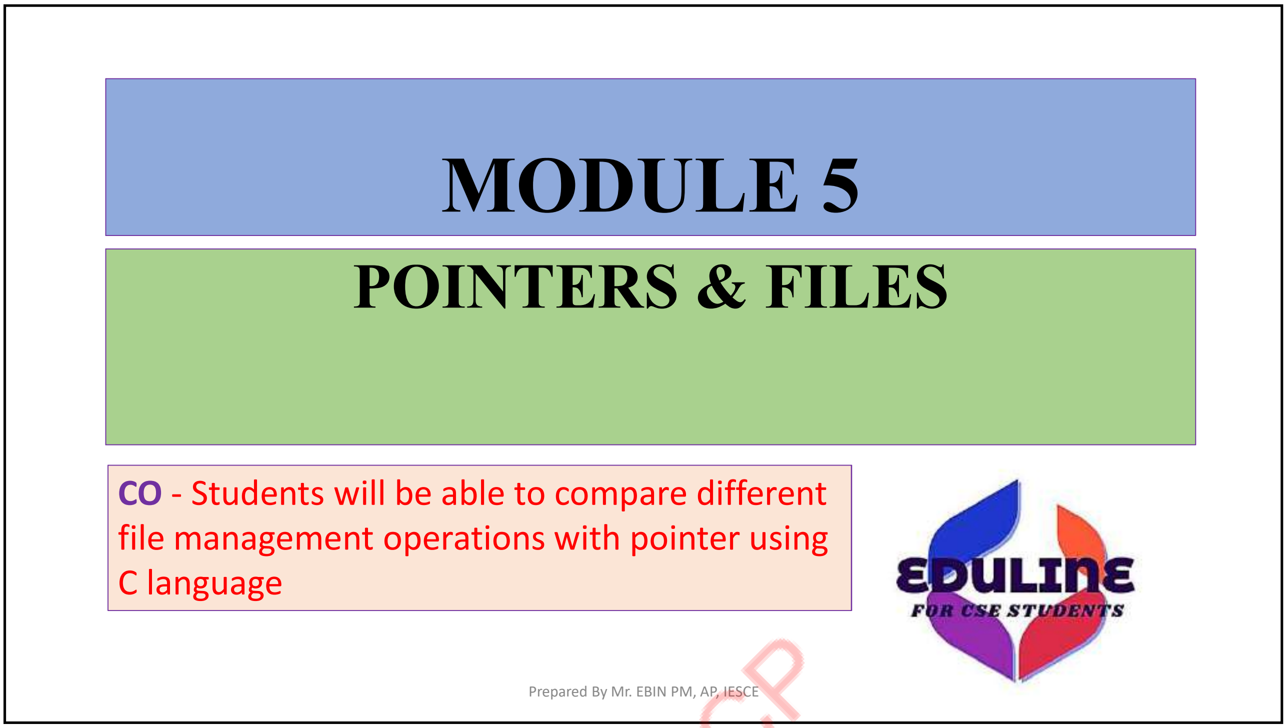
സഹായി

SFI GEC PALAKKAD

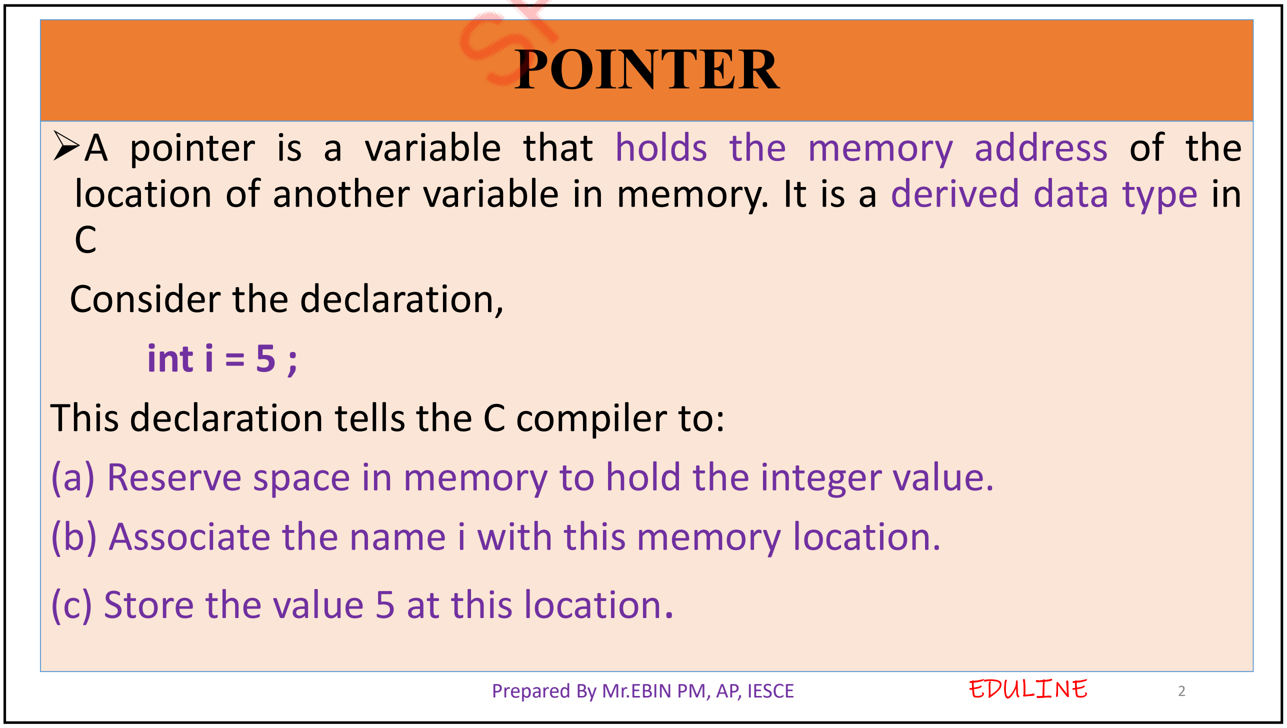


MODULE 5

POINTERS & FILES

CO - Students will be able to compare different file management operations with pointer using C language

Prepared By Mr. EBIN PM, AP, IESCE



POINTER

- A pointer is a variable that holds the memory address of the location of another variable in memory. It is a derived data type in C

Consider the declaration,

int i = 5 ;

This declaration tells the C compiler to:

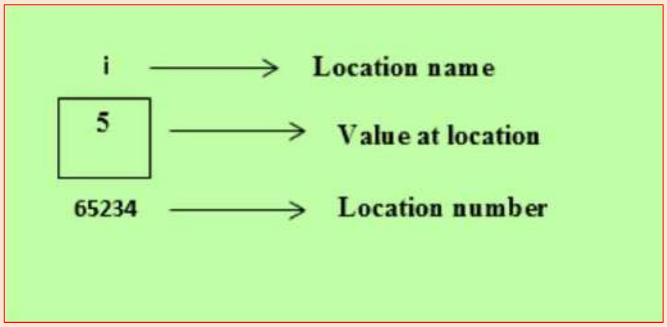
(a) Reserve space in memory to hold the integer value.

(b) Associate the name i with this memory location.

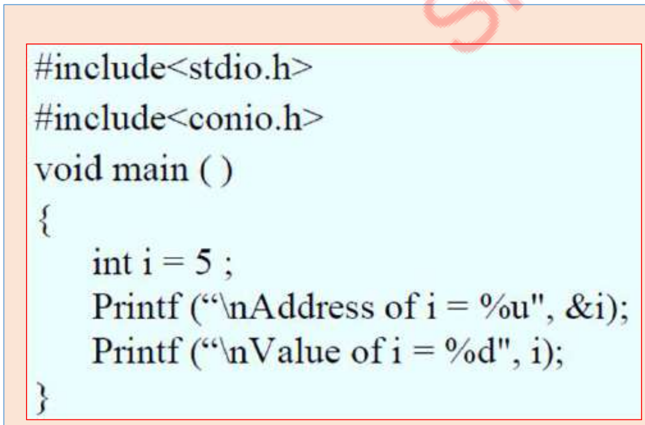
(c) Store the value 5 at this location.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 2

- We may represent i’s location in memory by the following memory map



- We see that the computer has selected memory location 65234 as the place to store the value 5.
- The important point is, i’s address in memory is a number. We can print this address number through the following program:



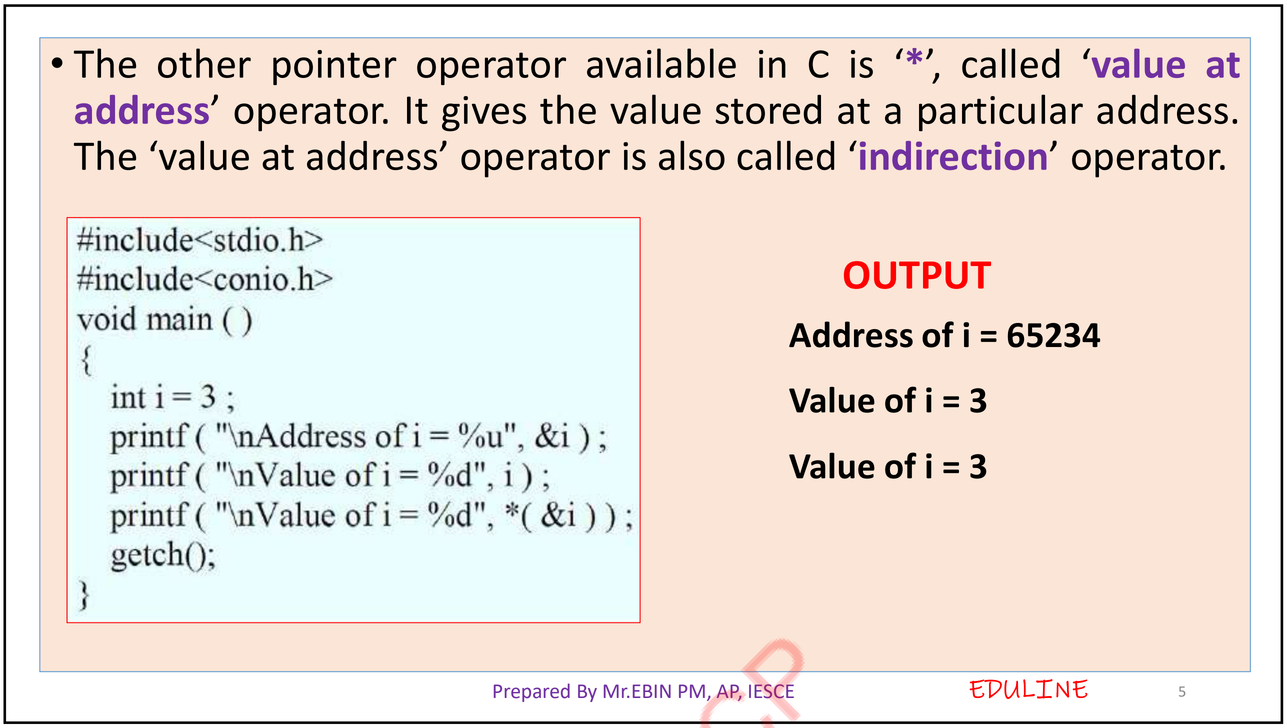
```
#include<stdio.h>
#include<conio.h>
void main ( )
{
    int i = 5 ;
    Printf (“\nAddress of i = %u", &i);
    Printf (“\nValue of i = %d", i);
}
```

OUTPUT

Address of i = 65234

Value of i = 5

- **&** used in this statement is C’s **address of** operator. The expression &i returns the address of the variable i.
- **%u** is a format specifier for printing an **unsigned integer**

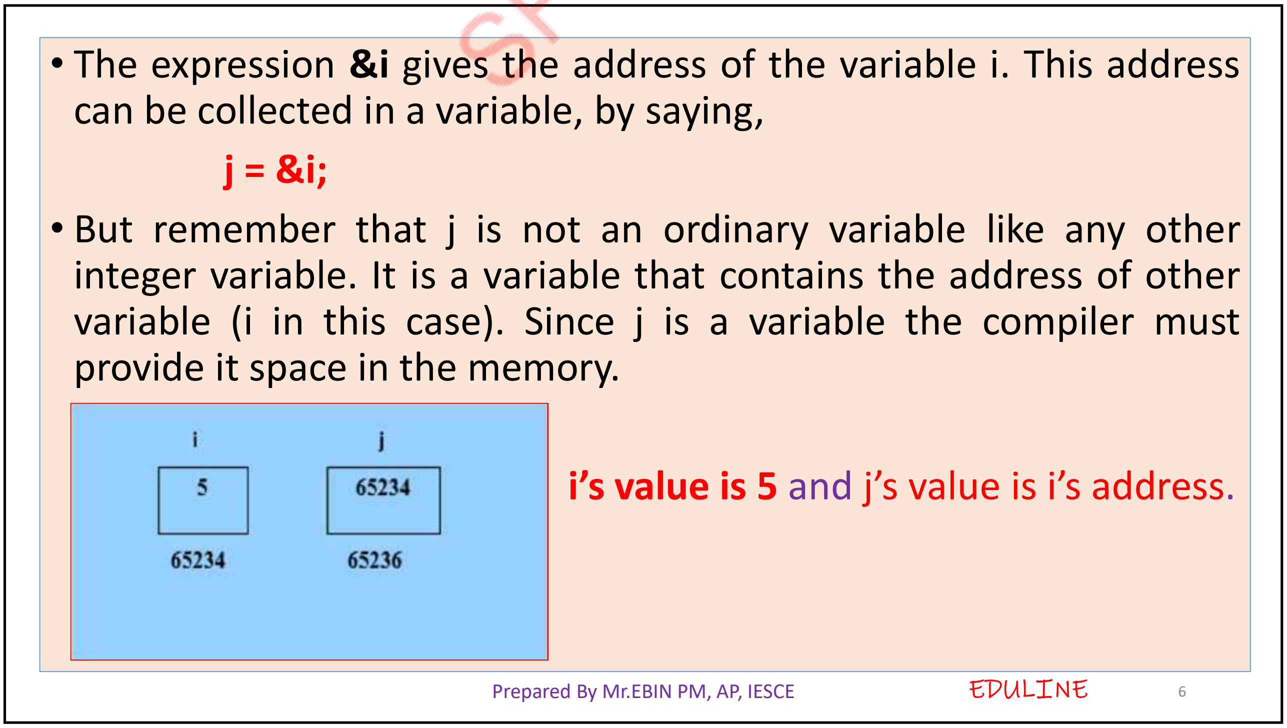


- The other pointer operator available in C is '*', called '**value at address**' operator. It gives the value stored at a particular address. The 'value at address' operator is also called '**indirection**' operator.

