C and Data Structures

E Balagurusamy

Vice Chancellor
Anna University, Chennai

NOTE TO THE USER

This CD-ROM contains chapter-wise program examples with source code. These are in addition to those given in the text. We hope the user finds them useful in enhancing his understanding of the text and also in developing his programming skills using C.

INTRODUCTION

This is an example for a simple substitution macro.

```
/************************************************************/
/* simple macro substitution */
/************************************************************/

#define AND &&
main()
{
  char a,b;

  a = 'A';
  b = 'A';

  if (a AND b) printf("\nSame");
}
```

This is an example for a simple substitution macro. This macro has a C function printf() in its body.

```
/************************************************************/
/* simple macro substitution with function */
/************************************************************/

#define AND &&
#define MESSAGE printf("\nSame...")
main()
{
  char a,b;

  a = 'A';
  b = 'A';

  if (a AND b) MESSAGE;
}
```

This is an example for a macro with argument. It calculates the area of a circle.

```
/************************************************************/
/* macro with arguments */
/************************************************************/

#define AREA(r) (3.14 * r * r)
main()
{
  float r;

  printf("\nEnter radius : ");
  scanf("%f",&r);
  printf("\nArea is : %f",AREA(r));

}
```

This is an example for a macro with arguments containing relational operators and nesting of macro. It checks whether the input value is a number or alphabet.

```
/**************************************************************/
/* macro with arguments - relational operators and nesting */
/**************************************************************/

#define AND &&
#define ISDIGIT(a) (a >= 48 AND a <= 57)
main()
{
  char r;

  printf("\nEnter a single character or number: ");
  r = getchar();
  if ISDIGIT(r)
  {
   printf("\nYou entered a number");
  }
  else
  {
   printf("\nYou entered an alphabet");
  }
}
```

CHAPTER 3
CONSTANTS, VARIABLES, AND DATA TYPES

1. Program shows the usage of the size of keyword in C to get the storage space used by various data types. Note that the results are different for a 16-bit environment (such as DOS) and 32-bit environment (such as Windows).

```
/****************************************************************/
/* Program to demonstrate the storage space used by data types */
/****************************************************************/

main()
{
   printf("\nStorage space (in bits) for various data types (32-bit environment)");
    printf("\n--------------------------------------------------------------------");
    printf("\nint                %5d",sizeof(int)*8);
    printf("\nsigned int         %5d",sizeof(signed int)*8);
    printf("\nunsigned int       %5d",sizeof(unsigned int)*8);
    printf("\nshort int          %5d",sizeof(short int)*8);
    printf("\nsigned short int   %5d",sizeof(signed short int)*8);
    printf("\nunsigned short int %5d",sizeof(unsigned short int)*8);
    printf("\nlong int           %5d",sizeof(long int)*8);
    printf("\nsigned long int    %5d",sizeof(signed long int)*8);
    printf("\nunsigned long int  %5d",sizeof(unsigned long int)*8);
    printf("\nfloat              %5d",sizeof(float)*8);
    printf("\ndouble             %5d",sizeof(double)*8);
    printf("\nlong double        %5d",sizeof(long double)*8);
    printf("\nchar               %5d",sizeof(char)*8);
    printf("\nunsigned char      %5d",sizeof(unsigned char)*8);
    printf("\n--------------------------------------------------------------------");
}
```

Continued in the next page

```
Storage space (in bits) for various data types (16-bit environment)
--------------------------------------------------------------------
int                   16
signed int            16
unsigned int          16
short int             16
signed short int      16
unsigned short int    16
long int              32
signed long int       32
unsigned long int     32
float                 32
double                64
long double           80
char                   8
unsigned char          8
```

```
Storage space (in bits) for various data types (32-bit environment)
--------------------------------------------------------------------
int                  32
signed int           32
unsigned int         32
short int            16
signed short int     16
unsigned short int   16
long int             32
signed long int      32
unsigned long int    32
float                32
double               64
long double          96
char                  8
unsigned char         8
--------------------------------------------------------------------
```

2. Program shows the usage of the char variable. Note that we have assigned the character variable a, the value ‘A’. When we print the value of the variable a as integer (%d), we get 65 as result, which is the ASCII code of the value of the variable. However, when we print the value as a character (%c), we get ‘A’ as the result. Also, we are able to increment the variable a as if it is a numeric variable. In this case, we get the next ASCII value into the variable which is 66 or ‘B’.

```
/****************************************************/
/*  Program to demonstrate usage of char data type */
/****************************************************/

main()
{
  char a;
  a = 'A';
  printf("\nValue of a is %c",a);
  printf("\nValue of a is %d",a);
  a = a + 1;
  printf("\nValues after incrementing\n");
  printf("\nValue of a is %c",a);
  printf("\nValue of a is %d",a);

}

Output

Value of a is A
Value of a is 65
Values after incrementing

Value of a is B
Value of a is 66
```

3. Program shows the usage of the typedef keyword. We have defined two data types of our own. One is an unsigned short int and another is a string data type. In C, we declare a string as a character array, which we will see later. Here, we have defined char* as a STRING data type. The exact meaning of char* will be seen later.
   In the program, we have assigned a huge value which cannot be contained in a short int. Hence, we get a junk result.

```
/*********************************************************/
/* PROGRAM TO SHOW USAGE OF TYPEDEF keyword             */
/*********************************************************/

typedef unsigned short int UNSHORTINT;
typedef char* STRING;

main()
{
 short int y = 56765;
 UNSHORTINT x = 56765;
 STRING s = "Test";

 printf("String is  %s Number is %d\n", s, y);
 printf("String is  %s Number is %d\n", s, x);
}

Output

String is  Test Number is -8771
String is  Test Number is 56765
```

4. Program shows the usage of the enum keyword. The program shows how the enum keyword assigns the first variable (MATHS) a value 2 and subsequent variable (PHYSICS) automatically a value 3. Also, look at the way we break this increment by starting another sequence with CHEMISTRY.

```
/****************************************************/
/* PROGRAM TO SHOW THE USAGE OF enum keyword        */
/****************************************************/

main()
{
 enum {MATHS = 2, PHYSICS, CHEMISTRY = 7, BOTANY, ZOOLOGY};

 printf("\nMATHS = %d\n", MATHS);
 printf("\nPHYSICS = %d\n", PHYSICS);
 printf("\nCHEMISTRY = %d\n", CHEMISTRY);
 printf("\nBOTANY = %d\n", BOTANY);
 printf("\nZOOLOGY = %d\n", ZOOLOGY);
}

Output

MATHS = 2

PHYSICS = 3

CHEMISTRY = 7

BOTANY = 8

ZOOLOGY = 9
```

5. Program shows the usage of the enum keyword. In this program we have extended the usage of the enum keyword. We have declared a variable to be of enum type. Note how we can use the individual components of the enumerated variable.

```
/*****************************************************/
/* PROGRAM SHOWING USAGE OF ENUM VARIABLE            */
/*****************************************************/

main()
{
 enum subject {MATHS = 2, PHYSICS, CHEMISTRY = 7, BOTANY, ZOOLOGY};

 enum subject mysub;

 mysub = MATHS + PHYSICS;

 printf("\nTotal credit for my subjects is %d", mysub);
 printf("\n\nZOOLOGY has a credit of %d", ZOOLOGY);

}

Output

Total credit for my subjects is 5

ZOOLOGY has a credit of 9
```

