UNIT – V

CHAPTER XI

POINTERS

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];

&x → pointing at array names

&(x+y) → pointing at expressions

illegal

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

2. *int q;*
   *int *p=&q*

3. *int x,*p=&x*

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).**

```
#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;
}
```

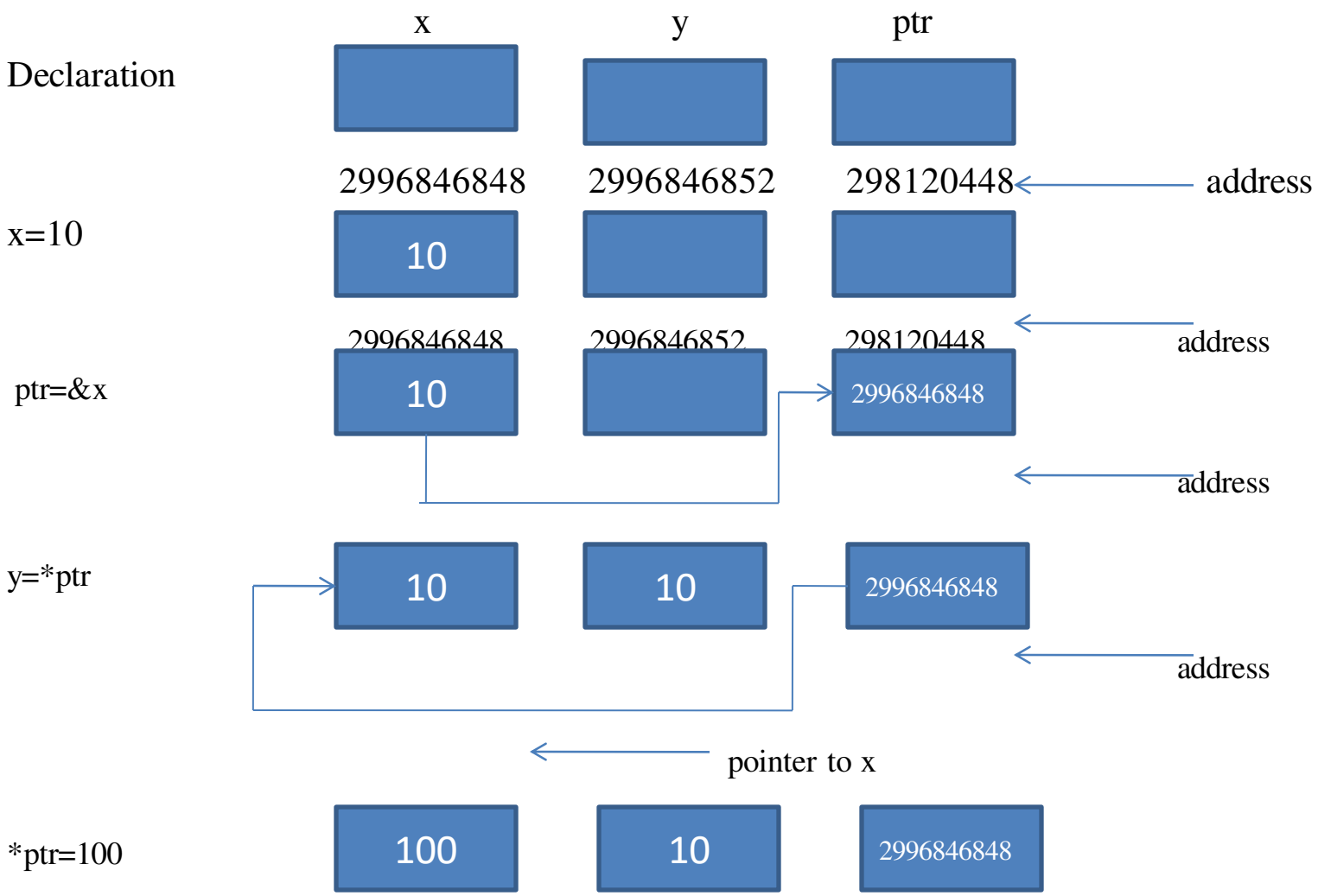
EXAMPLE

```
#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

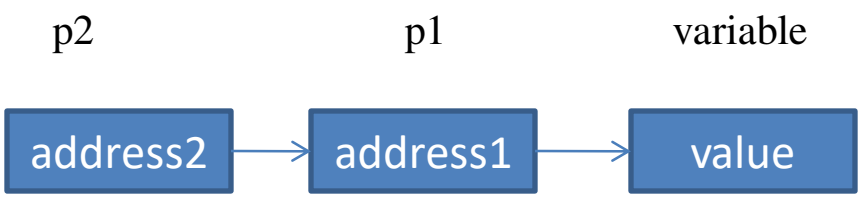
```
Value of x is 10

