IGATE

Speed.Agility.Imagination

C and Data Structure

Lab Book

Document Revision History

Date	Revision No.	Author	Summary of Changes
05-Apr-2009	0.1D	Kishori Khadilkar	Content Creation
01-Jul-2009		CLS Team	Review
25-May-2011	0.2D	Rathnajothi Perumalsamy	Revamp/Refinement
28-Sept-2012		Vaishali Kunchur	Revamped for additional assignments
17-Mar-015	0.3D	Vinod P Manoharan	Revamped for additional assignments and fixing existing errors.

Table of Contents

Getting Started

Overview

This lab book is a guided tour for learning C Programming. It comprises ‘To Do’ assignments. Work out the ‘To Do’ assignments that are given.

Setup Checklist for Data structure

Here is what is expected on your machine for the lab to work.

Minimum System Requirements

- Intel Pentium 90 or higher (P166 recommended)
- Microsoft Windows 95, 98, or NT 4.0, 2k, XP.
- Memory: 32MB of RAM (64MB or more recommended)

Please ensure that the following is done:

- Visual studio 2005 is installed on your machine.

Instructions

- Create a directory by your name in drive <drive>. In this directory, create a subdirectory C_assgn. For each lab exercise create a directory as lab <lab number>.
- If under the Windows OS, .prj is not created please create all the functions and the main function under a single .c file.

Learning More (Bibliography if applicable)

- “Pointers in C” by Kanetkar Yashawant
- “Test Your C” by Kanetkar Yashawant
- “C : The Complete Reference” by Schildt Herbert
- “Let Us C” by Kanetkar Yashawant
- “Programming in ANSI C” by Balagurusamy

Lab 1. Introduction to C operators

Goals	Write and execute simple C programs using various operators.
Time	120 Minutes

1. Write a function that inverts the bits of an unsigned char x and stores answer in y.

Your answer should print out the result in binary form although input can be in decimal form.

Your output should be like this:

```
x = 10101010 (binary)
x inverted = 01010101 (binary)
```

2. Write a program to perform the following task:
   a. Input two numbers and work out their sum, average and sum of the squares of the numbers.
3. Write a program that works out the largest and smallest values from a set of 10 inputted numbers.
4. Write a program to read a "float" representing a number of degrees Celsius, and print as a "float" the equivalent temperature in degrees Fahrenheit. Print your results in a form such as
   b. 100.0 degrees Celsius converts to 212.0 degrees Fahrenheit.
5. Write a program to read a positive integer at least equal to 3, and print out all possible permutations of three positive integers less or equal to than this value.
6. Write a program to read a number of units of length (a float) and print out the area of a circle of that radius. Assume that the value of pi is 3.14159
   a. Your output should take the form: The area of a circle of radius ... units is.... units.
   b. It should print an error message “Error: Negative values not permitted” if the input value is negative.
7. Given as input a floating (real) number of centimeters, print out the equivalent number of feet (integer) and inches (floating, 1 decimal), with the inches given to an accuracy of one decimal place.

   a. Assume 2.54 centimeters per inch, and 12 inches per foot.
   b. If the input value is 333.3, the output format should be:
   c. 333.3 centimeters is 10 feet 11.2 inches.

8. Given as input an integer number of seconds, print as output the equivalent time in hours, minutes and seconds. Recommended output format is something like 7322 seconds is equivalent to 2 hours 2 minutes 2 seconds.

Lab 2. Type Casting and Loop Constructs

Goals	Write and execute simple C programs to understand type conversion and type casting
Time	30 Minutes

1. 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.

2. Write a program to accept conversion choice from the user:
   a) centimeter to inch
   b) inch to centimeter
   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. Similarly work out for b) option.

   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.

Goals	Write and execute C programs to understand iterative constructs through while and do…while loops.
Time	60 Minutes

1 Modify the previous program 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.

2 Generate all permutation as

Example : (n = 3)

```
1   2   3
1   3   2
2   1   3
2   3   1
3   1   2
3   2   1
```

Try to generate for any n.

3 Print a palindrome as shown below:

```
MALAYALAM
 ALAYALA
  LAYAL
   AYA
    Y
```

Goals	Write and execute C programs to understand iterative constructs through for loops.
Time	90 Minutes

1 Write a program to generate the following output

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

2 Write a program to print prime numbers from 1 to 1000

3 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.

4 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.

5 Write a program to find hcf and lcm: The code below finds highest common factor and least common multiple of two integers. HCF is also known as greatest common divisor(GCD) or greatest common factor(gcf).

Lab 3. Functions

Goals	Write and execute C programs to perform string operations using standard library functions.
Time	120 min

1. Write an interactive C program that will encode or decode a line of text. To encode a line of text, proceed as follows:
   a. Convert each character including blank spaces to its ASCII equivalent.
   b. 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.
   c. 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 ACSII character.
   d. Print the characters that correspond to encoded ACSII values.
   e. Reverse the entire procedure to decode the line of text and print the original line of text again.

2. Write a program in C which performs the following operations on a string obtained. The following functions can be made for strings:
   a. Reverse string
   b. Check whether a string is palindrome or not
   c. Convert to lower case
   d. Convert to upper case
   e. Convert to title case
   f. Convert to sentence case
   g. Convert to toggle case
   h. Count vowels
   i. Count consonants
   j. Count digits
   k. Count words
   l. Extract n characters from the left
   m. Extract n characters from the right
   n. Extract n characters from m position
   o. Delete all spaces from the string
   p. Count the occurrence of a particular character

3. write a program to accept variable lenth of arguments , and display the sum of it all.

4. write a program to accept the list of names from the key board and pass it to a function with the help of variable lenth of arguments and display it all.

Lab 4. Arrays

Goals	Write and execute C programs to work with single dimensional arrays.
Time	90 Minutes

1. Write a program to store 11, 22, 33, 44, 55, 66 in an array.
    a. Find the sum of all the integers
    b. Insert a number at a specified position.
    c. Delete a number from a specified position.Write a program to display the minimum and maximum number from an array of 10 integers along with their position in the array.
2. Write a program to evaluate the expression $z=x^2+y^2$, where x and y are 2 are 2 arrays. Each array holds 10 user entered elements. Store the result in another array and display it.
3. 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.
4. Accept a 2D Matrix and store only its nonzero value in the manner describe below :-

```
   original Matrix          Converted matrix

  1  0  0  4               3   4   4 ---->1st row(see note)
  0  0  0  2      ==>       0   0   1
  0  6  0  0                0   3   4
                          1   3   2
                          2   1   6
```

Note :-
1st row: It contains dimension of first matrix (m, n) and No. of nonzero elements.