CHAPTER 4
OPERATORS AND EXPRESSIONS

1. We will now write a program that will accept a four-digit number and find the sum of its digits. The principle used in this program is the modulo operator, which yields a reminder when one number is divided by another number and integer division gives only an integer as result (quotient), truncating the decimal portion. You can extend this program using loops to find the sum of any number.

```
/**********************************************************************/
/*              PROGRAM TO FIND THE SUM OF DIGITS                     */
/**********************************************************************/
main()
{
 int number;
 int f1,f2,f3,f4,sum;

 printf("Enter a four digit number whose digits are to be added : ");
 scanf("%d",&number);

 f1 = number / 1000; /* integer division to extract first digit */
 f2 = number % 1000; /* extracts the last three digits */
 f2 = f2 / 100; /* integer division to extract first digit */
 f3 = number % 100; /* extract last two digits only */
  f3 = f3 / 10;
 f4 = number % 10;
  sum = f1 + f2 + f3 + f4;
 printf("\nSum of digits of %d is %d",number,sum);
 }

Output

Enter a four digit number whose digits are to be added : 2398
Sum of digits of 2398 is 22

Enter a four digit number whose digits are to be added : 9876
Sum of digits of 9876 is 30
```

2. Let us understand through another simple program, the result of expressions where we declare the variables as float. Note that when one of the operands is turned to be a float, we get a proper result.

```
/**********************************************************/
/*          FLOATING POINT DIVISION ILLUSTRATION          */
/**********************************************************/

main()
{
 float n1,n2,n3,n4,n5,n6,n7;

 n1 = 11/15;
 n2 = 11/15.0;
 n3 = 11.0/15;
 n4 = 11.0/15.0;
 n5 = 11/(float)15;
 n6 = 11.0/(float)15;
```

```
  n7 = (float)(11/15);

  printf("Operands are 11 and 15");
  printf("\nInteger over integer is %f",n1);
  printf("\nInteger over float is %f",n2);
  printf("\nFloat over integer is %f",n3);
  printf("\nFloat over float is %f",n4);
  printf("\nInteger over (floatcast)integer is %f",n5);
  printf("\nfloat over (floatcast)integer is %f",n6);
  printf("\n(floatcast)Integer over integer is %f",n7);
}

Output

Operands are 11 and 15
Integer over integer is 0.000000
Integer over float is 0.733333
Float over integer is 0.733333
Float over float is 0.733333
Integer over (floatcast)integer is 0.733333
float over (floatcast)integer is 0.733333
(floatcast)Integer over integer is 0.000000
```

3. Let us understand the shift operators through a simple program. When we right shift a value, a zero is added in the left of the binary equivalent of the number. This is indirectly dividing the number by 2. When we left shift a number, we indirectly multiply the number by 2. Also, when an odd number is shifted, the result is always rounded. For example, 12 >> 2 gives 3 (this is 12 / 2 = 6; 6 / 2 = 3 – one division by 2 for each shift), whereas 13 >> 1 will give only 6 (13 / 2 = 6 – decimal portion is truncated).

```
/*****************************************************/
/*                SHIFT OPERATORS                  */
/*****************************************************/

main()
{
   int a,b;

   printf("Enter a number (integer) : ");
   scanf("%d",&a);

   b = a >> 2;
   printf("\nValue when right shifted by 2 is %d",b);

   b = a << 3;
   printf("\nValue when left shifted by 3 is %d",b);
}

Output

Enter a number (integer) : 12
Value when right shifted by 2 is 3
Value when left shifted by 3 is 96

Enter a number (integer) : 13
Value when right shifted by 2 is 3
Value when left shifted by 3 is 104
```

4. Let us now understand the XOR (exclusive OR) operator. This is widely used for encoding and decoding purposes. When you XOR string (or character) with a character (this is called the key), you get a result. This is an encoded form. When you again XOR the result (encoded form) with the same key, you will get back the original character. If you XOR the encoded string with a different key, you will not get back the original string. The program shown below illustrates this concept.

```
/***********************************************************/
/*      USAGE OF XOR FOR ENCODING and DECODING          */
/***********************************************************/

main()
{
  char a = 'A';
  char b = '*';
  char c = '/';
  char d;

  d = a ^ b;
  printf("\n\n%c when encoded using %c gives %c",a,b,d);
  a = d ^ b;
  printf("\n\n%c when decoded using %c gives %c",d,b,a);

  a = d ^ c;
  printf("\n\n%c when decoded using %c gives %c",d,c,a);
}

Output

A when encoded using * gives k

k when decoded using * gives A

k when decoded using / gives D
```

CHAPTER 5
MANAGING INPUT AND OUTPUT OPERATIONS

1. When dealing with string input and output, the scanf and prinf will not be handy when the string has an embedded space character. For example, you may not be able to input the string “Programming Language”. If you do so, only the string “Programming” will be stored in the variable and the string “Language” will be left out. To avoid this, C has two other functions—gets() and puts(). Look at the program given. Note that printf() is capable of printing the string with embedded spaces.

```
/*******************************************************/
/*         STRING INPUT and OUTPUT                     */
/*******************************************************/

main()
{
 char mystring[100];

 printf("\nEnter a string : ");
 gets(mystring); /* gets is string input */
 printf("\nYou entered %s\n",mystring);

 puts("You entered");
 puts(mystring); /* puts is string output */
}

Output

Enter a string : programming language
You entered programming language
You entered
programming language
```

2. C supports two other indirect input and output functions. These are sprintf() and sscanf(). The sprintf() function behaves exactly same as the printf() function except that instead of sending the result to the screen, the sprintf() function stores the result in a string variable. The sscanf() function works in the reverse way of sprintf(), i.e. it reads from a string variable (as opposed to scanf() function which reads from keyboard) and stores the values in different variables. Look at the program given. The input variables a, b, c and d are stored in mystring through fprintf(). The sscanf() reads the mystring and distributes the values to the variables x, y, m and n.

```
/**********************************************************/
/* PROGRAM TO SHOW USAGE OF sprintf() and sscanf()        */
/**********************************************************/

main()
{
 int a,b,x,y;
 char resultstring[50];
 float c,d,m,n;

 printf("\nEnter two integers (separated by space): ");
```

```
  scanf("%d %d",&a,&b);
  printf("\nEnter two floats (separated by space): ");
   scanf("%f %f",&c,&d);

  sprintf(resultstring,"%d %d %f %f",a,b,c,d); /* sprintf adds to string */
  printf("Full string is : %s",resultstring);

  sscanf(resultstring,"%d %d %f %f",&x,&y,&m,&n); /* sscanf reads from string */
  printf("\nx=%d \ny=%d \nm=%f \nn=%f",x,y,m,n);
}

Output

Enter two integers (separated by space): 12 45
Enter two floats (separated by space): 1.23 5.67
Full string is : 12 45 1.230000 5.670000
x=12
y=45
m=1.230000
n=5.670000
```

3. Sometimes it may be desirable to just press a character without any Enter key pressed after that. The function getchar() requires you to press Enter key after you type a character. Also, the getchar() will allow you to enter more than one character before pressing Enter key, but will take into account the first character only. C provides two other single character input functions—getch() and getche(). They will exactly take only one character as input and also they do not require you to press the Enter key. The function getche() will echo what you press while getch() will not show on the screen. You can use putch() or putchar() to display the value. Look at the program.

```
/***************************************************************/
/*      PROGRAM TO ILLUSTRATE getch() and getche()           */
/***************************************************************/

main()
{
  char mychar;

  printf("Press a key (this will echo): ");
  mychar = getche();
  printf("\nYou pressed ");
  putch(mychar);

  printf("\nPress a key (this will not echo): ");
  mychar = getch();
  printf("\nYou pressed ");
  putch(mychar);
}

Output

Press a key (this will echo): A
You pressed A
Press a key (this will not echo):
You pressed K
```

