

Patni Computer Systems Ltd.

C Programming

Lab Guide



Copyright © 2005 Patni Computer Systems, Akruhi, MIDC Cross Road No. 21, Andheri (E), Mumbai 400 093. All rights reserved. No part of this publication reproduced in any way, including but not limited to photocopy, photographic, magnetic, or other record, without the prior agreement and written permission of Patni Computer Systems.

Patni Computer Systems considers information included in this document to be Confidential and Proprietary.

Table of Contents

Lab 1-1: Install and Setup Turbo C compiler	3
Lab 1-2: Introduction to C	11
Lab 2-1: C operators	15
Lab 2-2: Type Conversion and Type Casting	17
Lab 3-1: Understand C control flow constructs	19
Lab 3-2: Loop Constructs - while loop and do..while loop	22
Lab 3-3: Loop Constructs - for loop	24
Lab 4-1: Functions	27
Lab 4-2: Automatic and Static Storage Classes	31
Lab 4-3: Recursion	33
Lab 5-1: Arrays	35
Lab 5-2: Multi-dimensional arrays	37
Lab 5-3: Character Arrays and Strings	40
Lab 5-4: String Library functions	42
Lab 5-5: Two Dimensional Array of Characters	44
Lab 6-1: Pointers	47
Lab 6-2: Call by value vs. Call by reference	49
Lab 6-3: Arrays and pointers	51
Lab 6-4: Pointers to pointers	54
Lab 6-5: Command Line Arguments	57
Lab 7-1: Structures	60
Lab 7-2: More structures	62
Lab 8-1: Data structures	65
Lab 9-1: File Handling	72
Lab 9-2: More File Handling	74
Appendix I - Debugging Tips	84
Appendix II - Coding Standards	88
Appendix III - Table of Figures	95
Appendix IV - Table of Examples	96

Lab 1-1: Install and Setup Turbo C compiler

Goals	Learn to install and setup Turbo C compiler. Test the setup.
Time	15 Minutes
Lab Setup	PC with Internet connectivity

Install Turbo C compiler**I. Download Turbo C compiler**

1. Borland Turbo C is an old compiler for Windows platforms. The lab exercises are developed using Turbo C v2.01 compiler. This version of compiler is suitable for Windows 95/98, Windows NT and Windows 2K operating systems.
2. If you do not have a copy of installer, connect to internet and download a free copy of Turbo C compiler from
http://www.acms.arizona.edu/education/opti_583x/sample_code/Turbo C.exe

II. Install Turbo C

3. Double click on Turbo C.exe and extract the contents to a folder say, c:\tctemp
4. Double click on Install.exe and start the installation process. Follow the instructions provided at each step to complete the installation.

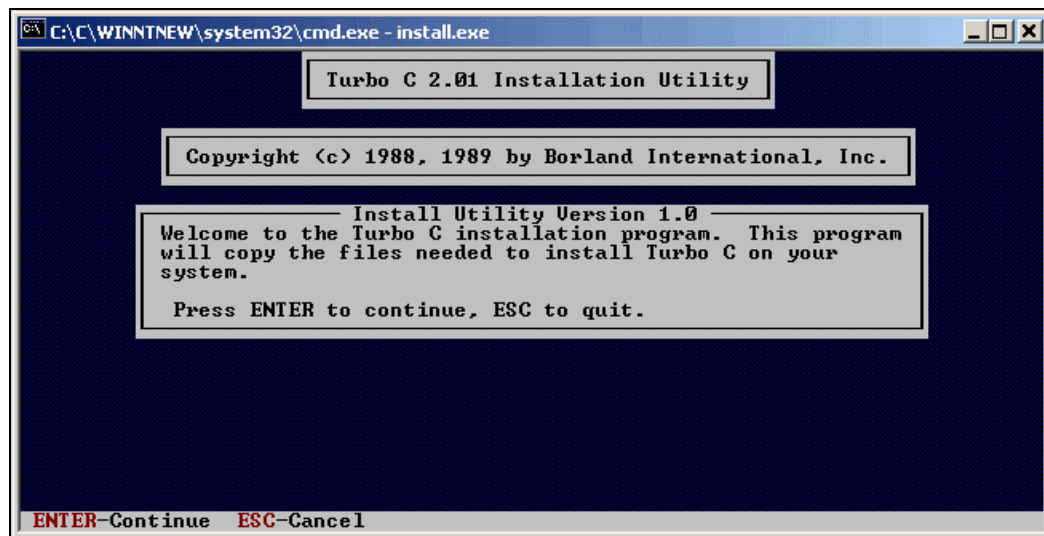


Figure 1-1.1: Turbo C Installation Utility

5. Enter the drive letter into which you would like to install Turbo C compiler and press enter.

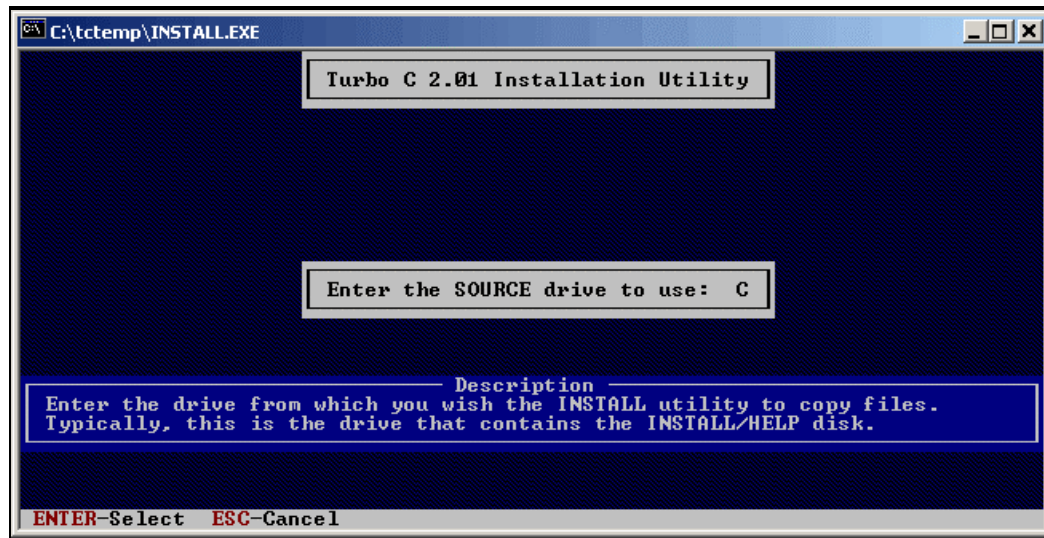


Figure 1-1.2: Set Drive

6. Enter the source path, which is the path where Turbo C.exe was extracted and press enter.

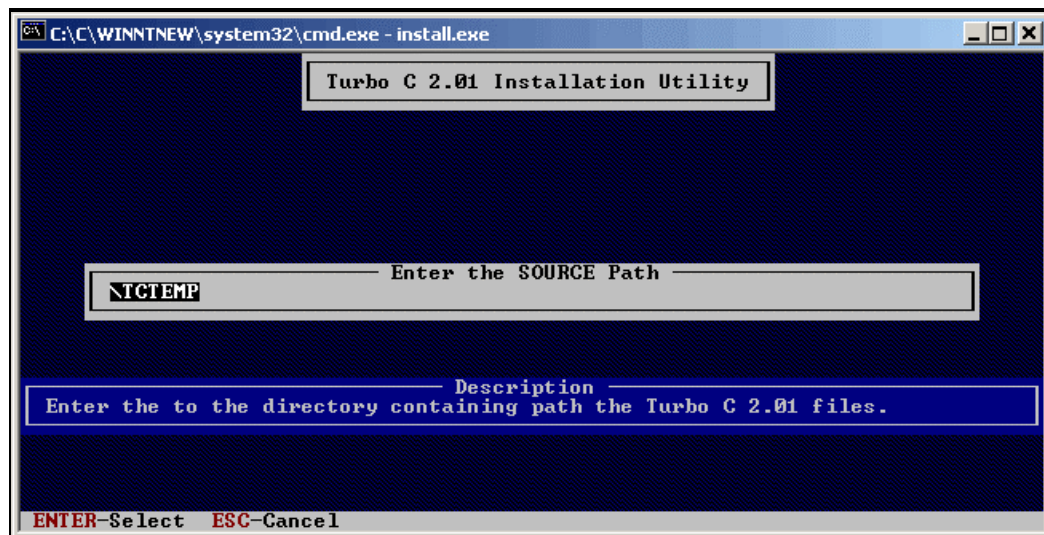


Figure 1-1.3: Set Source Path

7. Select the first option to install on hard disk and press enter.

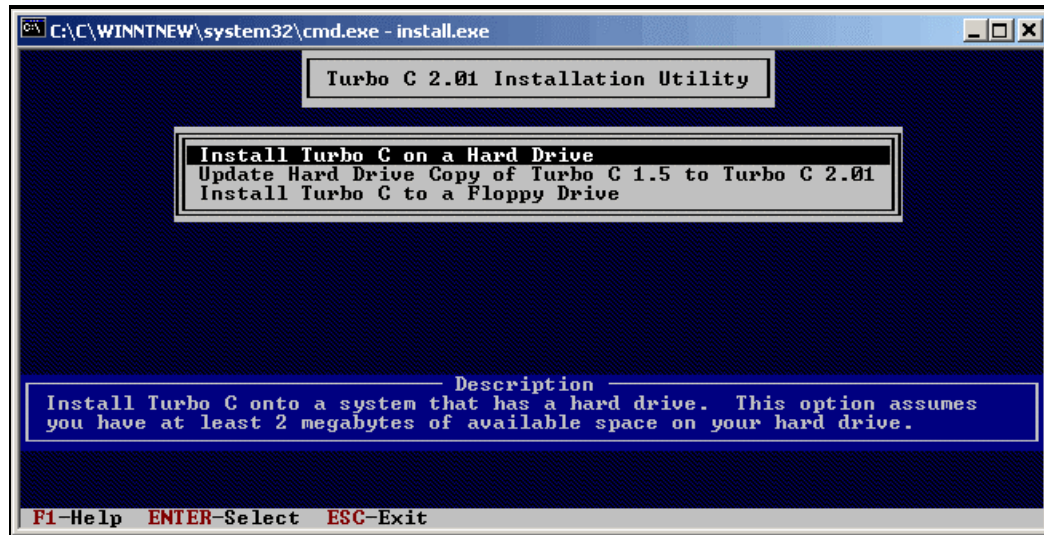


Figure 1-1.4: Install Turbo C

8. Select the directory where Turbo C compiler should be installed and press enter.

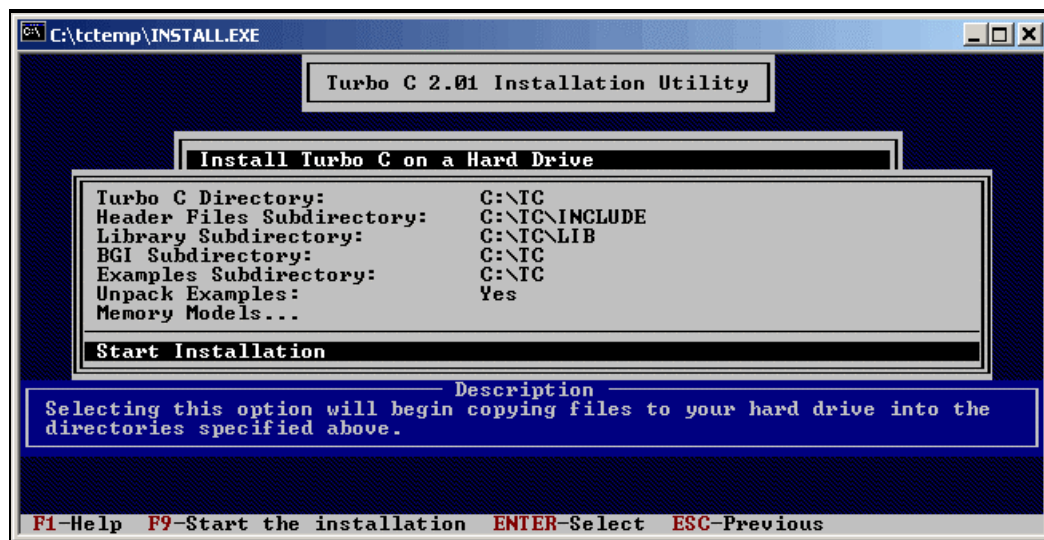


Figure 1-1.5: Start Installation

The required files will be copied and when the installation is completed you will see the prompt as shown below. Please update environment variable - PATH - as prompted.

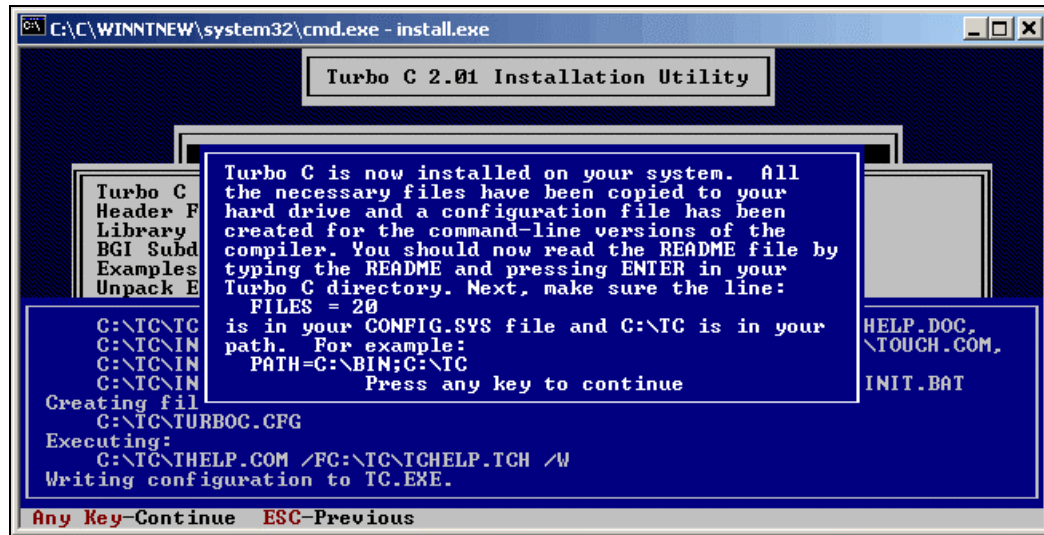


Figure 1-1.6: Installation Completed

I. Sample program and environment setting

1. Open command prompt and change the directory to your working directory and type TC.exe and press enter. This will open the editor and allows us to type the program.



Figure 1-1.7: Turbo C Environment

2. Type your C program and save the file using File -> Save menu option

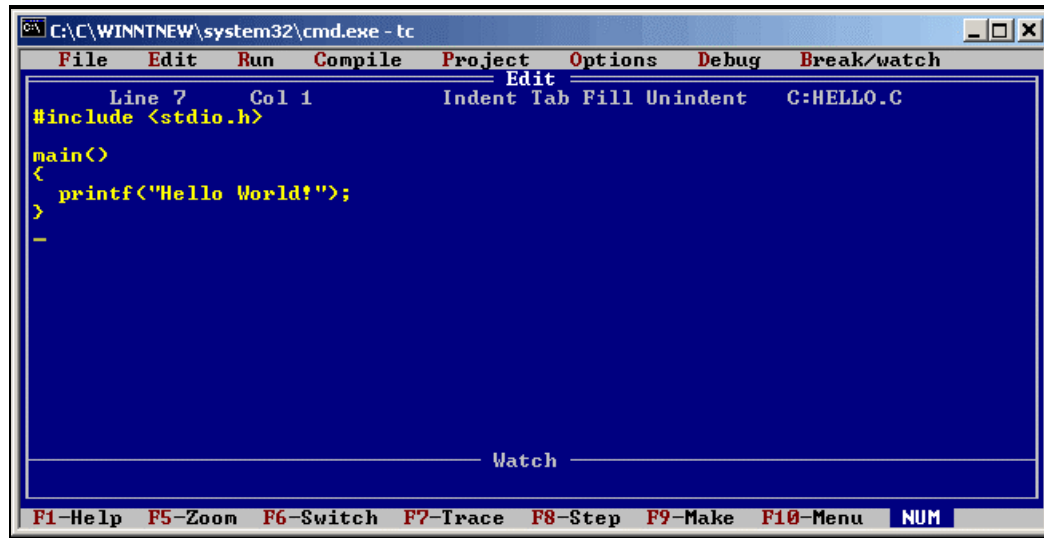


Figure 1-1.8: Hello World Program

3. To compile the program, the compiler environment should be configured. Select Options -> Directories menu option to display the directories currently configured. Ensure that Include directories points to "Include" directory and Library directories points to "Lib" directory from the installation directory.



Figure 1-1.9: Set Directories

- Now compile the program using Compile -> Compile to OBJ menu option. This will create a file with the same name as that of the source file and an extension "obj" in the current directory.

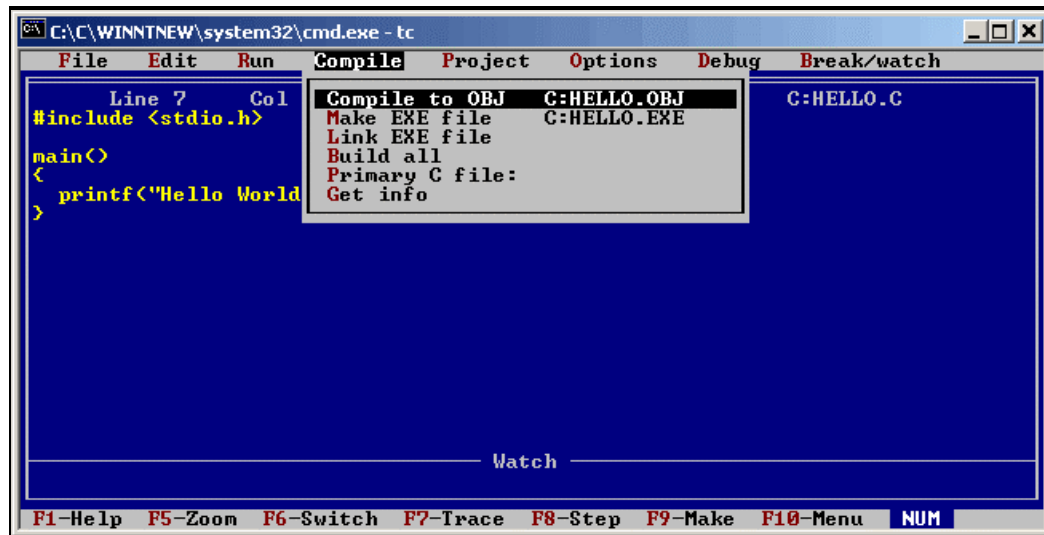


Figure 1-1.10: Compile to OBJ

- Next step is to generate the executable. Select Compile -> Make EXE file menu option. This will create a file with the same name as that of the source file and an extension "exe" in the current directory.

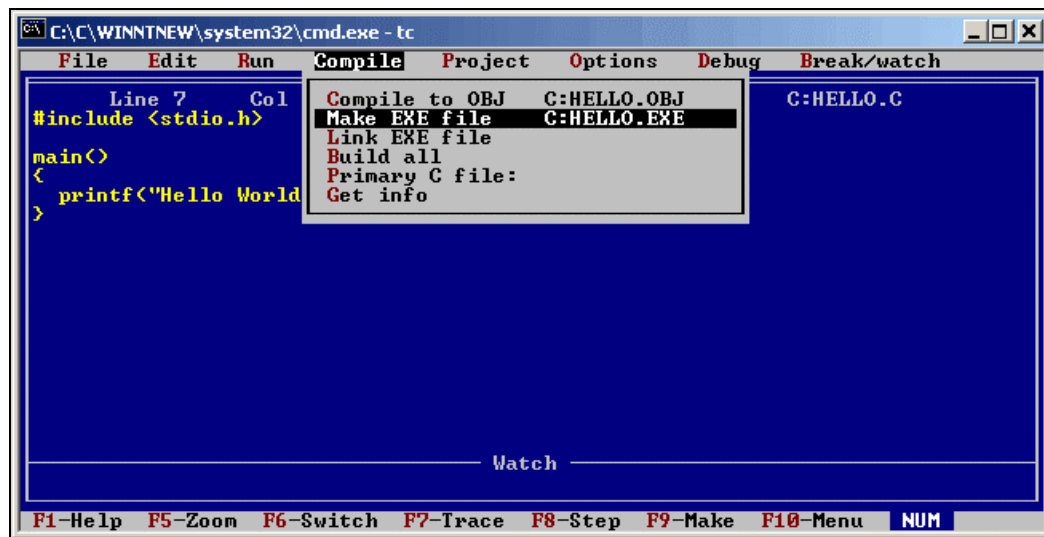


Figure 1-1.11: Make EXE File

6. Run the program using Run -> Run menu option to generate the output. Alternatively, you can also use CTRL+F9 shortcut key to perform this step.