Subsequent rows contain (row, col) of nonzero element & its value.

5. Write a program to merge two arrays into third array: Arrays are assumed to be sorted in ascending order. You enter two short sorted arrays and combine them to get a large array.

Goals	Write and execute C programs to understand character arrays and strings
Time	30 Minutes

1. Make a file which contains the functions to perform the following tasks and test it by including the file in another file which contains main()

    a. Accept an array
    b. Display an array
    c. Sort array
    d. Reverse array
    e. Count odd
    f. Count even
    g. Count prime
    h. Maximum value in the array
    i. Minimum value in the array
    j. Search a value in the array

Lab 5. Pointers

Goals	Write and execute C programs to work with Pointers.
Time	150 Minutes

1. Write a program to accept list of values [Numeric] from user and store in it, the size of the array will be defined by the user.
2. Write a program to accept list of values [Alphabet] from user and store it in a dynamic array.
3. Write a program to accept list of strings / names from the user via keyboard and store it in a dynamic multi-dimensional array.
4. Create a n X m matrix numerical array with the help of "n" and "m" given by the user via keyboard. And make sure both "n" and "m" are not equal. Accept values from the user and displays it.
5. Create a n X m matrix string array with the help of "n" and "m" given by the user via keyboard. And make sure both "n" and "m" are not equal. Accept values from the user and display it.

   Example : The Array should looks like for an input of 3 rows and 5 columns

   ```
   Rahul    rabino   vinod   raaju    reena
   Chandhu  kiran    kraunaa  kareena  meeru
   Mittu    mibin    madhu   chandrasekaran  ameerIslahi
   ```

   Each strings / name should be a variable length array.
6. Create a function which accepts the array of values via parameter, and display it.
7. Create a function which returns an array of values to the calling function.
8. Create a global user defined dynamic array, via one function defines it's size and create the allocation. Via another function accepts the input from the user and stores in it. Via the third function displays the values of the array with the help of return by reference method. Main() function should simply call all these functions to execte the memory allocation, accepting values from the user, storing in the memory, and displaying it in the screen. Make sure all the list of following should be part of the program.
   a. Static variable
   b. Global variable
   c. Call by value
   d. Call by address / pass by references
   e. Return by value
   f. Return by reference / address
   g. Dynamic array creation

Lab 6. Structures

Goals	Write and execute C programs to work with structures.
Time	60 Minutes

1. 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.

2. 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.

3. Modify the program you wrote in exercise 2. 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.

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.

Goals	Write and execute C programs to work with structures.
Time	30 Minutes

1. Write a program that read a number from command line input and generates a random floating point number in the range 0 - the input number.

2. Write a program that reads a number that says how many integer numbers are to be stored in an array, creates an array to fit the exact size of the data and then reads in that many numbers into the array.

3. Write a program to dynamically allocate an array and find the SUM of all the elements in the array,

Lab 7. File handling

Goals	Write and execute C programs to do File handling
Time	1.30 hrs

1. Write a program to read a file which consists of only numbers. If the read number is an even number it has to write in a separate file. If the number read is an odd number then it has to write in a different file.

2. Create an utility which takes a file name as input from the user at command line and counts the no. of characters, words and lines and displays them.

3. Write a program which takes two file names from the command line and compares them character by character and displays if the contents are same. If no then the position of the matched character and the mismatched character from both the files should be displayed. If one file reaches end of file appropriate message should be displayed while the mismatched character should be displayed from the other file.

4. Enhance the Employee program which is referred at Lab 6.1 and add code for the following tasks:
    a. Search Employee
    b. Modify Employee
    c. Delete Employee

Lab 8. Preprocessor

Goals	Write and execute simple C programs using Macros.
Time	90 Minutes

1. How would you arrange that the identifier MAXLEN is replaced by the value 100 throughout a program?
2. Define a preprocessor macro swap(t, x, y) that will swap two arguments x and y of a given type t.
3. Write a program to illustrate the difference between functions with Macros.
4. Write a program with the help of conditional macro to control the execution of the code block
5. Write a program with the help of #ifdefine and #undefine macros.

Lab 9. Sorting and Searching

Goals	Understand different sorting and searching algorithm.
Time	180 minutes

1.1: Bubble Sort

Step 1: Create a new C file named BubbleSort.c, and write the following code in it.

```
//include the required additional header files
#include <stdio.h>
//bubble sort function
void bubbleSort(int numbers[], int array_size)
{
   int i,j;
   int temp;
   //define bubble sort algorithm in this function
       for(i=0;i<array_size-1;i++)
      for(j=i+1;j<array_size;j++)
         if(numbers[i]>numbers[j]){
            temp=numbers[i];
            numbers[i]=numbers[j];
            numbers[j]=temp;}
       //first parameter is the array to be sorted
       //second parameter is the size of the input array
 }
```

Example 1: Bubble Sort

Step 2: Create header file bubblesort.h to include prototype of function.

```
#define NUM_ITEMS 5
void bubbleSort(int numbers[], int array_size);
```

Example 2: Bubble sort

Step 3: Create a new C source file named testBubbleSort.c, and write the following code.

```
#include<stdio.h>
#include
#include “bubbleSort.h”

int main()
{
 int numbers[NUM_ITEMS],i;
 //seed random number generator
 srand(getpid());
 for (i = 0; i < NUM_ITEMS; i++)
 {
    //generate elements of array randomly
     numbers[i] = rand();
     printf("%d\n",numbers[i]);
 }
//perform bubble sort on array
 bubbleSort(numbers, NUM_ITEMS);
 printf("\nDone with sort.\n");
 for (i = 0; i < NUM_ITEMS; i++)
     printf("%i\n", numbers[i]);
 return 0;
}
```

Example 3: Bubble Sort

1.2: Insertion Sort

Step 1: Create a new C file named InsertionSort.c, and write the following code in it.

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

void insertionSort(int numbers[], int array_size)
{
//first parameter is the array to be sorted
//second parameter is the size of the input array
<<to do>>
}
```

Example 4: Insertion sort

Step 2: Create Insertionsort.h file to add prototype of the following function:

```
#define NUM_ITEMS 5
void insertionSort(int numbers[], int array_size)
```

Example 5: Insertion sort

Step 3: Create a new C source file named testInsertionSort.c, and write the following code in it:

```
#include “InsertionSort.h”

int main()
{
     int numbers[NUM_ITEMS];
           int i;
           srand(getpid());
     /*srand seeds the random number generation
function rand so it does not produce the same sequence of numbers
The getpid() function returns the process ID of the calling process. */
 for (i = 0; i < NUM_ITEMS; i++)
 {
  //generate elements of array randomly
   numbers[i] = rand();
    printf("%d\n",numbers[i]);
 }
 insertionSort(numbers, NUM_ITEMS);
 printf("\nDone with sort.\n");

 for (i = 0; i < NUM_ITEMS; i++)
     printf("%i\n", numbers[i]);
 return 0;
}
```

Example 6: Insertion sort

1.3: Quick Sort

Step 1: Create a new C file named QuickSort.c, and write the following code in it. Also create QuickSort.h to add prototype of functions.