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



11.5 CHAIN OF POINTER

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



- 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

```
#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

```
#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
```

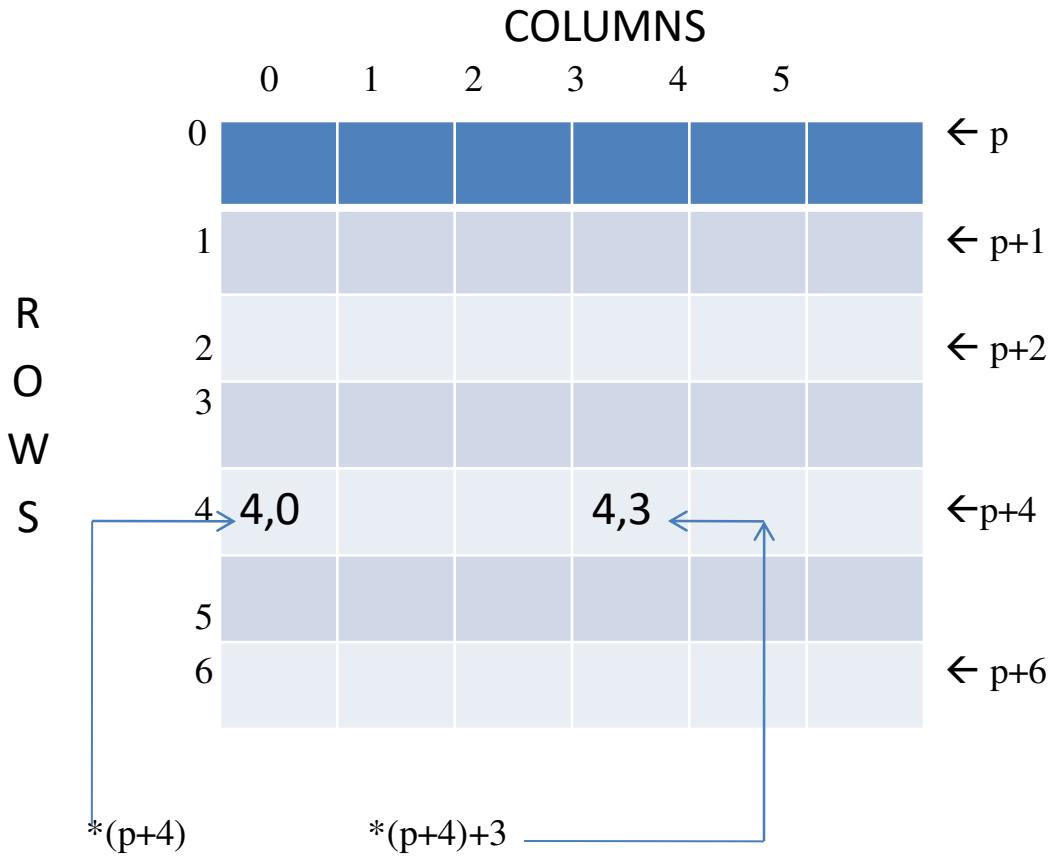
Example :2

```
#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)



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

```
#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) );
     }
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
```

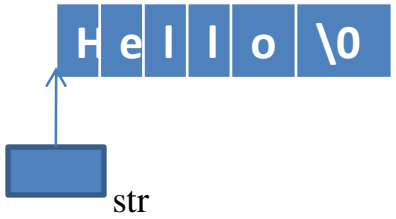
11.9 POINTERS AND CHARACTER STRINGS

- 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



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

char *str;

str= “hello”;

```
#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.

```
#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
```

UNIT – V

CHAPTER XII

FILES

12.1 INTRODUCTION

- **File Handling** is the storing of data in a file using a program.
- the programs store results, and other data of the program to a file using *file handling* in C.
- Also, we can extract/fetch data from a file to work with it in the program.

File operations

1. Naming a file
2. Opening an existing file
3. Reading data from an existing file
4. Writing data to a file
5. Closing the file

Two ways to perform the operations in c.

1. low-level i/o and uses UNIX
2. High-level i/o and uses functions in C's standard library.

High-level I/O functions

Function	Description
fopen()	function is used to create a new file or open an existing file in C.
fclose()	Closes as a file
fprintf ( )	write data into a file
fscanf ( )	read data from a file
putc ( )/ fputc()	write a character into a file
getc ( ) /fgetc()	read a character from a file
putw ( )	write a number into a file
getw ( )	read number from a file
fputs ( )	write a string into a file
fgets ( )	read a string from a file
fread()	read an entire record from a file
fwrite()	write an entire record into a file

12.2 DEFINING AND OPENING A FILE

- To store data in a file in the secondary memory, we must specify

1. Filename
2. Data structure
3. Purpose

- *file_name* – It is a string that specifies the name of the file that is to be opened or created using the fopen method.

  It may contain two parts

  A primary name and an optional period with the extension.

- Data structure of a file is defined as FILE in the library of standard I/O functions definition.
- Purpose-what we want to do with the file.

Example

*FILE *fp;*

fp = fopen("file_name", "mode");

1. fp as a file pointer to the data type FILE.
2. Filename- the file opened in the named filename and assign an identifier to the FILE type pointer fp.
3. Mode- It is a string (usually a single character) that specifies the mode in which the file is to be opened.

"r" – open for reading

"rb" – open for reading in binary mode

"w" – open for writing only

"wb" – open for writing in binary mode

"a" – open for append only

"ab" – open for append in binary

"r+" – open for reading and writing both

"rb+" – open for reading in binary mode

"w+" – open for writing and reading

"wb+" - open for writing and reading in binary mode

"a+" – open for read and append

"ab+" – open for read and append in binary

12.3 CLOSING A FILE

- A file must be closed as soon as the all operations are completed.

  fclose(file_pointer);

- Close file associated with the FILE pointer file_pointer.

- Once the file is closed, its file pointer can be reused for another file.
- All files are closed automatically whenever a program terminates.

Example