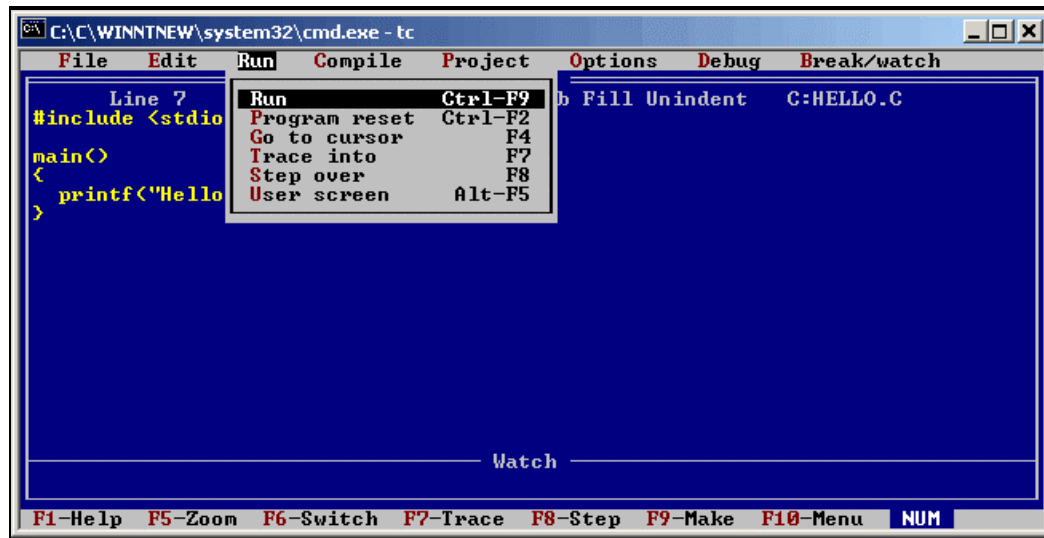


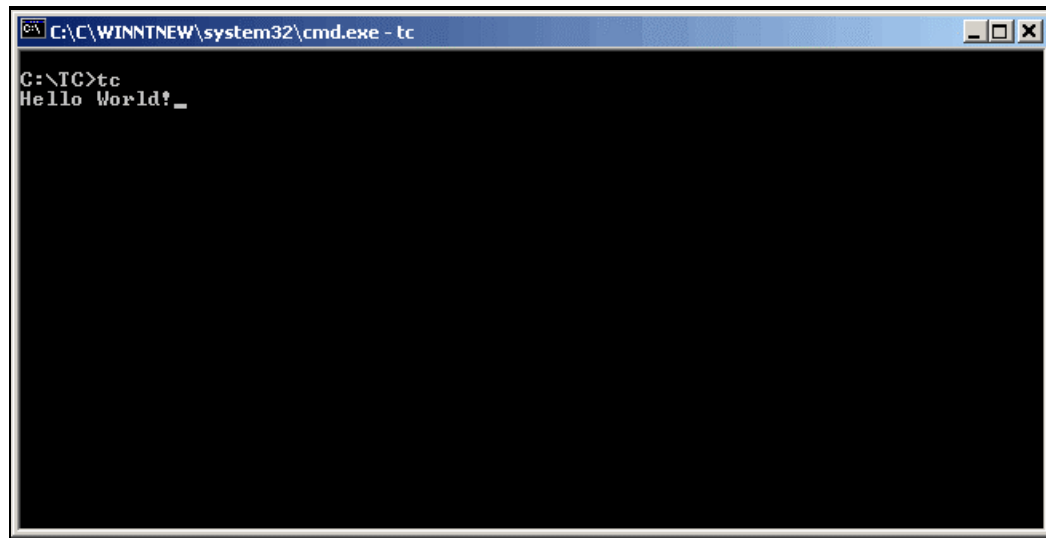
Figure 1-1.12: Run

7. In Turbo C, to view the output generated select Run -> User screen menu option. Alternatively, you can also use ALT+F5 shortcut key to perform this step.



Figure 1-1.13: User Screen

8. The output generated by the program is shown below. Press any key to come back to the Turbo C editor.



A screenshot of a Windows command prompt window. The title bar at the top reads "C:\C\WINNTNEW\system32\cmd.exe - tc". The command prompt shows the command "C:\TC>tc" and the output "Hello World!". The cursor is positioned at the end of the output line.

```
C:\C\WINNTNEW\system32\cmd.exe - tc
C:\TC>tc
Hello World!_
```

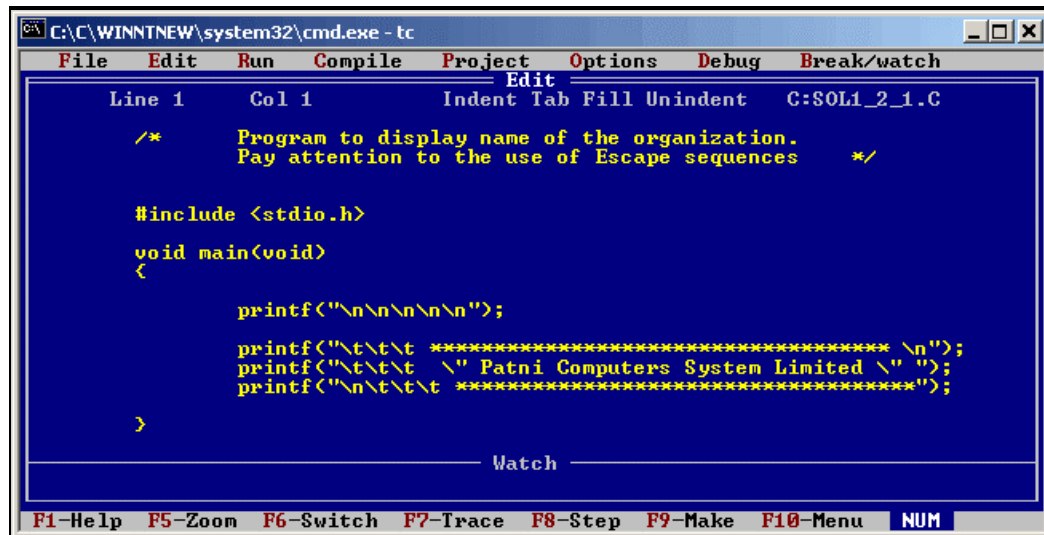
Figure 1-1.14: Hello World Program

Lab 1-2: Introduction to C

Goals	Write and execute simple C programs using built-in functions like scanf and printf
Time	30 Minutes
Lab Setup	PC having Turbo C installed.

I. Write a simple program to generate formatted output and display "Patni Computer Systems Limited"

1. Open the TC editor from your working directory and type the code as shown in the following screen shot. Formatted output can be generated with the use of \t, \n and \\" escape characters. These escape characters are used in standard printf function.



The screenshot shows the Turbo C editor window with the following code:

```
Line 1      Col 1      Indent Tab Fill Unindent      C:\SOL1_2_1.C
/*      Program to display name of the organization.
   Pay attention to the use of Escape sequences      */

#include <stdio.h>

void main(void)
{
    printf("\n\n\n\n\n");

    printf("\t\t\t ***** \n");
    printf("\t\t\t \" Patni Computers System Limited \"");
    printf("\n\t\t\t *****");

}

Watch
```

The bottom status bar shows: F1-Help F5-Zoom F6-Switch F7-Trace F8-Step F9-Make F10-Menu NUM

Figure 1-2.1: Formatted Output Program

- Save the file as SOL1_2_1.c and compile the program by choosing Compile -> Compile to OBJ menu.

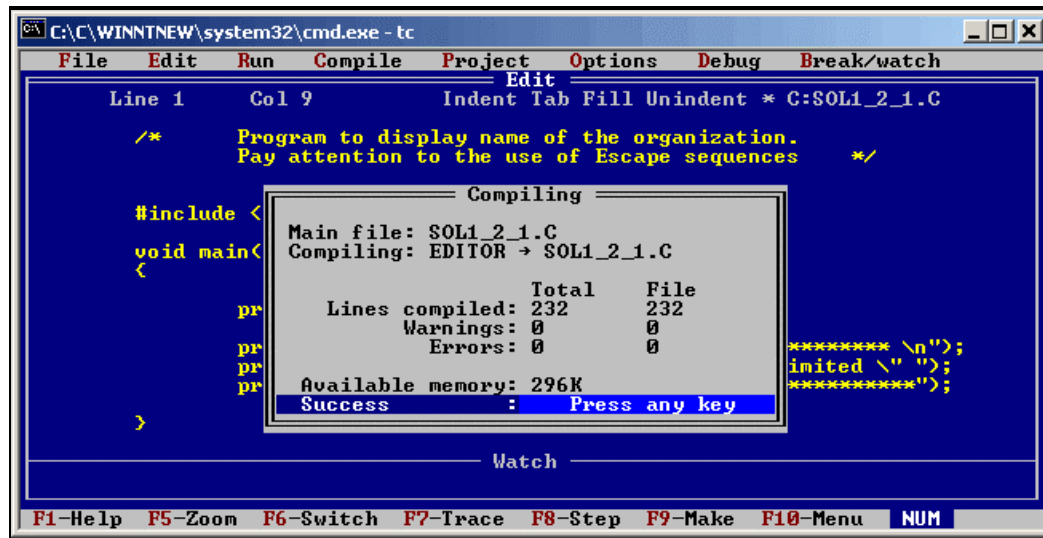


Figure 1-2.2: Formatted Output Program Compiled

Note: If your program contains any syntax errors, compiler will catch those errors and will display the summary of all the warnings and errors. You need to rectify these errors and recompile the program to proceed. *Always try to rectify the first error displayed in the Message window, as the subsequent errors are usually the consequences of the first one.*

- Generate the executable SOL1_2_1.exe by choosing Compile -> Make EXE file menu.

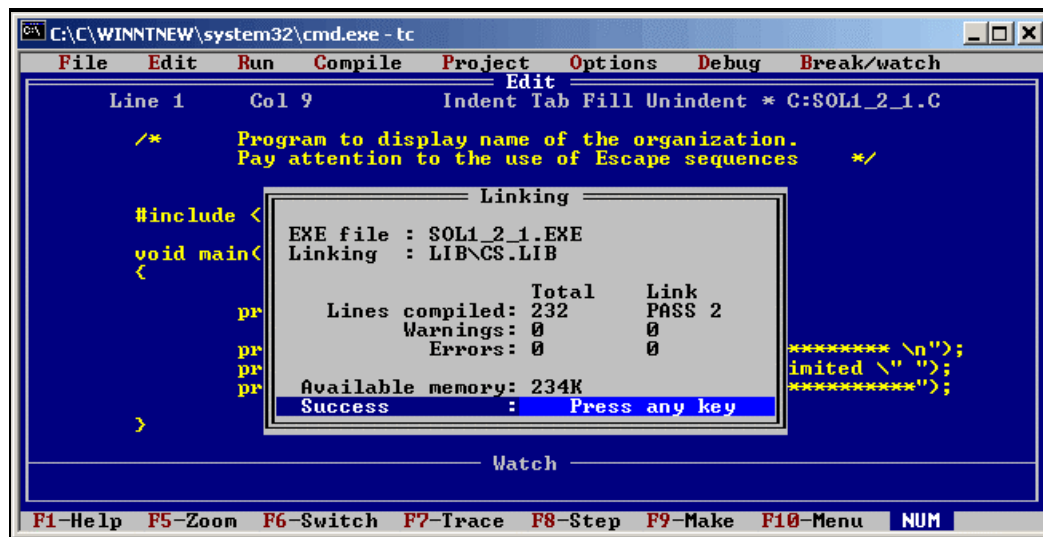


Figure 1-2.3: Formatted Output Program Linked

Note: If your environment settings are incorrect, linker will not be able to generate the required output file and it shows the summary of linker warnings and errors. You need to rectify these and rebuild to proceed.

4. Run the program by choosing Run -> Run menu.

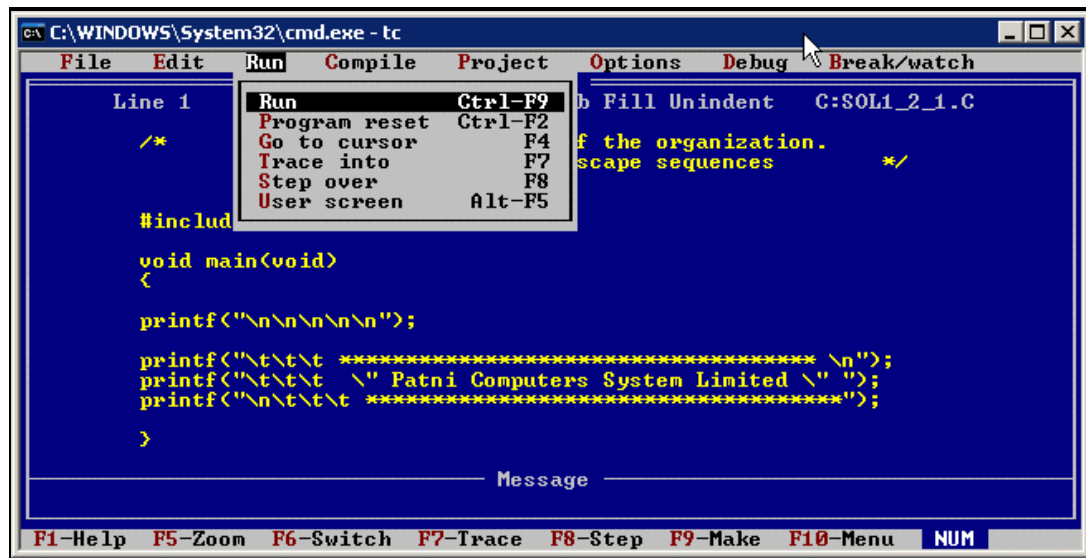


Figure 1-2.4: Run Formatted Output Program

5. See the output generated by choosing Run -> User screen.

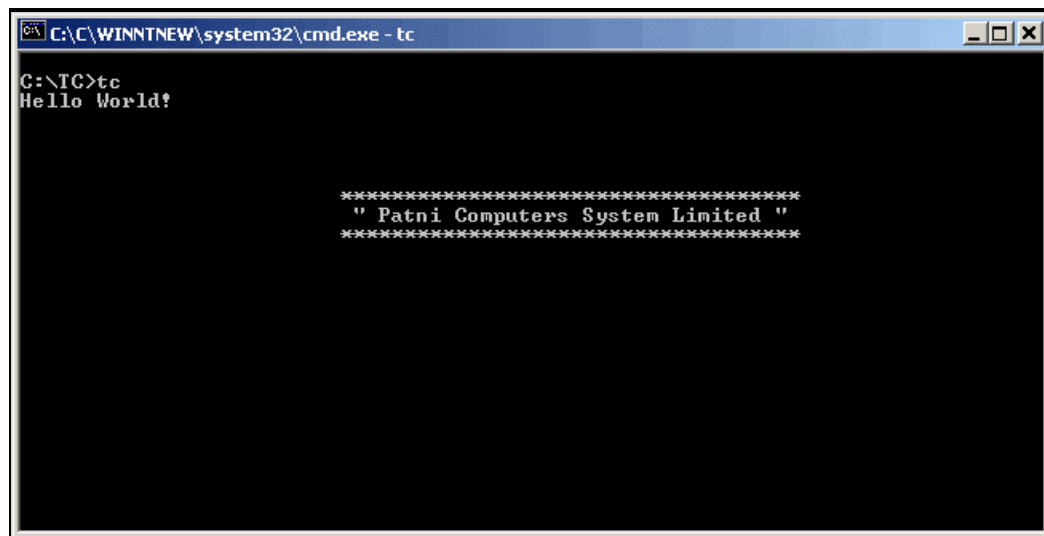


Figure 1-2.5: Formatted Output

Note: The output screen shows the output produced by the earlier programs since C programs executed in TC environment will not clear the output screen by default. You need to explicitly perform this task by executing `clrscr ()` library function.

II. Write a program to display your address in the following format.

```
-----  
| Amar Saghvi          |  
| Hiranandani Gardens  |  
| Powai                |  
| Mumbai - 400 021     |  
-----
```

<< To do >>

1. Open the TC editor from your working directory and write the main program to generate the required output.
2. Save the file as SOL1_2_2.c and compile the program by choosing Compile -> Compile to OBJ menu. Rectify the compiler errors, if any.
3. Generate the executable SOL1_2_2.exe by choosing Compile -> Make EXE file menu
4. Run the program by choosing Run -> Run menu
5. See the output generated by choosing Run -> User screen

III. Write a program to accept 2 integer values and display their addition. Repeat for various data types supporting numeric values.**<< To do >>**

1. Open the TC editor and write the main function.
2. Declare 3 variables for each of the data types supporting numeric values
e.g. int ia, ib, ic; float fa, fb, fc;
3. Write code to accept the values for 2 of the variables of same data type using scanf (). e.g. scanf("%d%d", &ia, &ib);
4. Write code to store addition of 2 variables in third one e.g. ic = ia + ib;
5. Write code to print value of third variable using printf () e.g. printf("%d", ic); Print the proper messages along with the desired result.
6. Repeat steps 3 to 5 for other data types.
7. Save the file as SOL1_2_3.c and compile the program by choosing Compile -> Compile to OBJ menu
8. Generate the executable SOL1_2_3.exe by choosing Compile -> Make EXE file menu
9. Run the program by choosing Run -> Run menu
10. See the output generated by choosing Run -> User screen or by pressing Alt+F5

Lab 2-1: C operators

Goals	Write and execute simple C programs using various operators.
Time	30 Minutes
Lab Setup	PC having Turbo C installed.

I. Write a C program to demonstrate working of bitwise shift operators. [SOL2_1_1.c]

1. Open Turbo C editor and type the following code in it

```
/* Sample program using bitwise shift operators */

#include<stdio.h>
#include<conio.h> /* header file for clrscr() library function*/

void main(void)
{
    int number;
    int result;

    clrscr();
    /* A library function to clear the output window.
       This may not work with other compilers */

    printf("\n Enter an integer : ");
    scanf("%d", &number);

    result = number<<3;
    /* Shift the bits of number to left 3 times*/
    printf("\n\n %d shifted to left thrice is %d", number, result);

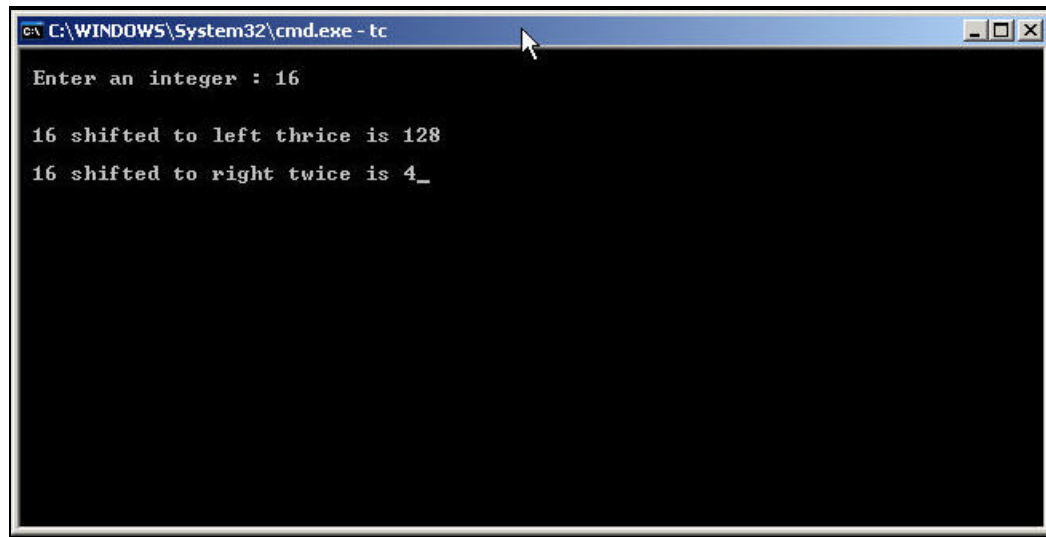
    result = number>>2;
    /* Shift the bits of number to right 2 times */
    printf("\n\n %d shifted to right twice is %d", number, result);

}
```

Example 2-1.1: Bitwise Shift Operators

2. Save the file as SOL2_1_1.c **and** compile the program by choosing Compile -> Compile to OBJ menu
3. Generate the executable SOL2_1_1.exe by choosing Compile -> Make EXE file menu
4. Run the program by choosing Run -> Run menu

5. See the output generated by choosing Run -> User screen or by pressing Alt+F5. A sample run of the above code is shown here. *Note the **doubling** and **halving** effects of bitwise shift operators.*



```
C:\WINDOWS\System32\cmd.exe - tc
Enter an integer : 16
16 shifted to left thrice is 128
16 shifted to right twice is 4_
```

Figure 2-1.1: Bitwise Shift Operators

- II. Write a C program to accept three numbers and display the minimum among them. Use ternary operator.

<< To do >>

1. Open the TC editor from your working directory and write the main program to generate the required output

Note: The syntax for ternary/conditional operator, `?:`, is as follows.

condition ? expression1 : expression2 ;

which means, if the specified condition is true execute expression1 else execute expression2. You may also nest the ternary operator to any level as shown here.

`cond1? (cond1_1 ? exp1_1 : exp1_2) : (cond1_2 ? exp2_1 : exp2_2) ;`

2. Save the file as SOL2_1_2.c and compile the program by choosing Compile -> Compile to OBJ menu
3. Generate the executable SOL2_1_2.exe by choosing Compile -> Make EXE file menu
4. Run the program by choosing Run -> Run menu
5. See the output generated by choosing Run -> User screen

Lab 2-2: Type Conversion and Type Casting

Goals	Write and execute simple C programs to understand type conversion and type casting
Time	20 Minutes
Lab Setup	PC having Turbo C installed.