```
//include the required additional header files
#include <stdio.h>
//helper function quickSort
void quickSort(int numbers[], int array_size)
{
            q_sort(numbers, 0, array_size - 1);
}

//quick sort function
void q_sort(int numbers[], int left, int right)
{
        //define variables to hold left, right and pivot values
        int pivot, l_hold, r_hold;
        //define quick sort algorithm in this function
        //first parameter is the array to be sorted
        //second parameter is the index of the element on the left side
        //third parameter is the index of the element on the right side
}
```

Example 7: Quick sort

Step 2: Create a new C source file named testQuickSort.c, and write the following code:

```
// include the required header files
//define array size and array
#define NUM_ITEMS 100
int main()
{
          int i;
      int numbers[NUM_ITEMS];
          //seed random number generator
          srand(getpid());
 //fill array with random integers
    <<to do>>
          //perform quick sort on array
          quickSort(numbers, NUM_ITEMS);
         //print the sorted array
 <<to do>>
          return 0;}
```

Example 8: Quick sort

1.4: Extra Assignments

Step 1: Write a C program that sorts the elements of a two dimensional array in the following manner:

- row wise
- column wise

1.5: Sequential Search

Step 1: Create a new C header file named SequenceSearch.c, and write the following code in it. Also create header file by name SequenceSearch.h to add prototype of function.

```
//include the required additional header files
#include <stdio.h>
//Search function
int Search(int numbers[], int array_size,int element)
{
         //define sequential search algorithm in this function
         //first parameter is the array to be sorted
         //second parameter is the size of the input array
     // Third parameter is element to be searched
   <<to do>>
   //return position of the number  if found -1 otherwise
}
```

Example 9: Sequential Search

Step 2: Create a new C source file named testSequenceSearch.c, and write the following code:

```
#include<stdio.h>
#include " SequenceSearch.h"
//include required header files

#define NUM_ITEMS 5
int main()
{
 int pos,num;
             //seed random number generator
 srand(getpid());
 int numbers[NUM_ITEMS];
 //fill numbers array with random integers
 <<to do>>
 //Accept element to be searched
 //perform Sequecntial search on array
  pos=search(numbers, NUM_ITEMS,num);
 // Display appropriate message
 return 0;
}
```

Example 10: Sequential search

1.6: Binary Search

Step 1: Create a new C file named BinarySearch.c, and write the following code in it. Also create BinarySearch.h file to add prototypes of function.

```
//include the required additional header files
#include <stdio.h>
//Search function
int Search(int numbers[], int array_size, int element)
{
             //define binary search algorithm in this function
             //first parameter is the array to be sorted
             //second parameter is the size of the input array
       // Third parameter is element to be searched
    <<to do>>
    //return position of the number  if found, -1 otherwise
}
```

Example 11: Binary Search

Step 2: Create a new C source file named testBinarySearch.c, and write the following code:

```
#define NUM_ITEMS 5
#include<stdio.h>
//include required header files
int main()

```

```
 { int pos,num;
```

```
int numbers[NUM_ITEMS];
//Accept array elements in the sorted order
 <<to do>>
    //Accept element to be searched
   //perform Binary search on array
   pos=search(numbers, NUM_ITEMS,num);
 // Display appropriate message
 return 0;
}
```

Example 12: Binary Search

Lab 10. Linked List

Goals	Understand linked list and different operations performed on the list
Time	120 minutes

1.1: Employee Linked List

Step 1: 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. 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.

Step 2: Create a new file named employee.c, 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 13: Linked List - getnode

```
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;
}
```

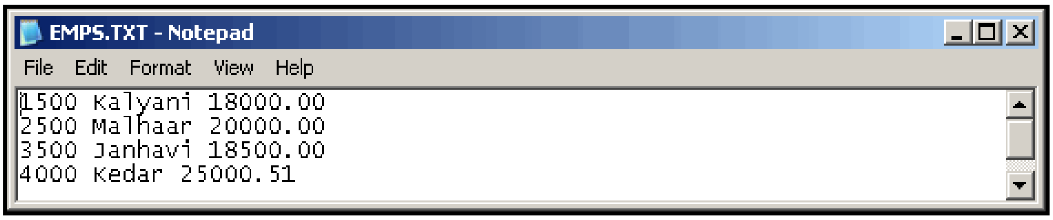
```
void displayall()
{
        struct emp *cur;
        if(list==NULL)
        {
                printf("The list is empty\n");
                return;
        }
        printf("\n\nEmpid Name                    Salary\n");
        for(cur=list;cur;cur=cur->next)
          printf("%4d  %-22s %8.2f\n",cur->empid,cur->name,cur->sal);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

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 %.2f\n", cur->empid, cur->name, cur->sal);
             cur = cur -> next ;
            }
```

```
    }
    fclose(FP);
}

void main(void)
{
    struct emp *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;

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

Example 14: Linked List

Step 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:



```
1500 Kalyani 18000.00
2500 Malhaar 20000.00
3500 Janhavi 18500.00
4000 Kedar 25000.51
```

Figure 1: Notepad

1.2: List of characters from string

Step 1: Write a C program to get a String from the user, and convert it to a linked list of characters and have options to convert the list back to String.