CHAPTER 7
DECISION MAKING AND LOOPING

1. We will now extend the program to find the sum of digits of a given number. The program uses a while loop to keep track of the number of the digit being added. The program also uses a function (which we will study later) to compute the powers of 10. This function (called power10) also uses a while loop. The program has a sprintf() to convert the number to a string so that its length can be got.

```
/*****************************************************************/
/* Sum of digits - PROGRAM TO ILLUSTRATE THE USAGE OF while loop */
/*****************************************************************/

#include <string.h>
main()
{
 int number,digit,len,sum;
 char n[20];
 int power10();
 printf("\nEnter a number whose digits are to be added : ");
 scanf("%d",&number);
 sprintf(n,"%d",number); /* convert to string to find number of digits */
 len = strlen(n); /* find the number of digits */
 len = len - 1; /* start off with a digit lesser */

 /* get first digit alone outside the loop */
 digit = number / power10(len);
 sum = digit;
 while (len > 0)
 {
  digit = number % power10(len);
  len = len - 1;
  digit = digit / power10(len);
  sum = sum + digit;
 }
 printf("\n Sum of digits of %d is %d",number,sum);
}

/* function to find the powers of 10 */
int power10(n)
int n;
{
  int p = 1;
  while (n > 0)
  {
    p = p * 10;
    n = n - 1;
  }
  return(p);
}

Output

Enter a number whose digits are to be added : 24578
 Sum of digits of 24578 is 26

Enter a number whose digits are to be added : 98765
 Sum of digits of 98765 is 35

Enter a number whose digits are to be added : 99999
 Sum of digits of 99999 is 45
```

2. We will now develop a program to find the square root of a number using the iterative Newton-Raphson method. The method is based on the following:

$$X_{curr} = 0.5 * (X_{prev} + number / X_{prev}) \text{ where } X_{prev\ (initial)\ =\ 1.0}$$

The second of the above expression will be executed in a loop till X_{prev} and X_{curr} matches. The value of either of the variables is the square root of the given number. We will use a do…while loop.

```
/*****************************************************************/
/*  PROGRAM TO COMPUTE SQUARE ROOT USING NEWTON-
RAPHSON METHOD   */
/*  PROGRAM ILLUSTRATES THE USAGE OF THE DO...WHILE
LOOP        */
/*****************************************************************/
main()
{
 float  xprev,xcurr,number;
 printf("\nEnter the number whose square root is required : ");
 scanf("%f",&number);
 xcurr = 1.0;
 do
 {
   xprev = xcurr;
   xcurr = 0.5 * (xprev + (number / xprev));
   printf("\n%f  %f",xcurr,xprev);
 }
 while(xprev != xcurr);
 printf("\n\nThe square root of the number is %f\n",xcurr);
}
Output
Enter the number whose square root is required : 4
2.500000  1.000000
2.050000  2.500000
2.000610  2.050000
2.000000  2.000610
2.000000  2.000000
The square root of the number is 2.000000
```

3. Let us now write a program to find all the factors of a given number. If the number is a prime number, the program displays that it is prime. The program is written using a **for** loop and uses the initialization using comma operator in the **for** statement itself.

```
/* Program to find the factors of number using for loop     */

main()
{
 int number,i,prime;

  printf("Enter the number whose factors are to be found : ");
  scanf("%d", &number);

  /* notice the use of comma operator in for statement */

  for (prime=1,i=2; i <= (number / 2); i++)
  {
    if ((number % i) == 0)
    {
      if (prime)
    printf("The factors of %d are: \n", number);
      prime = 0;
      printf("%d, ", i);
    }
  }
  if (prime)
    printf("%d is a prime number", number);
}

Output

Enter the number whose factors are to be found : 180
The factors of 180 are:
2, 3, 4, 5, 6, 9, 10, 12, 15, 18, 20, 30, 36, 45, 60, 90,

Enter the number whose factors are to be found : 78
The factors of 78 are:
2, 3, 6, 13, 26, 39,

Enter the number whose factors are to be found : 13
13 is a prime number
```

4. We will look at one more example using nested for loop. We will find out the list of numbers that are perfect within a given range. A perfect number is one whose sum of factors (including 1) is the number itself. For example, 6 = 1 + 2 + 3. The outer for loop enumerates the number range while the inner for loop identifies the factors one by one.

```
/****************************************************************/
/*  Program to illustrate nested for loop and also            */
/*  for loop with different increment                         */
/****************************************************************/

main()
{
 int i,sum,numbercurr;
 int number,endnumber;

 printf("\nEnter the maximum number upto which search will go on : ");
 scanf("%d",&endnumber);

 /* you need not search the odd numbers..they will never be perfect */

 for (number = 2 ; number < endnumber ; number = number + 2)
 {
   sum = 0;
   numbercurr = number;
   for (i = 1; i < numbercurr; i++)
   {
     if (number % i == 0)
     {
                               sum = sum + i;

      /* in this portion we find the corresponding other factor */

                     if (number / i != number)
      {
                   sum = sum + (number / i);
                    numbercurr = number / i;
                                              }
     }
   }

   if (sum == number)
    printf("%d is a perfect number\n", number);
 }
}
```

Output

```
Enter the maximum number upto which search will go on : 1000

6 is a perfect square
28 is a perfect square
496 is a perfect square
```

5. We will write a simple program that will demonstrate the usage of the break keyword. The program will accept a number and identify if it is a prime or composite number. Even if one factor is found, the program will break the **for** loop.

```
/*****************************************************/
/* PROGRAM TO SHOW USAGE OF break                   */
/*****************************************************/

main()
{
 int i, number, flag;

 printf("\nEnter a number to be tested : ");
 scanf("%d",&number);

 for (i = 2; i <= number / 2; i++)
 {
   flag = 0;
   if (number % i == 0)
   {
     flag = 1;
     break;
   }
 }
 if (flag == 1)
    printf("%d is a composite number",number);
 else
    printf("%d is a prime number",number);
}

Output

Enter a number to be tested : 13
13 is a prime number

Enter a number to be tested : 15
15 is a composite number
```

6. Binary to Decimal Converter—This program will accept a binary integer number and convert to its equivalent decimal integer number. The core logic is the separation of the digits in the input number.

```
/*****************************************************************/
/* Conversion of Binary integer to Decimal integer              */
/*****************************************************************/

#include <string.h>
main()
{
 int number,digit,len,sum;
 char n[20];
 int power();
 printf("\nEnter a binary number : ");
 scanf("%d",&number);
 sprintf(n,"%d",number); /* convert to string to find number of digits */
 len = strlen(n); /* find the number of digits */
 len = len - 1; /* start off with a digit lesser */

 /* get first digit alone outside the loop */
 digit = number / power(len,10);
 sum = digit * power (len,2); /* raise to power 2 */
 while (len > 0)
 {
   digit = number % power(len,10);
   len = len - 1;
   digit = digit / power(len,10);
   sum = sum + (digit * power (len,2));
 }
 printf("\n Decimal equivalent of %d is %d",number,sum);
}

/* function to find the powers */
int power(n,m)
int n,m;
{
  int p = 1;
  while (n > 0)
  {
    p = p * m;
    n = n - 1;
  }
  return(p);
}

Output

Enter a binary number : 1001
Decimal equivalent of 1001 is 9

Enter a binary number : 1111
Decimal equivalent of 1111 is 15
```