I. Write a program to demonstrate type conversion and type casting. [SOL2_2_1.c]

1. Open Turbo C editor and type the following code.

```
/* Sample program demonstrating Type Conversion and Type casting */

#include<stdio.h>
#include<conio.h>

void main(void)
{
    int    iNum;
    float  fVal=12.34;

    int    a=10,b=12,c=16,iSum;
    float  fAvg;

    clrscr();

    printf("\n Internal type conversion");
    iNum = fVal;
                    /*float value is converted to
                    integer and assigned to iNum */

    printf("\n float Value : %f ", fVal);
    printf("\n int   Number: %d ", iNum);

    printf("\n\n\n Explicit type casting");
    iSum= a + b + c;
    fAvg = (float)iSum / 3;
                    /* iSum type casted to float*/

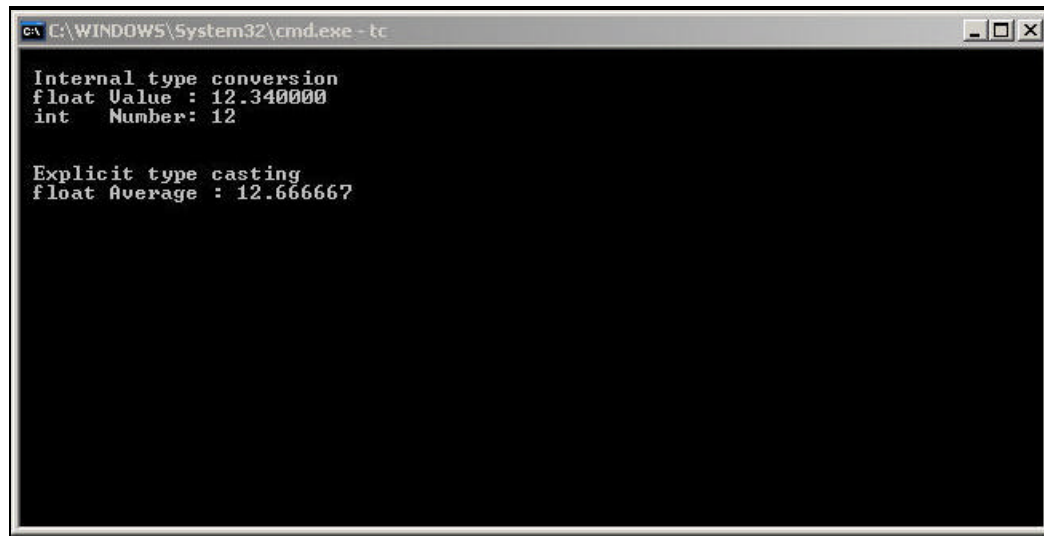
    printf("\n float Average : %f ", fAvg);

}
```

Example 2-2.1: Type Conversion & Type Casting

2. Save the file as SOL2_2_1.c and compile the program by choosing Compile -> Compile to OBJ menu
3. Generate the executable SOL2_2_1.exe by choosing Compile -> Make EXE file menu
4. Run the program by choosing Run -> Run menu

5. See the output generated by choosing Run -> User screen or by pressing Alt+F5. The output will be as follows



```
C:\WINDOWS\System32\cmd.exe - tc
Internal type conversion
float Value : 12.340000
int Number: 12

Explicit type casting
float Average : 12.666667
```

Figure 2-2.2: Type Conversion & Type Casting

- II. Write a C program to calculate simple interest. Accept the principal amount, rate of interest in percentage and number of years from the user. Assume these 3 quantities to be integers. The interest calculated will be a floating-point number.

<< To do >>

1. Open the TC editor from your working directory and write the program to generate the required output.
2. Save the file as SOL2_2_2.c and compile the program by choosing Compile -> Compile to OBJ menu
3. Generate the executable SOL2_2_2.exe by choosing Compile -> Make EXE file menu
4. Run the program by choosing Run -> Run menu
5. See the output generated by choosing Run -> User screen

- III. Write a program to accept the number of centimeters (a real number), and to print out the equivalent number of feet (integer) and inches (floating, 1 decimal), with the inches given to an accuracy of one decimal place.

Assume 2.54 centimeters per inch, and 12 inches per foot.

If the input value is 333.3, the output format should be:

333.3 centimeters is 10 feet 11.2 inches.

<<To do>>

1. Write the code to generate required output and save the file as SOL2_2_3.c
Note: printf ("%0.2f", fVal); will print the fVal to 2 decimal places
2. Compile and make exe SOL2_2_3.exe
3. Run the executable file and test your code with various input values

Lab 3-1: Understand C control flow constructs

Goals	Write and execute C programs to understand control flow constructs like if statement, if...else statement and switch statement.
Time	30 Minutes
Lab Setup	PC having Turbo C installed.

I. Write a C program to display a day of the week depending on its number.

1. Open a new file in Turbo C editor using File -> New menu and type the following code.

```
/* Sample program demonstrating use of switch statement */

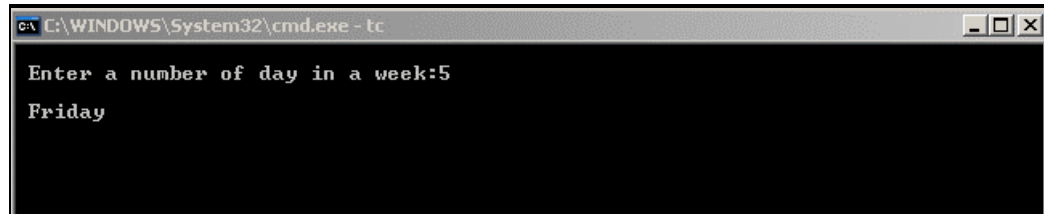
#include<stdio.h>
#include<conio.h>

void main(void)
{
    int iDay;
    clrscr();
    printf("\n Enter a number of day in a week:");
    scanf("%d", &iDay);

    switch(iDay)
    {
        case 1:
            printf("\n Monday ");
            break;
        case 2:
            printf("\n Tuesday ");
            break;
        case 3:
            printf("\n Wednesday ");
            break;
        case 4:
            printf("\n Thursday ");
            break;
        case 5:
            printf("\n Friday ");
            break;
        case 6:
            printf("\n Saturday ");
            break;
        case 7:
            printf("\n Sunday ");
            break;
        default:
            printf("\n Sorry there are 7 days in a week");
    }; /* End of switch statement */
} /* End of main */
```

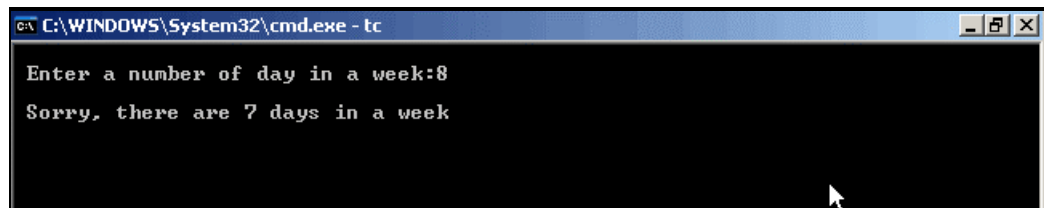
Example 3-1.1: Switch Statement

2. Save the file as SOL3_1_1.c **and** compile the program by choosing Compile -> Compile to OBJ menu
3. Generate the executable SOL3_1_1.exe by choosing Compile -> Make EXE file menu.
4. Run the program by choosing Run -> Run menu
5. See the output generated by choosing Run -> User screen or by pressing Alt+F5. A few sample runs of the above code are shown here.



```
C:\WINDOWS\System32\cmd.exe - tc
Enter a number of day in a week:5
Friday
```

Figure 3-1.1: Switch Statement - Output 1



```
C:\WINDOWS\System32\cmd.exe - tc
Enter a number of day in a week:8
Sorry, there are 7 days in a week
```

Figure 3-1.2: Switch Statement - Output 2

- II. Design a simple menu driven numeric calculator to display sum, difference, product, division and remainder of 2 integers. Perform arithmetic operations depending upon the operator chosen by the user.

- + Addition
- Subtraction
- * Multiply
- / Divide
- % Modulo operation

Give the appropriate error message if the operator chosen is not from the above list.

<<To do>>

1. Open a new file in Turbo C editor, include required header files and start coding the main function
2. Declare variables for numeric operands as well as a character variable to store user's choice for operator
3. Accept the values of operands and operator using scanf()
4. Write switch statement and case statements corresponding to each operator.
5. Write default case to handle wrong choice of the operator
6. Write appropriate code for each case and complete the program
7. Save the program as SOL3_1_2.c and Compile it.

8. Run the code and see the output.

III. Write a program to calculate the payable amount $\text{Payable Amount} = \text{Invoice Amt} - \text{Discount} + \text{Tax}$. You are given invoice no (an integer), customer code (a character) and invoice amount (a floating point). Discount is 3% of the amount for amounts exceeding 5000. Tax is 1% of the amount except for customer codes 'X', 'P'.

<< *To do* >>

1. Open a new file in Turbo C editor, include required header files and start coding the main function.
2. Declare variables and write code to accept values for invoice no., customer code and invoice amount.
3. Write code to calculate the payable amount as per the specified criteria.
4. Write code to display the calculated payable amount.
5. Save the file as SOL3_1_3.c and compile the same and Make SOL3_1_3.exe.
6. Run the exe and see the output.

Lab 3-2: Loop Constructs - while loop and do..while loop

Goals	Write and execute C programs to understand iterative constructs through while and do..while loops.
Time	30 Minutes
Lab Setup	PC having Turbo C installed.

I. Write a program to calculate the sum of digits of a number. [SOL3_2_1.c]

1. Open a new file and type the following code.

```
/* while loop to calculate sum of digits of an integer */

#include<stdio.h>
#include<conio.h>

void main(void)
{
    int iNum;
    int iCopy;
    int iSum;

    clrscr();

    printf("\n Enter a number:");
    scanf("%d", &iNum);

    iCopy = iNum;
    iSum = 0;

    while(iNum > 0)
    {
        iSum += iNum % 10;
        iNum /= 10;
    }

    printf("\n Sum of digits of %d is %d", iCopy, iSum);
}
```

Example 3-2.1: While Loop

2. Save the file as SOL3_2_1.c and compile the same.
3. Make exe named SOL3_2_1.exe.
4. Run the executable file and see the output.

- II. Write a program to reverse a given number e.g. if the given number is 7680 then the reversed number should be 867. [SOL3_2_2.c]

<<To do >>

1. Write a program to generate desired output and save it as SOL3_2_2.c
2. Compile, build (make exe) and run the same and see the output.

- III. Modify the program written for SOL3_2_2.c so that it executes repetitively until user chooses to stop i.e. Accept the option "Y" or "N" from the user to indicate whether he wants to continue or not. Make use of do..while loop to achieve the same. [SOL3_2_3.c]

<<To do>>

Lab 3-3: Loop Constructs - for loop

Goals	Write and execute C programs to understand iterative constructs through for loops.
Time	30 Minutes
Lab Setup	PC having Turbo C installed.

I. Write a program to generate the following output. [SOL3_3_1.C]

2	3	4	5
1	3	4	5
1	1	4	5
1	2	3	4

1. Write the following code in a new file and save it as SOL3_3_1.c

```
/* sample program demonstrating use of nested for loops and
continue control statement */

#include<stdio.h>
#include<conio.h>

void main(void)
{
    int i,j;
    clrscr();

    for(i=1;i<6;i++)
    {
        for(j=1;j<=5;j++)
        {
            if(i==j)
                continue;
            /* Skip the remaining statements from the loop
            and take the control back to the beginning of the loop*/

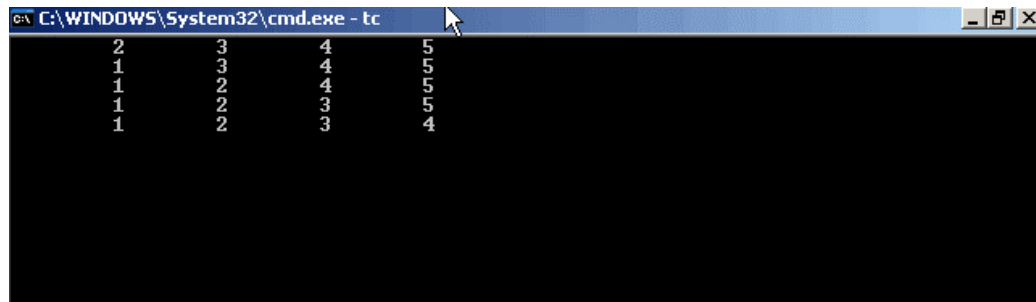
            printf("\t%d",j);
        } /* end of inner for loop */
        printf("\n");
    } /* end of outer for loop */

} /* end of main function */
```

Example 3-3.1: For Loop & Continue

2. Compile and make exe SOL3_3_1.exe.

3. Run the executable and see the output. The output will be as shown below.



```

C:\WINDOWS\System32\cmd.exe - tc
2      3      4      5
1      3      4      5
1      2      4      5
1      2      3      5
1      2      3      4
  
```

Figure 3-3.1: For Loop & Continue Statement

4. Replace the 'continue' control statement with 'break'. Compile, build and run the program again. Observe how does it affect the output. You may perform the step-by-step execution of the code and watch the values of i and j counter variables.

Note: The process of step-by-step execution is explained in Appendix - I.

- II. Write a program to print prime numbers from 1 to 1000. [SOL3_3_2.C]

<<To do >>

- III. Read a positive integer value, and compute the following sequence: If the number is even, halve it; if it's odd, multiply by 3 and add 1. Repeat this process until the value is 1, printing out each value. Finally print out how many of these operations you performed.

Typical output might be:

```

Initial value is 9
Next value is 28
Next value is 14
Next value is 7
Next value is 22
Next value is 11
Next value is 34
Next value is 17
Next value is 52
Next value is 26
Next value is 13
Next value is 40
Next value is 20
Next value is 10
Next value is 5
Next value is 16
Next value is 8
Next value is 4
Next value is 2
Final value 1, number of steps 19
  
```

If the input value is less than 1, print a message containing the word 'Error' and exit the execution. [SOL3_3_3.c]

<<To do >>

IV. Check whether a number is an Armstrong number or not. (An Armstrong number is the equal to sum of cubes of digits E.g. $153 = (1)^3 + (5)^3 + (3)^3$. [SOL3_3_4.c]

<<To do >>

Lab 4-1: Functions

Goals	Write and execute C programs to understand fundamentals of functions such as passing parameters and receiving value back.
Time	30 Minutes
Lab Setup	PC having Turbo C installed.

I. Write a C program with a function for calculation of compound interest.
[SOL4_1_1.c]

1. Open a new file in Turbo C editor and write the following code in it.

```
/*Calculation of Compound Interest*/
#include <stdio.h>
#include<conio.h>
#include<math.h>

int main()
{
    void ClCalc(double,double,int); /* Function prototype */

    double principal, rate;
    int years;
    char ch;
    clrscr();

    do
    {
        printf("\n Enter Principal Amount :");
        scanf("%lf",&principal);

        printf("\n Enter rate of interest in percentage:");
        scanf("%lf",&rate);
        rate = rate / 100.00;

        printf("\n Enter number of years :");
        scanf("%d",&years);

        ClCalc(principal,rate,years);      /*Function Call with actual
                                           arguments*/
        printf("\n Do you want to continue ?[Y/N] :");

        fflush(stdin); /* a library function to clear
                       the contents of input buffer */

        scanf("%c",&ch);
    }while(ch=='y' || ch=='Y');

    return(0); /* return 0 indicating successful execution */
}
```

Example 4-1.1: Functions - Compound Interest

```
void CCalc(double p, double r, int y) /* function definition with
formal arguments */
{
    double interest;
    int i;

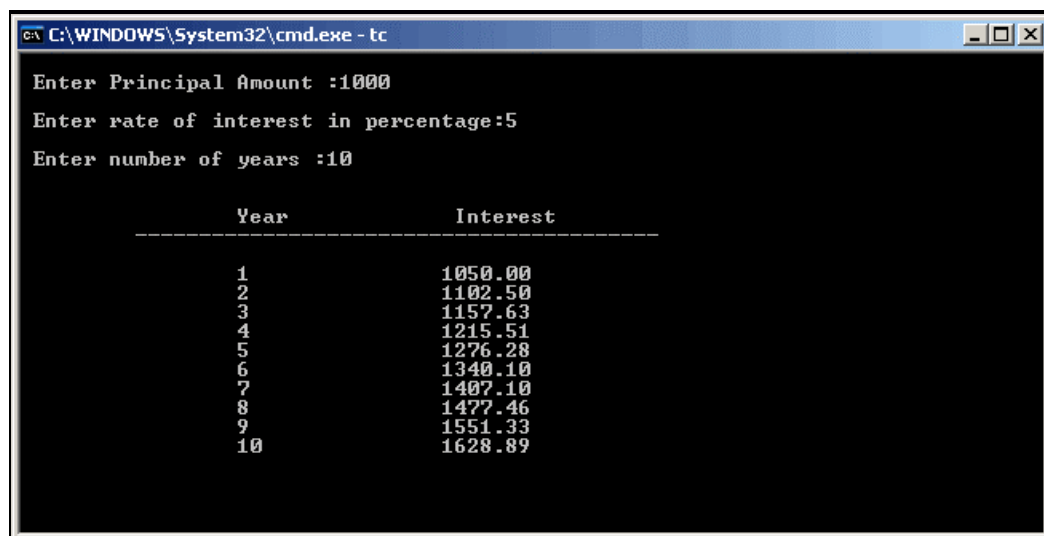
    printf("\n\t Year \t\t Interest");
    printf("\n\t -----");

    for ( i = 1; i <=y ; i++ )
    {
        interest = p * pow( 1.0 + r, i );
        /* double pow(double x, double y)
        is a library function declared in math.h
        which returns a double value corresponding
        to first argument raised to second */

        printf("\n\t %d \t\t %.2f ", i, interest);
    }
}
```

Example 4-1.1: Functions - Compound Interest Contd...

2. Save the file as SOL4_1_1.c and compile the program by choosing Compile -> Compile to OBJ menu
3. Generate the executable SOL3_1_1.exe by choosing Compile -> Make EXE file menu
4. Run the program by choosing Run -> Run menu
5. See the output generated by choosing Run -> User screen or by pressing Alt+F5. A sample run of the above code is shown here.



```
C:\WINDOWS\System32\cmd.exe - tc

Enter Principal Amount :1000
Enter rate of interest in percentage:5
Enter number of years :10

      Year          Interest
-----
      1             1050.00
      2             1102.50
      3             1157.63
      4             1215.51
      5             1276.28
      6             1340.10
      7             1407.10
      8             1477.46
      9             1551.33
     10             1628.89
```

Figure 4-1.1: Functions - Compound Interest

II. Write a function that accepts an integer argument and converts it into its absolute value. [SOL4_1_2.c]

1. Open a new file in Turbo C editor and type the following code in it.

```
/* call by value */

#include <stdio.h>
#include <conio.h>

void MakeAbs(int i)      /* i is local to MakeAbs-      Call by value */
{
    if (i < 0)
        i = -i;
    printf("\n Inside Function, absolute i = %d", i);
}

int main(void)
{
    int i; /* i local to main */
    clrscr();

    i = -5;
    printf("\n Before function, inside main i= %d", i);

    MakeAbs(i); /* function call */

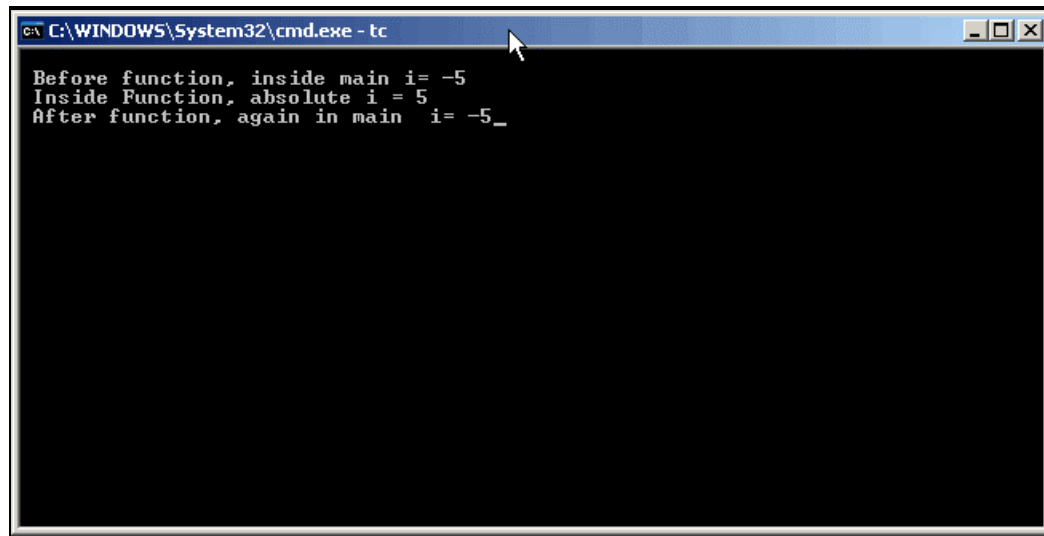
    printf("\n After function, again in main i= %d", i);

    return(0);
}
```

Example 4-1.2: Call by Value

2. Save the file as SOL4_1_2.c and compile the program by choosing Compile -> Compile to OBJ menu
3. Generate the executable SOL4_1_2.exe by choosing Compile -> Make EXE file menu
4. Run the program by choosing Run -> Run menu

5. See the output generated by choosing Run -> User screen or by pressing Alt+F5. The output of above code is shown here.



```
C:\WINDOWS\System32\cmd.exe - tc
Before function, inside main i= -5
Inside Function, absolute i = 5
After function, again in main i= -5_
```

Figure 4-1.2: Call by Value

- III. Read two integers, representing a rate of pay (rupees per hour) and a number of hours. Print out the total pay, with hours up to 40 being paid at basic rate, from 40 to 60 at rate-and-a-half, above 60 at double-rate. Print the pay as rupees to two decimal places. Terminate the loop when a zero rate is encountered. At the end of the loop, print out the total pay. The code for computing the pay from the rate and hours is to be written as a function. The recommended output format is something like:

```
Pay at 200 rupees/hr for 38 hours is 7600 rupees
Pay at 220 rupees/hr for 48 hours is 11440 rupees
Pay at 240 rupees/hr for 68 hours is 20640 rupees
Pay at 260 rupees/hr for 48 hours is 13520 rupees
Pay at 280 rupees/hr for 68 hours is 24080 rupees
Pay at 300 rupees/hr for 48 hours is 15600 rupees

Total pay is 92880 rupees
```

[SOL4_1_3.c]

<<To do >>

Lab 4-2:Automatic and Static Storage Classes

Goals	Write and execute C programs to understand Automatic and static storage classes.
Time	20 Minutes
Lab Setup	PC having Turbo C installed.