Lab 11. Stacks and Queues

Goals	Understand use of stacks and queues.
ime	90 minutes

1.1 Stacks using LinkedList

Step 1: Create a new C header file named Stacks.h, and write the following code in it.

```
struct stack
{
     int data;
     struct stack *next;
}*top;

void InitStack();

void Push();

int Pop();

int IsEmpty();
```

Example 15: Stack structure

Step 2: Create a new C source file named Stacks.c, and write the following code:

```
#include <stdio.h>
#include <stdlib.h>  /* for dynamic allocation */

typedef struct stack * stack_ptr;
#define NODEALLOC  (struct stack *) malloc (sizeof(struct stack))
//Initializing stack
 void InitStack()
{   top = NULL;  }

//Isempty Function
 int IsEmpty()
{ return (top == NULL);}

//push function
void Push (int num)
{
   stack_ptr newnode;
```

```
  newnode=NODEALLOC;
  newnode->next=NULL;
  newnode->data=num;
  if(top==NULL)
      top=newnode;
  else
   {
      newnode->next=top;
      top=newnode;
   }
 }

//pop function
int Pop()
{
   int num
   stack_ptr temp=top;
   num= top->data;
   top=top->next;
   free(temp);
   return(num);
}
```

Example 16: Stack operations

Step 3: Create a new C source file named Stacktest.c, and write the following code:

```
#include <stdio.h>
#include <string.h>  /* for strlen() */
#include "stacks.h"

main()
{
        int n,choice;
        initstack()   //initializing stack top
        do
        {
                printf(“\n1. Push \n 2. Pop \n  3. Exit\n”);
                printf(“\n Enter your choice”);
                scanf(“%d”,&choice);
                switch(choice)
                {
                        case 1:   //Push
                         printf(“Enter the element to be pushed :  ”);
                         scanf(“%d”,&n);
                         Push(n);
                         break;
                        case 2:  //pop
                         if(IsEmpty())
                                printf("\nThe stack is empty  \n");
                          else
                                printf("the poped element :  %d", Pop())
                        break;
                }
        }while(choice!=3)
        getch();
}//end of main
```

Example 17: Stack operations

1.2 Queue using LinkedList

Step 1: Write a C program queue.c to implement Queue.

```
#include<stdio.h.>

struct queue
{
    int data;
    struct queue *next;
}*front,*rear;

Typedef struct queue QUEUE;
//To initialize queue
void initqueue()
{
   front=rear=NULL;
 }

//to check for empty queue
int emptyqueue()
{
   return(front==NULL);;
}

//To remove element from queue
int remove()
{
     int num;
     QUEUE *temp=front;
     num=front->data;
     front=front->next ;
     free(temp);
     if(front==NULL)
                              rear=NULL;
     return(num);
}
```

```
//to add element in queue

void insert(int num)
{
    QUEUE *temp;
    temp=(QUEUE *) malloc(sizeof(QUEUE));
    temp->data=num;
    temp->next=NULL;
    if(front==NULL)
        rear=front=temp;
    else
    {
           rear->next=temp;
           rear=temp;
    }
}

main()
{
    int choice,num,n;
    initqueue();
    do
    {
       printf(“ \n\n 1: Add   \n 2:  Delete  \n 3: Exit\n”);
       printf(“Enter your choice  : “);
       scanf(‘%d”,&choice);
       switch(choice)
       {
         case 1:    //Add number in the queue
           printf(“Enter Element to be added in the queue”);
           scanf(“%d”,&num);
           insert(num);
           break;
         case 2:    //Remove data from the queue
           if(emptyqueue())
                                 printf(‘\n\n Queue is empty”);
           else
                                 printf(“\n\n The deleted element is %d “, dletequeue());
                                 break;
         case 3:
           exit();
       }// end of switch
    }while(choice!= 3)
}//end of main
```

Example 18: Queue

Assignment 1: Job Queue <<To Do>>

Step 1: Suppose there are several jobs to be performed with each job having priority value of 1, 2, 3, 4, etc. Write a program that receives the job descriptions and the priorities. Create as many queues as the number of priorities, and queue up the jobs into appropriate queues.

For example:

- Suppose the priorities are 1, 2, 3 and 4, and the data to be entered is as follows:
  - ABC,2,XYZ,1,PQR,1,RTZ,3,CBZ,2,QQQ,3,XXX,4,RRR,1
- Then arrange these jobs as shown below:
  - Q1: XYZ,1,PQR,1,RRR,1
  - Q2: ABC,2,CBZ,2
  - Q3: RTZ,3,QQQ,3
  - Q4: XXX,4
- The order of processing should be: Q1, Q2, Q3, Q4
- Write a program to simulate the above problem.

1.3: Stretched Assignments

Assignment 1: Write a program to make a list of people entering in the hall. The person who enters first in the hall will come out last. Maintain a list, and display the list in an order in which people should come out of the hall.

Lab 12. Trees

Goals	Understand tree data structure and operations on tree
Time	180 minutes

1.1 Binary Tree Traversal

Solution: Create a program to construct a Binary Tree and perform the Inorder, Postorder, and Preorder Traversal

Step 1: Create a C program called TreeTraversal.h, and include the following code into it:

```
#include <stdio.h>

struct btreenode
{
            btreenode *leftchild ;
            int data ;
            btreenode *rightchild ;
} *root ;

void InitTree( struct btreenode *root) ;
void buildtree (btreenode **root,int num ) ;
static void insert ( btreenode **sr, int num ) ;
void traverse( ) ;
static void inorder ( btreenode *sr  ) ;
static void preorder ( btreenode *sr ) ;
static void postorder ( btreenode *sr ) ;
static void del ( btreenode *sr ) ;
```