CHAPTER 8
ARRAYS

1. The next program is to find from a list of numbers, the smallest and largest numbers. The array is not sorted. The program should also show the positions of these numbers.

```
/*****************************************************************/
/* PROGRAM TO SHOW THE USAGE OF ONE-DIMENSIONAL ARRAY */
/*****************************************************************/

main()
{
 int i,smallest,largest,smallpos,largepos,count,getval;
 int numbers[100];

 /* Array input */
 printf("\nEnter number of numbers in the list : ");
 scanf("%d",&count);
 for (i=0; i<count; i++)
 {
  printf("\nEnter number[%d] : ",i);
  scanf("%d",&getval);
  numbers[i] = getval;
 }

 /* Array display */
 printf("\nGiven array is : ");
 for (i=0; i<count; i++)
   printf("%d,",numbers[i]);

 smallest = 9999; /* guess for smallest */
 largest = -9999; /* guess for largest */
 smallpos = largepos = 0;

 for (i=0; i<count; i++)
 {
  if (smallest > numbers[i])
   {
     smallest = numbers[i];
     smallpos = i;
   }

  if (largest < numbers[i])
   {
     largest = numbers[i];
     largepos = i;
   }
  }

  printf("\nSmallest number is %d found at position %d",smallest,smallpos);
  printf("\nLargest number is %d found at position %d",largest,largepos);
}
```

```
Output

Enter number of numbers in the list : 5
Enter number[0] : 3
Enter number[1] : 12
Enter number[2] : 56
Enter number[3] : 2
Enter number[4] : 44

Given array is : 3,12,56,2,44,
Smallest number is 2 found at position 3
Largest number is 56 found at position 2
```

2. The next program is to reverse the given array. The program simply reads each array element and swaps with the corresponding mirror element. While swapping, the program uses a temporary variable. The looping must be done only up to half of the array size.

```
/******************************************************************/
/* PROGRAM TO SHOW THE USAGE OF ONE-DIMENSIONAL ARRAY */
/******************************************************************/

main()
{
 int i,count,getval,temp,j;
 int numbers[100];

 /* Array input */
 printf("\nEnter number of numbers in the list : ");
 scanf("%d",&count);
 for (i=0; i<count; i++)
 {
  printf("\nEnter number[%d] : ",i);
  scanf("%d",&getval);
  numbers[i] = getval;
 }

 /* Array display */
 printf("\nGiven array is : ");
 for (i=0; i<count; i++)
   printf("%d,",numbers[i]);

 j = 1;
 for (i=0; i<count/2; i++)
 {
  temp = numbers[i];
  numbers[i] = numbers[count-j];
  numbers[count-j] = temp;
  j = j + 1;
 }
 printf("\nReversed array is : ");
 for (i=0; i<count; i++)
    printf("%d,",numbers[i]);
}

Enter number of numbers in the list : 5
Enter number[0] : 3
Enter number[1] : 12
Enter number[2] : 56
Enter number[3] : 2
Enter number[4] : 44

Given array is : 3,12,56,2,44,
Reversed array is: 44,2,56,12,3
```

3. This program is to find the LCM of two numbers. The program uses two arrays for multiples of each number. An array matching is done to find the common element. Though a greedy method to find LCM is available, this method is given here to highlight the usage of multiple single dimension arrays, array matching and nested for loops.

```
/****************************************************************/
/* PROGRAM TO FIND THE LCM OF TWO GIVEN NUMBERS              */
/* TWO SINGLE DIMENSION ARRAYS ARE USED - ONE FOR EACH MULTIPLE */
/* WHEN A MATCH BETWEEN ARRAYS IS FOUND, LCM IS GOT          */
/****************************************************************/

main()
{
 long int firstarr[100],secondarr[100];
 int firstnum,secondnum,i,j,multiple,flag,lcm;

 printf("\nEnter the first number : ");
 scanf("%d",&firstnum);
 printf("\nEnter the second number : ");
 scanf("%d",&secondnum);

 /* fill the arrays with the multiples */
 multiple = 0;
 for (i=0; i<100; i++)
 {
  multiple = multiple + firstnum;
  firstarr[i] = multiple;
 }

 multiple = 0;
 for (i=0; i<100; i++)
 {
  multiple = multiple + secondnum;
  secondarr[i] = multiple;
 }

 flag = 0;
```

```
  for (i=0; i < 100; i++)
  {
   for (j=0; j < 100; j++)
     if (firstarr[i] == secondarr[j])
     {
        flag = 1;
        lcm = firstarr[i];
        break;
     }
   if (flag == 1) break;
   }

   if (flag == 1)
    printf("\nLCM of %d and %d is : %d",firstnum,secondnum,lcm);
   else
    printf("\nLCM not lying within 100 numbers range");
}

Output

Enter the first number : 56
Enter the second number : 64
LCM of 56 and 64 is : 448
```

4. This program is to illustrate the usage of two-dimensional arrays to hold matrices. The program accepts two matrices and performs their addition.

```
/*******************************************************************/
/*  PROGRAM TO ADD TWO MATRICES                                  */
/*******************************************************************/


main()
{
   int arows,acols,brows,bcols,i,j,a[20][20],b[20][20],c[20][20];

           printf("\nEnter the number of rows in first matrix : ");
    scanf("%d",&arows);
           printf("\nEnter the number of columns in first matrix : ");
    scanf("%d",&acols);
           printf("\nEnter the number of rows in second matrix : ");
    scanf("%d",&brows);
           printf("\nEnter the number of columns in second matrix : ");
    scanf("%d",&bcols);

           if (arows == brows && acols == bcols)
    {
       ;
    }
    else
           {
                     printf("Addition of these two matrices is not possible" );
                     exit(0);
           }
   /* get the matrix */
          for(i=0; i < arows; i++)
            for(j=0; j < acols; j++)
      {
       printf("\nEnter element at row %d col %d for matrix A",i,j);
                      scanf("%d",&a[i][j]);
             }

          for(i=0; i < brows; i++)
            for(j=0; j < bcols; j++)
      {
       printf("\nEnter element at row %d col %d for matrix B",i,j);
                      scanf("%d",&b[i][j]);
             }

   /* perform addition */

          for(i=0; i < arows; i++)
           for(j=0;j < acols; j++)
                     c[i][j] = a[i][j] + b[i][j];

   /* print the matrices */

          printf("\nMatrix A is : \n");
          for(i=0; i < arows; i++)
    {
           for(j=0;j < acols; j++)
       printf("%d ",a[i][j]);
     printf("\n"); /* this is required for row break */
    }

          printf("\nMatrix B is : \n");
          for(i=0; i < brows; i++)
    {
           for(j=0;j < bcols; j++)
       printf("%d ",b[i][j]);
     printf("\n"); /* this is required for row break */
    }

   printf("\nMatrix sum is : \n");
          for(i=0; i < arows; i++)
    {
           for(j=0;j < acols; j++)
       printf("%d ",c[i][j]);
     printf("\n"); /* this is required for row break */
    }
}
```