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
   int i = 3 ;
   printf ( "\nAddress of i = %u", &i ) ;
   printf ( "\nValue of i = %d", i ) ;
   printf ( "\nValue of i = %d", *( &i ) ) ;
   getch();
}
```

OUTPUT

```
Address of i = 65234

Value of i = 3

Value of i = 3
```



- The expression **&i** gives the address of the variable i. This address can be collected in a variable, by saying,

j = &i;

- But remember that j is not an ordinary variable like any other integer variable. It is a variable that contains the address of other variable (i in this case). Since j is a variable the compiler must provide it space in the memory.

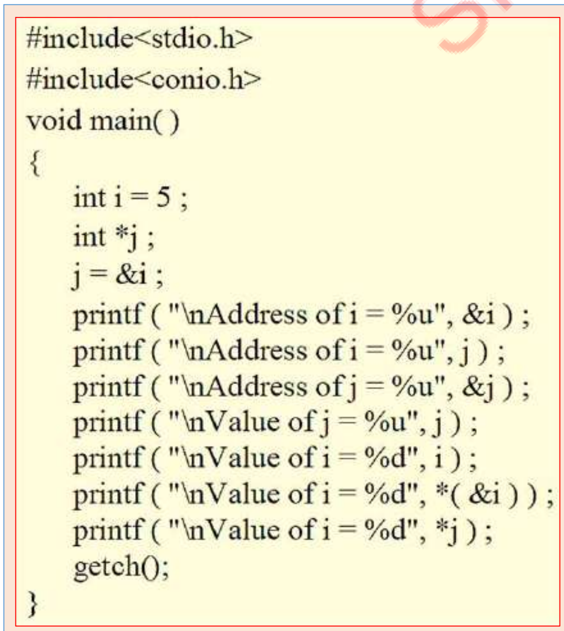
i	j
5	65234
65234	65236

i's value is 5 and j's value is i's address.

- we can’t use j in a program without declaring it. And since j is a variable that contains the address of i, it is declared as,

  int *j ;

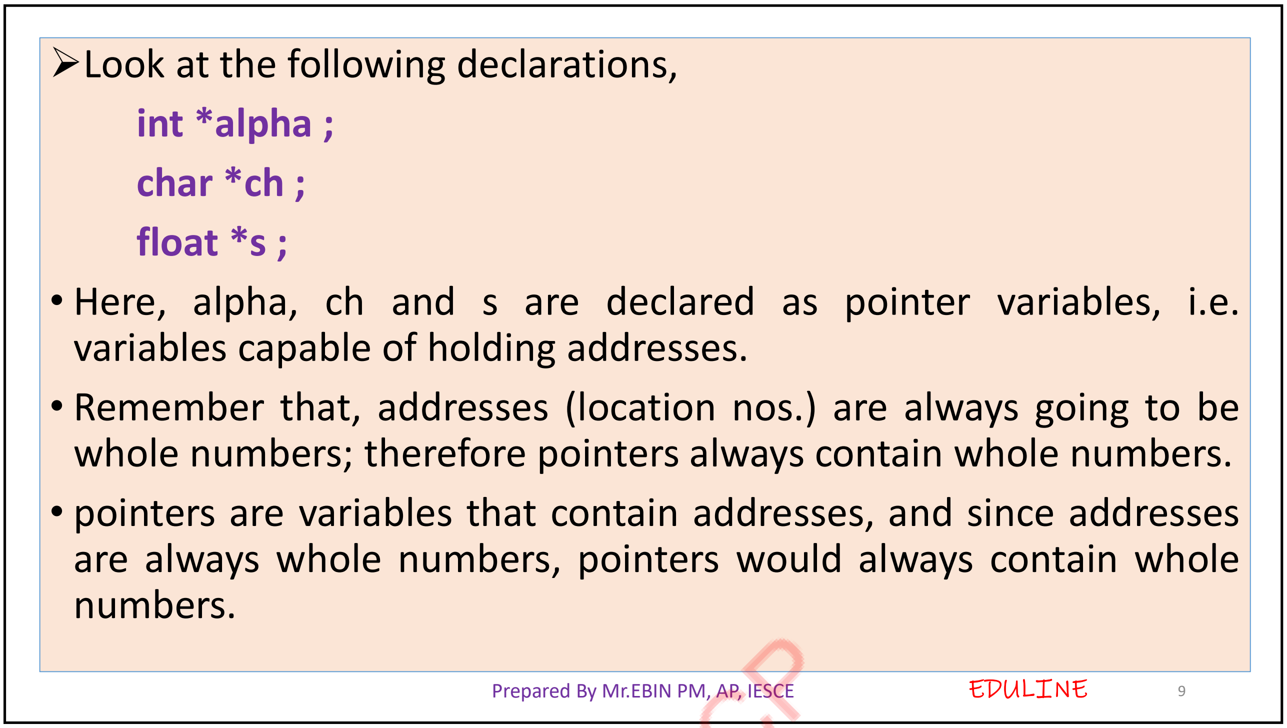
- This declaration tells the compiler that j will be used to store the address of an integer value. In other words j points to an integer
- Let us go by the meaning of *. It stands for ‘value at address’. Thus, int *j would mean, the value at the address contained in j is an int.



```
#include<stdio.h>
#include<conio.h>
void main( )
{
    int i = 5 ;
    int *j ;
    j = &i ;
    printf ( "\nAddress of i = %u", &i ) ;
    printf ( "\nAddress of i = %u", j ) ;
    printf ( "\nAddress of j = %u", &j ) ;
    printf ( "\nValue of j = %u", j ) ;
    printf ( "\nValue of i = %d", i ) ;
    printf ( "\nValue of i = %d", *( &i ) ) ;
    printf ( "\nValue of i = %d", *j ) ;
    getch();
}
```

OUTPUT

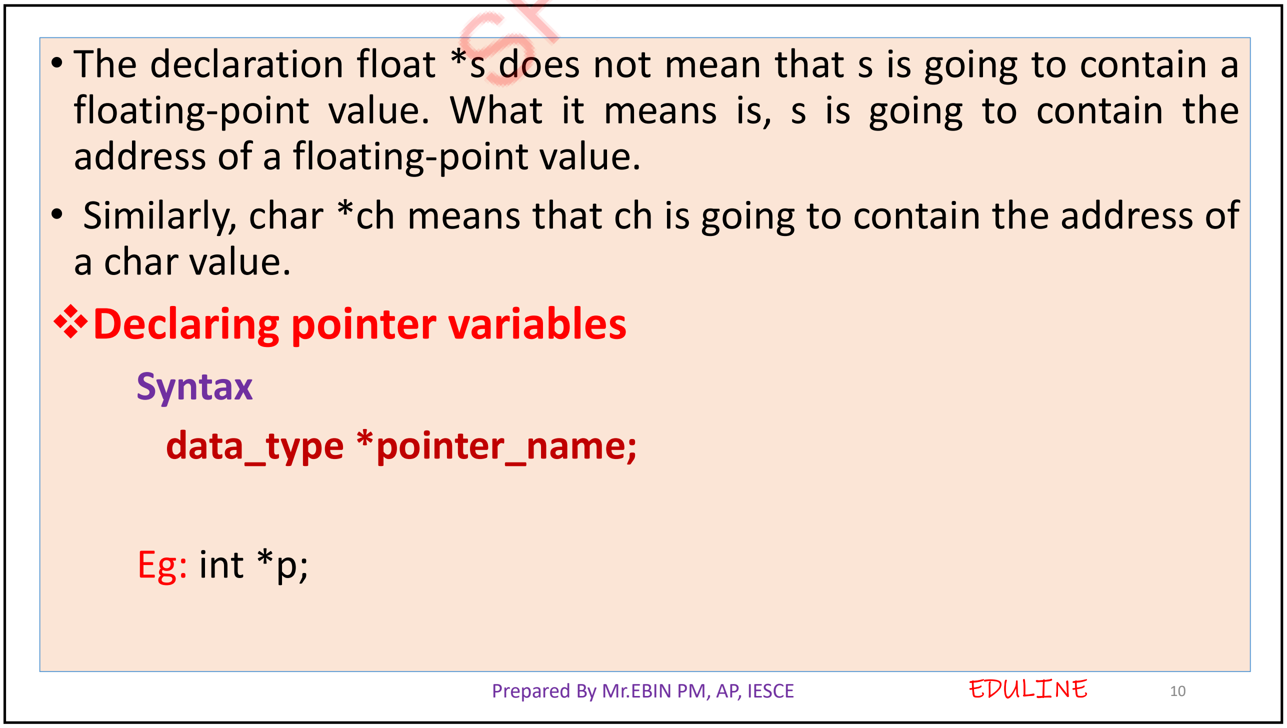
```
Address of i = 65234
Address of i = 65234
Address of j = 65236
Value of j = 65234
Value of i = 5
Value of i = 5
Value of i = 5
```



➢Look at the following declarations,