Example 19: Binary Tree

Step 2: Create a C program called TreeTraversal.c, and include the following code into it:

```
//include all header files
#includeTreeTraversal.h”

//Initializes tree
void InitTree( struct btreenode *root) ;
{
<<To Do>>
}

// build tree by calling insert( )
void buildtree (btreenode **root,int num )
{
```

```
//in buildtree we are changing the address of root this changed //address we want to get back
in main function. So we want to //point to root and not contents of root hence using
btreenode //**root
        insert ( root, num ) ;

}

// inserts a new node in a binary search tree
void  insert ( btreenode **sr, int num )
{
        if ( *sr == NULL )
        {
                *sr = new btreenode ;

                ( *sr ) -> leftchild = NULL ;
                ( *sr ) -> data = num ;
                ( *sr ) -> rightchild = NULL ;
                return ;
        }
        else  // search the node to which new node will be attached
        {
                // if new data is less, traverse to left
                if ( num < ( *sr ) -> data )
                        insert ( & ( ( *sr ) -> leftchild ), num ) ;
                else
                        // else traverse to right
                        insert ( & ( ( *sr ) -> rightchild ), num ) ;
        }
        return ;
}

// traverses btree
void traverse( btreenode *root)
{
        Printf("\nIn-order   Traversal: ") ;
        inorder ( root ) ;
        printf("\nPre-order  Traversal: ") ;
        preorder ( root ) ;

        printf("\nPost-order Traversal: ") ;
        postorder ( root ) ;
}

// traverse a binary search tree in a LDR (Left-Data-Right) fashion
void inorder ( btreenode *sr )
{
        if ( sr != NULL )
        {
                inorder ( sr -> leftchild ) ;

                // print the data of the node whose
                // leftchild is NULL or the path
                // has already been traversed
                printf("\t %d", sr -> data );
                inorder ( sr -> rightchild ) ;
        }
        else
                return ;
}

// traverse a binary search tree in a DLR (Data-Left-right) fashion
void preorder ( btreenode *sr )
{
        if ( sr != NULL )
        {
                // print the data of a node
                Printf("\t  %d", sr -> data) ;
                // traverse till leftchild is not NULL
                preorder ( sr -> leftchild ) ;
                // traverse till rightchild is not NULL
                preorder ( sr -> rightchild ) ;
        }
        else
                return ;
}

// traverse a binary search tree in LRD (Left-Right-Data) fashion
void postorder ( btreenode *sr )
{
        if ( sr != NULL )
        {
                postorder ( sr -> leftchild ) ;
                postorder ( sr -> rightchild ) ;
                printf( "\t  %d",sr -> data );
        }
        else
                return ;
}

// deletes nodes of a binary tree
void del ( btreenode *sr )
{
        if ( sr != NULL )
        {
                del ( sr -> leftchild ) ;
                del ( sr -> rightchild ) ;
        }
        free(sr);
}

void main( )
{
        btreenode *root ;
        int req, i = 1, num ;

        printf("Specify the number of items to be inserted: ") ;
```

```
            scanf(“%d”,&req) ;

            while ( i++ <= req )
            {
                        Printf( "Enter the data: " );
                        Scanf(“%d”, num) ;
                        buildtree (&root, num ) ;
            }

   traverse(root ) ;
   printf(“Do you want to delete the tree?(y/n)”);
   scanf(“\n%c”,&ans);
   if(ans==’Y’) || (ans==’y’)
        del(root);
}
```

Example 20: Binary Tree Traversal

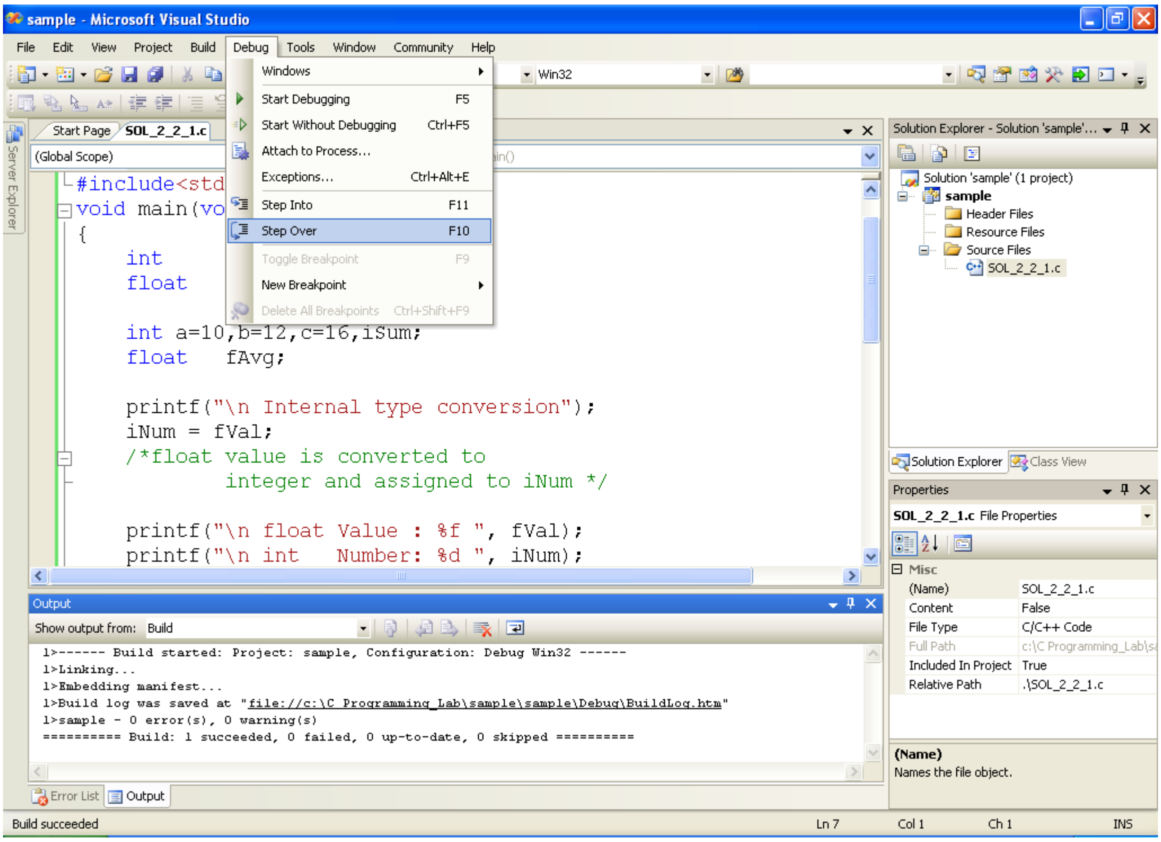
Assignment 1: Job Queue <<To Do>>

Step 1: Create a file word.txt. The file includes various words one word on each line. Read all words from file arrange it in binary tree format. Accept a new word from user and search if it is there in the file. Display appropriate message.