Output

```
Enter the number of rows in first matrix : 2
Enter the number of columns in first matrix : 2
Enter the number of rows in second matrix : 2
Enter the number of columns in second matrix : 2
Enter element at row 0 col 0 for matrix A 2
Enter element at row 0 col 1 for matrix A 3
Enter element at row 1 col 0 for matrix A 4
Enter element at row 1 col 1 for matrix A 5
Enter element at row 0 col 0 for matrix B 7
Enter element at row 0 col 1 for matrix B 8
Enter element at row 1 col 0 for matrix B 9
Enter element at row 1 col 1 for matrix B 10
Matrix A is :
2 3
4 5

Matrix B is :
7 8
9 10

Matrix sum is :
9 11
13 15
```

5. This program will find the transpose of a given matrix. The transpose is got by interchanging the rows and columns of a matrix. Note in the program, how we change the columns to be outer loop and rows to be the inner loop and change the subscript order to print the transpose.

```
/*******************************************************************/
/*  PROGRAM TO FIND MATRIX TRANSPOSE                          */
/*******************************************************************/

main()
{
  int arows,acols,brows,bcols,i,j,a[20][20];

          printf("\nEnter the number of rows in the matrix : ");
  scanf("%d",&arows);
          printf("\nEnter the number of columns in the matrix : ");
  scanf("%d",&acols);

  /* get the matrix */
          for(i=0; i < arows; i++)
            for(j=0; j < acols; j++)
    {
     printf("\nEnter element at row %d col %d ",i,j);
                  scanf("%d",&a[i][j]);
            }

  /* print the matrices */

          printf("\nInput matrix is : \n");
          for(i=0; i < arows; i++)
  {
           for(j=0;j < acols; j++)
      printf("%d ",a[i][j]);
   printf("\n"); /* this is required for row break */
  }
```

```
            printf("\nTranspose matrix is : \n");
            for(i=0; i < acols; i++)     /* note that we changes the rows */
    {
             for(j=0;j < arows; j++)
        printf("%d ",a[j][i]);  /*reverse printing */
     printf("\n"); /* this is required for row break */
    }
}

Output

Enter the number of rows in the matrix : 2
Enter the number of columns in the matrix : 3
Enter element at row 0 col 0 1
Enter element at row 0 col 1 2
Enter element at row 0 col 2 3
Enter element at row 1 col 0 4
Enter element at row 1 col 1 5
Enter element at row 1 col 2 6
Input matrix is :
1 2 3
4 5 6

Transpose matrix is :
1 4
2 5
3 6
```

6. This program is also a matrix related program. The program will compute the determinant of a given square matrix. Since the method is based on pivotal method, the diagonal elements should not be the same number. The matrix is declared a float data type since the pivoted matrix will definitely be a float data type.

```
/******************************************************************/
/* PROGRAM TO FIND THE DETERMINANT OF A SQUARE MATRIX             */
/******************************************************************/

main()
{
 int i,j,k,rows;
 float matrix[20][20];
 float pivot,determinant;

 /* accept the matrix details */
 printf("Enter the numbers of rows in the square matrix : ");
 scanf("%d",&rows);
 printf("\nEnter the matrix elements now :");
 for(i=0; i < rows; i++)
   for(j=0; j < rows;j++)
   {
    printf("\nEnter element at row %d column %d ",i,j);
           scanf("%f",&matrix[i][j]);
   }
 /* begin to compute the determinant */
 for(i=0; i < rows; i++)
 {
         for(j=0; j < rows; j++)
           if(j != i)
           {
             pivot = matrix[j][i] / matrix[i][i];
             for(k=0; k < rows; k++)
                     matrix[j][k]= matrix[j][k] - matrix[i][k] * pivot;
           }
 }

 determinant = 1;
 for(i=0; i < rows; i++)
         determinant = determinant * matrix[i][i];
 printf("The determinant is %6.2f\n",determinant);
}
```

Output

```
Enter the numbers of rows in the square matrix : 3
Enter the matrix elements now :
Enter element at row 0 column 0 2
Enter element at row 0 column 1 3
Enter element at row 0 column 2 5
Enter element at row 1 column 0 1
Enter element at row 1 column 1 2
Enter element at row 1 column 2 3
Enter element at row 2 column 0 3
Enter element at row 2 column 1 1
Enter element at row 2 column 2 3
The determinant is  -1.00
```

CHAPTER 10
USER-DEFINED FUNCTIONS

1. This program demonstrates a very simple function. It calls another function without any arguments and the function also does not return any value. A more complicated no return no argument function is in the next example.

```
/***************************************************/
/* a simple function - no arguments and return values */
/***************************************************/

main()
{
 printf("\nAbout to call a function");
 callfunction();
 printf("\nReturned from the function to main");
}

callfunction()
{
 printf("\nI am now in the called function");
}

Output

About to call a function
I am now in the called function
Returned from the function to main
```

2. This program demonstrates a function call with arguments passed to the function. The function gets two arguments from the main program and finds their maximum and displays the result.

```
/***************************************************/
/* a simple function - with arguments and no return values */
/***************************************************/

main()
{
 int a,b;
 printf("\nEnter the first number : ");
 scanf("%d",&a);
 printf("\nEnter the second number : ");
 scanf("%d",&b);
 findmax(a,b);
}

findmax(x,y)
int x,y;
{
 int c; /* this is a local variable */
 c = (x < y) ? y : x;
 printf("\nThe maximum is : %d",c);
}

Output

Enter the first number : 10
Enter the second number : 5
The maximum is : 10
```

3. This program demonstrates a function call with arguments passed to the function. The function also returns a result back to the main program. We will pass an integer to the function which will compute the sum from 1 up to the number like 1 + 2 + 3 + … + n.

```
/**************************************************/
/* a simple function - with arguments and with return values */
/**************************************************/

main()
{
  int n,sum;
  printf("\nEnter the number : ");
  scanf("%d",&n);
  sum = findsum(n);
  printf("\nSum is (1+2+3+...+%d) : %d",n,sum);
}

findsum(number)
int number;
{
  int mysum,i; /* these are local variables */
  mysum = 0;
  for (i=1; i <= number; i++)
    mysum = mysum + i;
  return(mysum);
}

Output

Enter the number : 10
Sum is (1+2+3+...+10) : 55
```

4. This program accepts a character and calls a function to convert it to uppercase. The argument data type and return data type are both characters.

```
/********************************************************/
/* passing of character datatype to function */
/********************************************************/

main()
{
  char a,b;
  char convert();
  printf("\nEnter a character to be converted to uppercase : ");
  a=getchar();
  b=convert(a);
  printf("\nUppercase of %c is %c",a,b);
}

char convert(c)
char c;
{
  char d;
  d = toupper(c);
  return(d);
}

Output

Enter a character to be converted to uppercase : a
Uppercase of a is A

Enter a character to be converted to uppercase : m
Uppercase of a is M
```

5. This program demonstrates a nested function call. The main program calls a function. The function in turn calls another function. These kinds of calls are nested function calls.

```
/*********************************************************/
/* NESTED FUNCTION CALLS */
/*********************************************************/

main()
{
  printf("\nThis is main program before calls");
  function_one();
  printf("\nThis is main program after call");
}

function_one()
{
  printf("\nThis is function_one before calls");
  function_two();
  printf("\nThis is function_one after call");
}

function_two()
{
  printf("\nThis is function_two");
}

Output

This is main program before calls
This is function_one before calls
This is function_two
This is function_one after call
This is main program after call
```