```
----------
----------
FILE *fp1,*fp2;
fp1=fopen("INPUT","w");
fp2=fopen("OUTPUT","r");
----------
----------
fclose(fp1);
fclose(fp2);
```

12.4 INPUT/OUTPUT OPERATIONS ON FILES

The getc() and putc() function

- The file is opened with mode w and filepointer fp1.

  putc(c,fp1);

- Writes a character contained in the character variable c to the file associated with FILE pointer fp1.

  c=getc(fp2);

- Reads a character from the file whose file pointer is fp2.
- The file pointer moves by one character position for every operations of getc() and putc().

Example

```
#include <stdio.h>
int main()
{
FILE *fp;
 char c;
  printf("Data Input\n");
   fp=fopen("INPUT.TXT","w");
  while ((c=getchar())  != EOF)
      putc(c,fp);
 fclose(fp);
 printf("Data Output\n");
 fp=fopen("INPUT.TXT","r");
  while ((c=getc(fp))!=EOF)
         printf("%c",c);
 fclose(fp);
   return 0;
}
```

get() and putw() functions

- Integer oriented functions.
- Used to read and write integer values.

putw(integer,fp);

integer_variable=getw(fp);

Example

```
#include <stdio.h>
int main()
{
FILE *f1;
 int n,i;
 printf("Data file\n");
  f1=fopen("DATA.TXT","w");
for (i=1;i<=10;i++)
   {
   scanf("%d",&n);
   if(n== -1) break;
    putw(n,f1);
   }      .
  fclose(f1);
printf("Data Output\n");
 f1=fopen("DATA.TXT","r");
while ((n=getw(f1))!=EOF)
    printf("%d",n);
 fclose(f1);
  return 0;
}
```

Output

Data file

1

2

3

4

5

6

-1

Data Output

1 2 3 4 5 6

fscanf() and fprintf() functions

- These functions are used performed I?O operations on files.

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

  fscanf(fp,control string",list);

- Where fp is a file pointer associate with a file that has been opened for writing.
- The control string contains output specifications for the items in the list.
- The list may include variables, constants and strings.

Example

```
#include <stdio.h>
int main()
{
FILE *fp;   int n,qty,i;
 float price,value;
 char item[10],filename[10];
 printf("Input file name\n");
 scanf("%s",filename);
 fp=fopen(filename,"w");
 printf("input inventory details\n");
 printf("Item Name   no  price   Quantity\n");
for(i=1;i<=3;i++)
 {
  scanf("%s  %d %f  %d", item, &n, &price, &qty);
  fprintf(fp,"%s  %d %f %d",item,n,price,qty);
  }
fclose(fp);
   printf("Data Output\n");
   fp=fopen(filename,"r");
   printf("Item Name   no  price   Quantity\n");
   for(i=1;i<=3;i++)
    {
      fscanf(fp,"%s  %d %f %d",item,&n,&price,&qty);
      printf("%s  %d %f %d\n",item,n,price,qty);
  }
fclose(fp);
return 0;
}
```

Output

Output

Input file name
pen
input inventory details
Item Name no price Quantity

note
1
50.50
100
pencil
2
56.90
290
paper
3
120.00
500
Data Output
Item Name no price Quantity

note 1 50.500000 100
pencil 2 56.900002 290
paper 3 120.000000 500

12.5 ERROR HANDLING DURING I/O OPERATIONS

Error situations

1. Trying to read beyond the end-of-file mark.
2. Device overflow
3. Trying to use a file that has not been opened.
4. Trying to perform an operations on a file, when the file is opened for another type of operations
5. Opening a file with an invalid filename
6. Attempting to write to a write-protected file.

- foef-use to test for an end of file condition.
- ferror-reports status of the file indicated.

Example