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 Debug -> Step over menu or press F10.



```
#include<std
void main(vo
{
     int
     float

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

     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);
```

```
1>------ Build started: Project: sample, Configuration: Debug Win32 ------
1>Linking...
1>Embedding manifest...
1>Build log was saved at "file://c:\C Programming_Lab\sample\sample\Debug\BuildLog.htm"
1>sample - 0 error(s), 0 warning(s)
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

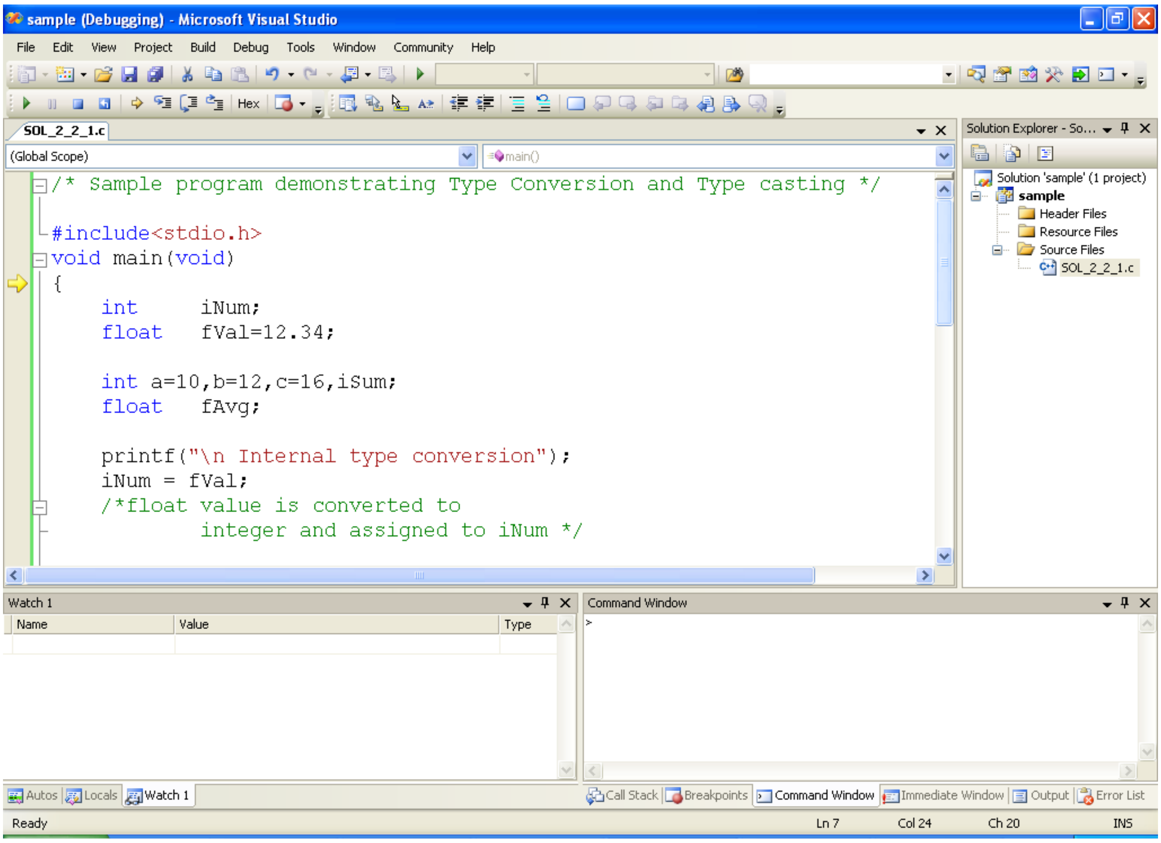
(Name)	SOL_2_2_1.c
Content	False
File Type	C/C++ Code
Full Path	c:\C Programming_Lab\s
Included In Project	True
Relative Path	.\SOL_2_2_1.c

(Name)
Names the file object.

Build succeeded   Ln 7   Col 1   Ch 1   INS

Figure 2: Step Over

2. The execution will start from main (). You can continue the execution by pressing the key, F11 every time. The line of code, which is going to get executed next, is highlighted.

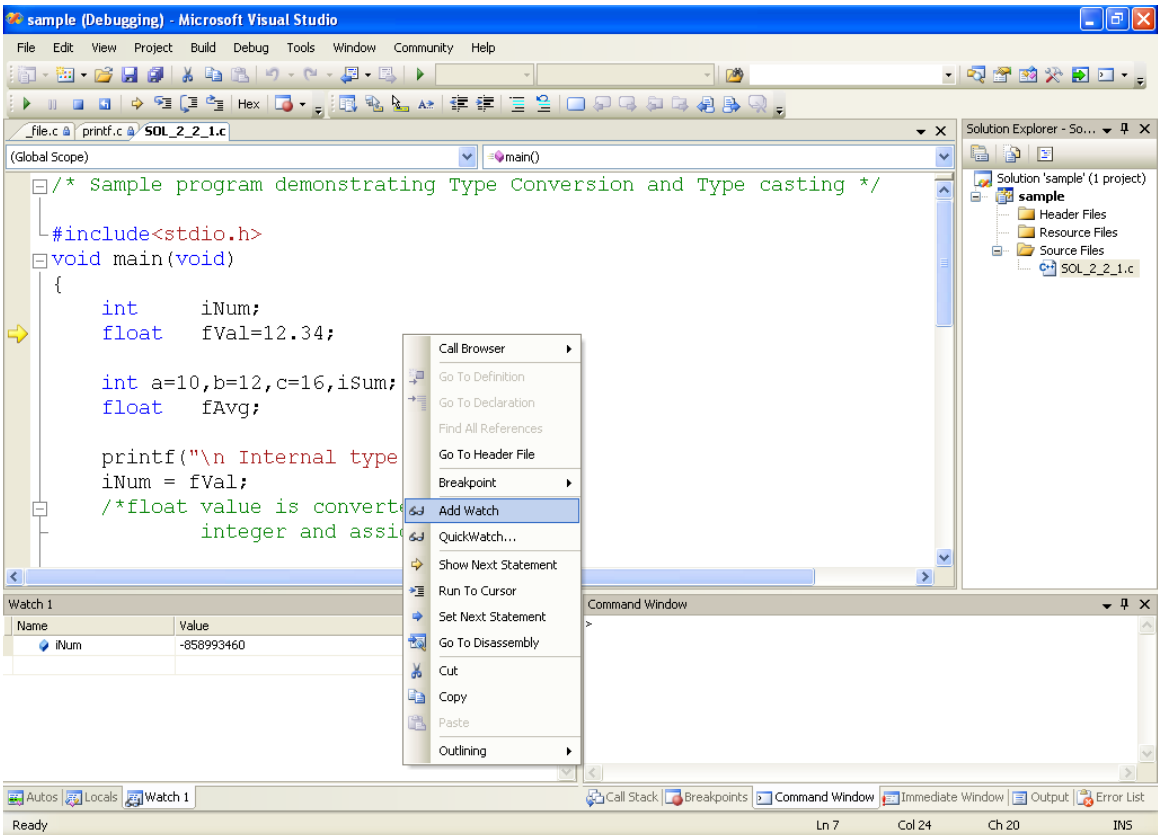


```
/* Sample program demonstrating Type Conversion and Type casting */

#include<stdio.h>
void main(void)
{
    int     iNum;
    float   fVal=12.34;

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

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

Figure 3: Start Debugging

i) Whenever there is a function call, choose Debug -> StepIn or press F11 to step into the function code. To skip the step-by-step execution of a function you may continue with F5.

ii) 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, right click on the editor in debug mode and select Add Watch from popup menu.