6. This program demonstrates the recursive function usage. The program computes the greatest common divisor (GCD) of two integers recursively.

```
int findgcd(int m,int y);
main()
{
          int firstno,secondno,gcd;
          printf("Enter the first number: ");
          scanf("%d",&firstno);
          printf("Enter the second number: ");
          scanf("%d",&secondno);

          if(firstno > secondno)
                    gcd = findgcd(firstno,secondno);
          else
                    gcd = findgcd(secondno,firstno);
          printf("The GCD of %d and %d is: %d",firstno,secondno,gcd);
}

int findgcd(int m,int n)
{
          int z;
          while(n!=0)
          {
              z = m % n;
              m = n;
              n = z;
              findgcd(m,n); /* this is a recursive call */
          }
          return(m);
}

Output

Enter the first number: 8
Enter the second number: 36
The GCD of 8 and 36 is: 4
```

7. This program demonstrates how to pass an array to a function. The program accepts three numbers in an array. The array is passed to a function that computes the average of the three numbers. The arguments are int data types while the result is a float.

```
float findaverage(int myarray[]);

main()
{
 int numbers[3],i;
 float average;

 for(i=0; i < 3; i++)
 {
  printf("\nEnter number[%d] : ",i);
  scanf("%d",&numbers[i]);
 }

 average = findaverage(numbers);
 printf("\nThe sum of the numbers is : %f",average);
}

float findaverage(int myarray[])
{
  int mysum = 0;
  int i;
  float average;

  for(i=0; i < 3; i++)
  {
    mysum = mysum + myarray[i];
  }

  return (mysum / 3.0);
}

Output

Enter number[0] : 1
Enter number[1] : 2
Enter number[2] : 3
The sum of the numbers is : 2.000000
```

8. This program demonstrates another interesting feature of auto. You can declare a variable with same within a function under different blocks. When the block is exited, the variable is destroyed. The program below explains the usage of different data types for auto.
   Assume that we declare a variable as extern. If we use the same variable name within a function, what will happen? The local variable will take precedence over the extern variable. This is demonstrated through the following program. We have defined a variable myvar as extern and used the same in main().

```
/**************************************************************/
/* using auto within program blocks */
/**************************************************************/

main()
{
  auto int myvar = 10;
  {
    auto float myvar = 20.0;
    {
      auto char myvar = 'A';
      printf("\nInnermost myvar (char) is %c",myvar);
    }
    printf("\nMiddle myvar (float) is %f",myvar);
  }
  printf("\nOutermost myvar (int) is %d",myvar);
}

Output

Innermost myvar (char) is A
Middle myvar (float) is 20.000000
Outermost myvar (int) is 10
```

CHAPTER 11
STRUCTURES AND UNIONS

1. This is a simple program that illustrates a structure. We have defined a structure called currency with two members such as rupees and paise.

```
/****************************************************************/
/* RUPEES AND PAISE STRUCTURE */
/****************************************************************/

struct currency
{
 int rupees;
 int paise;
};

main()
{
 struct currency money;
 printf("\nEnter Rupees : ");
 scanf("%d",&money.rupees);
 printf("\nEnter Paise : ");
 scanf("%d",&money.paise);

 printf("\nRs. %d.%2d",money.rupees,money.paise);
}

Output

Enter Rupees : 12
Enter Paise : 35
Rs. 12.35
```

2. This simple program illustrates how to initialize the members of a structure. Look at how different data types are initialized in this example.

```
/****************************************************************/
/* STRUCTURE INITIALIZATION */
/****************************************************************/

main()
{
 struct employee
  {
   int empno;
   char name[20];
   float basic;
  };

 struct employee emp1 = {100,"Muthu",5000.00};
 struct employee emp2 = {101,"Saravanan",6000.00};


 printf("\nEno : %d, Name : %s, Basic :
```

```
%5.2f",emp1.empno,emp1.name,emp1.basic);
 printf("\nEno : %d, Name : %s, Basic : %5.2f",emp2.empno,emp2.name,emp2.basic);

}

Output

Eno : 100, Name : Muthu, Basic : 5000.00
Eno : 101, Name : Saravanan, Basic : 6000.00
```

3. This simple program illustrates how to use a nested structure.

```
/****************************************************************/
/* Nested structures */
/****************************************************************/


 struct date
 {
   int day;
   int month;
   int year;
 };

 struct employee
 {
   int empno;
   char name[20];
   float basic;
   struct date joindate;
 };

main()
{

 struct employee emp1 = {100,"Muthu",5000.00,10,12,2002};

 printf("\nEno : %d, Name : %s, Basic : %5.2f",emp1.empno,emp1.name,emp1.basic);
 printf("\nJoindate %2d.%2d.",emp1.joindate.day,emp1.joindate.month);
 printf("%4d",emp1.joindate.year);

}

Output

Eno : 100, Name : Muthu, Basic : 5000.00
Joindate 10.12.2002
```

4. This is a simple program that illustrates how to use a union. Note that since the members share the same location, the last member's value has overridden other members' values.

```
typedef union date mydate;

union date
{
 int day;
 int month;
 int year;
};

main()
{
  mydate date1;
  date1.day = 15;
  date1.month = 12;
  date1.year = 2002;

  printf("\nDate is %2d.%2d.%4d", date1.day,date1.month,date1.year);
}

Output

Date is 2002.2002.2002
```

5. This is a simple program that illustrates how to use a union. Note that since the members share the same location, the last member's value will overwrite the other members' values (which we saw in the last example). By rearranging the order of usage of the members, you can prevent this. The example illustrates this.

```
typedef union date mydate;

union date
{
 int day;
 int month;
 int year;
};

main()
{
  mydate date1;
  date1.day = 15;
  printf("\nDate is %2d.", date1.day);
  date1.month = 12;
  printf("%2d.", date1.month);
  date1.year = 2002;
  printf("%4d", date1.year);
}

Output

Date is 15.12.2002
```

6. This is a simple program that illustrates an array of union. Note that this is same as a structure variable.

```
/****************************************************************/
/* Arrays in union */
/****************************************************************/

typedef union date mydate;

union date
{
 int day;
 int month;
 int year;
};

main()
{
 int i;
 mydate dates[2];

 dates[0].day = 10;
 dates[0].month = 12;
 dates[0].year = 2002;

 dates[1].day = 11;
 dates[1].month = 12;
 dates[1].year = 2003;

 for(i = 0; i < 2; i++)
 {
  printf("\nYear[%d] is %4d",i,dates[i].year);
 }
}

Output

Year[0] is 2002
Year[0] is 2003
```

7. This simple program illustrates how to place a union inside a structure.

```
struct mystructure
{
 union
 {
  int myval1;
 };
 int myval2;
};

main()
{
 struct mystructure struct1;
```

```
  printf("\nEnter a number (this will goto union) : ");
  scanf("%d", &struct1.myval1);

  printf("\nEnter a number (this will goto structure) : ");
  scanf("%d", &struct1.myval2);

  printf("\nUnion value is %d",struct1.myval1);
  printf("\nStructure value is %d",struct1.myval2);

}

Output

Enter a number (this will goto union) : 12
Enter a number (this will goto structure) : 13
Union value is 12
Structure value is 13
```

8. This simple program illustrates how to use bit fields in structures.

```
main()
{
  struct category
  {
    unsigned book : 4;
    unsigned  fine : 4;
  };

  struct category mycat;

  mycat.book = 2;
  mycat.fine = 3;

  printf("\nBook value is %d",mycat.book);
  printf("\nFine value is %d",mycat.fine);
}

Output

Book value is 2
Fine value is 3
```