```
int *alpha ;
char *ch ;
float *s ;
```

- Here, alpha, ch and s are declared as pointer variables, i.e. variables capable of holding addresses.
- Remember that, addresses (location nos.) are always going to be whole numbers; therefore pointers always contain whole numbers.
- pointers are variables that contain addresses, and since addresses are always whole numbers, pointers would always contain whole numbers.



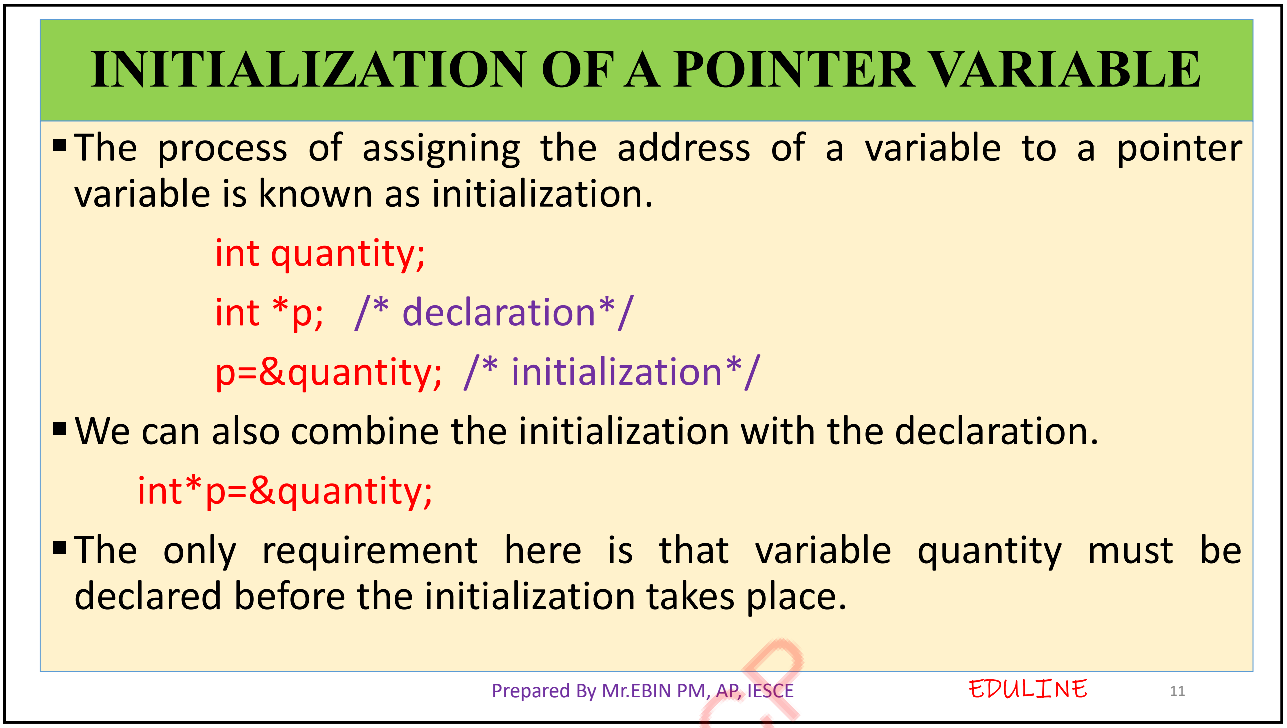
- The declaration float *s does not mean that s is going to contain a floating-point value. What it means is, s is going to contain the address of a floating-point value.
- Similarly, char *ch means that ch is going to contain the address of a char value.

❖Declaring pointer variables

Syntax

data_type *pointer_name;

Eg: int *p;



INITIALIZATION OF A POINTER VARIABLE

- The process of assigning the address of a variable to a pointer variable is known as initialization.

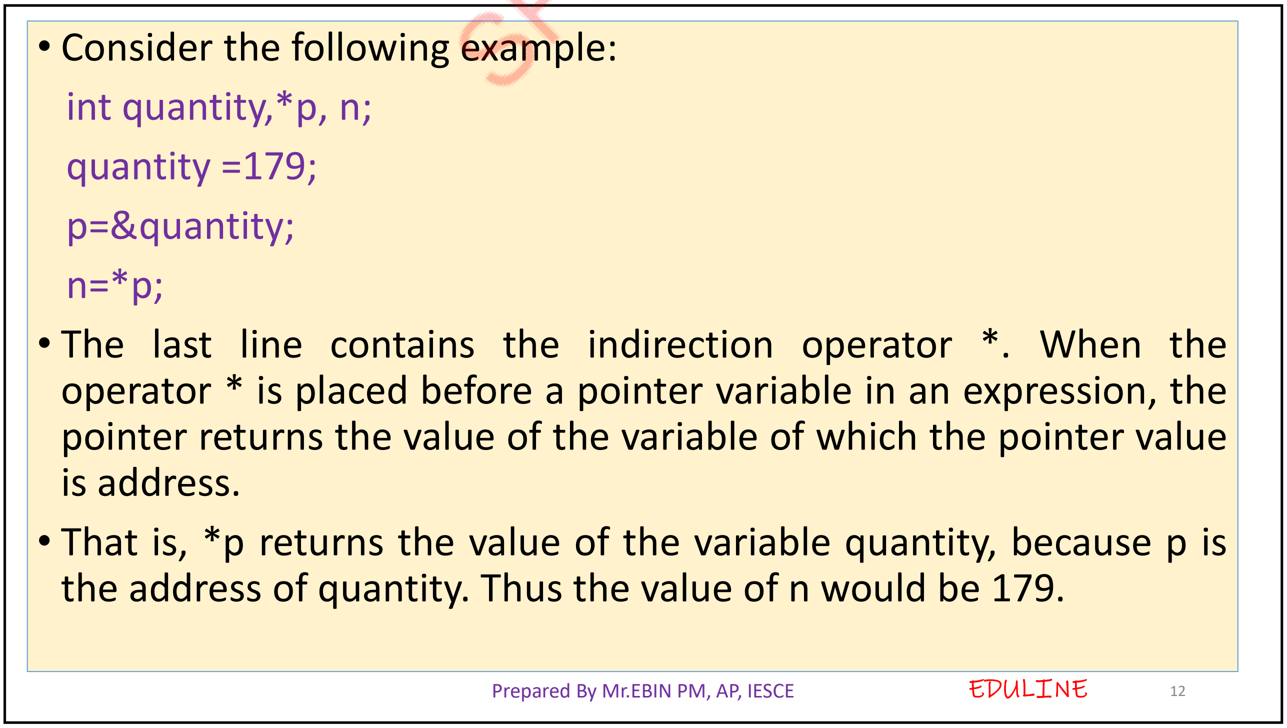
```
int quantity;
int *p;   /* declaration*/
p=&quantity;  /* initialization*/
```

- We can also combine the initialization with the declaration.

```
int*p=&quantity;
```

- The only requirement here is that variable quantity must be declared before the initialization takes place.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 11



- Consider the following example:

```
int quantity,*p, n;
quantity =179;
p=&quantity;
n=*p;
```

- The last line contains the indirection operator *. When the operator * is placed before a pointer variable in an expression, the pointer returns the value of the variable of which the pointer value is address.
- That is, *p returns the value of the variable quantity, because p is the address of quantity. Thus the value of n would be 179.

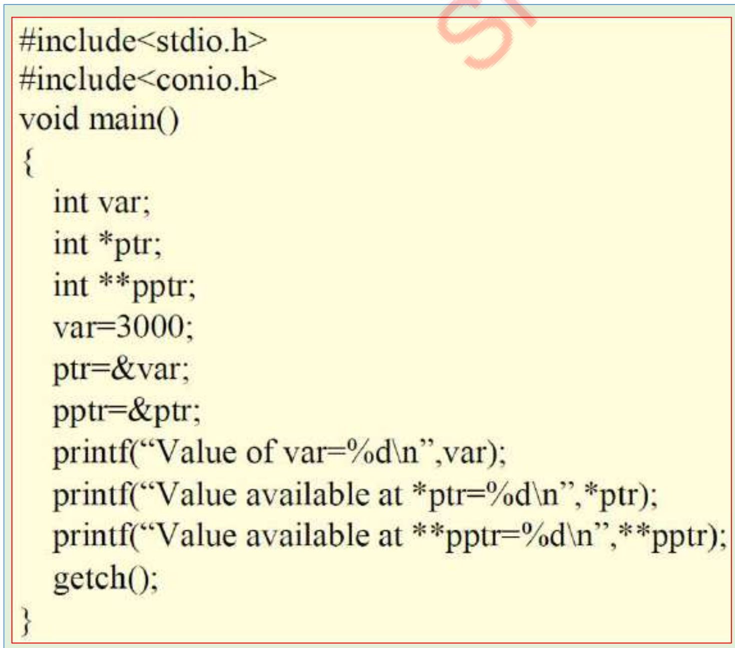
Prepared By Mr.EBIN PM, AP, IESCE EDULINE 12

POINTER TO A POINTER (Chain of Pointers)

- It is possible to make pointer to point to another pointer. Pointer, is a variable that contains address of another variable. Now this variable itself might be another pointer. Thus, we now have a pointer that contains another pointer's address. The following example should make this point clear.
- Observe how the variables p2 have been declared,

  int x, *p1, **p2 ;

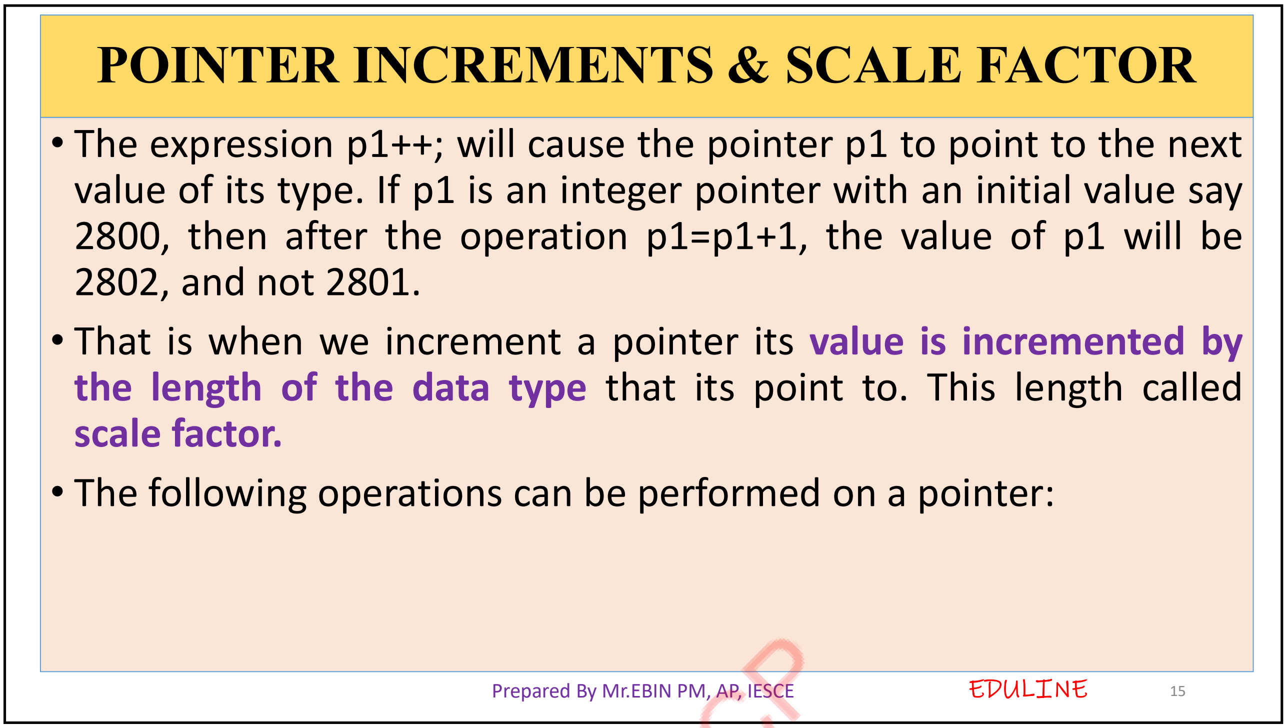
- Here, x is an ordinary int, p1 is a pointer to an int (often called an integer pointer), whereas p2 is a pointer to an integer pointer.
- The representation ****p2 is called multiple indirection**.



```
#include<stdio.h>
#include<conio.h>
void main()
{
   int var;
   int *ptr;
   int **pptr;
   var=3000;
   ptr=&var;
   pptr=&ptr;
   printf(“Value of var=%d\n”,var);
   printf(“Value available at *ptr=%d\n”,*ptr);
   printf(“Value available at **pptr=%d\n”,**pptr);
   getch();
}
```

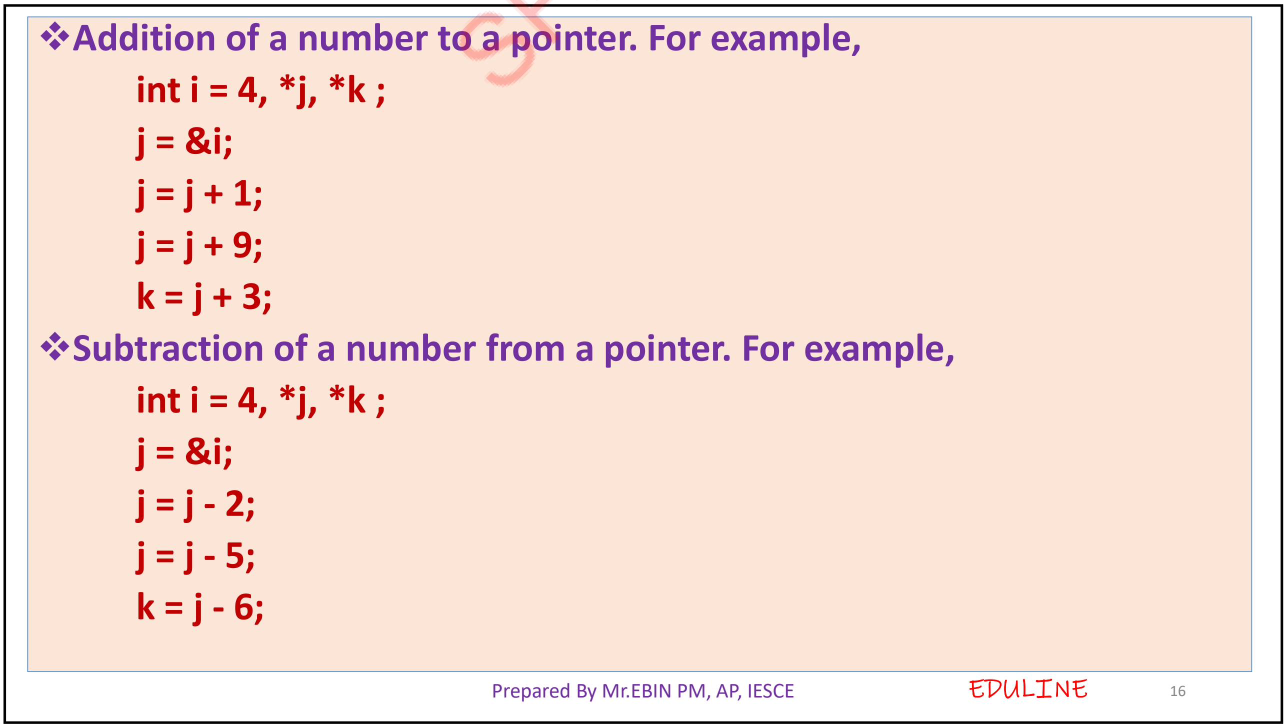
OUTPUT