I. Write a program to demonstrate working of static variables. [SOL4_2_1.C]

1. Open a new file in Turbo C editor and type the following code in it.

```
#include<stdio.h>
#include<conio.h>

void stat(void); /* Function prototype */

void main(void)
{
    int i;
    clrscr();

    for (i=0;i<5;++i)
        stat();
}

void stat(void)
{
    int auto_var = 0;
    static int static_var = 0;

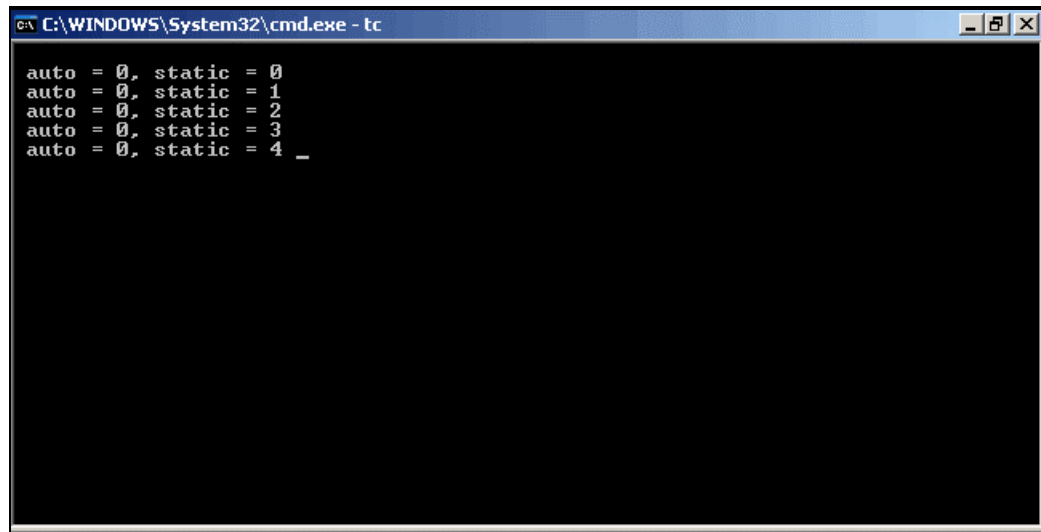
    printf("\n auto = %d, static = %d ", auto_var, static_var);

    ++auto_var;
    ++static_var;
}
```

Example 4-2.1: Static Variables

2. Save the file as SOL4_2_1.c and compile the program by choosing Compile -> Compile to OBJ menu
3. Generate the executable SOL4_2_1.exe by choosing Compile -> Make EXE file menu
4. Run the program by choosing Run -> Run menu

5. See the output generated by choosing Run -> User screen or by pressing Alt+F5. The output of above code is shown here.



```
C:\WINDOWS\System32\cmd.exe - tc
auto = 0, static = 0
auto = 0, static = 1
auto = 0, static = 2
auto = 0, static = 3
auto = 0, static = 4 _
```

Figure 4-2.1: Static Variables

- II. Write a program to generate the first n Fibonacci numbers, where n is a value specified by the user. [SOL4_2_2.C]

<<To do>>

1. The Fibonacci numbers from a sequence in which each number is equal to the sum of the previous 2 numbers. e. g. First 8 Fibonacci numbers are 1,1,2,3,5,8,13,21.
2. Open a new file named SOL4_2_2.c. write code for main() which will read in a value for n and then enter a loop that generates and writes out each of the Fibonacci numbers. Each pass of the loop will call a function Fibonacci() which in turn will calculate next term using previous 2 terms.
3. Inside function Fibonacci () maintain the previous terms as static variables and calculate the next one.
4. Complete the code, save and compile it to SOL4_2_2.obj.
5. Make exe SOL4_2_2.exe, run the same for various values of n.

Lab 4-3: Recursion

Goals	Write and execute C programs to understand recursive functions.
Time	30 Minutes
Lab Setup	PC having Turbo C installed.

I. Write a recursive function to multiply 2 positive integers using only addition operation. [SOL4_3_1.C]

1. Open a new file in Turbo C editor and type the following code in it.

```
#include <stdio.h>
#include <conio.h>

/* Return m*n. Computes the product recursively using addition */
unsigned int mult_rec(unsigned int m, unsigned int n)
{
    if (n == 0)
        return 0;
    else
        return m + mult_rec(m,n-1);
}

int main(void)
{
    unsigned int m,n;

    clrscr();

    printf("\n Enter m : ");
    scanf("%u", &m);
    printf("\n Enter n : ");
    scanf("%u", &n);

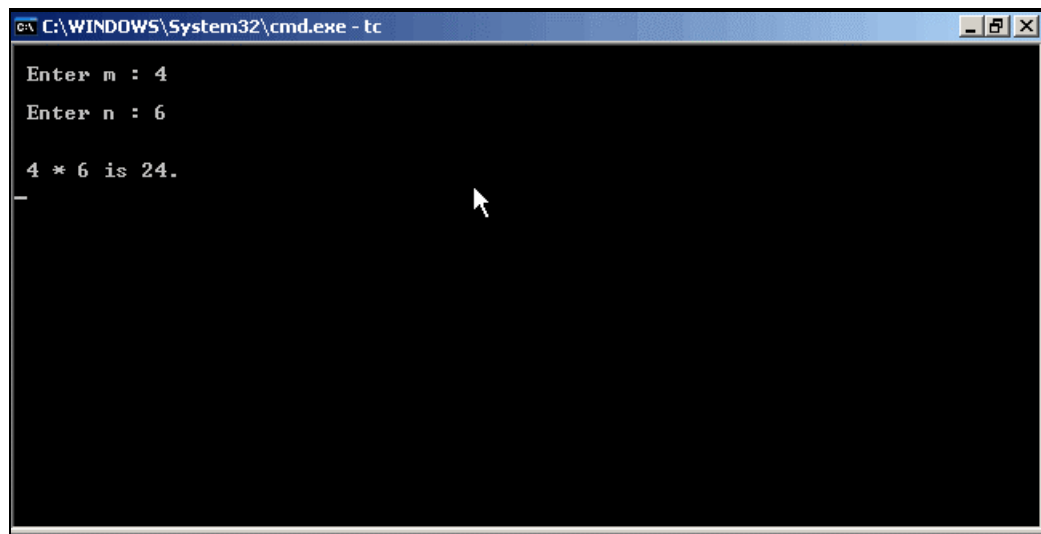
    printf("\n\n %u * %u is %u.\n", m, n, mult_rec(m, n) );

    return 0; /* no error */
}
```

Example 4-3.1: Recursion

2. Save the file as SOL4_3_1.c and compile the code using Compile -> Compile to OBJ menu
3. Generate the executable SOL4_3_1.exe by choosing Compile -> Make EXE file menu
4. Run the program by choosing Run -> Run menu

5. See the output generated by choosing Run -> User screen or by pressing Alt+F5. The output of above code is shown here.



```
C:\WINDOWS\System32\cmd.exe - tc
Enter m : 4
Enter n : 6
4 * 6 is 24.
```

Figure 4-2.2: Recursion

- II. Write a recursive function to calculate power of a number with the following prototype: `double power (double, int)`. Use only the multiplication operation. [SOL4_3_2.C]

<<To do>>

- III. Write a recursive function to calculate sum of first N positive integers. [SOL4_3_3.C]

<<To do >>

Lab 5-1: Arrays

Goals	Write and execute C programs to work with single dimensional arrays.
Time	30 Minutes
Lab Setup	PC having Turbo C installed.

I. Write a program to demonstrate single dimensional arrays. [SOL5_1_1.c]

1. Open a new file in Turbo C editor and type the following code in it.

```
#include<stdio.h>
#include<conio.h>

int main(void)
{
    int a[5]; /* a single dimensional array of 5 integers */

    int b[6] = { 11, 22, 33, 44, 55, 66 }; /* array initialization */

    float c[4] ; /* a 1-dimensional array of 4 floating point numbers*/

    int i ;
    float sum=0.0;

    clrscr();

    a[0] = 10; /* first element */
    a[1] = 20;
    a[2] = 30;
    a[3] = 50;
    a[4] = 80; /* last element */

    printf("\n Printing elements of array \'a\'");
    for (i=0; i<=4; i++)
    {
        printf("\n Element %d is %d at position %u", i, a[i], &a[i]);

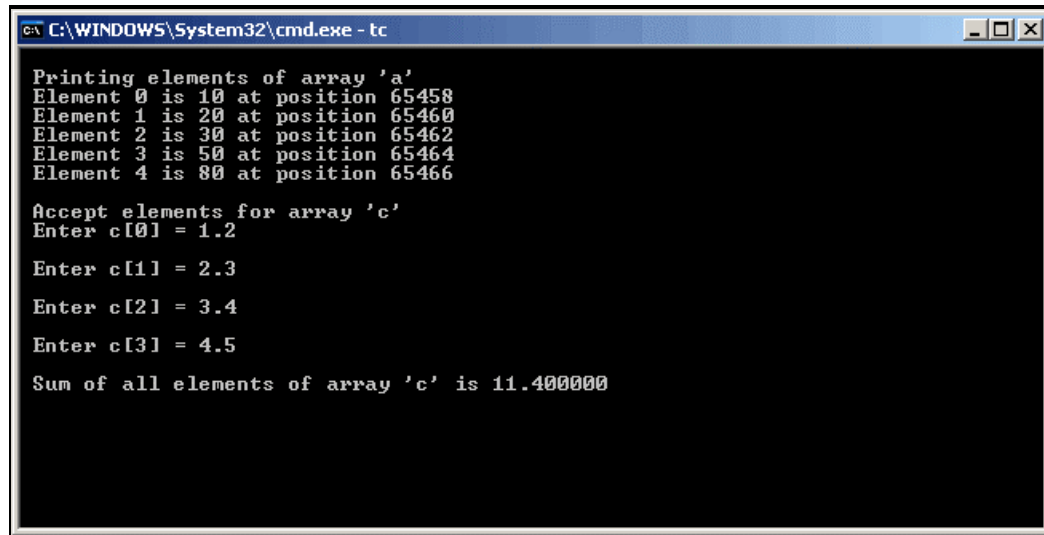
    }
    printf("\n\n Accept elements for array \'c\'");
    for (i=0; i<4; i++)
    {
        printf("\n Enter c[%d] = ", i);
        scanf("%f", &c[i]);
        sum = sum + c[i];
    }

    printf("\n Sum of all elements of array \'c\' is %f", sum);

    return(0);
}
```

Example 5-1.1: Single Dimensional Arrays

2. Save the file as SOL5_1_1.c and compile the code using Compile -> Compile to OBJ menu
3. Generate the executable SOL5_1_1.exe by choosing Compile -> Make EXE file menu
4. Run the program by choosing Run -> Run menu
5. See the output generated by choosing Run -> User screen or by pressing Alt+F5. The output of above code is shown here.



```
C:\WINDOWS\System32\cmd.exe - tc
Printing elements of array 'a'
Element 0 is 10 at position 65458
Element 1 is 20 at position 65460
Element 2 is 30 at position 65462
Element 3 is 50 at position 65464
Element 4 is 80 at position 65466

Accept elements for array 'c'
Enter c[0] = 1.2
Enter c[1] = 2.3
Enter c[2] = 3.4
Enter c[3] = 4.5
Sum of all elements of array 'c' is 11.400000
```

Figure 5-1.1: Single Dimensional Arrays

- II. Write a program to display the minimum and maximum number from an array of 10 integers along with their position in the array. [SOL5_1_2.C]

<<To do>>

- III. Write a program to store characters in an array and search the array for a given character. Also count the number of occurrences of the character in the array. [SOL5_1_3.C]

<<To do >>

1. Open a new file in Turbo C editor and start coding the main function
2. Declare a character array (str), and a character (ch) to be searched in the character array.
3. Declare an integer counter (cnt) to count the number of occurrences of ch in str. Initialize the counter to 0
4. Accept the values for str and ch
5. Set a loop, which will compare each character within str with ch. If they match, increment cnt by 1
6. Print the value of cnt. If cnt is 0, print an appropriate message.
7. Complete the code, save, compile, build and execute the same.

Lab 5-2: Multi-dimensional arrays

Goals	Write and execute C programs to understand multi-dimensional arrays
Time	30 Minutes
Lab Setup	PC having Turbo C installed.

I. Write a C program to perform multiplication for 2 integer matrices. [SOL5_2-1.C]

1. Open a new file in TC editor and type the following code in it.

```
#include<stdio.h>
#include<conio.h>
#define M 2
#define N 3
#define P 4

int main(void)
{
    int a[M][N] = { {1,2,3}, {4,5,6} }; /* initialization of 2-D integer array */
    int b[N][P], c[M][P];
    int i, j, k;

    clrscr();

    printf("\n Accepting input for matrix B ... \n");
    for (i=0; i<N; i++)
    {
        for(j=0; j<P; j++)
        {
            printf(" B[%d][%d] = ", i, j);
            scanf("%d", &b[i][j]);
        }
    }

    printf("\n Performing multiplication A * B ... \n");
    for (i=0; i<M; i++)
    {
        for(j=0; j<P; j++)
        {
            c[i][j]=0;
            for(k=0; k<N; k++)
            {
                c[i][j] += a[i][k]*b[k][j];
            }
        }
    }
}
```

Example 5-2.1: Multidimensional Arrays

```

printf("\n Array A is ... \n");
for (i=0; i<M; i++)
{
    for(j=0; j<N; j++)
        printf("\t %d", a[i][j]);
    printf("\n");
}

printf("\n Array B is ... \n");
for (i=0; i<N; i++)
{
    for(j=0; j<P; j++)
        printf("\t %d", b[i][j]);
    printf("\n");
}

printf("\n Array C is ... \n");
for (i=0; i<M; i++)
{
    for(j=0; j<P; j++)
        printf("\t %d", c[i][j]);
    printf("\n");
}

return(0);
}

```

Example 5-2.1: Multidimensional Arrays Contd...

2. Save the file as SOL5_2_1.C. Compile and make SOL5_2_1.exe.
3. Run the code for various values of Matrix B. A sample output is shown below.

```

C:\WINDOWS\System32\cmd.exe - tc
Accepting input for matrix B ... B[0][0] = 1
B[0][1] = 1
B[0][2] = 1
B[0][3] = 1
B[1][0] = 1
B[1][1] = 1
B[1][2] = 1
B[1][3] = 1
B[2][0] = 1
B[2][1] = 1
B[2][2] = 1
B[2][3] = 1

Performing multiplication A * B ...
Array A is ...
1      2      3
4      5      6
Array B is ...
1      1      1
1      1      1
1      1      1
Array C is ...
6      6      6      6
15     15     15     15

```

Figure 5-2.1: Multidimensional Arrays

Lab 5-3: Character Arrays and Strings

Goals	Write and execute C programs to understand character arrays and strings
Time	30 Minutes
Lab Setup	PC having Turbo C installed.

I. Write a program to demonstrate character array processing without using standard library functions. [SOL5_3_1.C]

1. Open a new file in Turbo C editor and type the following code in it.

```
#include<stdio.h>
#include<conio.h>

int main(void)
{
    char str1[10],str2[10],str3[20];
    int len=0, i = 0, j;
    char ch;
    clrscr();

    printf("\n Enter string1:");

    while( ((ch = getch()) != '\n') && i<9 ) /* Accept characters until
                                                Either Enter key is hit or counter exceeds 9,
                                                as the last character will be the null
                                                character*/
        str1[i++] = ch;

    str1[i]='\0'; /* Null terminating character array to make it a string */
    printf("\n You entered String1 as : %s", str1);

    while (str1[len] != '\0')
        len++;
    printf("\n Length of string1 is %d", len);

    for(i=0; str1[i] != '\0'; i++)
        str2[i] = str1[i];
    str2[i] = '\0';
    printf("\n\n String2 copied from string1 is %s", str2);

    fflush(stdin);
    printf("\n\n Enter string 2 :");
    scanf("%s",str2);

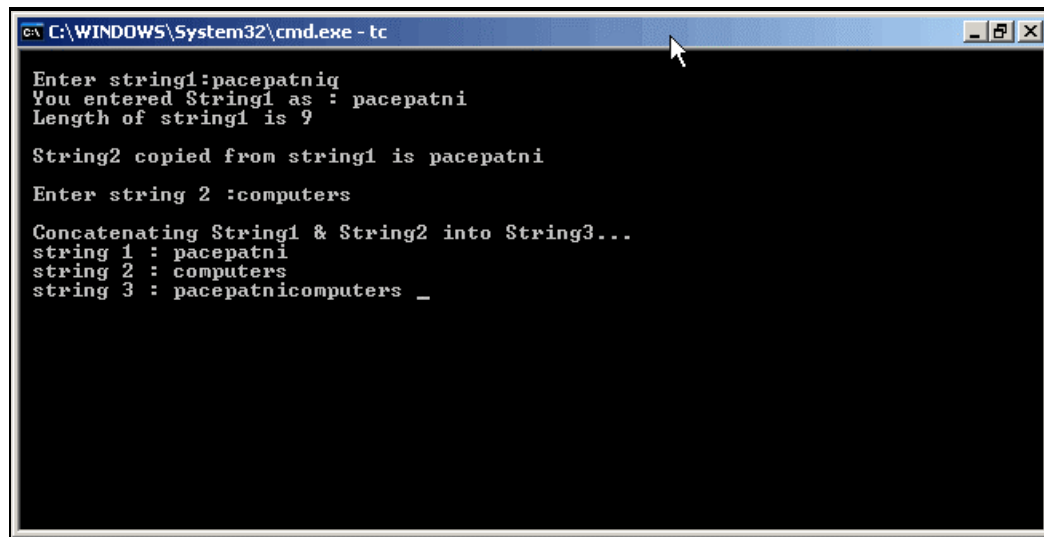
    printf("\n Concatenating String1 & String2 into String3...");
    for(i=0; str1[i] != '\0'; i++)
        str3[i] = str1[i];
    for(j=0; str2[j] != '\0'; j++, i++)
        str3[i] = str2[j];
```

Example 5-3.1: Character Arrays


```
    str3[i] = '\0';    /* Null terminate str3 to make it a string */  
    printf("\n string 1 : %s ", str1);  
    printf("\n string 2 : %s ", str2);  
    printf("\n string 3 : %s ", str3);  
    return(0);  
}
```

Example 5-3.1: Character Arrays Contd...

2. Save the file as SOL5_3_1.C. Compile the code and build an executable.
3. A sample run of the above code generates following output.



```
C:\WINDOWS\System32\cmd.exe - tc  
Enter string1:pacepatni  
You entered String1 as : pacepatni  
Length of string1 is 9  
String2 copied from string1 is pacepatni  
Enter string 2 :computers  
Concatenating String1 & String2 into String3...  
string 1 : pacepatni  
string 2 : computers  
string 3 : pacepatnicomputers _
```

Figure 5-3.1: Character Arrays

- II. Write a C program that will accept a line of text in an array and then write it out backwards in all capital letters. Allow the length of the line to be unspecified (terminated by a carriage return), but assume that it will not exceed 80 characters.

[SOL5_3_2.C]

<<To do>>

Lab 5-4: String Library functions

Goals	Write and execute C programs to perform string operations using standard library functions.
Time	30 Minutes
Lab Setup	PC having Turbo C installed.

I. Write a program to perform some string processing using library functions declared in the header file, string.h [SOL5_4_1.C]

1. Open a new file in Turbo c editor and type the following code in it.

```
#include<stdio.h>
#include<string.h>
#include<conio.h>

int main(void)
{
    char str1[10],str2[10],str3[20];

    clrscr();

    printf("\n Enter string1:");
    scanf("%s",str1);
    printf("\n Length of string1 is %d", strlen(str1));

    strcpy(str2,str1);
    printf("\n\n String2 copied from string1 is %s",str2);

    fflush(stdin);
    printf("\n\n Enter string 2 :");
    scanf("%s",str2);

    if(strcmp(str1,str2) == 0)
        printf("\n string 1 and string 2 match");
    else
        printf("\n string 1 and string 2 do not match");

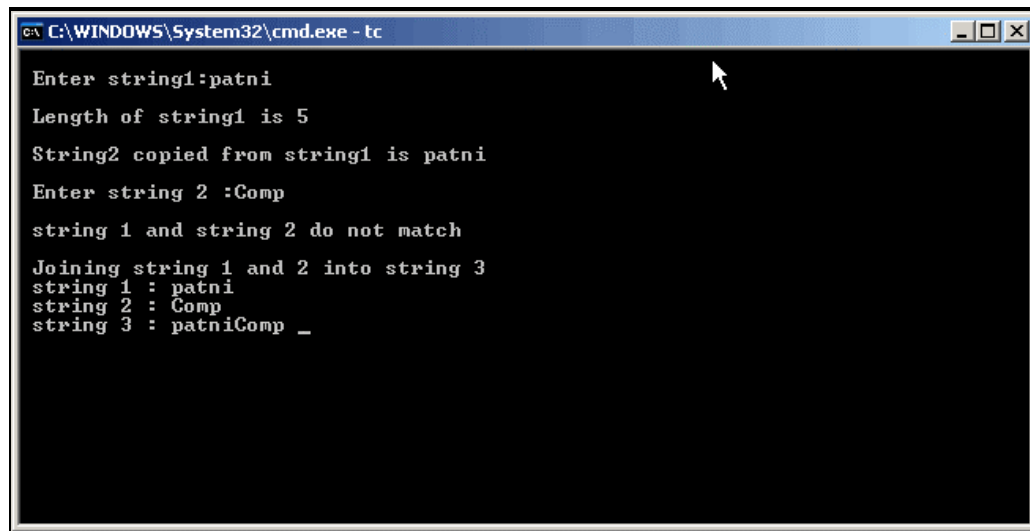
    printf("\n\n Joining string 1 and 2 into string 3");

    strcpy(str3,str1);
    strcat(str3,str2);

    printf("\n string 1 : %s ", str1);
    printf("\n string 2 : %s ", str2);
    printf("\n string 3 : %s ", str3);
    return(0);
}
```

Example 5-4.1: Library Functions for String Handling

2. Save the file as SOL5_4_1.C, compile and build the executable.
3. A sample output of the above code is shown here.



```
C:\WINDOWS\System32\cmd.exe - tc

Enter string1:patni
Length of string1 is 5
String2 copied from string1 is patni
Enter string 2 :Comp
string 1 and string 2 do not match
Joining string 1 and 2 into string 3
string 1 : patni
string 2 : Comp
string 3 : patniComp _
```

Figure 5-4.1: Library Functions for String Handling

- II. Write an interactive C program that will encode or decode a line of text. To encode a line of text, proceed as follows:
 - i. Convert each character including blank spaces to its ASCII equivalent.
 - ii. Generate a positive random integer. Add this integer to the ASCII equivalent of each character. The same random integer will be used for the entire line of text.
 - iii. Suppose the permissible values for ASCII code fall in the range of N1 and N2 (e.g. 0 to 255). If the number obtained in step ii above exceeds N2, then subtract the largest possible multiple of N2 from this number and then add the remainder to N1. Hence, the encoded number will always fall between N1 and N2, and will always represent some ASCII character.
 - iv. Print the characters that correspond to encoded ASCII values.
 - v. Reverse the entire procedure to decode the line of text and print the original line of text again.