CHAPTER 12
POINTERS

1. This simple program illustrates how to represent and access an array element using pointers. Note that the array name itself could be used as a pointer to the array or a separate pointer variable equated to the array can be used as the pointer to the array.

```
/******************************************************/
/* pointer to array - a simple example */
/******************************************************/

main()
{
 int myarray[5] = {10,2,3,4,5};
 int *myptr;

 myptr = myarray;

 printf("Value of myarray[0]: %d\n", myarray[0]);
 printf("Value of myarray[0]: %d\n", *myarray);
 printf("Value of myarray[0]: %d\n", *myptr);
}

Output

Value of myarray[0] : 10
Value of myarray[0] : 10
Value of myarray[0] : 10
```

2. This simple program illustrates how to use pointer arithmetic to access various elements of an array. A more detailed example is given subsequent to this example.

```
/*******************************************************/
/* pointer arithmetic to access array elements */
/*******************************************************/

main()
{
 int *myptr;
 int myarray[5] = {12,14,16,18,20};

 myptr = myarray;

 printf("\nThe pointer is at first element %d", *myptr);
 printf("\nAbout to increment the pointer");
 myptr++;
 printf("\nThe pointer is at next element %d", *myptr);
 printf("\nThis will access the fourth element which is %d", *(myptr+2));
}

Output

The pointer is at first element 12
About to increment the pointer
The pointer is at next element 14
This will access the fourth element which is 18
```

3. This simple program illustrates how to change the values of the array elements using pointers.

```
/*******************************************************/
/* modifying array elements using pointers */
/*******************************************************/

main()
{
 int *myptr;
 int myarray[5] = {12,14,16,18,20};

 myptr = myarray;

 printf("\nThe pointer is at first element %d", *myptr);
 printf("\nAbout to change the value of 2nd element %d",*(myptr+1));
 *(myptr+1) = 78;
 printf("\nThe value of 2nd element is now %d",*(myptr+1));
}

Output

The pointer is at first element 12
About to change the value of 2nd element 14
The value of 2nd element is now 78
```

4. This program is another example for pointer arithmetic. The program will accept an array. The printing of the array and the finding of the largest number in the array are done using pointer arithmetic.

```
main()
{
 int i,number,largest,*myptr,numbers[20];

 printf("\nEnter the number of elements in the array : ");
 scanf("%d",&number);

 for (i=0; i < number; i++)
 {
  printf("\nEnter number[%d] : ",i);
  scanf("%d",&numbers[i]);
 }

 printf("\nThe array is : ");
 myptr = numbers;
 for (i=0; i < number; i++)
    printf("%d, ",*myptr++);

 myptr = numbers;

 largest = 0;
 for (i=0; i < number; i++)
 {
  if (largest < *myptr) largest = *myptr;
  myptr++;
  }

  printf("\nThe largest element is : %d",largest);
}

Output

The array is : 12, 34, 2, 67, 12,
The largest element is : 67
```

5. This program is an illustration for pointer to two-dimensional array. Notice the way the array elements are printed.

```
main()
{
 int matrix[3][3]={{1,2,3},{4,5,6},{7,8,9}};

 int row,col;

 for (row = 0; row <= 2; row++)
 {
   printf("\n");
   for (col = 0; col <= 2; col++)
     printf("%d ",*(*(matrix + row) + col));
 }
}

Output

1 2 3
4 5 6
7 8 9
```

6. This program is an illustration for an array of pointers. Instead of containing the elements, the array has the pointer to various variables. When the variable's value changes the program reflects the change.

```
main()
{
 int *myarray[3];
 int var1 = 10;
 int var2 = 20;
 int var3 = 30;
 int i;

 myarray[0] = &var1;
 myarray[1] = &var2;
 myarray[2] = &var3;

 printf("\nThe array formed is : ");
 for(i = 0; i < 3; i++)
 {
  printf("%d,",*(myarray[i]));
  }

  var1 = 100; var2 = 200; var3 = 300;
  printf("\nThe new array formed is : ");
  for(i = 0; i < 3; i++)
  {
   printf("%d,",*(myarray[i]));
  }
}

Output

The array formed is : 10,20,30,
The new array formed is : 100,200,300,
```

7. This program is an illustration for an array of pointers for strings. It finds the position of the given substring using pointer arithmetic.

```
/************************************************************/
/* find substring position */
/************************************************************/

#include "string.h"
#include "stdio.h"
main()
{
  char *source = "This is the source string";
  char *search = "source";
  char *pos;
  int i;

  pos = strstr(source, search);

  if(pos==NULL)
  {
   printf("\n%s is not found",search);
  }
  else
  {
   i = source - pos;

   if(i < 0) i = -i;

   printf("%s is found at position %d",search,i);
  }
}

Output

Source is found at position 12
```

8. This program is an illustration for an array of pointers for strings.

```
/************************************************************/
/* array of pointers - swap two array elements */
/************************************************************/

main()
{
 char *myarray[] = {"first","second","third","fourth"};
 char *myvar;

 printf("\nBefore swapping %s : %s",myarray[1],myarray[2]);

 myvar = myarray[1];
 myarray[1] = myarray[2];
 myarray[2] = myvar;

 printf("\nAfter swapping %s : %s",myarray[1],myarray[2]);
}

Output

Before swapping second : third
After swapping third : second
```

9. This program is an illustration that explains how to use pointers to make a function return more than one value.

```
main()
{
 int a,b,c;
 int sum,product;

 printf("\nEnter the values separated by space bar");
 scanf("%d %d %d",&a,&b,&c);

 calculate(a,b,c,&sum,&product);

 printf("\nThe sum of the numbers : %d",sum);
 printf("\nThe product of the numbers : %d",product);

}

calculate(int a,int b,int c,int *mysum,int *myprod)
{
 *mysum = a + b + c;
 *myprod = a * b * c;
}

Output

Enter the values separated by space bar 1 2 4
The sum of the numbers : 7
The product of the numbers : 8
```

10. This program is an illustration that explains how to invoke a function using its pointer.

```
main()
{
 int myfunction();
 int (*myfuncptr)();
 int myarg = 10;

 myfuncptr = myfunction;
 (*myfuncptr)(myarg);
}

int myfunction(int param)
{
 printf("\nThis is from the function %d",param);
}

Output

This is from the function 10
```

11. This program is an illustration that explains how to make a function return a pointer. It converts a string to uppercase. Though this can be done in simpler ways, the idea is to explain the concept of returning a pointer from a function.

```
main()
{
 char *upper;
 char *upper_function();

 char input[20],output[20];

 printf("\nEnter a string : ");
 gets(input);
 upper = upper_function(input,output);
 printf("\nGiven string is %s",input);
 printf("\nConverted string is %s",upper);
}

char *upper_function(char *i,char *o)
{
 char *myptr;

 myptr = o;
 while (*i != '\0')
 {
  *o = toupper(*i);
  o++;
  i++;
 }
 *o = '\0';
 return(myptr);
}

Output

Enter a string : testing
Given string is testing
Converted string is TESTING
```

12. This program is an illustration that explains how to pass the pointer of a structure to a function.

```
struct currency
{
 int rupees;
 int paise;
};

main()
{
 struct currency mycurr = {123,25};
 showit(&mycurr);
}

showit(struct currency *myptr)
{
 printf("\nRupees %d.%d",myptr->rupees,myptr->paise);
}

Output

Rupees 100.25
```