```
Value of var=3000
Value available at *ptr=3000
Value available at **pptr=3000
```



POINTER INCREMENTS & SCALE FACTOR

- The expression p1++; will cause the pointer p1 to point to the next value of its type. If p1 is an integer pointer with an initial value say 2800, then after the operation p1=p1+1, the value of p1 will be 2802, and not 2801.
- That is when we increment a pointer its **value is incremented by the length of the data type** that its point to. This length called **scale factor.**
- The following operations can be performed on a pointer:



❖**Addition of a number to a pointer. For example,**

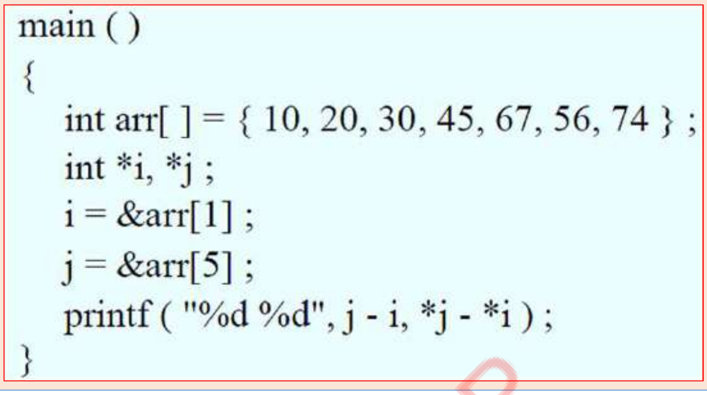
```
int i = 4, *j, *k ;
j = &i;
j = j + 1;
j = j + 9;
k = j + 3;
```

❖**Subtraction of a number from a pointer. For example,**

```
int i = 4, *j, *k ;
j = &i;
j = j - 2;
j = j - 5;
k = j - 6;
```

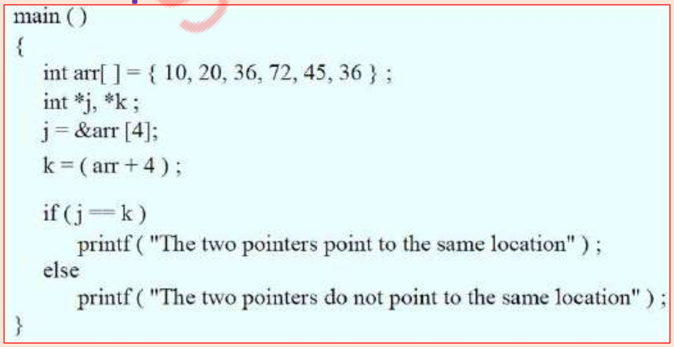
❖Subtraction of one pointer from another.

- One pointer variable can be subtracted from another provided both variables point to elements of the same array. The resulting value indicates the number of bytes separating the corresponding array elements. This is illustrated in the following program.



```
main ( )
{
    int arr[ ] = { 10, 20, 30, 45, 67, 56, 74 } ;
    int *i, *j ;
    i = &arr[1] ;
    j = &arr[5] ;
    printf ( "%d %d", j - i, *j - *i ) ;
}
```

❖Comparison of two pointer variables



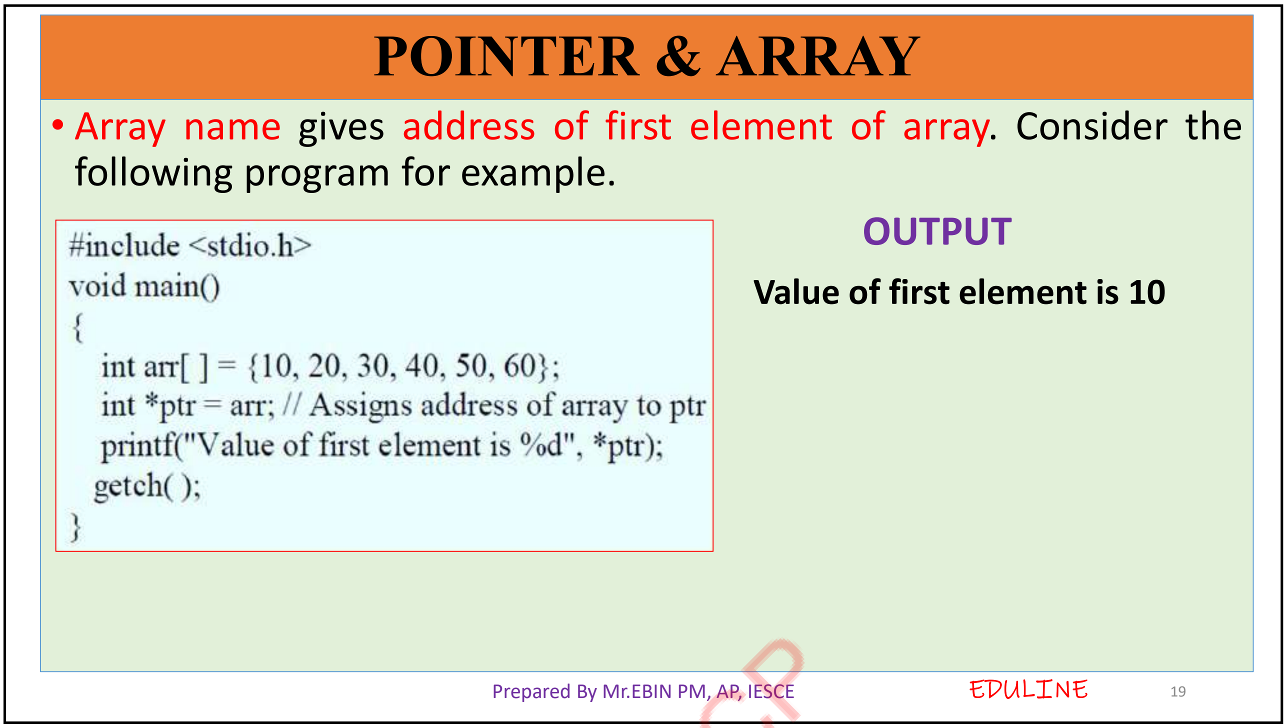
```
main ( )
{
    int arr[ ] = { 10, 20, 36, 72, 45, 36 } ;
    int *j, *k ;
    j = &arr [4];
    k = ( arr + 4 ) ;

    if ( j == k )
        printf ( "The two pointers point to the same location" ) ;
    else
        printf ( "The two pointers do not point to the same location" ) ;
}
```

➢Do not attempt the following operations on pointers... they would never work out.

(a) Addition of two pointers

(b) Multiplication of a pointer with a constant

(c) Division of a pointer with a constant



POINTER & ARRAY

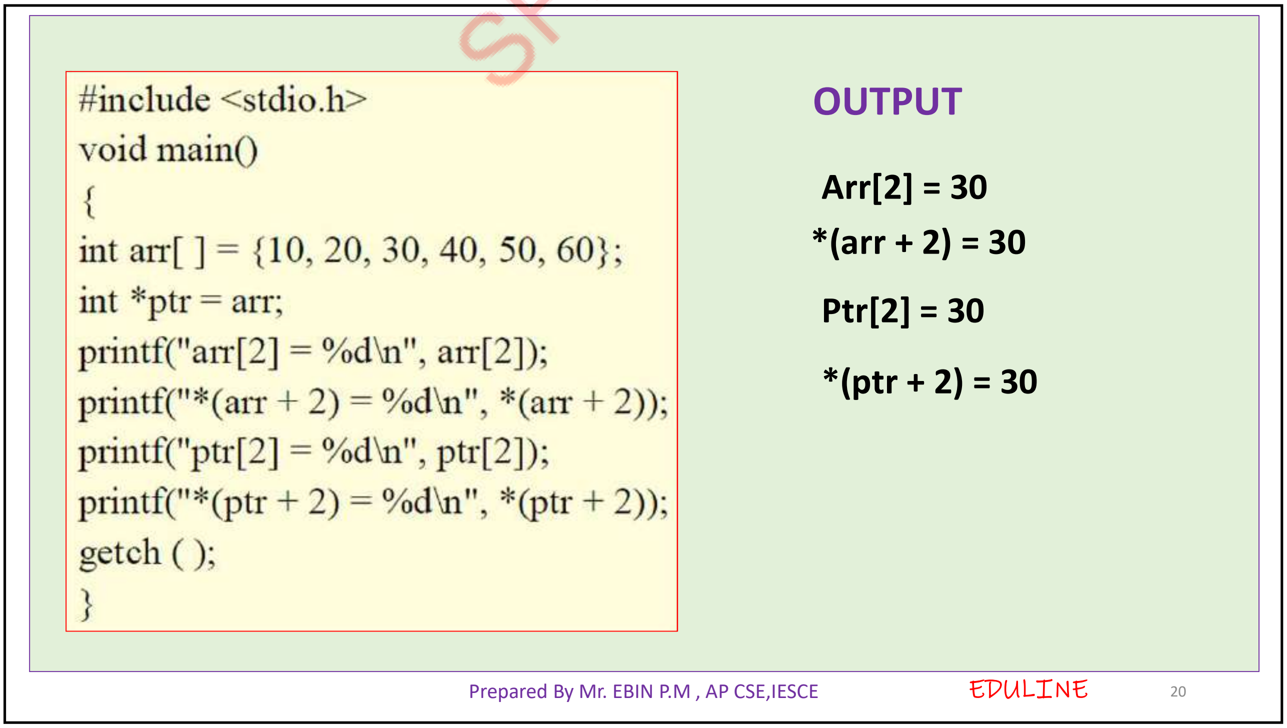
- Array name gives address of first element of array. Consider the following program for example.

```
#include <stdio.h>
void main()
{
   int arr[ ] = {10, 20, 30, 40, 50, 60};
   int *ptr = arr; // Assigns address of array to ptr
   printf("Value of first element is %d", *ptr);
  getch( );
}
```

OUTPUT

```
Value of first element is 10
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 19



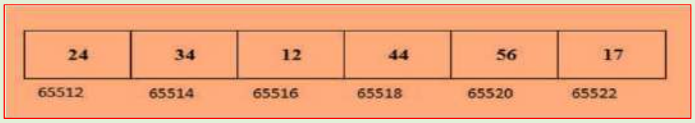
```
#include <stdio.h>
void main()
{
int arr[ ] = {10, 20, 30, 40, 50, 60};
int *ptr = arr;
printf("arr[2] = %d\n", arr[2]);
printf("*(arr + 2) = %d\n", *(arr + 2));
printf("ptr[2] = %d\n", ptr[2]);
printf("*(ptr + 2) = %d\n", *(ptr + 2));
getch ( );
}
```

OUTPUT

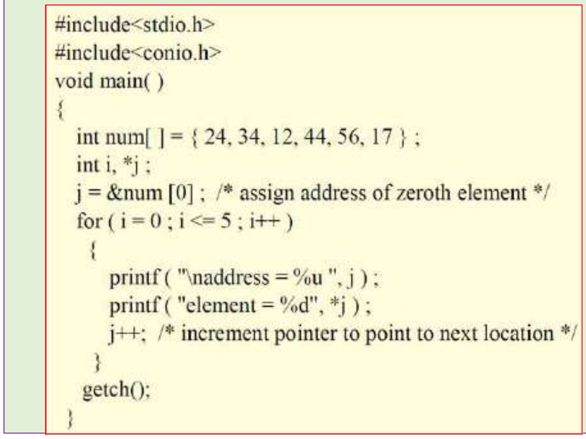
```
Arr[2] = 30
*(arr + 2) = 30
Ptr[2] = 30
*(ptr + 2) = 30
```

Prepared By Mr. EBIN P.M , AP CSE,IESCE EDULINE 20

- Suppose we have an array num [ ] = {24, 34, 12, 44, 56, 17}. The following figure shows how this array is located in memory.