[SOL5_4_2.C]

<<To do>>

Lab 5-5: Two Dimensional Array of Characters

Goals	Write and execute C programs to understand 2-dimensional arrays of characters.
Time	30 Minutes
Lab Setup	PC having Turbo C installed.

I. Write a program to sort the strings entered by a user alphabetically. [SOL5_5_1.C]

1. Open a new file in Turbo C editor and type the following code in it.

```
#include<stdio.h>
#include<string.h>
#include<conio.h>

int main(void)
{
    int i, num =0;
    char strs[10][20]; /* 10 strings. Each of maximum 20 characters*/

    void sort(int, char[][20]); /* Function prototype */

    clrscr();

    printf("\n Enter the strings to be sorted on seperate lines");
    printf("\n Type 'END ' when finished \n\n");

    do
    {
        printf(" String %d : ", num+1);
        scanf("%s", strs[num]);
    }while(strcmpi(strs[num++], "END"));

    sort(--num, strs);

    printf("\n Sorted list of strings is ...\n");
    for(i=0; i<num; i++)
        printf("\n String %d : %s ", i+1, strs[i]);

    return(0);
}
```

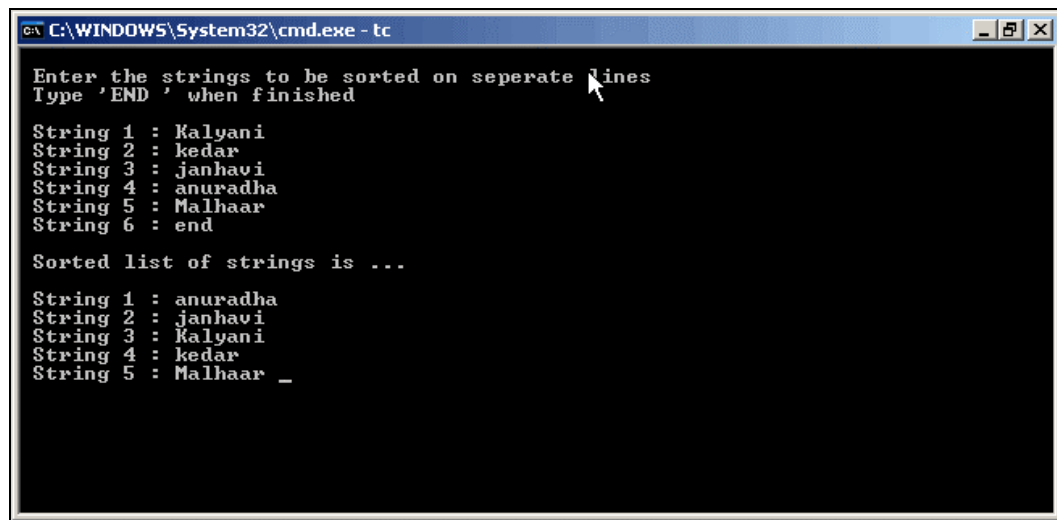
Example 5-5.1: Two Dimensional Array of Characters

```
void sort(int n, char s[][20])
{
    char temp[20];
    int i, j;

    for(i=0; i<n-1; i++)
    {
        for(j= i+1; j<n; j++)
        {
            if (strcmp(s[i], s[j]) > 0)
                /* Swap if earlier string is
                alphabetically larger than the latter one */
            {
                strcpy (temp, s[i]);
                strcpy(s[i], s[j]);
                strcpy(s[j], temp);
            }
        }
    }
}
```

Example 5-5.1: Two Dimensional Array of Characters Contd...

2. Save the file as SOL5-5_1.C, compile and make SOL5_5_1.exe.
3. Run the executable. A sample run of the code is shown below.



```
C:\WINDOWS\System32\cmd.exe - tc

Enter the strings to be sorted on seperate lines
Type 'END ' when finished

String 1 : Kalyani
String 2 : kedar
String 3 : janhavi
String 4 : anuradha
String 5 : Malhaar
String 6 : end

Sorted list of strings is ...

String 1 : anuradha
String 2 : janhavi
String 3 : Kalyani
String 4 : kedar
String 5 : Malhaar _
```

Figure 5-5.1: Two Dimensional Array of Characters

II. Consider the following list of countries and their capitals:

Australia	Melbourne
England	London
France	Paris
India	New Delhi
Italy	Rome
Japan	Tokyo
Sri Lanka	Colombo
United States	Washington, D. C.

Write an interactive C program that will accept the name of a country as input and write out the corresponding capital, and vice versa. Design the program so that it executes repetitively, until the word END is entered as input. [SOL5_5_2.C]

<<To do >>

1. Declare two 2-dimensional character arrays, first one for countries and the second one for capitals
2. Initialize both the arrays with given values
3. Accept a string, str
4. Search for the string str, in the first array. If str is found in the first array, at the position j, print the value from the second array from the same position j
5. Otherwise, search for the string str, in the second array. If str is found in the second array, at the position j, print the value from the first array from the same position j
6. If str is not found in any of the arrays, print appropriate message.
7. Repeat steps 3 to 6, until the string entered is END

Lab 6-1: Pointers

Goals	Write and execute C programs to understand basics of pointers
Time	30 Minutes
Lab Setup	PC having Turbo C installed.

I. Write a program to demonstrate simple pointer operations. [SOL6_1_1.c]

1. Open a new file in Turbo C editor and type following code in it.

```
#include<stdio.h>
#include<conio.h>

int main(void)
{
    int i=100,j=-200;           /* an integer*/
    int* p1;                   /* a pointer to an integer*/
    int *p2;

    clrscr();

    p1 = &i;                   /* p stores address of i */
    p2 = &j;

    printf("\n address of i, &i = %u", &i);
    printf("\n value of i, i = %d", i);

    printf("\n\n address of i, p1 = %u", p1);
    printf("\n value of i, *p1 = %d", *p1);

    printf("\n\n address of j, &j = %u", &j);
    printf("\n value of j, j = %d", j);

    printf("\n\n address of j, p2 = %u", p2);
    printf("\n value of j, *p2 = %d", *p2);

    printf("\n\n Difference in pointers, p2 - p1 is %d", p2-p1);
    printf("\n It is (p2 - p1)/sizeof(int) ");

    printf("\n\n Constant 2 added to pointer, (p1 + 2) is %u", (p1+2));
    printf("\n (p1 + c) is ( p1+ c * (sizeof (int) ) ");

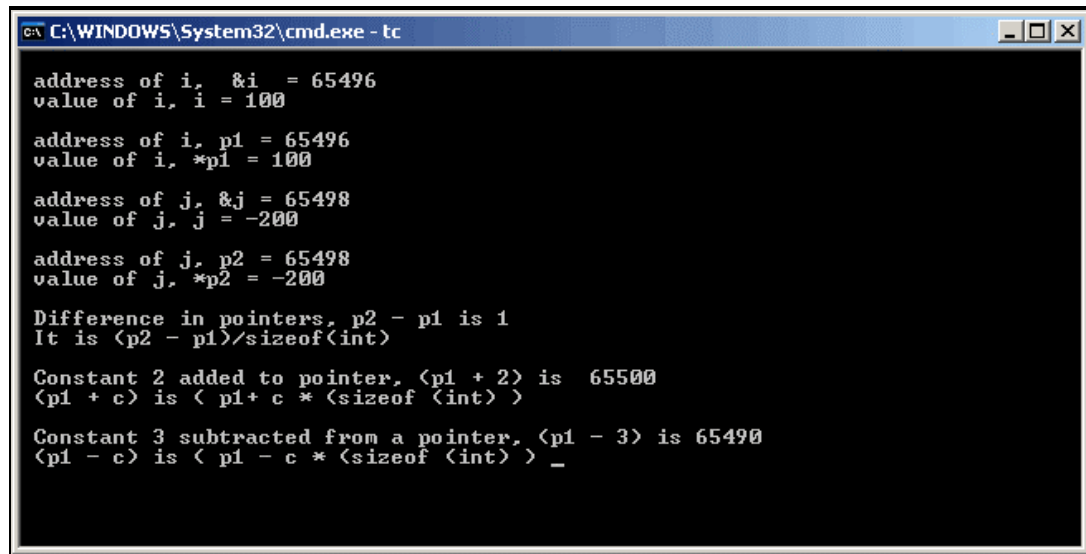
    printf("\n\n Constant 3 subtracted from a pointer, (p1 - 3) is %u",
    (p1-3));

    printf("\n (p1 - c) is ( p1 - c * (sizeof (int) ) ");

    return(0);
}
```

Example 6-1.1: Pointers

2. Save the file as SOL6_1_1.c, compile and build SOL6_1_1.exe.
3. A sample output is shown below.



```
C:\WINDOWS\System32\cmd.exe - tc

address of i, &i = 65496
value of i, i = 100

address of i, p1 = 65496
value of i, *p1 = 100

address of j, &j = 65498
value of j, j = -200

address of j, p2 = 65498
value of j, *p2 = -200

Difference in pointers, p2 - p1 is 1
It is (p2 - p1)/sizeof(int)

Constant 2 added to pointer, (p1 + 2) is 65500
(p1 + c) is ( p1+ c * (sizeof (int) )

Constant 3 subtracted from a pointer, (p1 - 3) is 65490
(p1 - c) is ( p1 - c * (sizeof (int) ) _
```

Figure 6-1.1: Pointers

Lab 6-2: Call by value vs. Call by reference

Goals	Write and execute C programs to understand methods of passing parameters to functions.
Time	20 Minutes
Lab Setup	PC having Turbo C installed.

I. Implement and test functions to demonstrate call by value and call by reference method of parameter passing. [SOL6_2_1.c]

1. Open a new file in Turbo C editor and type the following code in it.

```
#include <stdio.h>
#include <conio.h>

void MakeAbs_val(int i) /* Call by value */
{
    if (i < 0)
        i = -i;
    printf("\n Inside Function, absolute value = %d", i);
}

void MakeAbs_ref(int* p) /* Call by reference */
{
    if (*p < 0)
        *p = -*p;
    printf("\n Inside Function, absolute value = %d", *p);
}

int main(void)
{
    int i; /* i is local to main */
    clrscr();
    i = -5;
    printf("\n Call by value");

    printf("\n Before function, inside main i= %d", i);
    MakeAbs_val(i); /* function call */

    printf("\n After function, again in main i= %d", i);

    i = -5;
    printf("\n\n\n Call by reference/ address");

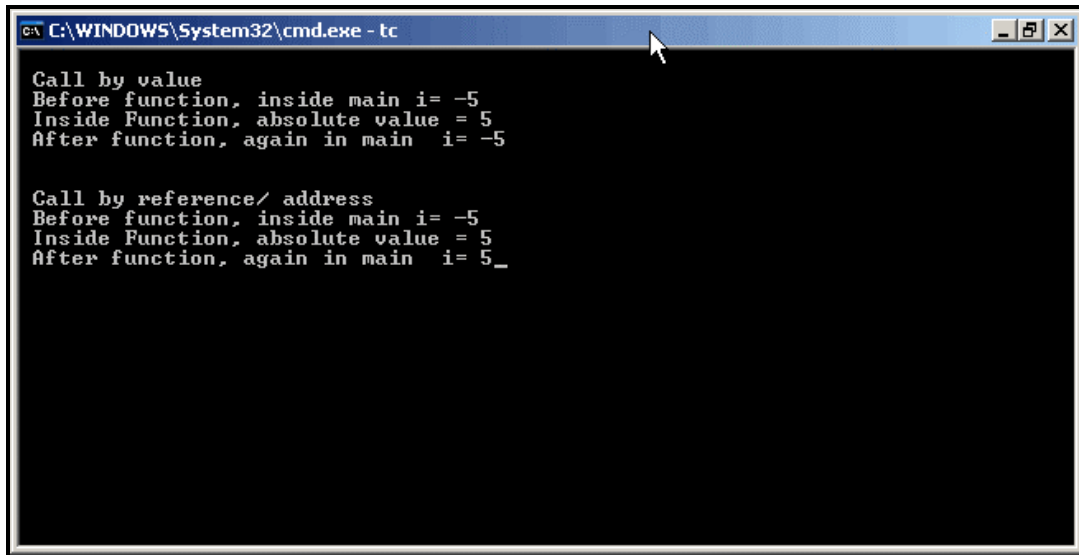
    printf("\n Before function, inside main i= %d", i);

    MakeAbs_ref(&i); /* function call */

    printf("\n After function, again in main i= %d", i);
    return(0);
}
```

Example 6-2.1: Call by Reference

2. Save the file as SOL6_2_1.C, compile and build the same.
3. A sample run of the executable is shown here.



```
C:\WINDOWS\System32\cmd.exe - tc

Call by value
Before function, inside main i= -5
Inside Function, absolute value = 5
After function, again in main i= -5

Call by reference/ address
Before function, inside main i= -5
Inside Function, absolute value = 5
After function, again in main i= 5_
```

Figure 6-2.1: Call by Reference

Lab 6-3: Arrays and pointers

Goals	Write and execute C programs to understand handling arrays using pointer-offset notations.
Time	30 Minutes
Lab Setup	PC having Turbo C installed.

I. Write a program to demonstrate single dimensional array-handling using pointers.

1. Open a new in Turbo C editor and type the following code in it.

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>

int main(void)
{
    float c[5]; /* a single dimensional array of 5 floating point numbers*/
    char str[5]={ 'p', 'a', 't', 'h', 'i' }; /* char array initialization */
    int i;
    float sum=0.0;
    void make_caps(char s[], int n); /* function prototype */

    clrscr();
    printf("\n\n Accept elements for array \c\n");
    for (i=0; i<5; i++)
    {
        printf("\n Enter c[%d] = ", i);
        scanf("%f", (c+i)); /* array name is a pointer to first element*/
        sum = sum + *(c+i);
    }

    printf("\n Sum of all elements of array \c\ is %f", sum);

    printf("\n\n Character array before function call \n");
    for (i=0; i<5; i++)
        printf("%c", str[i]);

    make_caps(str, 5); /* passing an array to a function */

    printf("\n\n Character array after function call \n");
    for (i=0; i<5; i++)
        printf("%c", *(str+i)); /* array name is a pointer to first element*/
    return(0);
}

void make_caps(char s[], int n)
{
    int i;
    for (i=0; i<n; i++)
        s[i] = toupper(*(s+i));
}
```

Example 6-3.1: Arrays & Pointers

Note: While executing this program, you may encounter the following run-time error message with Borland C/C++ and Turbo C/C++

**scanf : floating point formats not linked
Abnormal program termination**

as shown in the snapshot below.

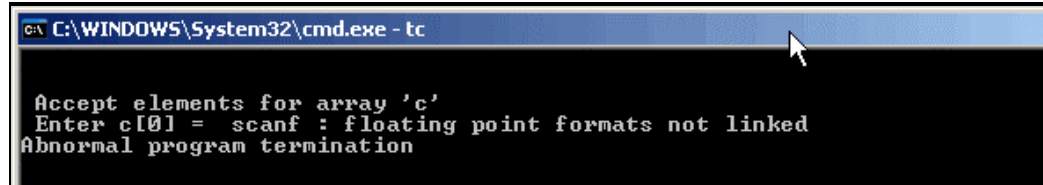


Figure 6-3.1: Arrays & Pointers - Output 1

"Floating point formats not linked" is a Borland run-time error (Borland C or C++, Turbo C or C++). Borland's compilers do not link in the floating-point library unless you need it. One common case is where you don't call any floating-point functions, but you have %f or other floating point formats in scanf () or printf () calls. The cure is to call a floating-point function, or at least force one to be present in the link e.g. you may define a dummy function like dummyFunction() in your program and do not call it anywhere in your program

```
void dummyFunction ( void )  
{  
    float dummyFloat;  
    scanf("%f", &dummyFloat);  
    printf("%f", dummyFloat);  
}
```

Once done, the output of above program will be as follows:

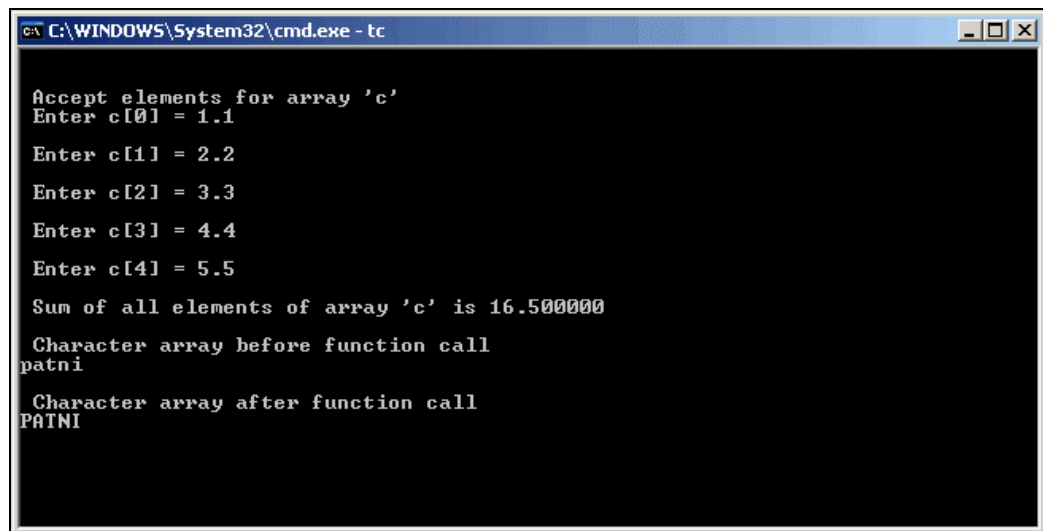


Figure 6-3.2: Arrays & Pointers - Output 2

- II. Write a function `void split (int array[], int pivot)` using pointers to partition the given array into two parts: one with all elements whose values are \leq pivot and the other one with all elements whose values are $>$ pivot. The array should be partitioned in place. For example, if the array is

13	-42	8	35	-7	46	28	-19
----	-----	---	----	----	----	----	-----

and the pivot given is 10, then the function should turn the array above into

-19	-42	8	-7	35	46	28	13
-----	-----	---	----	----	----	----	----

^

Within which all elements before ^ are ≤ 10 and all elements after ^ are > 10 . Write a main program to test your function. [SOL6_3_2.c]

<<To do>>

- III. Write a function `int replace_char(char *str, char old_char, char new_char)` which replaces each occurrence of the character `old_char` with the character `new_char` in the string `str`. The function should return the number of characters actually replaced. Write a program to test your function.

<<To do>>

Lab 6-4: Pointers to pointers

Goals	Write and execute C programs to understand pointers to pointers.
Time	30 Minutes
Lab Setup	PC having Turbo C installed.

I. Demonstrate a pointer to a pointer. [SOL6_4_1.c]

1. Open a new file in Turbo C editor and type following code in it.

```
#include<stdio.h>

int main(void)
{
    int n = 100;
    int *pt = &n;
    int **pt_to_pt = &pt;

    printf("\n\n Value of n is....");
    printf("\n n = %d ", n);
    printf("\n *pt = %d ", *pt);
    printf("\n **pt_to_pt = %d", **pt_to_pt);

    printf("\n\n Address of n is....");
    printf("\n &n = %u ", &n);
    printf("\n pt = %u ", pt);
    printf("\n *pt_to_pt = %u", *pt_to_pt);

    printf("\n\n Value of pt is address of n and it is....");
    printf("\n pt = %u ", pt);
    printf("\n *pt_to_pt = %u", *pt_to_pt);

    printf("\n\n Address of pt is....");
    printf("\n &pt = %u ", &pt);
    printf("\n pt_to_pt = %u", pt_to_pt);

    printf("\n\n Value of pt_to_pt is address of pt and it is....");
    printf("\n &pt = %u ", &pt);
    printf("\n pt_to_pt = %u", pt_to_pt);

    printf("\n\n Address of pt_to_pt is....");
    printf("\n &pt_to_pt = %u ", &pt_to_pt);

    return(0);
}
```

Example 6-4.1: Pointers to Pointers

2. Save the file as SOL6_4_1.C , compile it and build the executable.

3. The output generated will be as shown below.