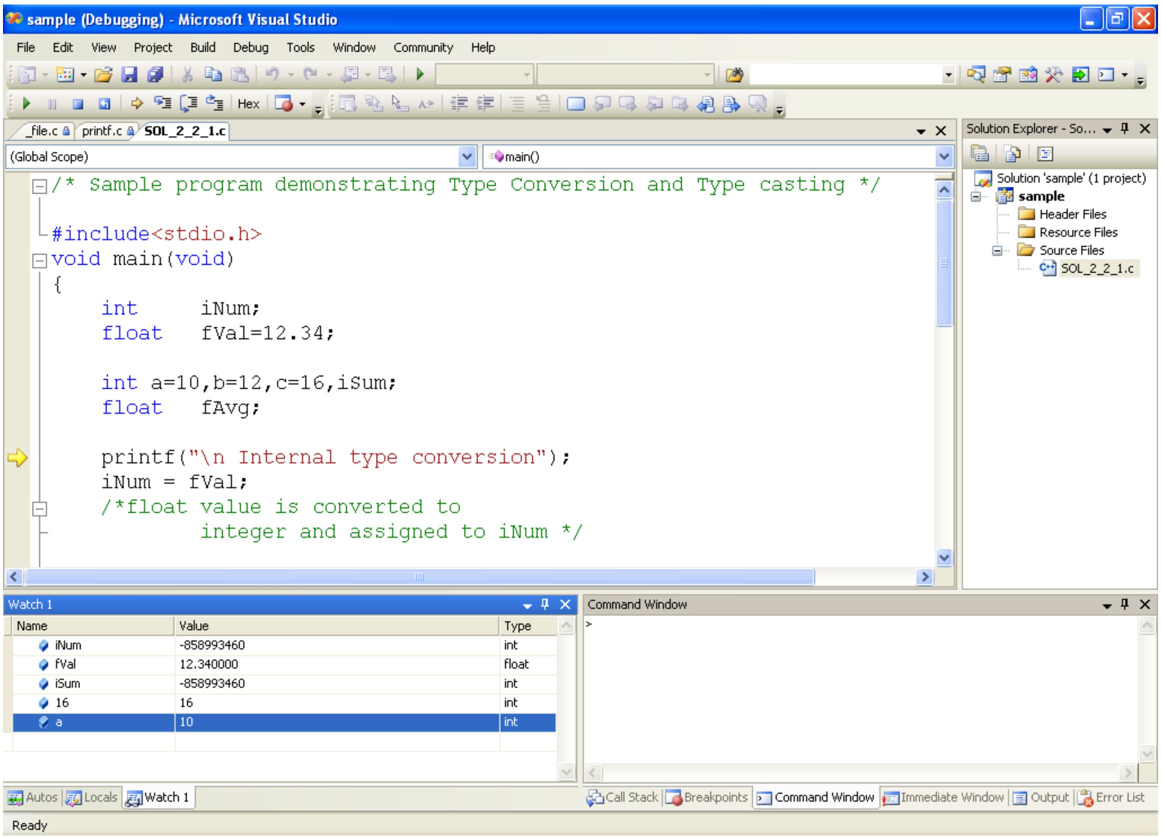
```
/* Sample program demonstrating Type Conversion and Type casting */

#include<stdio.h>
void main(void)
{
    int      iNum;
    float    fVal=12.34;

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

    printf("\n Internal type
    iNum = fVal;
    /*float value is converte
            integer and assi
```

Watch 1

Name	Value
iNum	-858993460

Figure 4: Add Watch Menu

iii) 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.


sample (Debugging) - Microsoft Visual Studio
File Edit View Project Build Debug Tools Window Community Help
Hex
_file.c printf.c SOL_2_2_1.c
(Global Scope)
main()
Solution Explorer - So...
Solution 'sample' (1 project)
sample
Header Files
Resource Files
Source Files
SOL_2_2_1.c
Watch 1
Command Window
Autos Locals Watch 1
Call Stack Breakpoints Command Window Immediate Window Output Error List
Ready

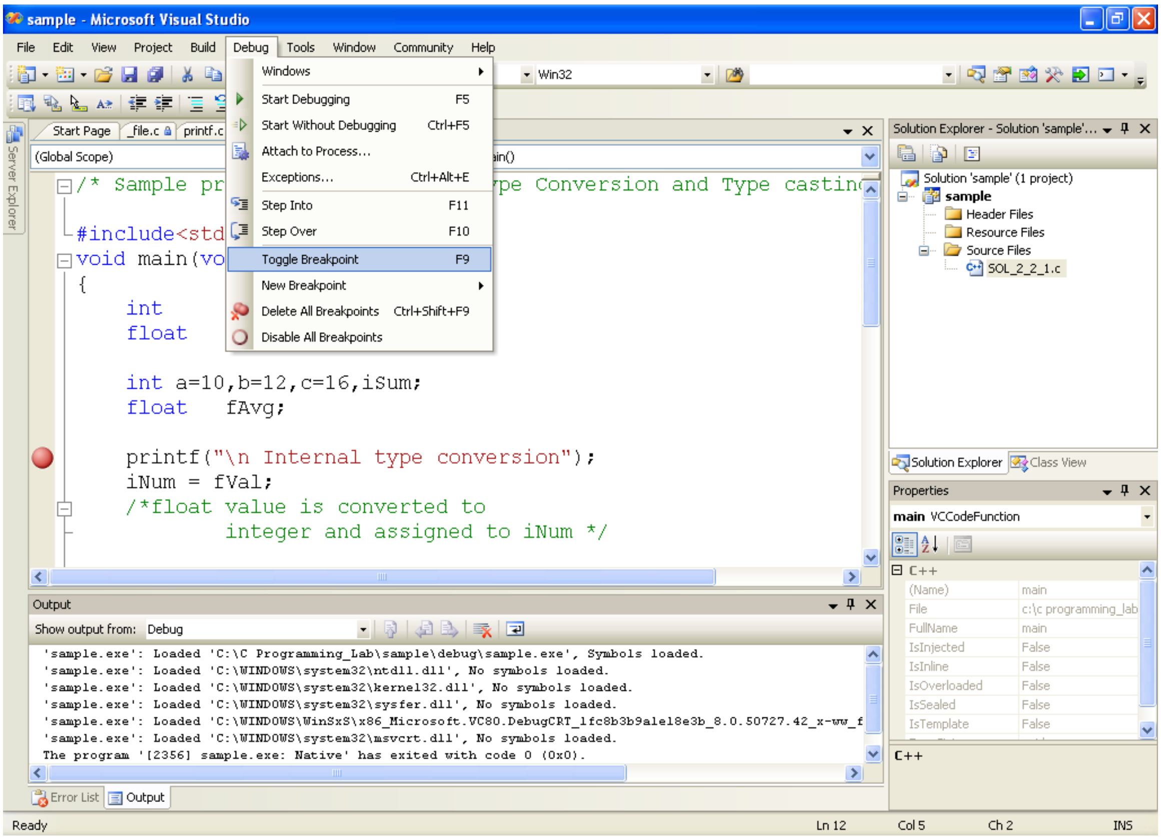
```
/* Sample program demonstrating Type Conversion and Type casting */

#include<stdio.h>
void main(void)
{
    int     iNum;
    float   fVal=12.34;

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

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

Name	Value	Type
iNum	-858993460	int
fVal	12.340000	float
iSum	-858993460	int
16	16	int
a	10	int

Figure 5: Watch Window

v) At any point of time, if you want to skip the step-by-step execution, and continue with the normal execution, choose the Debug -> Start without Debugging or press Ctrl+F5.

vi) You can insert a break –point at a particular statement in the code by selecting Debug -> Toggle breakpoint menu option or by pressing F9 while your cursor is placed at that statement.