24	34	12	44	56	17
65512	65514	65516	65518	65520	65522



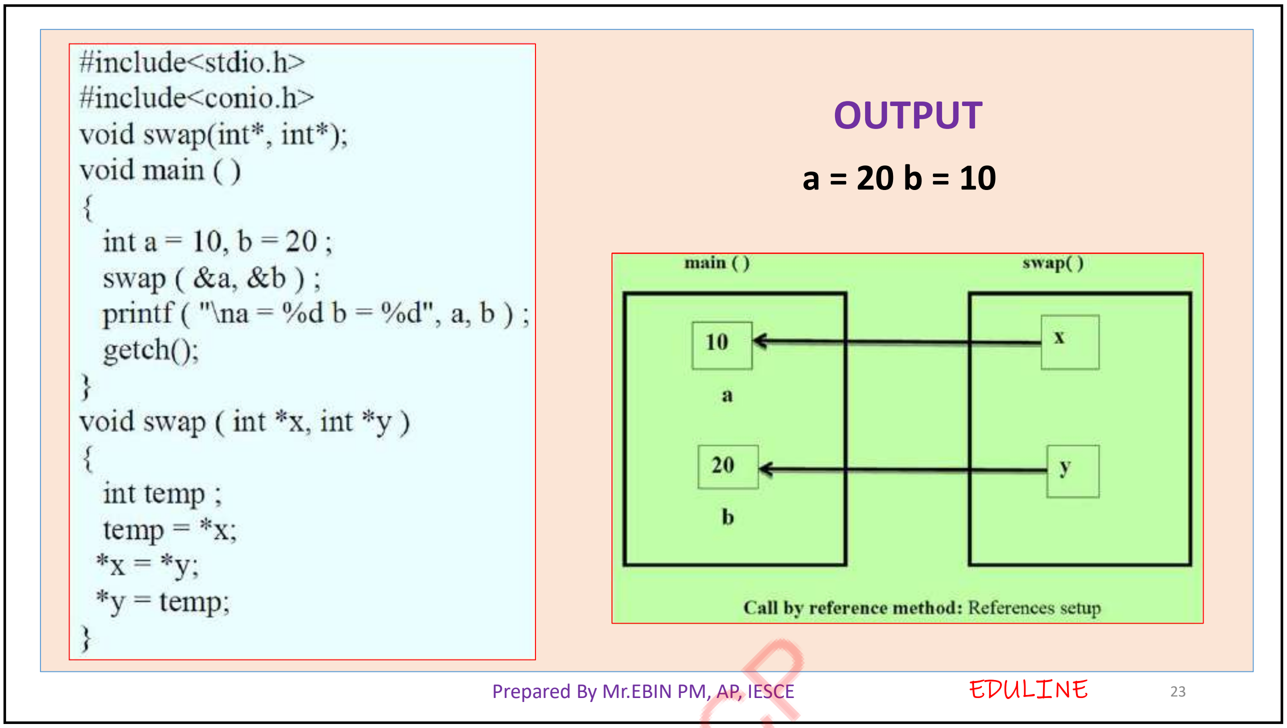
```
#include<stdio.h>
#include<conio.h>
void main( )
{
   int num[ ] = { 24, 34, 12, 44, 56, 17 } ;
   int i, *j ;
   j = &num [0] ;  /* assign address of zeroth element */
   for ( i = 0 ; i <= 5 ; i++ )
     {
        printf ( "\naddress = %u ", j ) ;
        printf ( "element = %d", *j ) ;
        j++;  /* increment pointer to point to next location */
     }
   getch();
 }
```

OUTPUT

```
address = 65512 element = 24
address = 65514 element = 34
address = 65516 element = 12
address = 65518 element = 44
address = 65520 element = 56
address = 65522 element = 17
```

PASS BY REFERENCE (CALL BY REFERENCE)

- In this method the addresses of actual arguments in the calling function are copied into formal arguments of the called function.
- That is the same variables value can be accessed by any of the two names: The original variables name and the reference name.
- We are actually passes the address.
- The following program illustrates this fact
- Note that this program manages to exchange the values of **a** and **b** using their addresses stored in **x** and **y**.
- Usually in C programming we make a call by value. This means that in general you cannot alter the actual arguments



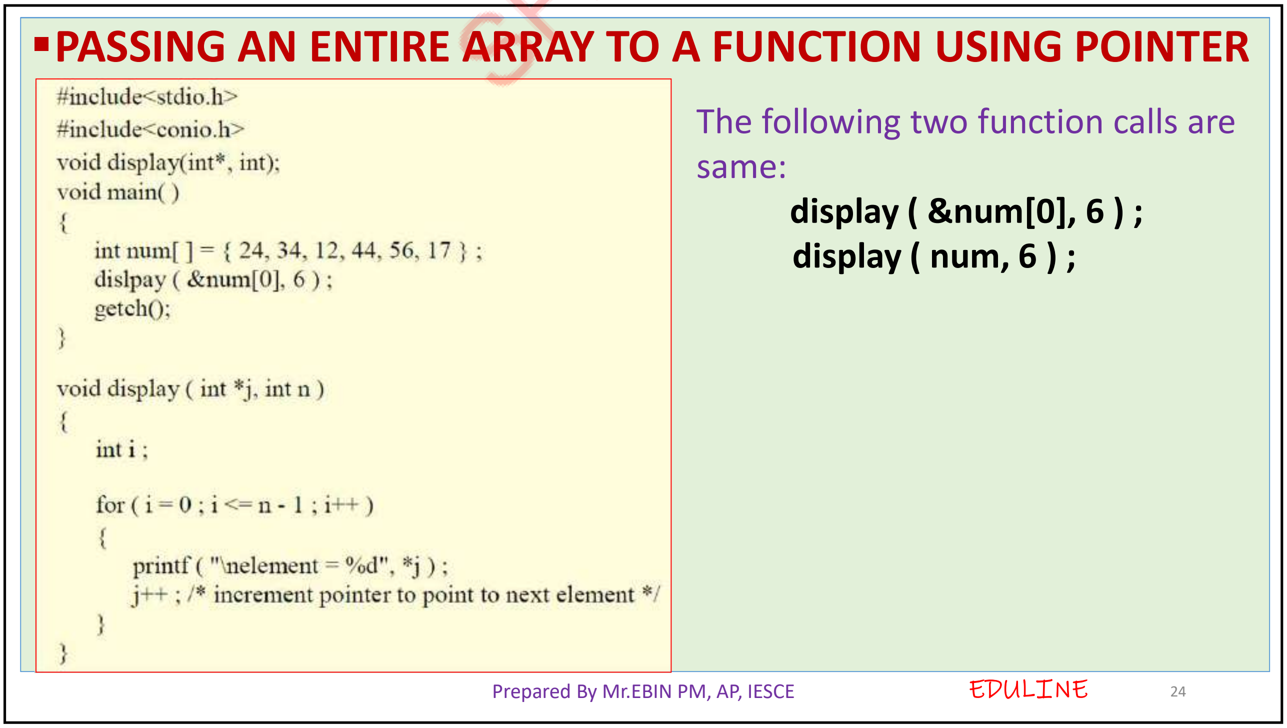
```
#include<stdio.h>
#include<conio.h>
void swap(int*, int*);
void main ( )
{
  int a = 10, b = 20 ;
  swap ( &a, &b ) ;
  printf ( "\na = %d b = %d", a, b ) ;
  getch();
}
void swap ( int *x, int *y )
{
  int temp ;
  temp = *x;
 *x = *y;
 *y = temp;
}
```

OUTPUT

```
a = 20 b = 10
```

Call by reference method: References setup

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 23



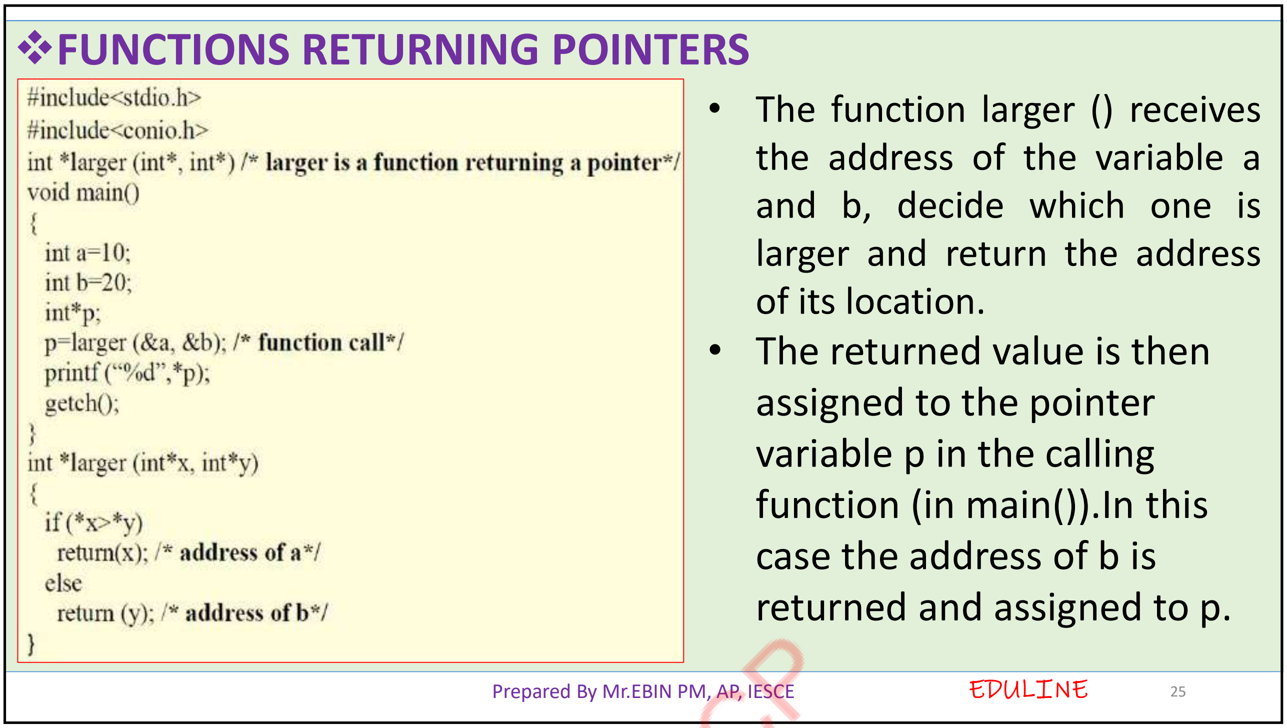
▪PASSING AN ENTIRE ARRAY TO A FUNCTION USING POINTER

```
#include<stdio.h>
#include<conio.h>
void display(int*, int);
void main( )
{
    int num[ ] = { 24, 34, 12, 44, 56, 17 } ;
    dislpay ( &num[0], 6 ) ;
    getch();
}

void display ( int *j, int n )
{
    int i ;

    for ( i = 0 ; i <= n - 1 ; i++ )
    {
        printf ( "\nelement = %d", *j ) ;
        j++ ; /* increment pointer to point to next element */
    }
}
```

The following two function calls are same:

display (&num[0], 6) ;
display (num, 6) ;

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 24

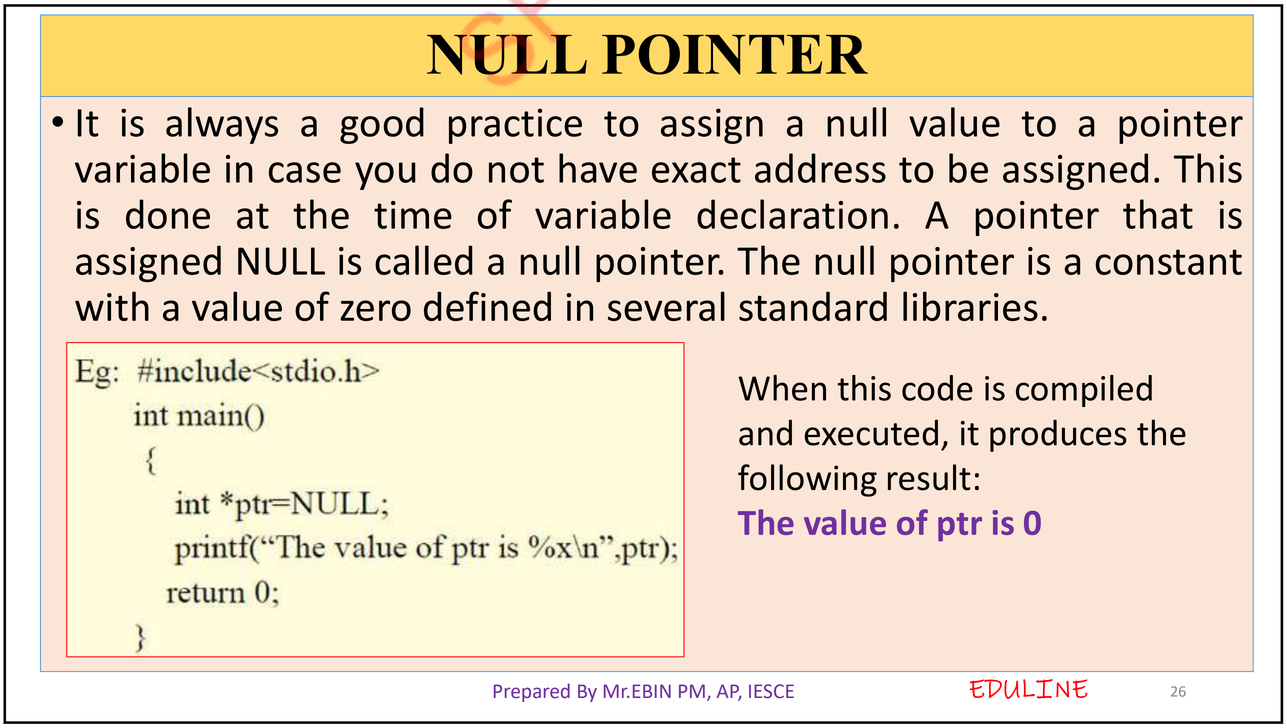


❖FUNCTIONS RETURNING POINTERS

```
#include<stdio.h>
#include<conio.h>
int *larger (int*, int*) /* larger is a function returning a pointer*/
void main()
{
  int a=10;
  int b=20;
  int*p;
  p=larger (&a, &b); /* function call*/
  printf ("%d",*p);
  getch();
}
int *larger (int*x, int*y)
{
  if (*x>*y)
    return(x); /* address of a*/
  else
    return (y); /* address of b*/
}
```