```

C:\WINDOWS\System32\cmd.exe - tc

Value of n is....
n = 100
*pt = 100
**pt_to_pt = 100

Address of n is....
&n = 65494
pt = 65494
*pt_to_pt = 65494

Value of pt is address of n and it is...
pt = 65494
*pt_to_pt = 65494

Address of pt is....
&pt = 65496
pt_to_pt = 65496

Value of pt_to_pt is address of pt and it is....
&pt = 65496
pt_to_pt = 65496

Address of pt_to_pt is....
&pt_to_pt = 65498
  
```

Figure 6-4.1: Pointers to Pointers

II. Demonstrate working with a 2- dimensional array using pointer-offset notation.
[SOL6_4_2.C]

1. Open a new file in Turbo C editor and type following code in it.

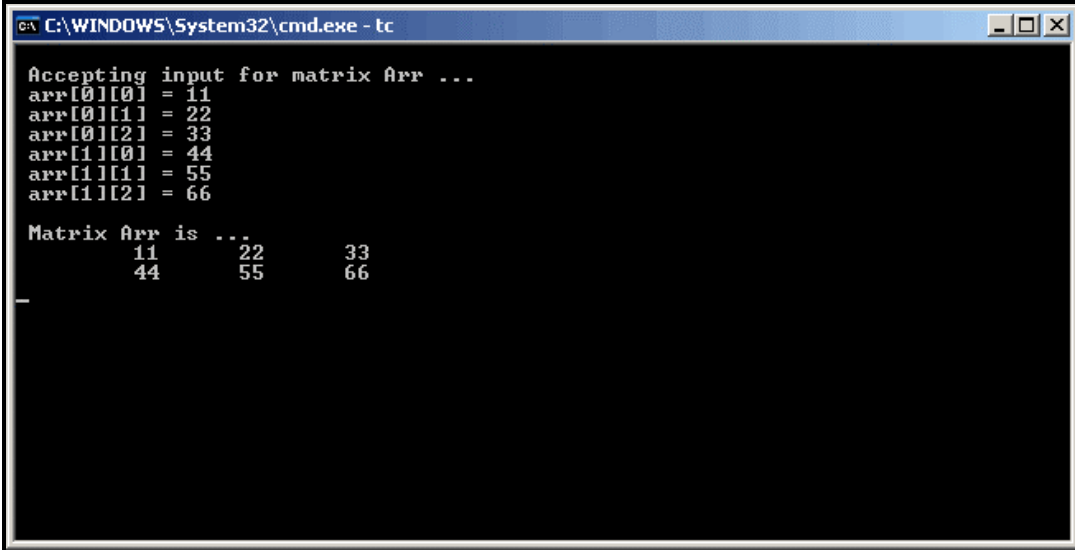
```

#include<stdio.h>
#include<conio.h>

int main(void)
{
    int arr[2][3]; /* 2-D integer array */
    int i, j;
    clrscr();
    printf("\n Accepting input for matrix Arr ... \n");
    for (i=0; i<2; i++)
    {
        for(j=0; j<3; j++)
        {
            printf(" arr[%d][%d] = ",i,j);
            scanf("%d", (*(arr+i)+j));
        }
    }
    printf("\n Matrix Arr is ... \n");
    for (i=0; i<2; i++)
    {
        for(j=0; j<3; j++)
            printf("\t %d", (*(arr+i)+j) );
        printf("\n");
    }
    return(0);
}
  
```

Example 6-4.2: Two Dimensional Arrays & Pointers

2. Save the file as SOL6_4_2.C. Compile and build the same.
3. A sample output is shown here.



```
C:\WINDOWS\System32\cmd.exe - tc
Accepting input for matrix Arr ...
arr[0][0] = 11
arr[0][1] = 22
arr[0][2] = 33
arr[1][0] = 44
arr[1][1] = 55
arr[1][2] = 66

Matrix Arr is ...
    11    22    33
    44    55    66
```

Figure 6-4.2: Two Dimensional Arrays & Pointers

Lab 6-5: Command Line Arguments

Goals	Write and execute C programs to understand command line arguments
Time	30 Minutes
Lab Setup	PC having Turbo C installed.

- I. Write a C program to sort an integer array. Accept the number of elements to be sorted from command line and allocate exact amount of memory dynamically. [SOL6_5_1.C]

1. Open a new file in Turbo C editor and type the following code in it.

```
#include<stdio.h>
#include<alloc.h>

void insertion_sort(int *p, int x)
{
    int j,k,temp;
    for (j=0;j<x;j++)
    {
        for (k=j+1;k<x;k++)
        {
            if(*(p+j) > *(p+k))
            {
                temp = *(p+j);
                *(p+j)=*(p+k);
                *(p+k)= temp;
            }
        }
    }
}
```

Example 6-5.1: Command Line Arguments

```
int main(int argc, char* argv[])
{
    int i, n, *ip;

    if(argc != 2)
    {
        printf("\n Wrong no. of arguments. Please enter 2
arguments");
        exit(1);
    }

    n = atoi(argv[1]); /* convert second agument to numeric value*/

    if( ! n)
    {
        printf("\n Wrong argument. Second argument should be
numeric");
        exit(1);
    }

    printf(" *** Sorting an array using Insertion sort ****");
    printf("\n No. of elements to be sorted : %d", n);

    ip = (int*)malloc(n * sizeof(int))    ;

    for(i=0;i<n;i++)
    {
        printf("\n Enter element %d :", i);
        scanf("%d", (ip+i));
    }

    printf("\n\n Array before sorting\n");
    for(i=0;i<n;i++)
        printf("%d\t", *(ip+i));

    insertion_sort(ip,n);

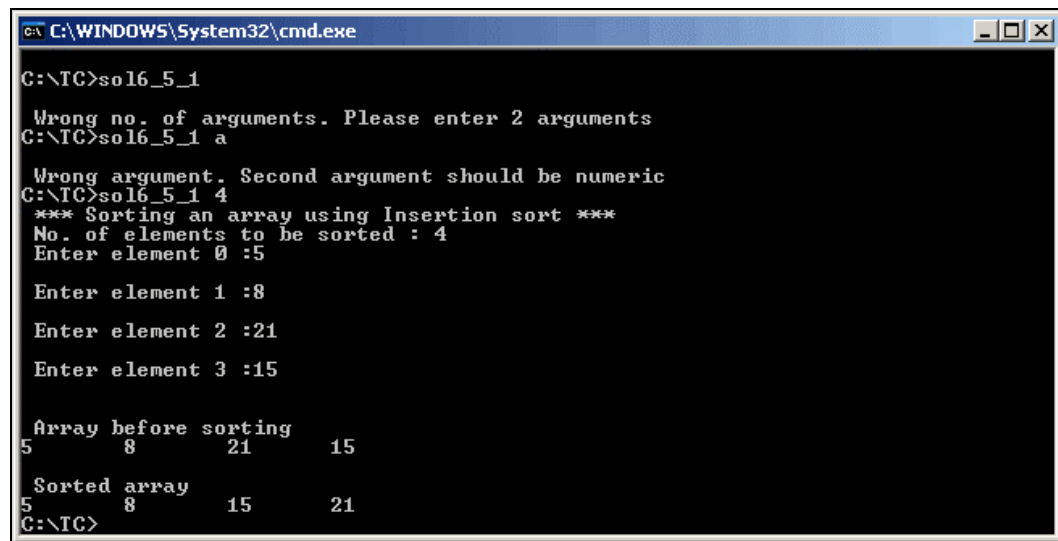
    printf("\n\n Sorted array \n");
    for(i=0;i<n;i++)
        printf("%d\t", ip[i]);

}
```

Example 6-5.1: Command Line Arguments Contd...

2. Save the file as SOL6_5_1.C, compile and build the same to generate an executable.

3. A few sample runs of the code are shown below.



```
C:\WINDOWS\System32\cmd.exe
C:\TC>sol6_5_1
Wrong no. of arguments. Please enter 2 arguments
C:\TC>sol6_5_1 a
Wrong argument. Second argument should be numeric
C:\TC>sol6_5_1 4
*** Sorting an array using Insertion sort ***
No. of elements to be sorted : 4
Enter element 0 :5
Enter element 1 :8
Enter element 2 :21
Enter element 3 :15
Array before sorting
5      8      21      15
Sorted array
5      8      15      21
C:\TC>
```

Figure 6-5.1: Command Line Arguments

- II. Write a program to calculate sum of all the integer arguments supplied at command line e.g.

```
C:\TC> SOL6_5_2      1      606      30      29
Would yield the output of 666.      [SOL6_5_2.c]
<<To do>>
```

Lab 7-1: Structures

Goals	Write and execute C programs to work with structures.
Time	30 Minutes
Lab Setup	PC having Turbo C installed.

- I. Define a structure to represent an employee record containing information like employee id, name and salary for an employee. Use this structure to store and print the values for the same. [SOL7_1_1.C]
1. Type the following code in a file named SOL7_1_1.c. Save, compile and build the same.

```
#include<stdio.h>
#include<conio.h>

/*structure definition */
struct emp
{
    int empid;
    char name[20];
    float sal;
} e1={1234,"John",1500.00}; /* Structure initialization */

int main(void)
{
    struct emp e2, e3={2222,"Tom",2500.00};
                                /* Structre initialization */

    clrscr();

    printf("\n Employee record e1 is :");
    printf("\n %d \t %s \t %f", e1.empid, e1.name, e1.sal);

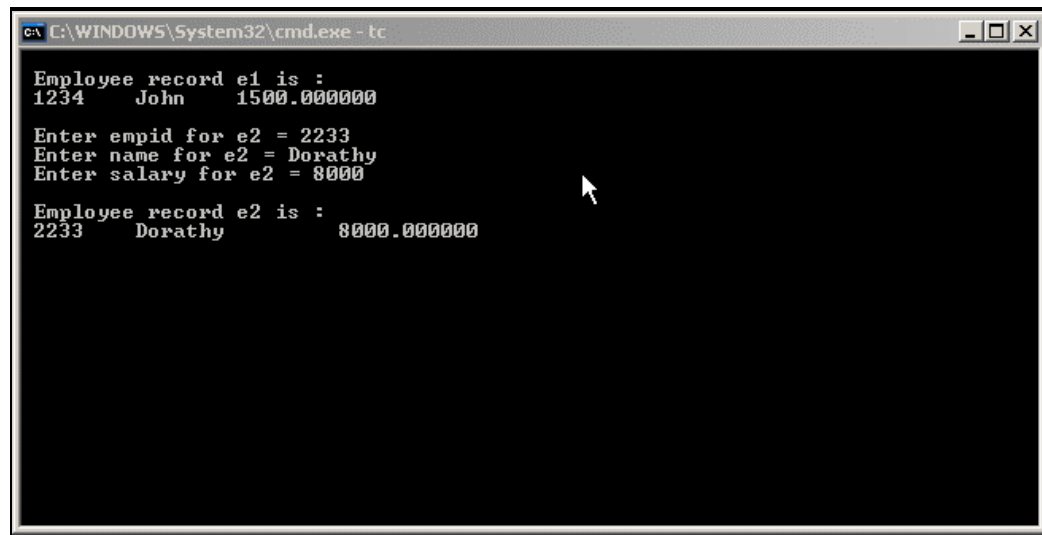
    printf("\n\n Enter empid for e2 = ");
    scanf("%d", &e2.empid);
    printf(" Enter name for e2 = ");
    scanf("%s", e2.name);
    printf(" Enter salary for e2 = ");
    scanf("%f", &e2.sal);

    printf("\n Employee record e2 is :");
    printf("\n %d \t %s \t %f", e2.empid, e2.name, e2.sal);

    return(0);
}
```

Example 7-1.1: Structures

2. The sample output for above code is shown below.



```
C:\WINDOWS\System32\cmd.exe - tc
Employee record e1 is :
1234   John   1500.000000

Enter empid for e2 = 2233
Enter name for e2 = Dorathy
Enter salary for e2 = 8000

Employee record e2 is :
2233   Dorathy 8000.000000
```

Figure 7-1.1: Structures

- II. Write a program that defines a structure containing 3 elements that describe an entry in your telephone book: one is the name of your friend not exceeding 20 characters, his/her telephone number and date of birth (which is a structure variable containing 3 integers for day, month and year). Use the same for storing and displaying entries for a friend. [SOL7_1_2.C]

<<To do>>

Lab 7-2: More structures

Goals	Understanding pointers as structure members and pointers to structures.
Time	30 Minutes
Lab Setup	PC having Turbo C installed.

- I. Define a structure to represent an employee record containing information like employee id, name and salary for an employee. Use this structure to store and print the values for the same. Implement name as a character pointer (char*) and allocate exact amount of memory required for the name entered by a user. Implement a function to raise the salary for an employee. [SOL7_2_1.C]
1. Type the following code in a file named SOL7_2_1.c. Save, compile and build the same.

```
#include<stdio.h>
#include<string.h>
#include<alloc.h>
#include<conio.h>

/*structure definition */
struct emp
{
    int empid;
    char *name;           /* struct containing a pointer */
    float sal;
};

int main(void)
{
    struct emp e1;        /* struct variable */
    struct emp *eptr1;     /* pointer to struct */

    void raise_sal(struct emp*, float);
                        /* function prototype*/

    char name_buf[50];

    clrscr();

    printf("\n Enter empid for e1 = ");
    scanf("%d", &e1.empid);

    /* accept the name in character buffer */
    printf(" Enter name for e1 = ");
    fflush(stdin);
    scanf("%[^\n]", name_buf); /*Accept string until carriage return*/
```

Example 7-2.1: Structures & Pointers

```
/* allocate memory required for name */
e1.name = (char*) malloc (sizeof(char) * strlen(name_buf) + 1 );
strcpy(e1.name , name_buf);

printf(" Enter salary for e1 = ");
scanf("%f", &e1.sal);

printf("\n Employee record e1 is :");
printf("\n %d \t %s \t %f", e1.empid, e1.name, e1.sal);

raise_sal(&e1,1000);          /* call by reference */

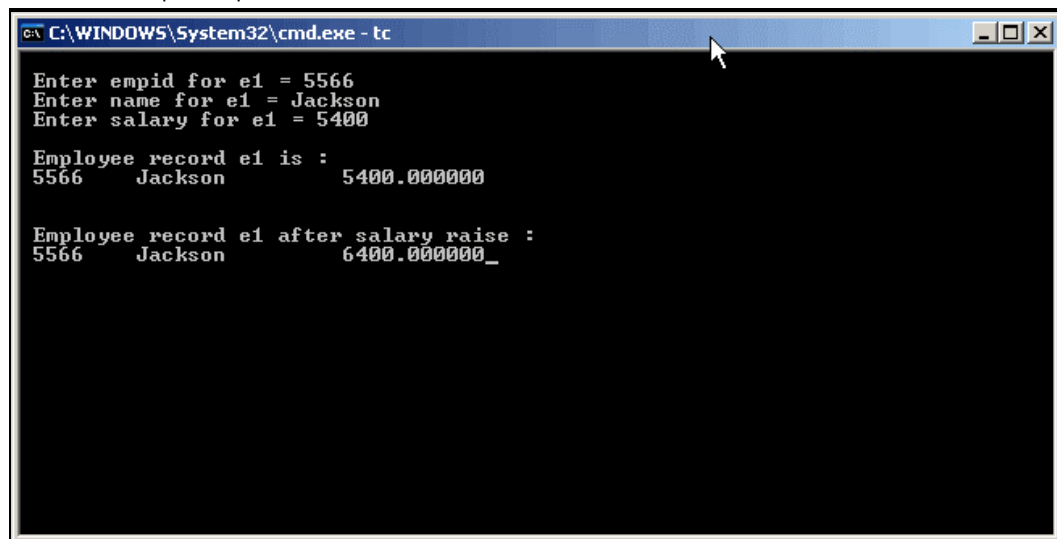
printf("\n\n Employee record e1 after salary raise :");
printf("\n %d \t %s \t %f", e1.empid, e1.name, e1.sal);

free(e1.name); /* deallocate memory for name */
return(0);
}

void raise_sal(struct emp* eptr, float raise)
{
    eptr->sal = eptr->sal + raise;    /* -> member access operator */
}
```

Example 7-2.1: Structures & Pointers Contd...

2. A sample output of above code is shown below.



```
C:\WINDOWS\System32\cmd.exe - tc

Enter empid for e1 = 5566
Enter name for e1 = Jackson
Enter salary for e1 = 5400

Employee record e1 is :
5566    Jackson    5400.000000

Employee record e1 after salary raise :
5566    Jackson    6400.000000_
```

Figure 7-2.1: Structures & Pointers

- II. Write a program that defines a structure containing 2 members that describe an entry in a telephone book: one is the name of a person, another is his/her telephone number. Implement name as a `char *` and allocate memory dynamically to hold the name entered by the user. Use the same for storing and displaying information for 2 persons. [SOL7_2_2.C]

<< To do>>

- III. Modify the program you wrote in exercise II. First it will ask how many entries will be needed. Then it will allocate an array of struct's and allow you to enter the data. Finally it will print and deallocate the array.

Note: Do not forget to release the memory allocated for the names before deallocating the entire structure array. [SOL7_2_3.C]

<<To do >>

Note: The language definition states that for each pointer type, there is a special value--the "null pointer"--which is distinguishable from all other pointer values and which is "guaranteed to compare unequal to a pointer to any object or function". That is, the address-of operator `&` will never yield a null pointer, nor will a successful call to `malloc`. (`malloc` does return a null pointer when it fails, and this is a typical use of null pointers: as a "special" pointer value with some other meaning, usually "not allocated" or "not pointing anywhere yet.")

A null pointer is conceptually different from an uninitialized pointer. A null pointer is known not to point to any object or function; an uninitialized pointer might point anywhere

Lab 8-1: Data structures

Goals	Writing and executing C programs to work with linked lists.
Time	60 Minutes
Lab Setup	PC having Turbo C installed.

- I. Write a program to perform basic linked list operations such as inserting and deleting nodes and displaying the list. Each node in the linked list should contain an employee id and his/her name not exceeding 20 characters. The list must be maintained in ascending order of the employee ids [SOL8_1_1.C]

1. Open a new file in Turbo C editor and type the following code in it.

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>

struct emp
{
    int empid;
    char name[20];
    float sal;
    struct emp *next;
};

struct emp *list;          /* global pointer to beginning of list */

struct emp *getnode() /*creates a node and accepts data */
{
    struct emp *temp;
    temp=(struct emp *)malloc(sizeof(struct emp));
    printf("Enter the employee id:");
    scanf("%d",&temp->empid);
    fflush(stdin);
    printf("Enter the name:");
    scanf("%s",temp->name);
    fflush(stdin);
    printf("Enter salary:");
    scanf("%f",&temp->sal);
    fflush(stdin);
    temp->next=NULL;
    return temp;
}
```

Example 8-1.1: Linked List

```

/* search returns address of previous node; current node is */
struct emp * search(int cd,int *flag)
{
    struct emp *prev,*cur;
    *flag=0;
    if (list==NULL)          /* list empty */
        return NULL;
    for(prev=NULL,cur=list;(cur) && ((cur->empid) < cd);
prev=cur,cur=cur->next); /*move forward to come to exact position*/
    if((cur) && (cur->empid==cd))
        /* node with given empid exists */
        *flag=1;
    else
        *flag=0;
    return prev;
}

int insert(struct emp *new)
{
    struct emp *prev;
    int flag;
    if (list==NULL)          /* list empty */
    {
        list=new;
        return 0;
    }
    prev = search(new->empid,&flag);
    if(flag == 1)              /* duplicate empid */
        return -1;

    if(prev==NULL)            /*insert at beginning */
    {
        new->next=list;
        list=new;
    }
    else                       /* insert at middle or end */
    {
        new->next=prev->next;
        prev->next=new;
    }
    return 0;
}

void displayall()
{
    struct emp *cur;
    if(list==NULL)
    {
        printf("The list is empty\n");
        return;
    }
    printf("\n\nEmpid Name                Salary\n");
    for(cur=list;cur;cur=cur->next)
    {
        printf("%4d  %-22s %8.2f\n",cur->empid,cur->name,cur->sal);
    }
}

```

Example 8-1.1: Linked List Contd...

```
int delete(int cd)
{
    struct emp *prev,*temp;
    int flag;
    if (list==NULL)                /* list empty */
        return -1;

    prev=search(cd,&flag);
    if(flag==0)                    /* empid not found */
        return -1;

    if(prev==NULL)                /* node to delete is first node (as flag is 1) */
    {
        temp=list;
        list=list->next;
        free(temp);
    }
    else
    {
        temp = prev->next;
        prev->next = temp->next;
        free(temp);
    }
    return 0;
}

void main(void)
{
    struct emp *new;
    int choice=1,cd;
    list=NULL;
    clrscr();
    do
    {
        printf("\n\n\t\t\tMenu\n\n");
        printf("\t\t\t1.Insert\n");
        printf("\t\t\t2.Delete\n");
        printf("\t\t\t3.Display list\n");
        printf("\t\t\t0.Exit\n");
        printf("\n\n\t\t\t..... choice:");
        scanf("%d",&choice);
        fflush(stdin);
    }
```

Example 8-1.1: Linked List Contd...

```
switch(choice)
{
case 1: new=getnode();
        if(insert(new)== -1)
            printf("error:cannot insert\n");
        else
            printf("node inserted\n");
        break;
case 2 :
        printf("Enter the employee id of the record to delete:") ;
        scanf("%d",&cd);
        fflush(stdin);
        if(delete(cd)==-1)
            printf("deletion failed\n");
        else
            printf("node deleted\n");
        break;
case 3 :
        displayall();
        break;
case 0 :
        exit();
}
}while(choice !=0);
}
```

Example 8-1.1: Linked List Contd...

2. Save the file as SOL8_1_1.c, compile it to the object code and build the SOL8_1_1.exe.
3. Execute the program and try out the various options.