13. This program is an illustration that explains how to pass variable number of arguments to a function using pointers. This uses stdarg.h for this purpose.

```
#include "stdarg.h"
main()
{
 int sum;

 sum = computesum(4,2,3,4,5);
 printf("\nSum of the 4 numbers is %d",sum);

 sum = computesum(3,2,3,4);
 printf("\nSum of the 3 numbers is %d",sum);
}

computesum(int no_of_args,...)
{
 int mysum,i,myparam;

 va_list myptr;
 va_start(myptr,no_of_args);
 mysum=0;

 for(i=0; i < no_of_args; i++)
 {
  myparam = va_arg(myptr,int);
  mysum = mysum + myparam;
 }

 return(mysum);
}
```

Output

```
Sum of the 4 numbers is 14
Sum of the 3 numbers is 9
```

CHAPTER 13
FILE MANAGEMENT IN C

1. You can also use fputc() to write to a file and fgetc() to read from a file. This example illustrates these functions.

```
/* writing and reading from a file using fgetc and fputc() */

#include <stdio.h>

main()
{
 FILE *myfile;
 char text[100];
 int i;

 myfile = fopen("example.txt", "w+");

 printf("Enter a sentence to store in the file : ");
 gets(text);

 for(i=0; text[i]; i++)
  {
   fputc(text[i],myfile);
  }

 fclose(myfile);
 myfile = fopen("example.txt", "r");

 printf("\n The file has : ");

 while(!feof(myfile))
  {
   printf("%c", fgetc(myfile));
  }

 fclose(myfile);
}

Output

Enter a sentence to store in the file : this is an example for fgetc() and fputc()
The file has : this is an example for fgetc() and fputc()
```

2. This example illustrates how you can append (add) to the contents of an existing file.

```
/****************************************************************/
/* illustrating file append operation */
/****************************************************************/

#include <stdio.h>
main()
{
 FILE *myfile;
```

```
  char text[100];
  int i;

  myfile = fopen("example.txt", "a");

  printf("Enter a sentence to store in the file : ");
  gets(text);

  for(i=0; text[i]; i++)
  {
   fputc(text[i],myfile);
  }

  fclose(myfile);
  myfile = fopen("example.txt", "r");

  printf("\n The file has : ");

  while(!feof(myfile))
  {
   printf("%c", fgetc(myfile));
  }
  fclose(myfile);
}

Output

Enter a sentence to store in the file : This is the new line added

this is an example for fgetc() and fputc()
this is the new line added
```

3. This example illustrates how to use fputs() and fputs() to write to a file. Unlike character based I/O (fgetc), these are string based functions. Note the usage of feof() to check end of file.

```
/**************************************************************/
/* usage of fgets and fputs */
/**************************************************************/

#include <stdio.h>
main()
{
  FILE *myfile;
  char *text;

  myfile = fopen("example.txt", "w");

  printf("Enter a line to be stored : ");
  gets(text);
  fputs(text, myfile);
  fclose(myfile);

  myfile = fopen("example.txt", "r");
  printf("\nThe file has : ");
  while(!feof(myfile))
  {
    printf("%s", fgets(text, 10, myfile));
  }
  fclose(myfile);
}

Output

Enter a line to be stored : usage of fputs and fgets
The file has : usage of fputs and fgets
```

4. This example shows how to copy one file to another using fread() and fwrite(). Notice the usage of two file pointers, one for input file and other for output file.

```
/****************************************************************/
/* fread and fwrite for multiple line read and write */
/* program which copies from one file to another file */
/****************************************************************/

#include <stdio.h>
main()
{
  FILE *inputfile;
  FILE *outputfile;
  char text[100];
  int chars;

  inputfile = fopen("size.c", "r");
  outputfile = fopen("exam.txt", "w");

  /* in fread, 1 indicates how many characters to be read at a time and */
  /* 50 indicates how many characters to be read from the file */

  chars = fread(text, 1, 100, inputfile);
  fwrite(text, 1, chars, outputfile);

  fclose(inputfile);
  fclose(outputfile);

  outputfile = fopen("exam.txt", "r");
  chars = fread(text, 1, 100, outputfile);
  printf("%s", text;
  fclose(outputfile);
}

Output

/*****************************************************************/
/* Program to demonstrate the sto
```

5. This example is to generate a calendar for a given month and year. The output is stored in a file. This is an example for formatted outputs.

```
#include <stdio.h>
#define TRUE 1
#define FALSE 0

int check_leap (int year);
void construct (FILE *outputfile, int year,int month, int day_code, int
leap);

main()
{
  FILE *outputfile;
  int year,week_day_number,leap,month;
   int a,b,c;
  outputfile = fopen ("cal.txt", "w");
  printf("\nEnter year : ");
  scanf("%d",&year);
  printf("\nEnter month : ");
  scanf("%d",&month);

  /* use Zeller's congreunce to find the first day of the year */
  a = (year - 1.0) / 4.0;
  b = (year - 1.0) / 100.0;
  c = (year - 1.0)/ 400.0;
  week_day_number = (year + a - b + c) % 7;
  leap = check_leap (year);
  construct(outputfile, year,month, week_day_number, leap);
}

int check_leap (int year)
{
  if ((year % 4 == 0) && (year % 100 != 0) || (year%400 == 0))
    return TRUE;
  else
    return FALSE;
}

void construct (FILE *outputfile, int year,int month, int
week_day_number, int leap)
{
           int  days,daycount;
    switch ( month )
    {
      case 1:
        days = 31;
        break;

      case 2:
        days = leap ? 29 : 28;
        break;
      case 3:
        days = 31;
        break;

      case 4:
        days = 30;
        break;

      case 5:
        days = 31;
        break;

      case 6:
        days = 30;
        break;

      case 7:
        days = 31;
        break;

      case 8:
        days = 31;
        break;

      case 9:
        days = 30;
        break;

      case 10:
        days = 31;
        break;

      case 11:
        days = 30;
        break;

      case 12:
        days = 31;
        break;
    }
    fprintf(outputfile,"\n\nSun |Mon |Tue |Wed |Thu |Fri |Sat\n" );
    for ( daycount = 1; daycount <= 1 + week_day_number * 5; daycount++ )
       fprintf(outputfile," " );

    for ( daycount = 1; daycount <= days; daycount++ )
    {
      fprintf(outputfile,"%2d", daycount );
      if ( ( daycount + week_day_number ) % 7 > 0 )
         fprintf(outputfile," | " );
       else
         fprintf(outputfile, "\n " );
    }
    week_day_number = ( week_day_number + days ) % 7;
}
```

CHAPTER 14
DYNAMIC MEMORY ALLOCATION AND LINKED LISTS

1. This example is to illustrate the usage of malloc for characters.

```
/*****************************************************************/
/* usage of malloc in characters */
/*****************************************************************/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

main()
{
 char *first = "Programming ";
 char *second = "Language";
 char *final;

 final = (char *)malloc((strlen(first) + strlen(second) + 1)*sizeof(char));

 strcpy(final, first);
 strcat(final, second);

  printf("first: %s\n", first);
 printf("second: %s\n", second);
 printf("concatenated: %s\n", final);

 free(final);
}

Output

first: Programming
second: Language
concatenated: Programming Language
```

2. This example is to illustrate the usage of calloc for characters.

```
/*****************************************************************/
/* usage of calloc for strings */
/*****************************************************************/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

main()
{
 char *first = "Programming ";
 char *second = "Language";
 char *final;

 final = (char *)calloc(strlen(first) + strlen(second) + 1, sizeof(char));
```

```
  strcat(final, first);
  strcat(final, second);

  printf("first: %s\n", first);
  printf("second: %s\n", second);
  printf("concatenated: %s\n", final);

  free(final);

}
```

Output

```
first: Programming
second: Language
concatenated: Programming Language
```