- The function larger () receives the address of the variable a and b, decide which one is larger and return the address of its location.
- The returned value is then assigned to the pointer variable p in the calling function (in main()).In this case the address of b is returned and assigned to p.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 25



NULL POINTER

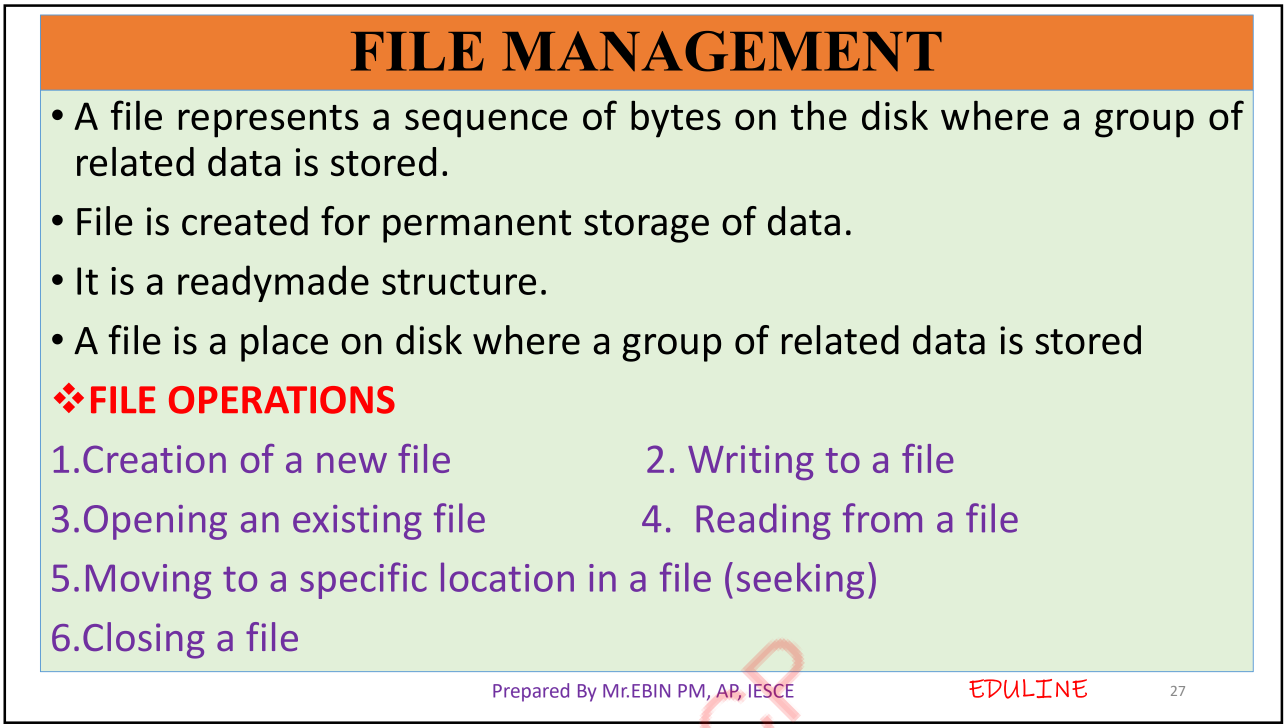
- It is always a good practice to assign a null value to a pointer variable in case you do not have exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a null pointer. The null pointer is a constant with a value of zero defined in several standard libraries.

```
Eg:  #include<stdio.h>
        int main()
         {
            int *ptr=NULL;
            printf("The value of ptr is %x\n",ptr);
           return 0;
         }
```

When this code is compiled and executed, it produces the following result:

The value of ptr is 0

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 26



FILE MANAGEMENT

- A file represents a sequence of bytes on the disk where a group of related data is stored.
- File is created for permanent storage of data.
- It is a readymade structure.
- A file is a place on disk where a group of related data is stored

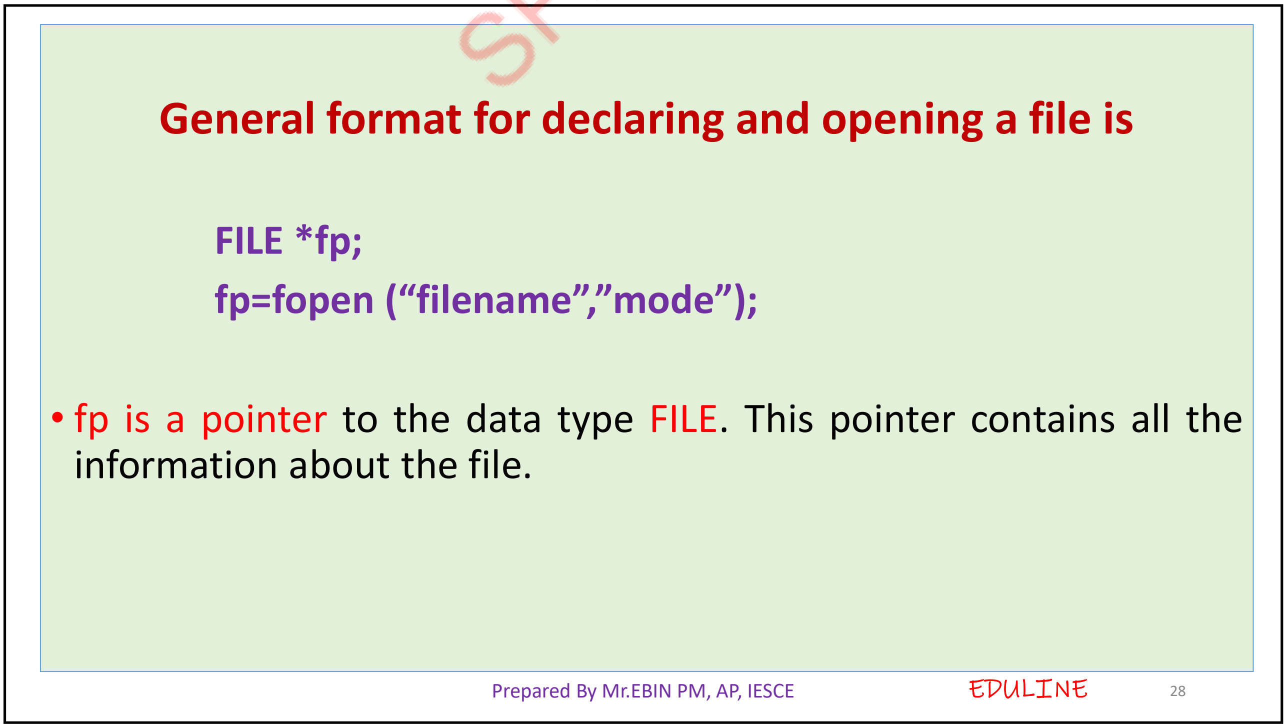
❖**FILE OPERATIONS**

1.Creation of a new file 2. Writing to a file

3.Opening an existing file 4. Reading from a file

5.Moving to a specific location in a file (seeking)

6.Closing a file

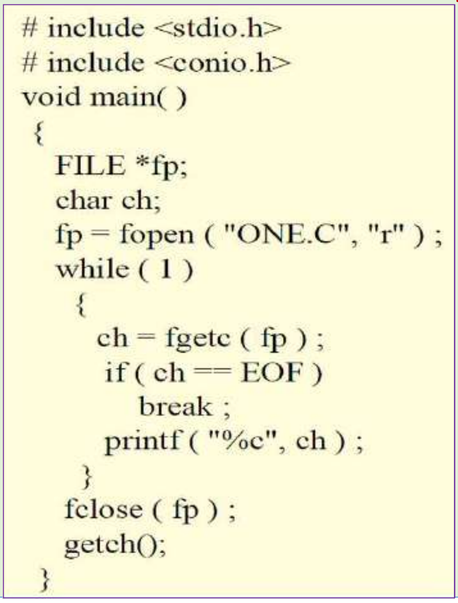


General format for declaring and opening a file is

```
FILE *fp;
fp=fopen (“filename”,”mode”);
```

- fp is a pointer to the data type FILE. This pointer contains all the information about the file.

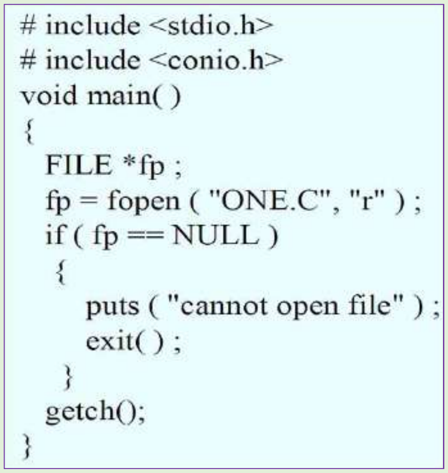
❖program to read a file and display its contents on the screen



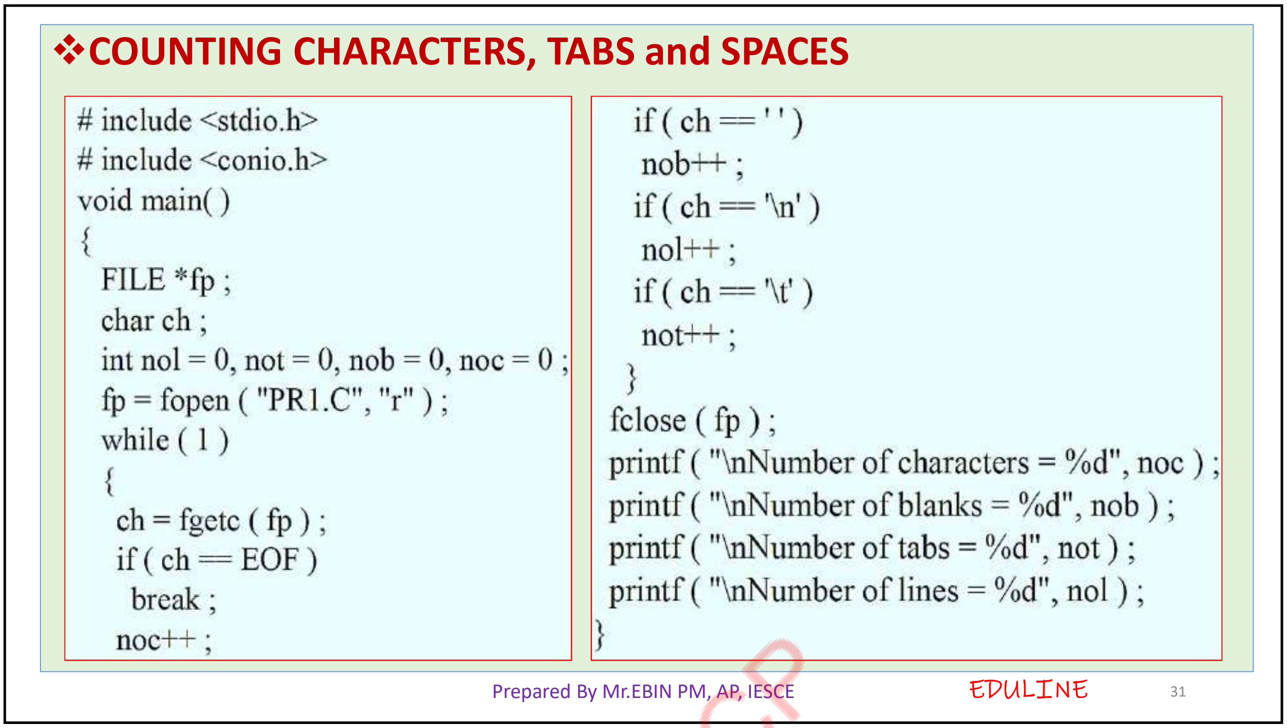
```
# include <stdio.h>
# include <conio.h>
void main( )
 {
   FILE *fp;
   char ch;
   fp = fopen ( "ONE.C", "r" ) ;
   while ( 1 )
     {
       ch = fgetc ( fp ) ;
        if ( ch == EOF )
           break ;
        printf ( "%c", ch ) ;
     }
    fclose ( fp ) ;
    getch();
 }
```

❖Trouble in Opening a File

- It is important for any program that accesses disk files to check whether a file has been opened successfully before trying to read or write to the file.
- If the file opening fails due to any of the several reasons , the fopen( ) function returns a value NULL (defined in “stdio.h” as **#define NULL 0**)