4. A sample run of the executable SOL8_1_1.exe is shown here.



```
C:\WINDOWS\System32\cmd.exe - tc

Menu
1.Insert
2.Delete
3.Display list
0.Exit

The list is empty          ..... choice:3

Menu
1.Insert
2.Delete
3.Display list
0.Exit

..... choice:1_

Enter the employee id:18480
Enter the name:Anuradha
Enter salary:25000
node inserted

Menu
1.Insert
2.Delete
3.Display list
0.Exit

..... choice:1

Enter the employee id:15000
Enter the name:Sarah
Enter salary:30000
node inserted

Menu
1.Insert
2.Delete
3.Display list
0.Exit

..... choice:1

Enter the employee id:19000
Enter the name:Jayant
Enter salary:20000
node inserted

Menu
1.Insert
2.Delete
3.Display list
0.Exit
```

Figure 8-1.1: Linked List

```
..... choice:3

Empid Name          Salary
15000 Sarah          30000.00
18480 Anuradha       25000.00
19000 Jayant         20000.00

Menu
1.Insert
2.Delete
3.Display list
0.Exit

..... choice:2
Enter the employee id of the record to delete:12000
deletion failed

Menu
1.Insert
2.Delete
3.Display list
0.Exit

..... choice:2
Enter the employee id of the record to delete:15000
node deleted

Menu
1.Insert
2.Delete
3.Display list
0.Exit

..... choice:3

Empid Name          Salary
18480 Anuradha       25000.00
19000 Jayant         20000.00

Menu
1.Insert
2.Delete
3.Display list
0.Exit

..... choice:0
```

Figure 8-1.1: Linked List Contd...

- II. Write a menu driven program to perform the following linked list operations. Each node in the list will represent a telephone book entry consisting of the name of a person and his telephone number.

1. Add a member
2. Query a member
3. Delete a member
4. Modify a member
5. List all the members
6. Exit

[SOL8_1_2.C]

<<To do >>

Lab 9-1: File Handling

Goals	Write and execute C programs to perform various file handling operations.
Time	30 Minutes
Lab Setup	PC having Turbo C installed.

I. Write a program to demonstrate input/output operations on ASCII files.

1. Open a new file in Turbo C editor and type following code in it.

```
/*Program to make copies of itself */
# include <stdio.h>
void main(void)
{
    FILE *fs,*ft1,*ft2;
    char ch, s[80];
    fs=fopen("C:\\TC\\Solved\\SOL9_1_1.C","r");
        /* open source file for input. Absolute path mentioned here*/

    if(fs==NULL)
    {
        puts("Cannot open source file");
        exit(0);
    }

    ft1=fopen("solved\\Copy1.c","w"); /* Open target file for output.
                                      Relative path */
    if(ft1==NULL)
    {
        puts("Cannot open target file");
        fclose(fs); /* close the source file */
        exit(0);
    }

    while((ch=getc(fs))!=EOF) /* read a character from source file */
        putc(ch,ft1); /* write the character to target file*/

    fclose(ft1); /* close first target file */

    ft2=fopen("solved\\Copy2.c","w"); /* open second target file */
    if(ft2==NULL)
    {
        puts("Cannot open target file");
        fclose(fs); /* close source file */
        exit(0);
    }
    fseek(fs,0,SEEK_SET); /* Set the get pointer back to the
                           beginning of the source file*/
    while(fgets(s,79,fs) !=NULL) /* read a string form source file*/
        fputs(s,ft2); /* write a string to second target file*/

    fclose(ft2); /* Close second target file*/
    fclose(fs); /* Close source file */
}
```

Example 9-1.1: File Handling

2. Save the file as SOL9_1_1.C. The file should be saved at path C:\TC\Solved\Sol9_1_1.C or otherwise modify the absolute and relative paths mentioned in the program.
3. Compile and build the same to generate SOL9_1_1.exe.
4. Run the executable.
5. You will find 2 files viz. Copy1.C and Copy2.C created under the folder C:\TC\Solved (or under the path you had mentioned.)

- II. Write a program that reads strings from stdin and writes them to a file. Have the program finish and close the file when the string entered is "end". Read the same file and print each line to stdout.

Note that stdout (the device that puts() and printf() write to) is a FILE* that is set by the system when starting your program (the same is true for stdin). When you start your program, stdout is the screen, and stdin is the keyboard, by default.

<<To do>>

- III. Write a C program to print any ASCII file. At the end, display the total number of characters, lines in the file. Use command line argument to accept any number of file names e.g. more file1 file2 file3

<<To do >>

Lab 9-2: More File Handling

Goals	Writing and executing C programs to perform formatted and unformatted file input/output.
Time	45 Minutes
Lab Setup	PC having Turbo C installed.

- I. The employee information (empid, name, sal) is stored in a text file EMPS.TXT. Write a program to read all the records from this file and store them in a linked list fashion. Do the necessary update in the linked list such as adding new records, deleting and modifying existing records etc. Once done, save all the records back to the EMPS.TXT file. [SOL9_2_1.C]
1. Open a new file named SOL9_2_1.c in Turbo C environment and type in the following code.

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>

FILE *FP; /* global pointer to file */

struct emp
{
    int empid;
    char name[20];
    float sal;
    struct emp *next;
};

struct emp *list; /* global pointer to beginning of list */

struct emp * getnode() /*creates a node and accepts data from user*/
{
    struct emp *temp;
    temp=(struct emp *)malloc(sizeof(struct emp));
    printf("Enter the employee id:");
    scanf("%d",&temp->empid);
    fflush(stdin);
    printf("Enter the name:");
    scanf("%s",temp->name);
    fflush(stdin);
    printf("Enter salary:");
    scanf("%f",&temp->sal);
    fflush(stdin);
    temp->next=NULL;
    return temp;
}
```

Example 9-2.1: Files & Linked List

```

struct emp * getrecord() /*creates a node and reads a record from file */
{
    struct emp *temp;
    temp=(struct emp *)malloc(sizeof(struct emp));

    fscanf(FP,"%d %s %f\n", &temp->empid, temp->name, &temp->sal);

    temp->next=NULL;
    return temp;
}

/* Search returns address of previous node; current node is */
struct emp * search(int cd,int *flag)
{
    struct emp *prev,*cur;
    *flag=0;
    if (list==NULL) /* list empty */
        return NULL;
    for(prev=NULL,cur=list;(cur) && ((cur->empid) < cd);
    prev=cur,cur=cur->next);
        if((cur) && (cur->empid==cd))
            /* Node with given empid exists */
            *flag=1;
        else
            *flag=0;
    return prev;
}

int insert(struct emp *new)
{
    struct emp *prev;
    int flag;
    if (list==NULL) /* list empty */
    {
        list=new;
        return 0;
    }
    prev = search(new->empid,&flag);
    if(flag == 1) /* duplicate empid */
        return -1;

    if(prev==NULL) /*insert at beginning */
    {
        new->next=list;
        list=new;
    }
    else /* insert at middle or end */
    {
        new->next=prev->next;
        prev->next=new;
    }
    return 0;
}

```

Example 9-2.1; Files & Linked List Contd...

```

void displayall()
{
    struct emp *cur;
    if(list==NULL)
    {
        printf("The list is empty\n");
        return;
    }
    printf("\n\nEmpid Name                Salary\n");
    for(cur=list;cur;cur=cur->next)
        printf("%4d  %22s %8.2f\n",cur->empid,cur->name,cur->sal);
}

int delete(int cd)
{
    struct emp *prev,*temp;
    int flag;
    if (list==NULL)                /* list empty */
        return -1;

    prev=search(cd,&flag);
    if(flag==0)                    /* empid not found */
        return -1;

    if(prev==NULL) /* node to delete is first node (as flag is 1) */
    {
        temp=list;
        list=list->next;
        free(temp);
    }
    else
    {
        temp = prev->next;
        prev->next = temp->next;
        free(temp);
    }
    return 0;
}

void BackToFile() /*save entire list to the file*/
{
    struct emp * cur =list;
    FP = fopen("EMPS.TXT", "w");

    if (FP == NULL)
        printf("Unable to open emps.txt\n");
    else
    {
        while ( cur != NULL )
        {
            fprintf(FP,"%d %s %8.2f\n", cur->empid, cur->name, cur->sal);
            cur = cur -> next ;
        }
    }
    fclose(FP);
}

```

Example 9-2.1: Files & Linked List Contd...

```
void main(void)
{
    struct node *new;
    int choice=1,cd;

    list=NULL;
    FP = fopen("EMPS.TXT", "r");
    clrscr();

    if (FP == NULL)
        printf("Unable to open file emps.txt\n");
    else
    {
        /* populate the list from file records*/
        while(!feof(FP))
        {
            new=getrecord();
            if(insert(new)== -1)
                printf("error:cannot insert\n");
            else
                printf("File Record inserted as node\n");
        }
        fclose(FP);
    }

    do
    {
        printf("\n\t\t\t\tMenu\n\n");
        printf("\t\t\t\t1.Insert\n");
        printf("\t\t\t\t2.Delete\n");
        printf("\t\t\t\t3.Display list\n");
        printf("\t\t\t\t0.Exit\n");
        printf("\n\t\t\t\t..... choice:");
        scanf("%d",&choice);
        fflush(stdin);

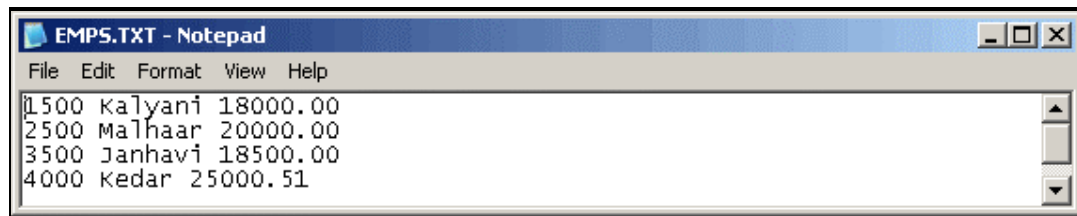
        switch(choice)
        {
            case 1:
                new=getnode();
                if(insert(new)== -1)
                    printf("error:cannot insert\n");
                else
                    printf("node inserted\n");
                break;
            case 2 :
                printf("Enter the employee id of the record to delete:");
                scanf("%d",&cd);
                fflush(stdin);
                if(delete(cd)==-1)
                    printf("deletion failed\n");
                else
                    printf("node deleted\n");
                break;
        }
    }
}
```

Example 9-2.1: Files & Linked List Contd...

```
        case 3 :
            displayall();
            break;
        case 0 :
            BackToFile();    /* save the list back to the file*/
            exit(0);
    }
}while(choice !=0);
}
```

Example 9-2.1: Files & Linked List Contd...

2. Save and Compile and build SOL9_2_1.exe.
3. Create a text file named EMPS.TXT in the folder where SOL9_2_1.exe is stored. Add a few entries in the file e.g. a snapshot of EMPS.TXT before executing the program may be as follows.

**Figure 9-1.1: EMPS.TXT - Snapshot 1**

4. Execute the program and perform some modifications to the linked list. A few snapshot of the execution are shown here.

```

C:\WINDOWS\System32\cmd.exe - tc
File Record inserted as node
File Record inserted as node
File Record inserted as node
File Record inserted as node

Menu
1.Insert
2.Delete
3.Display list
0.Exit

..... choice:3

Empid Name          Salary
1500  Kalyani        18000.00
2500  Malhaar         20000.00
3500  Janhavi          18500.00
4000  Kedar             25000.51

Menu
1.Insert
2.Delete
3.Display list
0.Exit

..... choice:1
Enter the employee id:3000
Enter the name:Anuradha
Enter salary:25000
node inserted

Menu
1.Insert
2.Delete
3.Display list
0.Exit

..... choice:2
Enter the employee id of the record to delete:1500
node deleted

Menu
1.Insert
2.Delete
3.Display list
0.Exit

..... choice:3

Empid Name          Salary
2500  Malhaar         20000.00
3000  Anuradha         25000.00
3500  Janhavi          18500.00
4000  Kedar             25000.51

Menu
1.Insert
2.Delete
3.Display list
0.Exit

..... choice:0_

```

Figure 9-2.2: Files & Linked List

5. The snapshot of EMPS.TXT after executing the program is

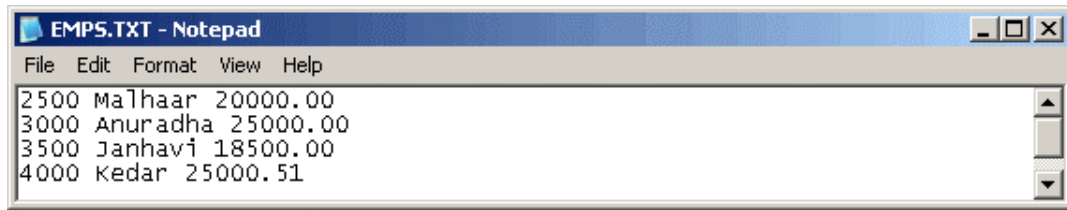


Figure 9-2.3: EMPS.TXT - Snapshot 2

- II. The telephone book entries (name, telephone number) are stored in a text file TBOOK.TXT. Write a program to read all the records from this file and store them in a linked list fashion. Do the necessary update in the linked list such as adding new records, deleting and modifying existing records etc. Use menu driven loop to perform these operations. Once done, save all the records back to the TBOOK.TXT file. [SOL9_2_2.C]

<<To do>>

III. Demonstrate unformatted file input/output using fread() and fwrite. [SOL9_2_3.C]

1. Open a new file and write following code in it.

```
/*The program receives records from the keyboard,
writes them to a file and reads them back from the file */
#include<stdio.h>
#include<conio.h>
void main(void)
{
    FILE *fp;
    char another='Y';
    struct employee
    {
        char name[20];
        int eid;
        float sal;
    } e;

    clrscr();

    fp=fopen("emp.dat","w"); /* open file for output*/
    if(fp==NULL)
    {
        puts("Cannot open file");
        exit(0);
    }
    while(another=="Y" || another == 'y')
    {
        /* Accepting the record form the user */
        fflush(stdin);
        printf("\n Enter name = ");
        scanf("%[^\n]",e.name);
        printf(" Enter employee id = ");
        scanf("%d",&e.eid);
        printf(" Enter basic salary = ");
        scanf("%f",&e.sal);

        fwrite(&e,sizeof(e),1,fp); /*Add an entire record to the file*/

        printf("Want to add another record (Y/N) : ");
        fflush(stdin);
        another=getchar();
    }
    fclose(fp);
}
```

Example 9-2.2 : Unformatted File I/O

```
fp=fopen("emp.dat","r"); /* open the file for input*/
if(fp==NULL)
{
    puts("Cannot open file");
    exit(0);
}

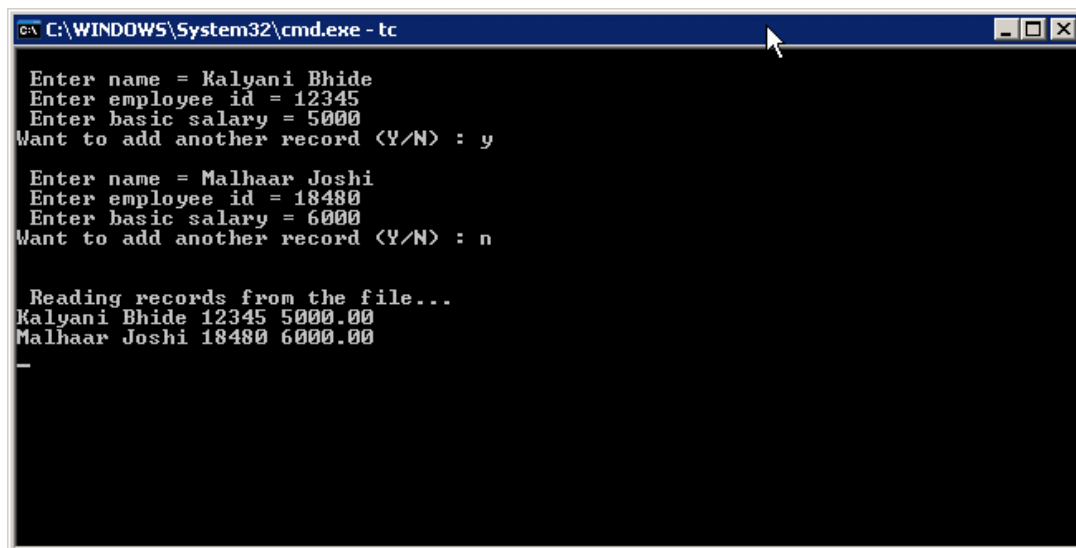
printf("\n\n Reading records from the file...\n");
while(fread(&e,sizeof(e),1,fp)==1)
    /* read an entire record from the file
    */
    printf("%s %d %.2f \n",e.name,e.eid,e.sal);

fclose(fp);
}
```

Example 9-2.2: Unformatted File I/O Contd...

Note: The functions `fread()` and `fwrite()` are generally used with binary files as they perform unformatted input/output. It is ideal for inserting structures/arrays containing numeric data.

2. Save, compile and build the executable.
3. A sample run is shown in the following snapshot.



```
C:\WINDOWS\System32\cmd.exe - tc

Enter name = Kalyani Bhide
Enter employee id = 12345
Enter basic salary = 5000
Want to add another record <Y/N> : y

Enter name = Malhaar Joshi
Enter employee id = 18480
Enter basic salary = 6000
Want to add another record <Y/N> : n