```
/* Sample pr                       pe Conversion and Type castin

#include<std
void main(vo
{
    int
    float

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

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

Output

Show output from: Debug

```
'sample.exe': Loaded 'C:\C Programming_Lab\sample\debug\sample.exe', Symbols loaded.
'sample.exe': Loaded 'C:\WINDOWS\system32\ntdll.dll', No symbols loaded.
'sample.exe': Loaded 'C:\WINDOWS\system32\kernel32.dll', No symbols loaded.
'sample.exe': Loaded 'C:\WINDOWS\system32\sysfer.dll', No symbols loaded.
'sample.exe': Loaded 'C:\WINDOWS\WinSxS\x86_Microsoft.VC80.DebugCRT_1fc8b3b9a1e18e3b_8.0.50727.42_x-ww_f
'sample.exe': Loaded 'C:\WINDOWS\system32\msvcrt.dll', No symbols loaded.
The program '[2356] sample.exe: Native' has exited with code 0 (0x0).
```

Figure 3: Breakpoints

vii) 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.

viii) To remove the breakpoint, place the cursor at the statement having the break point and select Debug -> Toggle breakpoint menu or press F9.

ix) To remove all the break points from the code, select Debug -> Delete all breakpoints menu option.

x) At any point of time, if the program execution hangs you can return to Edit window from the User screen by selecting Debug -> Stop debugging or by pressing Shift+F5.

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.

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:

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.

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.

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.
    a. 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.
    - Standardize use of abbreviations.
    - Avoid use of obscure abbreviations.
    - Do not abbreviate InputCharacter as inch.
    - Abbreviate common part of mnemonics
    - 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

    i. p ------ pointer
    ii. c ------ char
    iii. I, 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.
3. for (i = 0; i < ARRBOUND; i++)
4. All the symbolic constant names should be in capitals only. E.g.
5. #define ARRBOUND 25 /* comment */
6. *The enumeration data type should be used to declare variables that take discrete set of values. e.g.
7. 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;
```

Programming style:

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;
```

White Spaces:

i) Add space before and after all the binary operators except "." and "->" for structure members.

ii) Unary operators should not be separated from their single operand.

iii) Horizontal and vertical spacing is equally important.

iv) After each keyword one space is required.

v) 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)
{
        ..........
}
```

vi) Use angle brackets to include a system header file
- Angle brackets should be used to include system header files.
  #include <someFile.h>

vii) Use quotes to include a user defined header file
- Quotes should be used to include user header files.
  #include "someFile.h"

viii) Every “if” statement written with the use of {} braces is the best practice. It will ensure the maintainability and readability of the code.

```
// Method 1
if (condition)
function1(a, b);
for (i < 7 && j > 8 || k == 4)
function2(i);
```

In Method1, naming conventions is not followed and {} braces is not used.

```
// Method 2
if (condition)
{
    get_details(emp_id, name);
}
for ((i < 7) && ((j < 8) || (k == 4)))
{
    display_info(emp_id);
}
```

In Method2, We have named functions using a verb (get, display), used variable names that describe the data they contain, and used brackets to help show what the for condition is. Not only that, but we have also included the braces for the control structures and used indentation to show which code appears under each structure.

Comments:

Comments are intended to help the reader understand the code. It is necessary to document source code. The following rules should be followed to achieve this:

- Keep code and comments visually separate.
- Use header comments for all files and functions.
- Use block comments regularly. Use trailing comments for special items.

File Header:

Each file shall begin with a File Header. The following file header template shall be used:

```
/**********************************<File
Header>**************************************/
/*!
 @file <filename>
 @brief Abstract: <Abstract>
          [There must always be one line between the abstract & description]
 Description: <Description>
 @note <Note>
 @todo <Todo>
 Author :<author
 Date: <date>
 Change History : <change history>
*/
/*****************************************************************************************
**/
```

Example:

```
/***********************************<File
Header>***************************************/
/*!
 @file SystemManager.c
 @brief Abstract: This file contains the implementation of the SystemManager.

 Description: The SystemManager provides a generalized mechanism to report
 various system conditions.
 @note Refer to System Manager Macros section in the FTS Software Coding Standards
 for more information about when to use the System Manager.
 @todo Unit test and review the code.
 */
/*****************************************************************************************
**/
```

Function/Method Headers

Each function or method shall be preceded by a function header. The following function header template shall be used:

```
/**********************************<Function
Header>************************************/
/*!
 @fn <FunctionName>
 @brief Abstract: <Abstract>
[There must always be one line between the abstract & description]
 Description: <Description>
 @param <Parameter> <ParameterDescription>
 @return <Return>
 @remarks <Remark>
 @warning <Warning>
 @see <SeeAlso>
 */
/*****************************************************************************************
**/
```

Example:

```
/************************************<Function
Header>**************************************/
/*!
 @fn factorial
 @brief Abstract: This method accepts a limit number, and prints the factorial of a number
 @param limit Limit of the factorial number.
 @return This method returns the result of the factorial of a number
  @warning **NOT_UNIT_TESTED**
 */
/*********************************************************************************************
**/
```

Structure Header

1) Each structure and union shall have a header.

2) The following class header template shall be used:

```
/**************************************<Structure
Header>***************************************/
/*!
 @struct <structurename>
 @brief Abstract: <Abstract>
          [There must always be one line between the abstract & description]
 Description: <Description>
 @remarks <Remarks>
 @warning <Warning>
 @see <SeeAlso>
 */
/*********************************************************************************************
**/
```

You can substitute the keyword union instead of struct for union header

Appendix III - Table of Figures