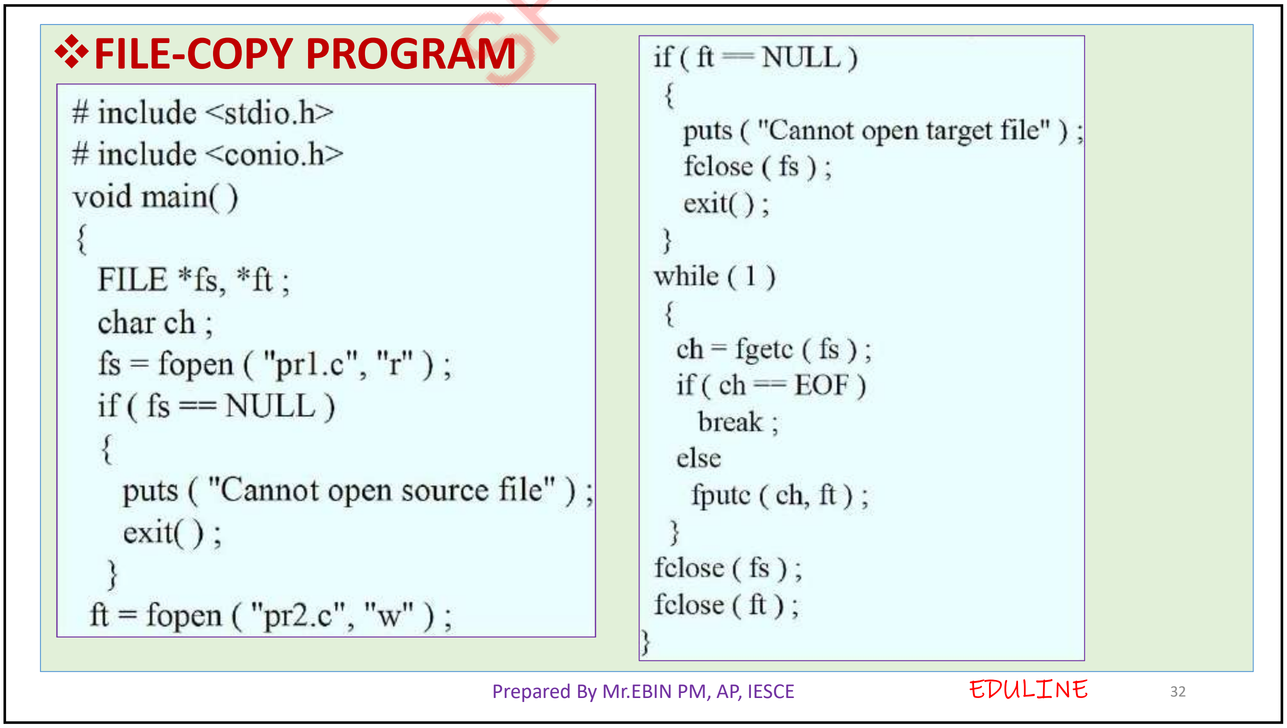
```
# include <stdio.h>
# include <conio.h>
void main( )
{
   FILE *fp ;
   fp = fopen ( "ONE.C", "r" ) ;
   if ( fp == NULL )
    {
       puts ( "cannot open file" ) ;
       exit( ) ;
     }
   getch();
}
```



❖COUNTING CHARACTERS, TABS and SPACES

```
# include <stdio.h>
# include <conio.h>
void main( )
{
  FILE *fp ;
  char ch ;
  int nol = 0, not = 0, nob = 0, noc = 0 ;
  fp = fopen ( "PR1.C", "r" ) ;
  while ( 1 )
  {
   ch = fgetc ( fp ) ;
   if ( ch == EOF )
    break ;
   noc++ ;
   if ( ch == ' ' )
    nob++ ;
   if ( ch == '\n' )
    nol++ ;
   if ( ch == '\t' )
    not++ ;
  }
 fclose ( fp ) ;
 printf ( "\nNumber of characters = %d", noc ) ;
 printf ( "\nNumber of blanks = %d", nob ) ;
 printf ( "\nNumber of tabs = %d", not ) ;
 printf ( "\nNumber of lines = %d", nol ) ;
}
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 31

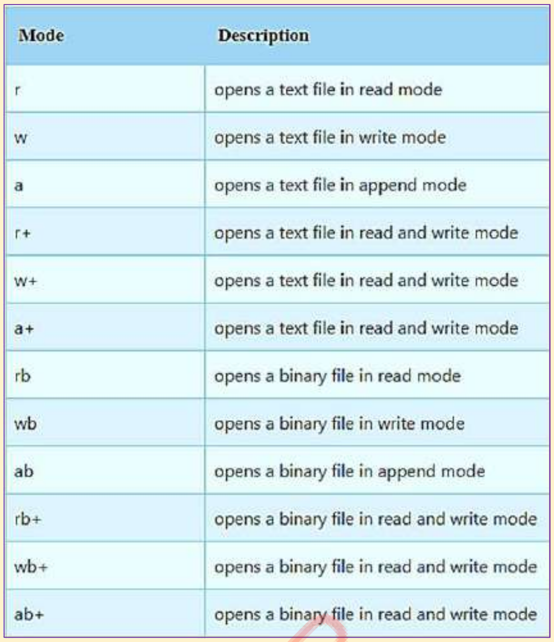


❖FILE-COPY PROGRAM