Reading records from the file...
Kalyani Bhide 12345 5000.00
Malhaar Joshi 18480 6000.00
-
```

Figure 9-2.4: Unformatted File I/O

- IV. Think of the possible modifications you will need in the code of `SOL9_2_2.c` if you were to use `fread()` and `fwrite()` instead of `fscanf()` and `fprintf()` library functions.

Appendix I - Debugging Tips

In the development stage, you need to debug your code a number of times. A step-by-step execution of the code helps you monitor the execution path that gets followed. Monitoring/watching the values of the variables helps you catch the bugs (logical errors) in the code.

1. To start executing the code in a step-by-step mode, choose Run -> Step over menu or press F8.

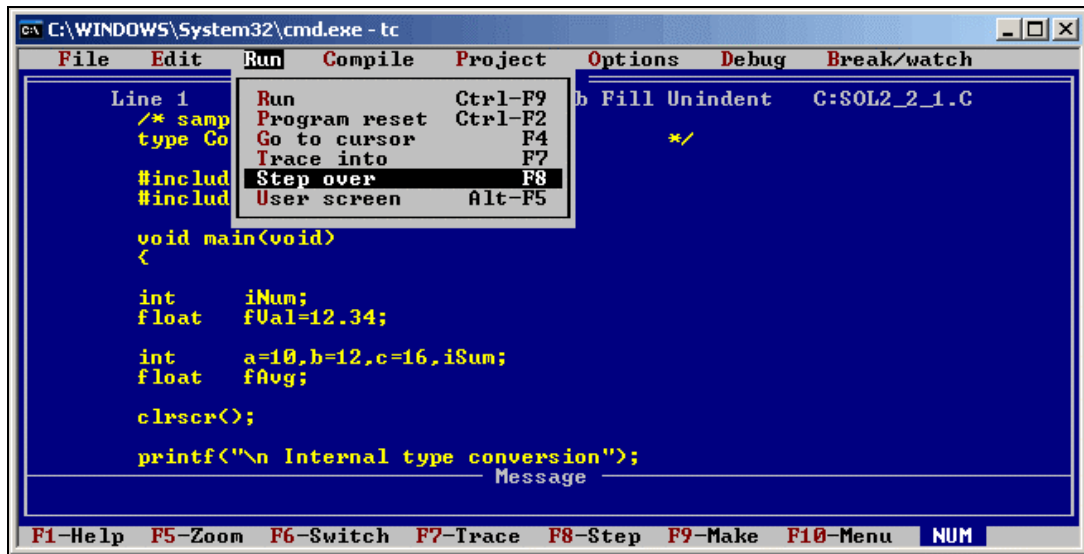


Figure App-I.1: Step Over

2. The execution will start from main (). You can continue the execution by pressing the key, F8 every time. The line of code, which is going to get executed next, is highlighted.

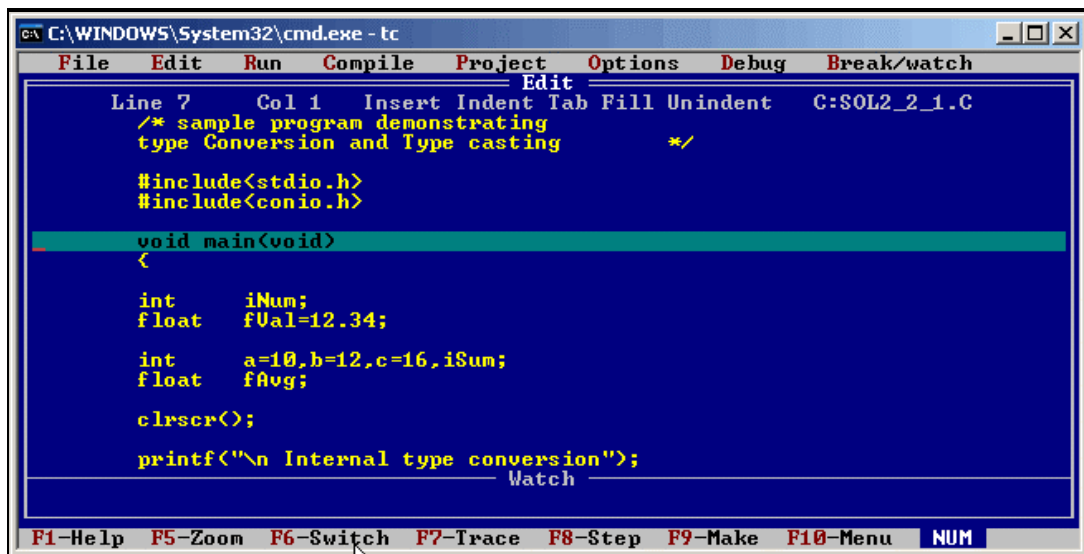


Figure App-I.2: Start Debugging

4. Whenever there is a function call, choose Run -> Trace into menu or press F7 to step into the function code. To skip the step-by-step execution of a function you may continue with F8.
5. To monitor/display the current value of a variable throughout the program execution you can add a watch on that variable. To add a watch, select Break/Watch -> Add Watch menu or press Ctrl-F7.

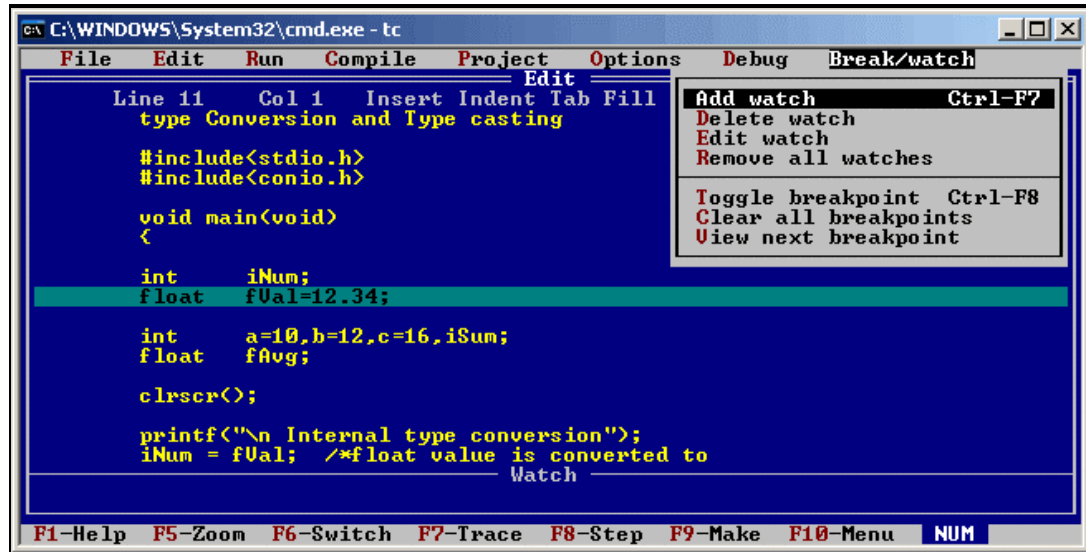


Figure App-I.3: Add Watch Menu

6. An 'Add Watch' window pops up. Type the variable name you want to watch and press Enter.

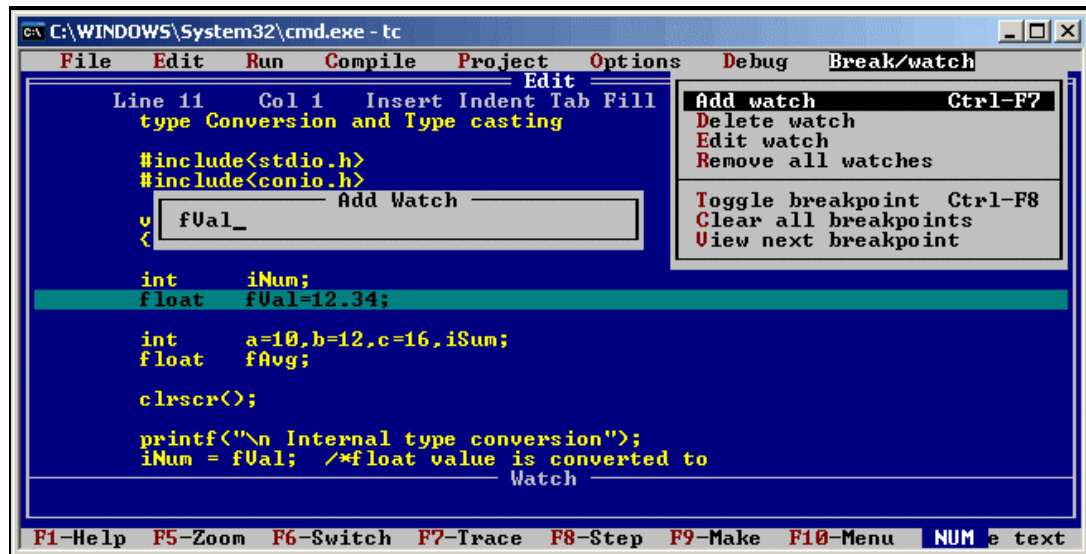


Figure App-I.4: Add Watch

7. The variable and its value are now displayed in the Watch window as shown below. You can add a watch on multiple variables in the same way.

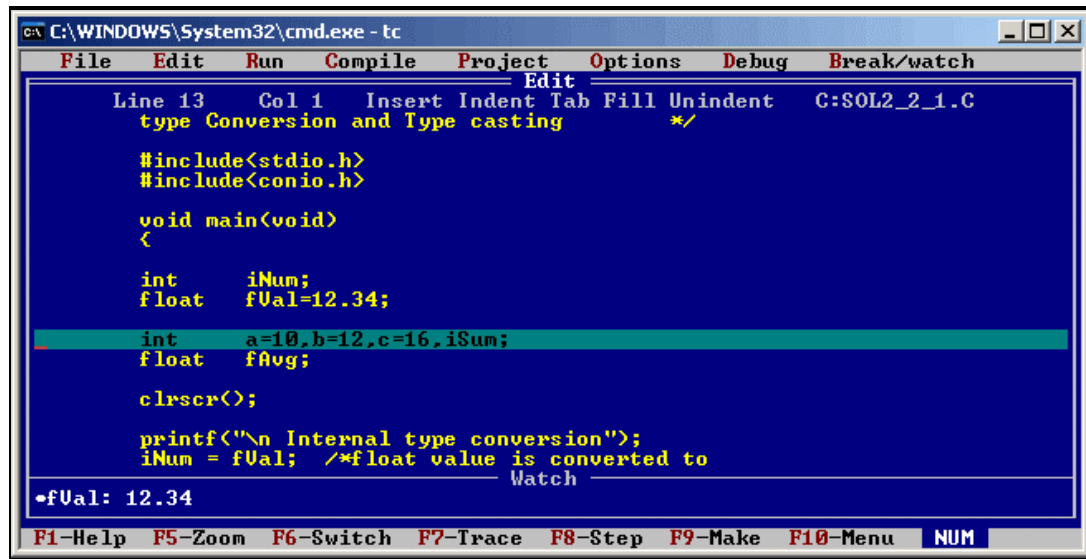


Figure App-I.5: Watch Window

8. At any point of time, if you want to skip the step-by-step execution, and continue with the normal execution, choose the Run -> Run menu or press Ctrl-F9.
9. You can insert a break -point at a particular statement in the code by selecting Break/Watch -> Toggle breakpoint menu option or by pressing Ctrl-F8 while your cursor is placed at that statement.

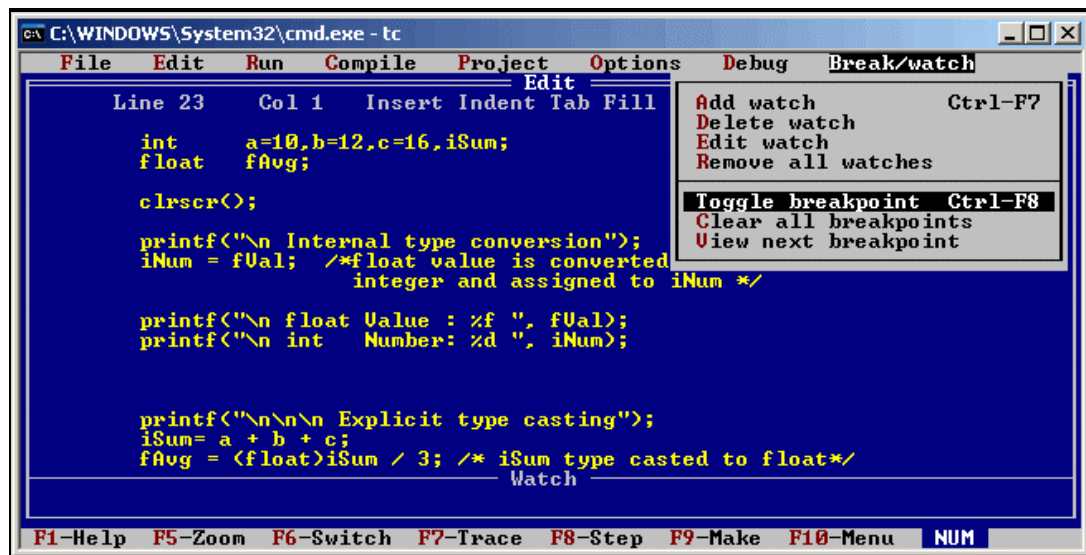


Figure App-I.6: Breakpoints

10. When you run the program, the execution will proceed normally till the first breakpoint is reached. As soon as the break point is encountered, the execution

halts at that point. You may continue with step-by-step mode of execution from that point onwards.

11. To remove the breakpoint, place the cursor at the statement having the break point and select Break/Watch -> Toggle breakpoint menu or press Ctrl-F8.
12. To remove all the break points from the code, select Break/Watch -> Clear all breakpoints menu option.
13. At any point of time, if the program execution hangs you can return to Edit window from the User screen by pressing Ctrl-Pause/Break keys simultaneously.

Appendix II - Coding Standards

The purpose of this document is to serve as a guiding standard for C programmers. This document will aid in achieving coding consistency and ANSI C conformance. It will also enhance the portability and maintainability of C source code. Any client specific standards are to override the standards given here in the event of contradiction between the two.

I. Naming conventions

The names of all variables, functions, etc., should be informative and meaningful. It is worth the time to devise and use informative names because good mnemonics lead to clear documentation, easy verification, maintenance and improved readability. The following rules should be followed to achieve this.

i. **File Names**

1. File name must be of the format : <basename>.<ext>
2. All the characters forming the basename and the extension must be in lowercase.
3. Basename length must not be longer than 10 characters out of which first 8 characters should be unique and extension must not be longer than 3 characters.
4. Basename should always start with an alphabet and not with a digit or an underscore. It should not end with an underscore.
5. As far as possible digits should be avoided. If used they should be at the end of the basename.
6. Name of the file should always be related to the purpose of that file.
7. File name should never conflict with any system file names.

ii. Source Code File Names :

1. Files belonging to a module should have a common prefix added to the basename, which would indicate the module to which the file belongs.
2. If a file contains only one function then the name of the file can be the same as the name of the function.

iii. Identifier Names

- General

1. The names should be no longer than 31 characters.
2. Avoid similar looking names. e.g. systst and sysstst
3. Names for similar but distinct entities will be distinct. e.g. 'goat' and 'tiger' instead of `animal_1` and `animal_2`
4. Consider limitations such as allowable length and special characters of compiler/linker on local machines.
5. Names will not have an underscore at their beginning or end.
6. Names should be related to the purpose of the identifiers.
7. Underscore '_' should be used to separate only significant words in names.
8. Avoid names that differ only in presence or absence of underscore '_'
9. Avoid names that differ only in case, eg. foo, Foo.
10. Avoid names conflicting with standard library names, variables and keywords.
11. Standardize use of abbreviations.

12. Avoid use of obscure abbreviations.
13. Do not abbreviate InputCharacter as inch.
14. Abbreviate common part of mnemonics
15. e.g. ``next_char'` is preferred over ``n_character'`

- **Local variables**

1. Local variables should be in lower case only.
2. Use of standard short names is allowed when the scope of the variable is limited to a few lines (about 22 lines e.g. following are a few frequently used short names

```
p ----- pointer
c ----- char
l, j --- index
```

- **Global Variables:**

1. Relate names of variables to the name of the module where defined.
2. Append `_g` to the end of the name. Eg. `int file_section_g;`

- **Constants (#defines and enums) :**

1. Use of constants must be strictly avoided as they convey little meaning about their use. Instead use symbolic constants.
2. There can be exceptions to this usage where 0 and 1 may appear as themselves e.g.

```
for (i = 0; i < ARRBOUND; i++)
```
3. All the symbolic constant names should be in capitals only. E.g.

```
#define ARRBOUND 25 /* comment */
```
4. *The enumeration data type should be used to declare variables that take discrete set of values. e.g.

```
enum { BLACK = 0, WHITE = 1 } colors;
```

- **Tags:**

1. This includes typedefs, structs, enums and unions.
2. Tags should be in lower case only with `_t` as a suffix. e.g.

```
typedef struct coord_t
{
    int x;
    int y;
}
```

- **Typedef names:**

1. Typedef names should be in lower case only.

- **Macros:**

1. Append ``_M'` to the macro names.
2. Macro name should be in all capitals. e.g.

```
#define MOD_M(a,b) \
< (b) ? ((b) - (a)) : ((a) - (b))
```

- **Functions:**

1. Relate the name of the function to the name of the module it belongs to. Convention for the abbreviation should be the same as that for the file names. e.g. function belonging to obj module will have Objxxxx.
2. Use capitalization to separate significant words. e.g. 'displayroutine' should be written as 'DispRoutine'

- **Pointers:**

1. Append an '_p' to pointer names. e.g.
char *name_p;

II. Programming style :

i. General Rules :

1. Line length must be less than or equal to 80 characters.
2. File length should be less than or equal to 1000 lines.
3. Each function within a file should be limited to 200 lines.
4. Storage class, type and variables must be vertically aligned. Each item should start from the same column.
5. For union/enum/struct adopt following rules :

```
[ typedef ] enum/struct/union [ tag ]
{
...
...
}
```
6. Function prototypes and definitions should be as per the following style.

```
< return type >
< function name > (
int count; /* the counter */
...      /* ...      */
...)     /* ...      */
```
7. Indentation should be in multiples of 4 spaces. One shouldn't use tabs for indentation.
8. Any start and end brace should be alone on separate lines, and should be vertically aligned with the control keyword. e.g.

```
if (count != 0)
{
.....
}
```
9. If a group of functions have alike parameters or local variables, then they should have same names. e.g.

Use

```
int
func1(
    int file_err)
```

```
int
func2(
    int file_err)
```

Instead of

```
int
func1(
    int file_problem)

int
func2(
    int file_invalid)
```

10. Avoid hiding identifiers across blocks. e.g.

```
/* Bad practice */

int a;

.....

.....

{
float a; /* This hides 'a' above */

.....
.....
}
```

11. For pointer definitions attach "" to variable names and not to the types. e.g.
Use

```
char *char_p;

instead of

char* char_p;
```

12. Put unrelated variables on the separate lines. e.g.

Use

```
int i, j, k;
int count;
int index;
int flag;
```

Instead of

```
int i, j, k;
int count, index, flag;
```

13. All the logical blocks should be separated by appropriate blank lines. There should be three blank lines between the definitions of any two functions and at least one blank line between other major blocks.
14. Static, automatic and register variable declarations should be demarcated by blank lines.
15. Variables declared with the register storage class must be declared in decreasing order of importance to ensure that the compiler allocates a register to the most important variables.
16. Avoid simple blocks i.e. without any control statements.

17. Do not depend upon any implicit types. E.g.

Use

```
static int a;
```

Instead of

```
static a;
```

18. Null body of the control statement must always be on a separate line and explicitly commented as `/* Do nothing */`.

Use

```
while (...)
{
    ;           /* Do nothing */
}
```

instead of

```
while (...) ;
```

19. One line can have at the most one statement except when the multiple statements are closely related.
20. Return value of functions must be used.
21. An 'if' statement with 'else-if' clauses should be written with the else conditions left justified. The format then looks like a generalized switch statement. e.g.

```
if (count == 0)
{
    .....
}
else if (count < 0)
{
    .....
}
else if (count > 0)
{
    .....
}
```

22. The body of every control statement must be a compound statement (enclosed in braces even if there is only one statement.)
23. Fall through in "case" statement must be commented.
24. "default" must be present at the end of "case" statement.

25. Condition testing must be done on logical expressions only. eg. With 'count' having the declaration:

```

int count;

Use
    if (count == 0)
        /* THIS REFLECTS THE NUMERIC (NOT BOOLEAN) NATURE OF THE
        TEST */
    {
        ...
    }

```

Instead of

```

    if (!count)
    {
        ...
    }

```

26. "goto"s could be used only under error/exception conditions and that too only in forward direction. However, it is to be generally avoided.
27. Labels should be used only with gotos. Labels should be placed at the first indentation.
28. If some piece of code is deleted or commented while modifying the file, the identifiers, which become unused must be deleted or commented respectively.
29. Check the side effects of ++ and --.
30. Use "," as a statement separator only when necessary.
31. Use ternary operator only in simple cases.
32. Repeat array size in the array declarations if the array was defined with size. E.g.

```

main()
{
    int arr[10];

    .....
    func(arr);
}

int
func(
    int passed_arr[10])
{
    .....
}

```

33. Initialize all the global variable definitions.
34. Do not assume the value of uninitialized variables.
35. Functions and variables used only in a particular file should be declared as static in that file.
36. Prototype a function before it is used.
37. Declare a Boolean type "bool" in a global include file. e.g.

```

typedef int  bool;
#define FALSE 0
#define TRUE 1

```

Instead of checking equality with TRUE, check inequality with FALSE. This is because FALSE is always guaranteed to have a unique value, zero but this is not the case with TRUE. e.g.

Use

```
if (f() != FALSE)
```

instead of

```
if (f() == TRUE)
```

38. No tab characters should be saved in the file.

39. Data declarations should be written vertically aligned w.r.t. data types and identifier names both. e.g.

```
int          count;
struct item_t item;
char         *buffer;
```

ii. **White Spaces:**

1. Add space before and after all the binary operators except "." and "->" for structure members.
2. Unary operators should not be separated from their single operand.
3. Horizontal and vertical spacing is equally important.
4. After each keyword one space is required.
5. Long statements should be broken on different lines, splitting out logically separate parts and continue on the next line with an indentation of 4 spaces from the first line. e.g.

Use

```
if (next_p == (node_p) NULL
    && count < now && now <= MAXALLOT)
{
    .....
}
```

instead of

```
if (next_p == (node_p) NULL && count < now
    && now <= MAXALLOT)
{
    .....
}
```

Appendix III - Table of Figures

Figure 1-1.1: Turbo C Installation Utility	3
Figure 1-1.2: Set Drive	4
Figure 1-1.3: Set Source Path	4
Figure 1-1.4: Install Turbo C	5
Figure 1-1.5: Start Installation	5
Figure 1-1.6: Installation Completed	6
Figure 1-1.7: Turbo C Environment	6
Figure 1-1.8: Hello World Program	7
Figure 1-1.9: Set Directories	7
Figure 1-1.10: Compile to OBJ	8
Figure 1-1.11: Make EXE File	8
Figure 1-1.12: Run	9
Figure 1-1.13: User Screen	9
Figure 1-1.14: Hello World Program	10
Figure 1-2.1: Formatted Output Program	11
Figure 1-2.2: Formatted Output Program Compiled	12
Figure 1-2.3: Formatted Output Program Linked	12
Figure 1-2.4: Run Formatted Output Program	13
Figure 1-2.5: Formatted Output	13
Figure 2-1.1: Bitwise Shift Operators	16
Figure 2-2.2: Type Conversion & Type Casting	18
Figure 3-1.1: Switch Statement - Output 1	20
Figure 3-1.2: Switch Statement - Output 2	20
Figure 3-3.1: For Loop & Continue Statement	25
Figure 4-1.1: Functions - Compound Interest	28
Figure 4-1.2: Call by Value	30
Figure 4-2.1: Static Variables	32
Figure 4-2.2: Recursion	34
Figure 5-1.1: Single Dimensional Arrays	36
Figure 5-2.1: Multidimensional Arrays	38
Figure 5-3.1: Character Arrays	41
Figure 5-4.1: Library Functions for String Handling	43
Figure 5-5.1: Two Dimensional Array of Characters	45
Figure 6-1.1: Pointers	48
Figure 6-2.1: Call by Reference	50
Figure 6-3.1: Arrays & Pointers - Output 1	52
Figure 6-3.2: Arrays & Pointers - Output 2	52
Figure 6-4.1: Pointers to Pointers	55
Figure 6-4.2: Two Dimensional Arrays & Pointers	56
Figure 6-5.1: Command Line Arguments	59
Figure 7-1.1: Structures	61
Figure 7-2.1: Structures & Pointers	63
Figure 8-1.1: Linked List	69
Figure 8-1.1: Linked List Contd...	70
Figure 9-1.1: EMPS.TXT - Snapshot 1	78
Figure 9-2.2: Files & Linked List	79
Figure 9-2.3: EMPS.TXT - Snapshot 2	80
Figure 9-2.4: Unformatted File I/O	82
Figure App-I.1: Step Over	84
Figure App-I.2: Start Debugging	84
Figure App-I.3: Add Watch Menu	85
Figure App-I.4: Add Watch	85
Figure App-I.5: Watch Window	86
Figure App-I.6: Breakpoints	86

Appendix IV - Index of Examples

Example 2-1.1: Bitwise Shift Operators	15
Example 2-2.1: Type Conversion & Type Casting.....	17
Example 3-1.1: Switch Statement	19
Example 3-2.1: While Loop	22
Example 3-3.1: For Loop & Continue	24
Example 4-1.1: Functions - Compound Interest	27
Example 4-1.2: Call by Value	29
Example 4-2.1: Static Variables	31
Example 4-3.1: Recursion.....	33
Example 5-1.1: Single Dimensional Arrays.....	35
Example 5-2.1: Multidimensional Arrays.....	37
Example 5-3.1: Character Arrays.....	40
Example 5-3.1: Character Arrays Contd... ..	41
Example 5-4.1: Library Functions for String Handling.....	42
Example 5-5.1: Two Dimensional Array of Characters.....	44
Example 6-1.1: Pointers.....	47
Example 6-2.1: Call by Reference	49
Example 6-3.1: Arrays & Pointers	51
Example 6-4.1: Pointers to Pointers	54
Example 6-4.2: Two Dimensional Arrays & Pointers	55
Example 6-5.1: Command Line Arguments	57
Example 7-1.1: Structures	60
Example 7-2.1: Structures & Pointers	62
Example 8-1.1: Linked List	65
Example 9-1.1: File Handling	72
Example 9-2.1: Files & Linked List	74
Example 9-2.2 : Unformatted File I/O	81