```
#include <stdio.h>
int main()
{
 char *filename;
   FILE *fp1,*fp2;
 int i,n;
 fp1=fopen("TEST.TXT","w");
for (i=10;i<=100;i+=10)
    putw(i,fp1);
 fclose(fp1);
  printf("Input filename");
s:scanf("%s",filename);
if ((fp2=fopen(filename,"r"))== NULL)
    {
   printf("Cannot open the file\n");
    printf("Try again\n");
     goto s;
   }
Else
  for(i=10;i<=100;i+=10)
        {
         n=getw(fp2);
if (feof(fp2))
         {
           printf("Run out of data\n");
           break;
         }
         else              .
      printf("%d\n",n);
         }
fclose(fp2);
       return 0;
}
```

Output
Input filenametet
Cannot open the file
Try again
TEST.TXT

10
20
30
40
50
60
70
80
90
100

12.6 RANDOM ACCESS FILE

- functions for random access file processing.

1. fseek()
2. ftell()
3. rewind()

fseek()

This function is used for seeking the pointer position in the file at the specified byte.

Syntax

fseek(file_ptr, offset, position);

file_ptr :pointer to the file concerned.

offset: is a number or variale of type long.

the no. positions (bytes) to be moved from the location specified by position.

position: is an integer number.

The position can take one of the three values.

The offset may be positive, meaning move forwards, or negative, meaning move backwards

value	meaning
0	Beginning of the file
1	Current position
2	End of the file

.

Statement	Meaning
fseek(fp,0l,0);	Got o the beginning
fseek(fp,0l,1);	Stay at the current position
fseek(fp,0l,2);	Go to the end of the file
fseek(fp,m,0);	Move to (m+1)th byte in the file
fseek(fp,m,1);	Go forward by m bytes
fseek(fp,-m10);	Go backward by m bytes from the current position
fseek(fp,-m,2);	Go backward by m bytes from the end

Example

```
#include <stdio.h>
int main ()
 {
FILE *fp;
  int c;
 fp = fopen("OUTPUT.txt","w+");
  fputs("Random Access file example", fp);
fseek( fp, 0, SEEK_SET );
  printf("Random Access file contains\n");
  while(1)
 {
 c = fgetc(fp);
   if( feof(fp))
 {
  break;
   }
  printf("%c", c);
  }
fputs("\nC Programming E Balagurusamy", fp);
fputs("\nsecond edition", fp);
printf("\nThe current position of the file
      pointer is: %ld\n", ftell(fp));
rewind(fp);
 printf("The current position of the file pointer
      is: %ld\n", ftell(fp));
 printf("After rewrite the contents\n");
printf("Read the Random Access File\n");
 while(1)
  {
  c = fgetc(fp);
 if( feof(fp))
 {
 break;
  }
printf("%c", c);
  }
 fclose(fp);
 return(0);
}
```

Output

Random Access file contains

Random Access file example

The current position of the file pointer is: 70

The current position of the file pointer is: 0

After rewrite the contents

Read the Random Access File

Random Access file example

C Programming E Balagurusamy

second edition

12.7 COMMAND LINE ARGUMENTS

What is a command line argument?

- It is a parameter supplied to a program when the program is invoked.
- This parameter may represent a filename the program should process.
- The command line arguments are handled using main() function arguments where **argc** refers to the number of arguments passed, and **argv[]** is a pointer array which points to each argument passed to the program.
- In order to access the command line arguments, we must declare the main() function and its parameters are

```
main(int argc,char *argv[])
{
-------------
-------------
}
```

```
#include <stdio.h>
void main( int argc, char *argv[] )
{
FILE *fp;
int i;
Char word[15];
fp=fopen(argv[1],"w");
printf("No. of arguments in command line =%d\n",argc);
for(i=2;i<argc;i++)
{
 fprintf(fp,"%s",argv[1]);
}
fclose(fp);
printf("contents of %s file \n",argv[1]);
fp=fopen(argv[1],"r");
for(i=2;i<argc;i++)
{
fscanf(fp,"%s",word);
printf("%s",word);
}
fclose(fp);
printf("\n\n");
for(i=0;i<argc;i++)
 printf("%s\n",argv[i]);
}
```

References

1. 1. E. Balagurusamy, ”Programming in ANSI C”, Seventh Edition, McGraw Hill Education India Private Ltd, 2017.
2. . https://www.tutorialspoint.com/cprogramming/
3. https://ecomputernotes.com/what-is-c/function-a-pointer/advantages-of-using-pointers
4. https://www.includehelp.com/c/accessing-the-value-of-a-variable-using-pointer-in-c.aspx
5. https://www.programiz.com/c-programming/c-pointers-arrays
6. https://www.studytonight.com/c/pointer-with-function-in-c.php