```
# include <stdio.h>
# include <conio.h>
void main( )
{
  FILE *fs, *ft ;
  char ch ;
  fs = fopen ( "pr1.c", "r" ) ;
  if ( fs == NULL )
  {
    puts ( "Cannot open source file" ) ;
    exit( ) ;
  }
 ft = fopen ( "pr2.c", "w" ) ;
 if ( ft == NULL )
 {
   puts ( "Cannot open target file" ) ;
   fclose ( fs ) ;
   exit( ) ;
 }
 while ( 1 )
 {
  ch = fgetc ( fs ) ;
  if ( ch == EOF )
   break ;
  else
   fputc ( ch, ft ) ;
 }
 fclose ( fs ) ;
 fclose ( ft ) ;
}
```

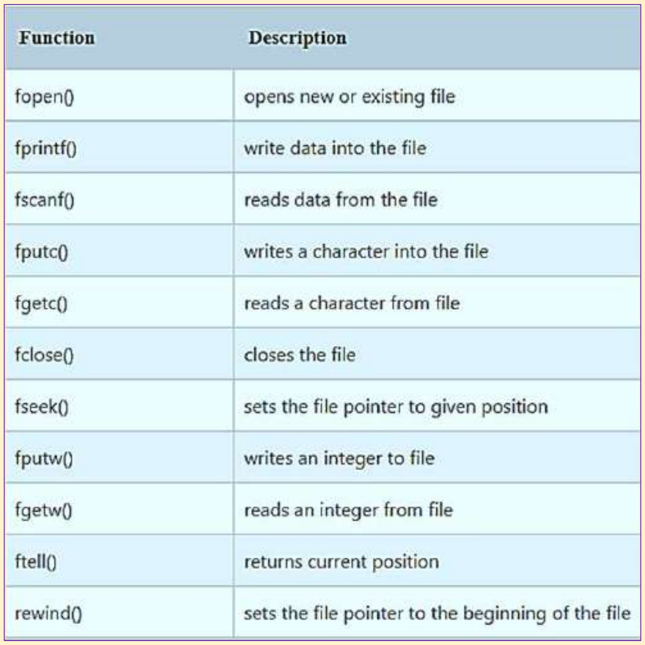
Prepared By Mr.EBIN PM, AP, IESCE EDULINE 32

FILE OPENING MODE

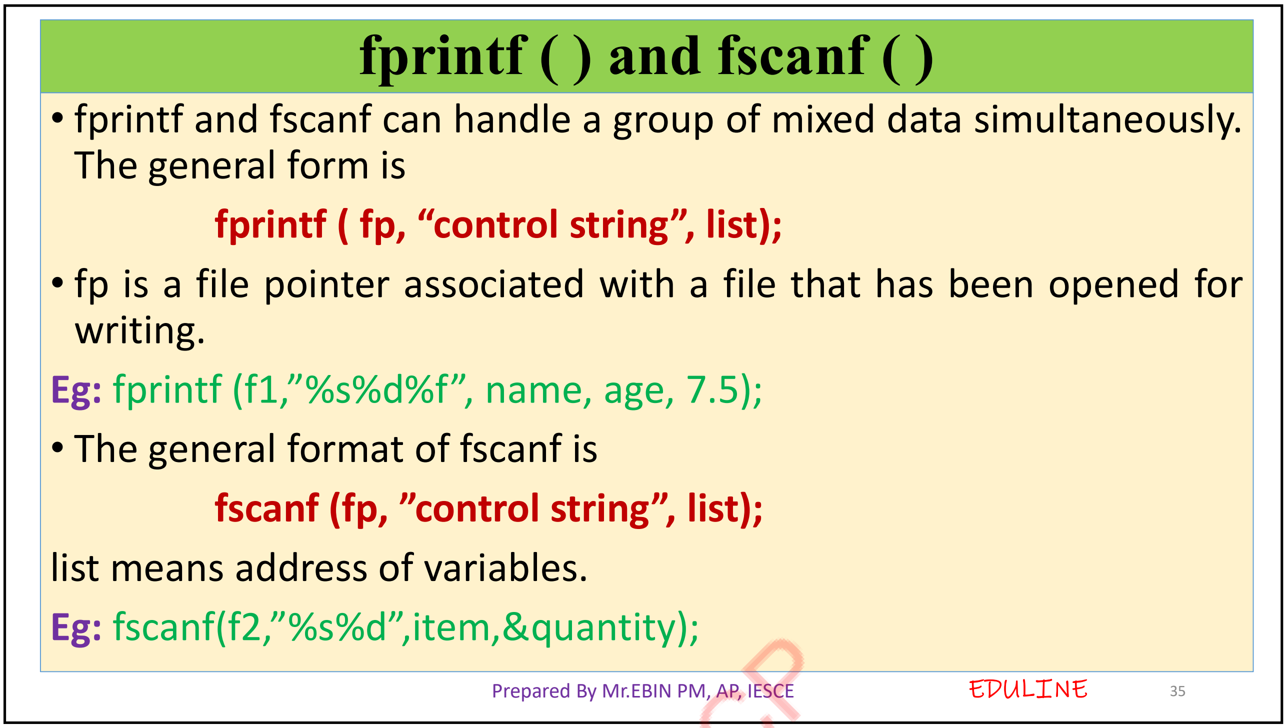


Mode	Description
r	opens a text file in read mode
w	opens a text file in write mode
a	opens a text file in append mode
r+	opens a text file in read and write mode
w+	opens a text file in read and write mode
a+	opens a text file in read and write mode
rb	opens a binary file in read mode
wb	opens a binary file in write mode
ab	opens a binary file in append mode
rb+	opens a binary file in read and write mode
wb+	opens a binary file in read and write mode
ab+	opens a binary file in read and write mode

FUNCTIONS FOR FILE HANDLING



Function	Description
fopen()	opens new or existing file
fprintf()	write data into the file
fscanf()	reads data from the file
fputc()	writes a character into the file
fgetc()	reads a character from file
fclose()	closes the file
fseek()	sets the file pointer to given position
fputw()	writes an integer to file
fgetw()	reads an integer from file
ftell()	returns current position
rewind()	sets the file pointer to the beginning of the file



fprintf ( ) and fscanf ( )

- fprintf and fscanf can handle a group of mixed data simultaneously. The general form is

fprintf (fp, "control string", list);

- fp is a file pointer associated with a file that has been opened for writing.

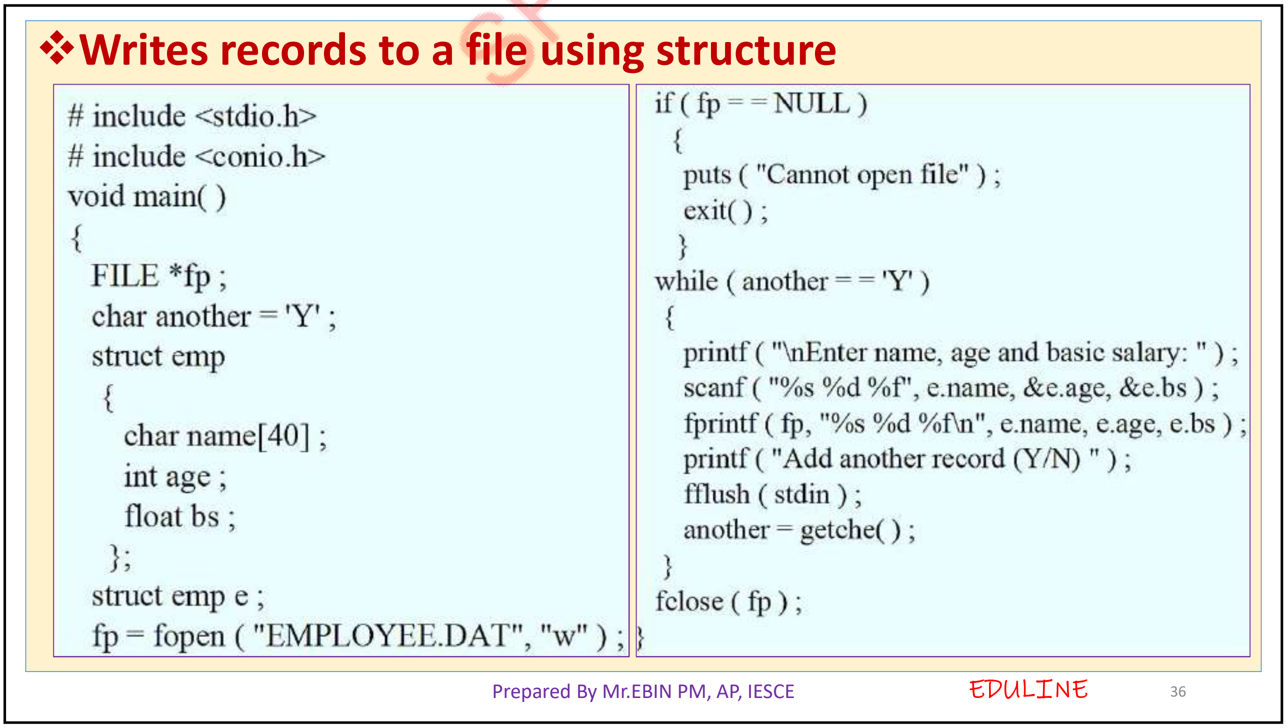
Eg: fprintf (f1,"%s%d%f", name, age, 7.5);

- The general format of fscanf is

fscanf (fp, "control string", list);

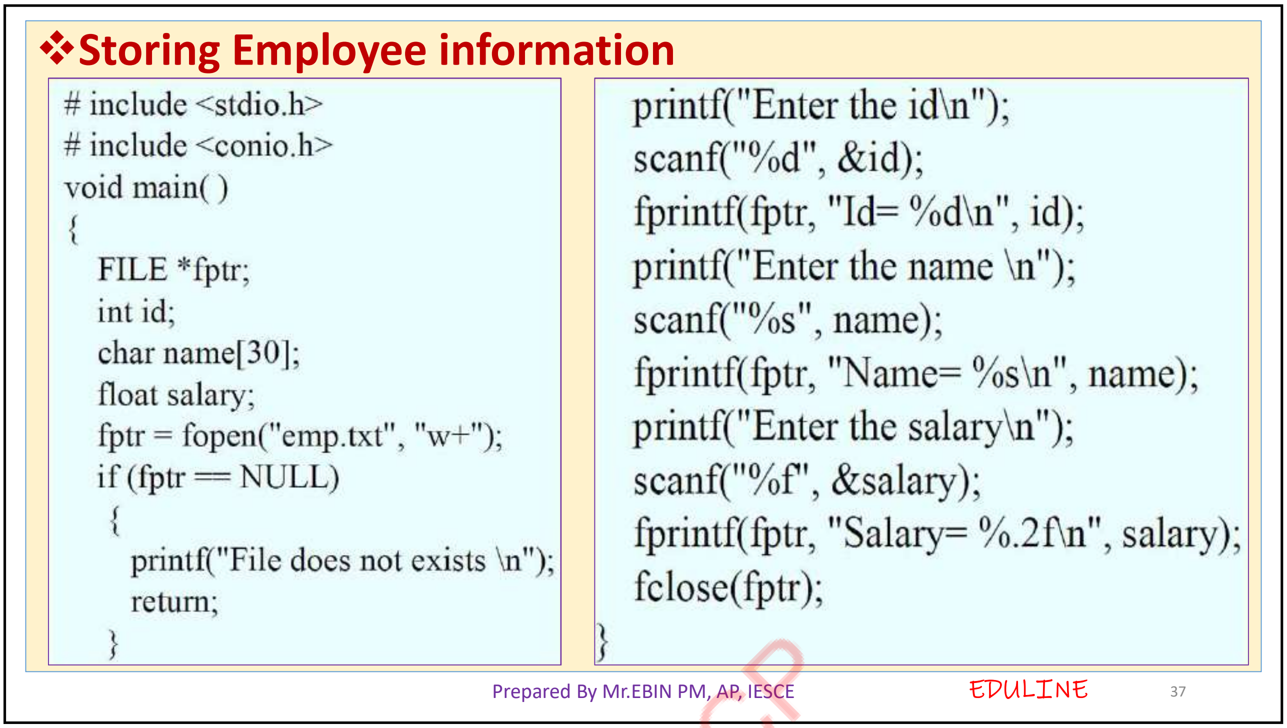
list means address of variables.

Eg: fscanf(f2,"%s%d",item,&quantity);



❖Writes records to a file using structure

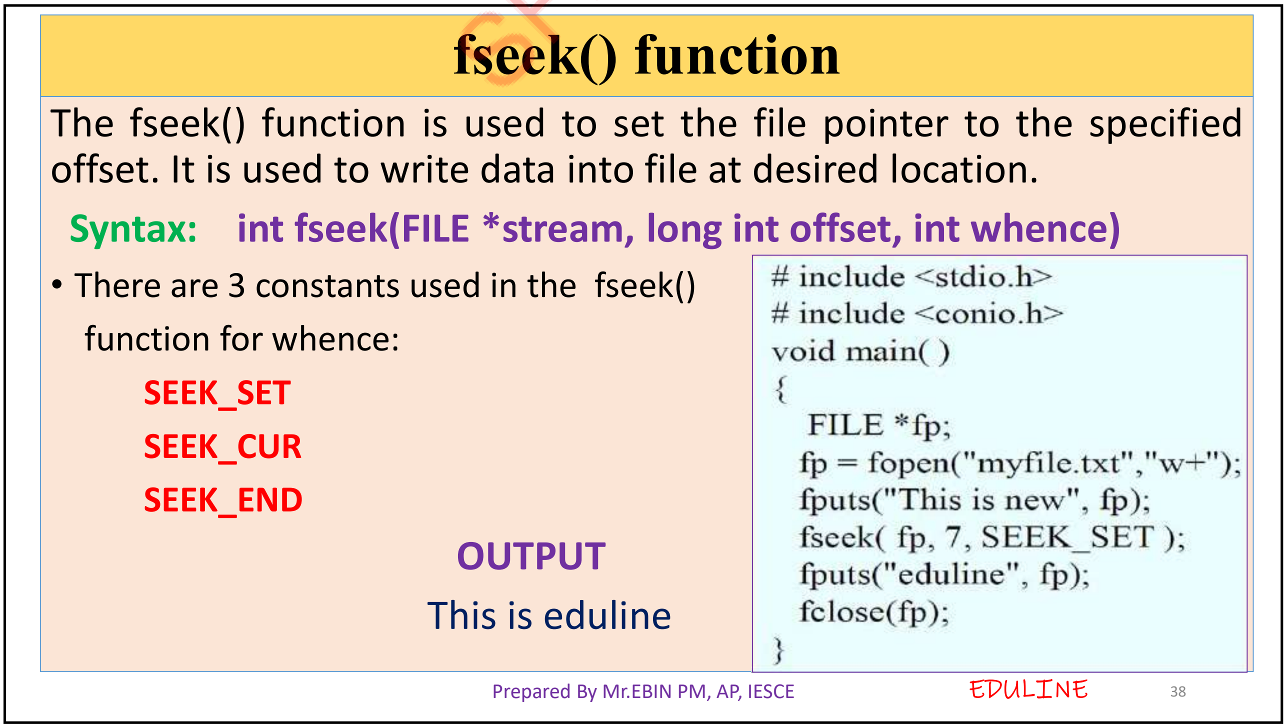
```
# include <stdio.h>
# include <conio.h>
void main( )
{
  FILE *fp ;
  char another = 'Y' ;
  struct emp
   {
     char name[40] ;
     int age ;
     float bs ;
   };
  struct emp e ;
  fp = fopen ( "EMPLOYEE.DAT", "w" ) ;
if ( fp = = NULL )
  {
   puts ( "Cannot open file" ) ;
   exit( ) ;
  }
while ( another = = 'Y' )
 {
   printf ( "\nEnter name, age and basic salary: " ) ;
   scanf ( "%s %d %f", e.name, &e.age, &e.bs ) ;
   fprintf ( fp, "%s %d %f\n", e.name, e.age, e.bs ) ;
   printf ( "Add another record (Y/N) " ) ;
   fflush ( stdin ) ;
   another = getche( ) ;
 }
fclose ( fp ) ;
}
```



❖Storing Employee information

```
# include <stdio.h>
# include <conio.h>
void main( )
{
   FILE *fptr;
   int id;
   char name[30];
   float salary;
   fptr = fopen("emp.txt", "w+");
   if (fptr == NULL)
    {
      printf("File does not exists \n");
      return;
    }
   printf("Enter the id\n");
   scanf("%d", &id);
   fprintf(fptr, "Id= %d\n", id);
   printf("Enter the name \n");
   scanf("%s", name);
   fprintf(fptr, "Name= %s\n", name);
   printf("Enter the salary\n");
   scanf("%f", &salary);
   fprintf(fptr, "Salary= %.2f\n", salary);
   fclose(fptr);
}
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 37



fseek() function

The fseek() function is used to set the file pointer to the specified offset. It is used to write data into file at desired location.

Syntax: **int fseek(FILE *stream, long int offset, int whence)**

- There are 3 constants used in the fseek() function for whence:

  SEEK_SET

  SEEK_CUR

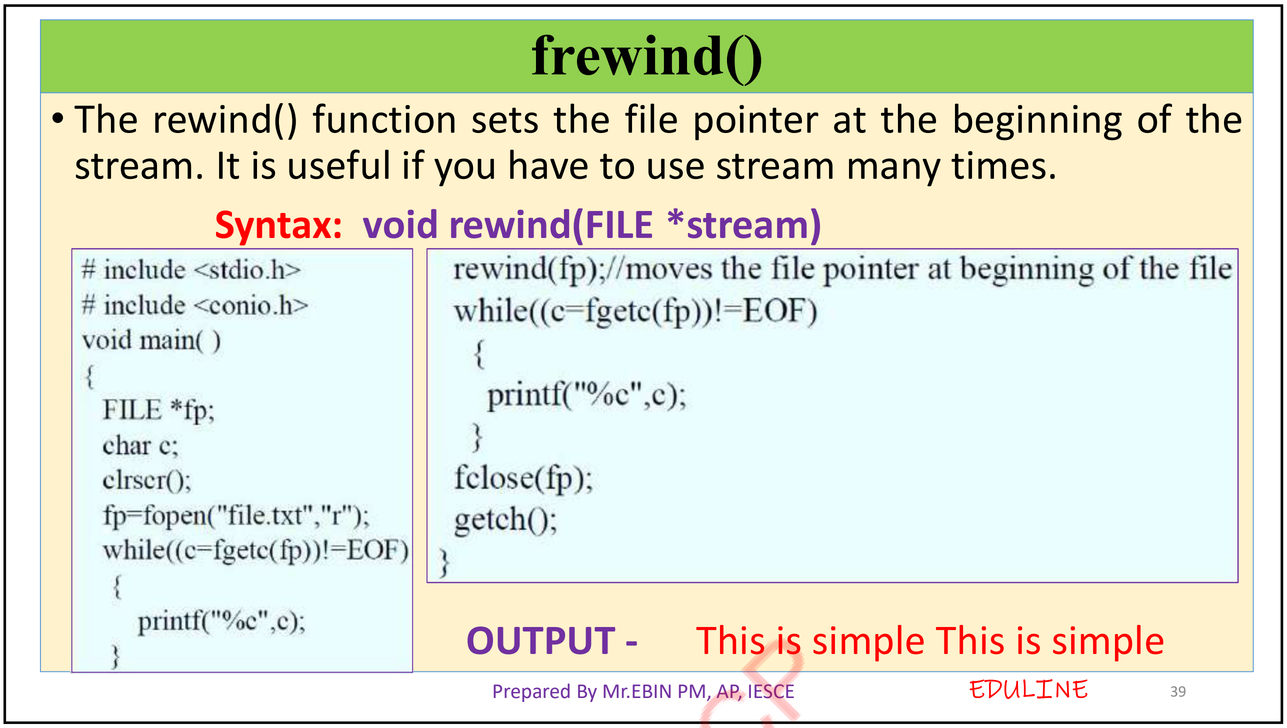
  SEEK_END

OUTPUT

```
This is eduline
```

```
# include <stdio.h>
# include <conio.h>
void main( )
{
   FILE *fp;
   fp = fopen("myfile.txt","w+");
   fputs("This is new", fp);
   fseek( fp, 7, SEEK_SET );
   fputs("eduline", fp);
   fclose(fp);
}
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 38



frewind()

- The rewind() function sets the file pointer at the beginning of the stream. It is useful if you have to use stream many times.

Syntax: void rewind(FILE *stream)

```
# include <stdio.h>
# include <conio.h>
void main( )
{
  FILE *fp;
  char c;
  clrscr();
  fp=fopen("file.txt","r");
  while((c=fgetc(fp))!=EOF)
   {
      printf("%c",c);
   }
```

```
rewind(fp);//moves the file pointer at beginning of the file
while((c=fgetc(fp))!=EOF)
  {
   printf("%c",c);
  }
fclose(fp);
getch();
}
```

OUTPUT - This is simple This is simple

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 39