    return 0;
}
```

Output

arr[2][3]=44Address of 0 th array 2692284448

arr[0][0]=11

arr[0][1]=22

arr[0][2]=33

arr[0][3]=44

Address   of 1 th array 2692284464

arr[1][0]=55

arr[1][1]=66

arr[1][3]=88

Address of 2 th array 2692284480

arr[2][0]=11

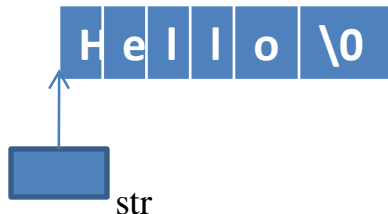arr[2][1]=66

arr[2][2]=77

arr[2][3]=44

11.9 POINTERS AND CHARACTER STRINGS

- C supports an alternate method to create strings **using pointer variables of type char.**

Example

char *str= "Hello";

- This creates a string for the literal and then stores its address in the pointer variable str.
- The pointer str now points to the first character of the string "Hello" as

str

We can also use runtime assignment for giving values to a string pointer.

char *str;
str= "hello";

```c
#include <stdio.h>
#include <string.h>
 int main ()
 {
 char name[25];
 char *ptr;
 strcpy(name,"gaccbe");
  ptr=name;
 while(*ptr !='\0')
    {
     printf("\n %c is stored at address %u",*ptr,ptr);
     ptr++;
    }
  return 0;
}
```
Output
g is stored at address 3432464000

a is stored at address 3432464001

c is stored at address 3432464002

c is stored at address 3432464003

b is stored at address 3432464004

e is stored at address 3432464005

11.10 ARRAY OF POINTERS

Example

char name[4][25];

- The name is a table containing four names, each with maximum of 25 characters.
- The total storage requirements is 75 bytes.
- The individual strings will of equal lengths.

Example

char *names[4] = {

     "Anu",

    "Banu",

     "Chandru",

    "Deepak"

    };

- Declares name to be an array of four pointers to characters, each pointer pointing to a particular name.

```c
#include <stdio.h>
 const int MAX = 4;
 int main ()
 {
char *names[] = { "Anu", "Banu", "Chandru",
     "Deepak" };
 int i = 0;
 for ( i = 0; i < MAX; i++)
  {
printf("Value of names[%d] = %s\n", i, names[i] );
}
 return 0;
}
```

Output

Value of names[0] = Anu
Value of names[1] = Banu
Value of names[2] = Chandru
Value of names[3] =Deepak

11.11 POINTERS AS FUNCTION ARGUMENTS

- Pointer as a function parameter is used to hold addresses of arguments passed during function call.

- This is also known as **call by reference**.

- When a function is called by reference any change made to the reference variable will effect the original variable.

EXAMPLE

```
#include <stdio.h>
void exchange(int *a, int *b);
int main()
{
 int m = 10, n = 20;
printf("m = %d\n", m);
printf("n = %d\n\n", n);
swap(&m, &n);
printf("After Swapping:\n\n");
printf("m = %d\n", m);
 printf("n = %d", n);
 return 0;
 }
void exchange (int *a, int *b)
{
 int temp;
temp = *a;
 *a = *b;
 *b = temp;
 }
```

Output

m = 10

 n = 20

 After Swapping:

 m = 20

 n = 10

11.12 FUNCTIONS RETURNING POINTERS

- A function can return a single value by its name or return multiple values through pointer parameters.
- A function can also return a pointer to the calling function.
- Local variables of function doesn't live outside the function.
- They have scope only inside the function.
- Hence if you return a pointer connected to a local variable, that pointer will be pointing to nothing when the function ends.

```
#include <stdio.h>
int* larger(int*, int*);
void main()
{
 int a = 10;
int b = 20;
int *p;
p = larger(&a, &b);
printf("%d is larger",*p);
}
 int* larger(int *x, int *y)
 {
 if(*x > *y)
     return x;
Else
     return y;
 }
Output
20 is larger
```

11.13 POINTERS TO FUNCTIONS

- It is possible to declare a pointer pointing to a function which can then be used as an argument in another function.

- A pointer to a function is declared as follows,

  *type (*pointer-name)(parameter)*

Example

  int (*sum)(); →legal declaration of pointer to function

  int *sum(); →This is not a declaration of pointer to function.

- A function pointer can point to a specific function when it is assigned the name of that function.

  int sum(int, int);

  *int (*s)(int, int);*

  s = sum;

- s is a pointer to a function sum.

- sum can be called using function pointer s along with providing the required argument values.

  s (10, 20);

Example

```
#include <stdio.h>
int sum(int x, int y)
{
return x+y;
}
int main( )
{
 int (*fp)(int, int);
fp = sum;
 int s = fp(10, 15);
printf("Sum is %d", s);
 return 0;
}
```

Output

25

11.14 POINTERS AND STRUCTURES

- We know that the name of an array stands for the address of its zero-th element.
- Also true for the names of arrays of structure variables.

Example

struct inventory
{
    int no;
    char name[30];
    float price;
} product[5], *ptr ;

- The name product represents the address of the zero-th element of the structure array.
- ptr is a pointer to data objects of the type struct inventory.
- The assignment

    ptr = product ;

will assign the address of product [0] to ptr.

- Its member can be access

    ptr –>name ;
    ptr –> no ;
    ptr –> price;

The symbol "–>" is called the arrow operator or member selection operator.

- When the pointer ptr is incremented by one (ptr++) :The value of ptr is actually increased by sizeof(inventory).
- It is made to point to the next record.
- We can also use the notation

    *(*ptr).no;*

- When using structure pointers, we should take care of operator precedence.
- Member operator "." has higher precedence than "*".
- ptr –> no and (*ptr).no mean the same thing.
- ptr.no will lead to error.

- The operator "–>" enjoys the highest priority among operators
- ++ptr –> no   will increment roll, not ptr.
- (++ptr) –> no   will do the intended thing.

Example

```
void main ()
{
struct book
{
char name[25];
char author[25];
int edn;
};
struct book b1 = { "Programming in C",
    "E Balagurusamy", 2 } ;
 struct book *ptr ;
 ptr = &b1 ;
printf ( "\n%s %s edition %d ", b1.name,
    b1.author, b1.edn ) ;
printf ( "\n%s %s edition  %d", ptr-
    >name,    ptr->author, ptr->edn ) ;
}
```

Output

Programming in C E Balagurusamy edition 2
Programming in C E Balagurusamy edition 2